LA INSEGURIDAD DE LAS CONTRASEÑAS SEGURAS

Manual del correcto desarrollo de plataformas de registro

DIEGO CELDRÁN MORELL

Índice

Pueden revisar el post resumido y simplificado desde:
http://www.diegoceldran.es/la-inseguridad-de-las-contrasenas-seguras/

En el día de hoy, daremos una vuelta por lo que coloquialmente se conoce como las mayores metidas de pata de los desarrolladores web y las geniales ideas que se han tenido a lo largo del tiempo para hacer más segura la estancia de los usuarios en nuestras aplicaciones web.

Metodologías

Primeras metodologías

Creo que sobra hablar sobre las primeras metodologías en el desarrollo de este tipo de interfaces de login las cuales se remontan incluso a antes del descubrimiento del tan apreciado SQL Injection el que apareció entorno al 1998, es decir, hace ya casi 20 años y sigue siendo la vulnerabilidad más común en entornos web.

Esta ya de por sí, hicieron inútiles le seguridad de millares de aplicaciones web, pero para el caso de hoy supongamos que ya no hay nadie tan inútil de no colocar un mísero addslashes() en el formulario de login de una web PHP.

Las metodologías de por aquel entonces, en un para aquel entonces infantil internet, fueron los primeros pasos para lo que serían más de 2 décadas de lo que ahora se conoce como metodologías ágiles que son principalmente lo que nos dan trabajo a los hackers, por lo que tal vez tengamos que estarles agradecidos.

La visión de por aquel entonces era la de quiero que los usuarios se registren, pues pongo un panel de registro y ya está, y cada cual que coloque la contraseña que ellos quieran.

Esto produjo una cantidad ingente de usuarios que terminaban utilizando contraseña que les fuese fácilmente recordables y lo más sencillas posibles para no olvidarse de ella, esto es un hecho que cabría recalcar que se sigue dando en muchas ocasiones en la actualidad, estas contraseñas tan usuales son las que ya todos conocemos y absolutamente todo el mundo (casi todos al menos) saben que no se han de utilizar.

Por esta época, “hackear” una cuenta de alguien era tan fácil como probar las contraseñas más comunes y utilizadas para estas situaciones.

contraseña password 1234 (nickname) “ “

Y pocas más, o sino, siempre podías optar por utilizar un diccionario de palabras comunes o directamente uno de palabras aleatorias y realizar un ataque de fuerza bruta.

Recordemos que nos estamos remontando a antes del sql injection, sino también podríamos añadir el ‘ or ‘1’=’1 a la lista de las contraseñas maestras para acceder a absolutamente cualquier web con cualquier usuario.

No obstante, y “por suerte” las cosas han ido evolucionando para adaptarse a los nuevos tiempos con las nuevas amenazas de internet.

Metodologías un poco más actuales

En mi humilde opinión, internet no ha evolucionado tanto como la gente acostumbra a creer, o mejor dicho, internet sí que ha evolucionado, y muy deprisa además, el problema es que ha evolucionado más rápido de lo que lo han hecho las mismas personas que lo usan.

Con la llegada de XSS, SQL Injection, LDAP Injection y al menos una docena más de vulnerabilidades algo más modernas, los programadores más metidos en el ámbito de la seguridad han ido adaptándose para crear códigos cada vez más seguros al igual que los hackers por crear técnicas cada vez más avanzadas.

Tampoco es que se esforzaran mucho porque recordemos que una de las medidas de seguridad que se implementaron para evitar los sql injection fue el magic_quotes_gpc que básicamente ejecutaba un addslashes() en todas las peticiones GET, POST y COOKIE, por ende, la manera de evitar un SQL Injection fue actualizar PHP, por lo que si tenías una aplicación antigua que no usaba addslashes() y actualizabas te protegía por defecto las configuraciones de PHP ya que estuvo activada por defecto hasta la versión 5.4, y si la usabas junto a esta se producía un doble escape, al fin y al cabo como podéis comprobar, es una solución a medias que se han de tomar cuando desarrollas software empleando unas metodologías de desarrollo de software débiles.

Como bien ya he dicho antes, SQL Injection es un problema, pero no tanto como los propios usuarios que manejan internet, con todas estas nuevas metodologías de “seguridad”, se solucionaron unos problemas, que fueron los de SQL Injection, LDAP Injection, XSS (en gran parte de la web al menos) pero se olvidaron de los usuarios y sus contraseñas y de lo débiles que estas eran, por lo que prácticamente la única medida que se empleó fue la de empezar a hashear las contraseñas de los usuarios en la base de datos para que si les realizan un SQL Injection y les realizan un Dump de la base de datos completa y roban toda la información, al menos que la tuvieran hasheada para que les costara un poco más averiguarla.

Esto, por otra parte resulta inútil, puesto que aún sin saber el método de hasheo, un atacante podría registrarse, obtener su hash de la base de datos, y probar con distintos algoritmos de hash para obtener el algoritmo que se está utilizando y más adelante realizar el mismo proceso generando hashes de contraseñas aleatorias y cotejándolas con las que ya se tienen para empezar a obtener todas las contraseñas sin hashear y al mismo tiempo aprovechar para crearte tu propia base de datos con hashes y las contraseñas sin hashear y el algoritmo para utilizarla más tarde con otras bases de datos.

También hemos de recordar que hasta hace, relativamente poco, lo más frecuente era hashear la contraseña en md5 que posteriormente se comprobó que es vulnerable a ataques por colisión, es decir, hay más de una clave con el mismo hash la cual se puede terminar por romper.

También cabría recalcar que, en teoría, cualquier hash es vulnerable a ataques por colisión puesto que basan en devolver una cadena de una longitud fija, por lo que, en teoría, para por ejemplo un sha256 devuelve una cadena de una longitud de 256 bits, o 32bytes, que representados en hexadecimal suponen 64 caracteres (en hexadecimal) si nosotros empezásemos a sacar la combinatoria de todas las contraseñas posibles de 33 caracteres o más, en teoría se debería de repetir, pero claro, 256^33 suponen algo menos de 3 * 10^79, demasiadas contraseñas como para probarlas todas (por ahora).

Metodologías “actuales”

A día de hoy, ya sabemos cómo evitar ataques SQL Injection, XSS, LDAP Injection, LFI, RFI, XSF, CSRF, XFS, XZS, XAS, XRS, XSSDoS, etc…

O al menos se supone que ya deberíamos de saber, por el momento evidenciaremos el hecho que la mayoría de las personas en general y una gran parte de la gente que desarrolla aplicaciones web no sepa que son entre al menos una y ninguna de estas vulnerabilidades.

En la actualidad ya somos un poco más conscientes de que hemos de proteger a los usuarios, no solo de posibles atacantes y amenazas en general sino también protegerlos de ellos mismos, hecho casi, casi, casi imposible de realizar ya que para la mayoría de vulnerabilidades hoy en día basta con poco más que utilizar una función, en PHP por ejemplo addslashes() para las cadenas o intval() para los valores enteros, sim embargo, para proteger a un usuario de sí mismo, es bastante más difícil.

En la actualidad, hemos ido desarrollando una serie de estrategias y metodologías pensando siempre en que el usuario utilice contraseñas cada vez más difícil, cosa que más tarde explicaremos porque no es algo seguro, ni tan si quiera es una solución válida para proteger a nuestros usuarios.

Recuerden están palabras:

“Reglas de seguridad mínimas”

Si han trabajado con SQL recordarán el “WHERE” que se utiliza filtrar resultados poniéndoles condiciones, por ejemplo:

“SELECT * FROM products WHERE price > 8 and price < 12”.

Y sí, soy totalmente consciente de que se podría haber utilizado el operador BETWEEN, pero para el ejemplo es irrelevante.

Las condiciones mínimas de seguridad que se suelen aplicar a los formularios de registro son los que veremos aquí a continuación:

- La longitud mínima de la contraseña ha de ser de al menos 8 caracteres.
- La contraseña ha de contener al menos un número.
- La contraseña ha de contener al menos una minúscula.
- La contraseña ha de contener al menos una mayúscula.

Y por lo general también es habitual toparse con sitios en donde también has de incluir un carácter especial en tu contraseña.

Mediante esta serie de filtros, hemos intentando asegurarnos de los usuarios que se registran en nuestras aplicaciones utilizan contraseñas que, a priori, deberían de suponer un problema para un atacante que intentase averiguar la contraseña de uno de nuestros usuarios, y digo, “a priori” porque tal y como veremos a continuación no siempre es algo efectivo y tal y como demostraré después, puede resultar una ayuda al atacante.

Primero, veamos qué es exactamente lo que se trata de conseguir con esta serie de técnicas en nuestras aplicaciones web.

Para un atacante, realizar un ataque de fuerza bruta supone probar muchas contraseñas, ya sean por diccionario o generando contraseñas aleatorias, en matemáticas hay una cosa llamada combinaciones y permutaciones que son básicamente lo mismo solo que la diferencia entre ambas es que en la combinación no importa el orden y en las permutaciones si, y dentro de las permutaciones existen las permutaciones con repeticiones y sin ellas.

Lo que se trata de conseguir con utilizar una contraseña de 8 caracteres es que se tengan que probar las contraseñas con una longitud de hasta, al menos 8 caracteres, ahora con esto en mente pongamos nos en el lugar de un atacante y hagamos unas cuentas para ver a ver cuántas contraseñas hay que testar para asegurarnos de dar con una con una longitud de 8 caracteres, esto, en matemáticas, supondría que le número de contraseñas a testar sería (n+r-1)!/r!(n-1)! donde n es el número de caracteres a usar en la combinación y r, en este caso la longitud de la cadena que en este caso son 8 caracteres.

En conclusión:

Para calcular el número de contraseñas a testar:

$$x = \frac{(n + r - 1)!}{r!\,(n - 1)!}$$

Donde:

x = número de combinaciones posibles.
n = número de caracteres a utilizar en la permutación.
r = longitud se la cadena a escoger.

Ahora, vamos a calcular.

Para una contraseña de 8 caracteres tendríamos:

Solo para números:
$x = \frac{(10+8-1)!}{8!9!}$ = 24.310 combinaciones

Solo para minúsculas o mayúsculas:
$x = \frac{(28+8-1)!}{8!27!}$ = 23.535.820 combinaciones

Solo para minúsculas y mayúsculas:
$x = \frac{(56+8-1)!}{8!55!}$ = 3.872.894.697 combinaciones

Para minúsculas, mayúsculas y números:
$x = \frac{(66+8-1)!}{8!65!}$ = 13.442.126.049 combinaciones

Para los 256 caracteres de ASCII:
$x = \frac{(256+8-1)!}{8!255!}$ = 509.850.594.887.712 combinaciones

En la mayor parte de los casos supondría 13.442.126.049 combinaciones para cubrir la mayoría de los casos, y aún bastante más si también le incluimos que tenga caracteres especiales, estas son un montón de contraseñas posibles, y se disparan si ampliamos aún más la longitud de la contraseña, por pura curiosidad si calculásemos el máximo de combinaciones posibles que

podríamos realizar antes de encontrar otro hash exactamente idéntico con el algoritmo de hash sha256 sería:

$$x = \frac{(256 + 32 - 1)!}{32!\,255!}$$

Para los que no tengáis ganas de echar las cuentas eso equivale a 2.83 * 10^42 combinaciones posibles, es decir, casi un 3 seguido de 42 ceros.

Esto, ya de por sí, haría prácticamente inviable realizar cualquier ataque de fuerza bruta, y más teniendo en cuenta que existe un tiempo de respuesta entre el cliente-servidor, lo cual lo convierte en una limitación física, puesto que si se pudiesen realizar 100 peticiones por segundo a un servidor, para una contraseña de 8 caracteres con minúsculas, mayúsculas y números, supondría una demora de 1.555,8 días, o lo que es lo mismo, algo más de 4 años.

A parte de estas limitaciones, muchos optan también por establecer medidas como los tan conocidos “max attempt login” que son los errores máximos que se permite a un usuario equivocarse en su login antes de bloquearlo, esto generalmente se lleva a cabo bloqueando la ip del equipo que se está intentando loguear durante “x” tiempo.

Metodologías del atacante

Las metodologías del atacante, son los distintos pasos que un atacante sigue a la hora de intentar conseguir acceso (en este caso a una plataforma web).

Los ataques contra formularios, han ido evolucionando con el paso del tiempo, y ya no es tan común ver a servidores probando millones de contraseñas para obtener acceso a la cuenta de una persona, para un atacante, suele haber 2 formas distintas de obtener acceso a la cuenta de un usuario, obviando las que requieren de un ataque directo al servidor o a la plataforma como tal.

Ahora, redactaremos las distintas formas que tiene un atacante de obtener acceso suplantando así a un usuario.

Forma fácil

Para un atacante, la forma más sencilla de obtener una contraseña, es haciendo que él propio usuario se la dé, y para esto existe algo denominado ingeniería social.

La ingeniería social básicamente son un conjunto de técnicas que nos permiten a nosotros engañar a un usuario para conseguir que nos dé su contraseña, aún sin el siquiera saberlo, por ejemplo, creando una web de fishing como si fueran nuestra página, realizarnos un DNS Spoofing desde su propia red y conseguir que la web que visite sea la del atacante en vez de la nuestra y consecuentemente una vez que inicie sesión almacenar la contraseña.

De esta manera tan sencilla el atacante conseguiría tanto el nombre de usuario como la contraseña del usuario en cuestión.

Dentro de la ingeniería social hay miles de formas distintas, por lo que resultaría prácticamente imposible abarcarlas todas en este paper.

Otra forma también muy común en este tipo de situaciones es sencillamente troyanizar a la víctima, de tal manera que el atacante pueda tener acceso completo a todo lo que realiza la víctima, inclusive tener acceso a la contraseña almacenadas en el navegador.

Existen también centenares de herramientas para realizar esto, incluyendo SpyNet que nos permite crear nuestra propia botnet de equipos infectados con nuestro malware.

También existen otras como por ejemplo realizar un ataque Man in the Middle (Ataque de hombre en el medio), un Man in the Browser (Ataque de hombre en el navegador), etc… pero todos ellos se basan básicamente en robar le la contraseña al usuario para posteriormente acceder como ella en nuestra plataforma, además, la mayoría de los usuarios reutilizan sus contraseñas por lo que de hackear otro servidor que no sea el nuestro y obtener su nombre de usuario y/o su contraseña de allí, es muy posible que también pueda acceder un atacante en nuestra web como si de este se tratara.

Y por si esto fuera poco, a menudo estas listas de usuarios acaban en internet, y a menudo resulta sencillo encontrar listas enteras de usuarios y contraseñas públicos en internet haciendo tan solo un poco de Google Hacking, por ejemplo, si buscamos por:

program*url host login password allintext:"@outlook.com"

De inmediato nos saldrán una gran cantidad de resultados donde aparecerán contraseñas y usuarios de gente con una cuenta de correo electrónico de *@outlook.com, y como bien ya he dicho antes, la gente suele reutilizar sus contraseñas por lo que si lo encontramos aquí no sería raro pensar que también valdría para nuestra plataforma.

Y como esto, también sería posible encontrar webs que se dedican profesionalmente a la venta de listas enteras de usuarios y contraseñas donde puede que tú también estés.

Forma sofisticada

La forma sofisticada es otra también muy común, que es la que se basa en realizar búsquedas a través de internet para información acerca del usuario en cuestión y confeccionar diccionarios de contraseñas en función a los datos que conocemos sobre este usuario, y gracias a redes sociales, blogs y demás, casi la totalidad de la información acerca de un usuario la podemos encontrar en escasos minutos buscando un poco por internet.

Recordemos que estamos hablando de un formulario en donde se requiere un usuario y contraseña, método que tal y como veremos es seguramente el menos recomendable de ellos.

Este tipo de técnicas de recolección de información se denominan footprinting y fingerprinting y existen centenares de formas y herramientas distintas para realizarlas, básicamente consisten en una búsqueda de información a través de internet sobre algo en concreto, la principal diferencia entre ambos, es que el footprinting se realiza mediante la búsqueda de información desde principalmente internet y en el fingerprinting este proceso se toma para obtener información directamente sobre los propios servidores de una organización, este paso es mayoritariamente llevado a cabo para auditorías o ataques directamente sobre un servidor o aplicación más que para un ataque dirigido a una determinada persona.

Algunas herramientas y métodos que nos ayudan a realizar esta tarea de footprinting pueden ser:

- Maltego
- Foca
- Parámetros avanzado de Google, Bing, etc… (también conocidos como dorks)
- Facebook
- Twitter
- Instagram
- (y en general cualquier red social)

E incluso mediante el análisis de servidores y documentos en los que esta persona haya formado parte, de ahí la importancia de los metadatos de los documentos e imágenes puesto que pueden contener nombres de usuarios, fechas, ubicaciones, etc…

Y del lado del fingerprinting podríamos tener:

- Nmap
- Algunos módulos de Metasploit
- Y un largo etc…

Inclusive revisar en Github y tal vez hasta la misma herramienta creada por mí, BrutiFramework pudiesen servir de ayuda para la búsqueda y recogida de información.

Una vez que se ha recogido toda la información podemos volver a lo mismo de antes, a las combinaciones y permutaciones, en este caso combinaciones sin repeticiones, en esta podríamos poner todos los datos que hubiésemos encontrados y crear un diccionario con palabras y combinaciones de todo lo que hemos ido recolectando, inclusive mi propia herramienta, BrutiFramework, tiene un módulo para la creación de diccionarios a partir de estos datos que hace este proceso de manera totalmente automatizada.

No obstante, a pesar de crear nuestro diccionario, podríamos contar con que actualmente la mayoría de los usuarios utiliza las mismas técnicas para crear sus contraseñas, pero de esto ya hablaremos más adelantes, por el momento, creemos el diccionario.

Para calcular el tamaña de nuestro diccionario con la mayor precisión posible podemos utilizar la fórmula de las permutaciones sin repeticiones para calcular cuántos resultados nos saldrían, su fórmula es la siguiente:

$$x = \frac{n!}{(n-r)!}$$

Una formula sencilla en donde las letras equivalen a lo mismo que antes.

Y en nuestra recopilación de información escasamente utilizaremos unos 20 datos diferentes tales como su nombre, apellidos, fecha de nacimiento, nombre de sus hijos, fecha de nacimiento de sus hijos fecha en la que se casó, año actual, color favorito, etc… esto ya es hilar muy muy fino, también podamos suponer el agregarle 15 caracteres especiales de los más comunes que se suelen utilizar, ¡!¿?-_.,"'#@$/& pese a que las más habituales son .-_ y eso en la mayoría de los casos, primero hagamos los cálculos con los 20 datos y los 15 caracteres, en la combinación habrá unos 35 caracteres posibles y vamos a hacer los cálculos con una longitud de 1 a 4 para cubrir de lejos todos los datos posibles, a eso le podemos quitar los 15 caracteres de las 15 contraseñas que saldrán ellos solos, por lo tanto se quedaría que:

$$x = 35 + \frac{35!}{(35-2)!} + \frac{35!}{(35-3)!} + \frac{35!}{(35-4)!} - 15$$

Eso equivalen a 1.297.120 de posibilidades teniendo casi un 100% de posibilidades de encontrar la combinación correcta, ahora bien, la mayoría de las contraseñas más seguras pueden presentar a lo sumo una combinación de entre 3 y 4 palabras, por ejemplo, AlfonsoPablo_1964, si se afina mucho un hacker con práctica solo probaría las contraseñas más posibles y muy posiblemente diese con la contraseña correcta en no más de 5-10 minutos.

Y como bien hemos dicho antes, hacer la combinación con esos 15 caracteres especiales es crear el diccionario queriendo que tener la contraseña si o si, pero si hacemos lo mismo solo con los 3 caracteres más probables:

$$x = 23 + \frac{23!}{(23-2)!} + \frac{23!}{(23-3)!} + \frac{23!}{(23-4)!} - 3$$

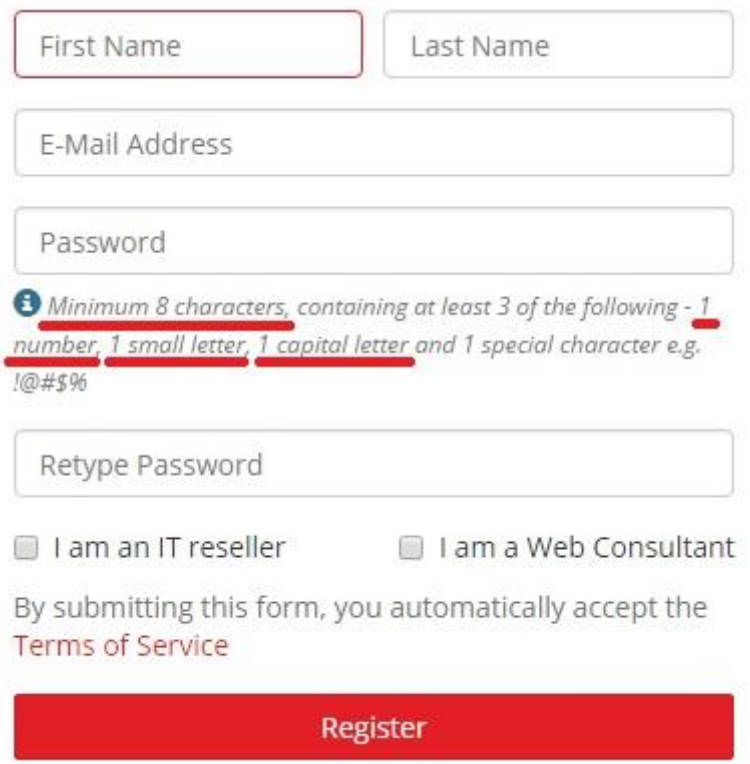
Se reduce directamente a tan solo una quinta parte de lo que teníamos antes dando a penas unas 223.672 algo que tal vez pudiese llevar un tiempo, pero tampoco tardaría demasiado en un ataque por diccionario y se podría cruzar por tor para que cambiase de nodo cada 3 intentos por lo que no tardaría demasiado tiempo en averiguar la contraseña de ese usuario.

Y esto para realizar un ataque de fuerza bruta, porque como ya bien digo, un hacker con experiencia podría sacarla en apenas 5-10 minutos.

Metodologías de desarrollo correctas

Llevar unas correctas metodologías en el desarrollo de un software, en este caso de una aplicación web, es algo fundamental para evitar más que probables riesgo en la seguridad de nuestros usuarios.

Dejando de lado el evitar vulnerabilidades SQL Injection, LDAP Injection, XSS entre otras muchas vulnerabilidades hemos de pensar en la seguridad del usuario frente a posibles amenazas como por ejemplo los ataques de fuerza bruta, ahora redactaremos correctas prácticas a la hora hacer más seguro los formularios de registro y login de nuestra aplicación web, pero para ello, primero veremos porqué puede suponer un riesgo para nuestros la implementación de los conocimos requisitos mínimos de seguridad de una contraseña.



Esta imagen es una captura recién sacada del formulario de registro de Acunetix, tal y como se puede apreciar tiene condiciones mínimas de seguridad para las contraseñas, esto traducido al lenguaje de un atacante significa, puedo quitarle contraseñas a mi lista de contraseñas ya que sé que todas las contraseñas de todos los usuarios registrados en Acunetix cumplen estas condiciones, de tal manera que podríamos descontar aprox. un 70% de las contraseñas de nuestro diccionario para quedarnos con muchas menos.

Esto, sería un ejemplo de algo que no es lo más recomendable para integrar en nuestro formulario de registro, por ejemplo, antes teníamos 223.672 combinaciones posibles, pues si contamos con que la contraseña ha de tener minúsculas, mayúsculas, números y un carácter especial sabemos que el número de valores a coger en la permutación, serían entre 3 y 4, como las probabilidades juegan a nuestro favor, primero testearíamos las de 3 combinaciones, por ejemplo, “Pablo“ “_” “1986” formarían parte de ese conjunto siendo esta a su vez una de las

combinaciones más probables que se suelen usar en contraseñas, por lo que si hacemos el cálculo nos saldrá que:

$$x = \frac{23!}{(23-3)!} - 3$$

Este, sería el conjunto de combinaciones más probable para un usuario registrado en esta plataforma.

Si realizamos el cálculo, nos darán 10.623 que suponen las contraseñas más probables para el usuario en cuestión al que estemos auditando, no obstante si nos fijamos en la imagen anterior pone 3 de las siguientes condiciones, por experiencia sé que la mayor parte de los usuarios que se registran en cualquier sitio y han de escoger al menos x de z condiciones, el mayoría de los casos cogerán un número x de condiciones, y más en concreto en este caso, tenderán a coger aquellos a lo que más acostumbrados, es decir, números, mayúsculas y minúsculas, por lo tanto nuestra ecuación podríamos reducirla a simplemente coger una longitud de 2, en vez de 3, transformando la ecuación de antes nos dará que:

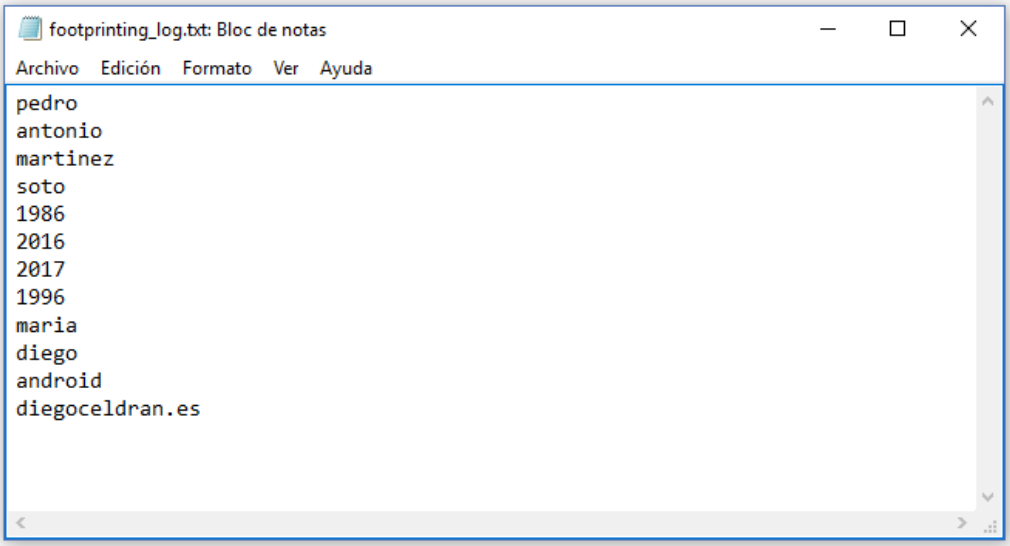
$$x = \frac{23!}{(23-2)!} - 3$$

Dan unos 503 resultados pese a que ese cálculo es incorrecto puesto que estamos teniendo en cuenta también las palabras todo en minúsculas, si estas no las tuviésemos en cuenta el número de palabra a escoger se reduciría de 23 a aprox. 16-18 palabras para realizar la permutación, si re-hacemos el cálculo nos dará lo siguiente:

$$x = \frac{18!}{(18-2)!} - 3$$

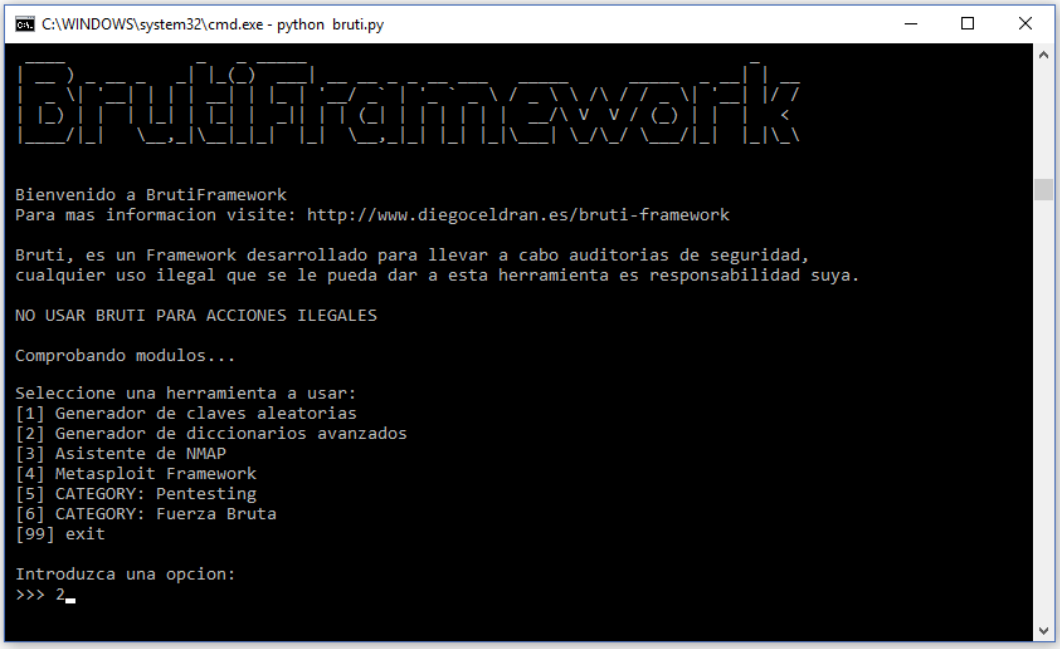
Para los de la LOMCE eso supone unas 303 combinaciones posibles.

Tal vez os preguntéis si realmente es tan sencillo realizar un diccionario con todas estas palabras, y la respuesta es sí, sin llegar más lejos con BrutiFramework (al final del paper tenéis los enlaces hace la(s) herramientas) podemos generar un diccionario de una forma muy sencilla, simplemente creamos un documento de texto con la lista de los datos sobre esa persona que tenemos, descargamos el módulo de “dictionary_advanced_maker” también desde la página web, lo agregamos a BrutiFramework, abrimos BrutiFramework, Iniciamos el módulo y sencillamente seguimos los pasos para la creación del diccionario.

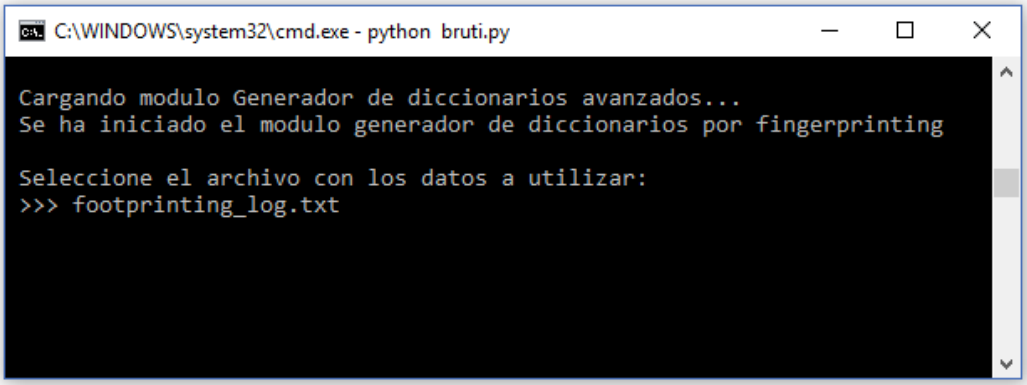


Ya hemos creado la lista de datos, Bruti automáticamente pondrá las mayúsculas y algún otro dato que el detecte que se suele utilizar para este tipo de situaciones y no están en la lista.

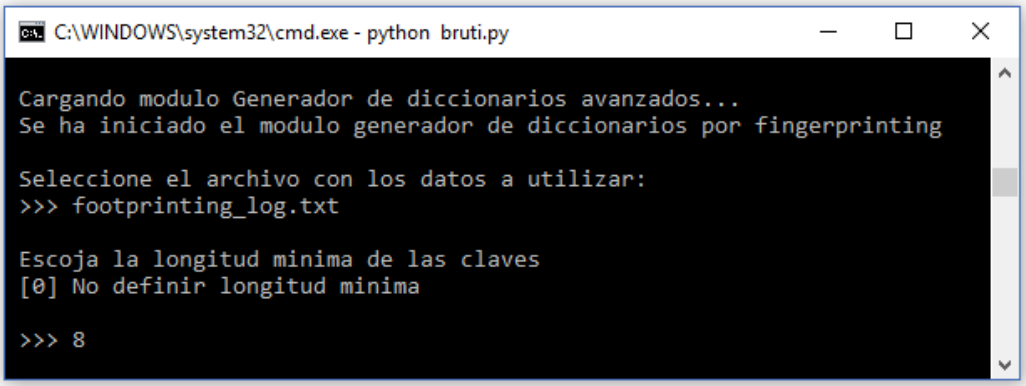
Iniciamos el módulo desde BrutiFramework.



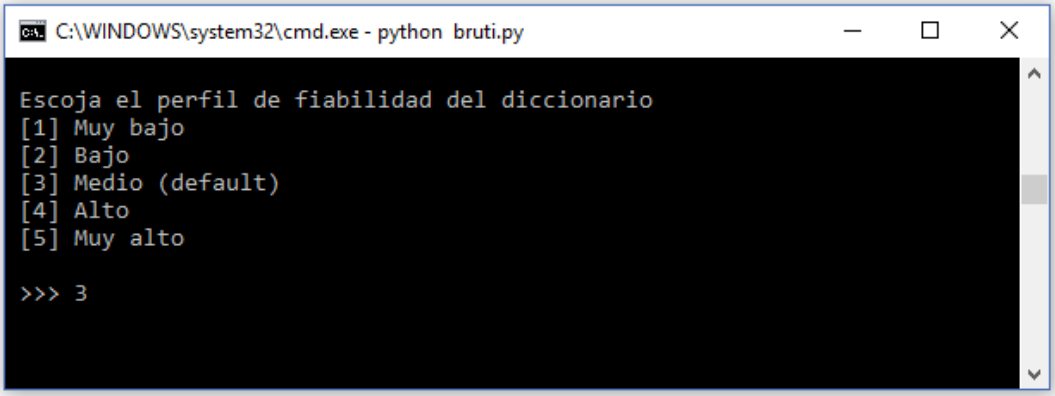
Le indicamos la ruta al archivo de nuestra lista de datos sobre el usuario:



Definimos la longitud mínima y máxima de las contraseñas:



Le indicamos el perfil de fiabilidad con el que queremos que trabaje:



Seleccionamos el archivo de salida y listo, ya tendríamos nuestro diccionario completamente creado con una fiabilidad bastante elevada de que la contraseña del usuario, pueda estar en nuestro diccionario.

A continuación, veremos unas correctas prácticas a la hora de crear un formulario de registro y login, también veremos el código en PHP.

Evitando ataques automatizados

Creo que no hace falta ni nombrarlo, es importante implementar sistema del tipo, “Max Login Attempts” para lo que existen distintas formas de realizar esto, pero, desde hace ya tiempo se tiene la costumbre de integrar sistemas como los captcha en los formularios de búsqueda, login, registro, etc… Esto evitará que ataquen nuestros sistemas con herramientas automatizadas y resultan bastante sencillos de integrar.

En este caso veremos brevemente como podemos integrar Captcha a nuestro formulario de login.

Lo primero será registrarnos en http://recaptcha.net/ simplemente nos pedirán unos pocos datos para registrarnos, una vez registrados obtendremos una clave pública y una privada especial para cada dominio, por ejemplo:

Public Key: XXXXXXXXXXXXXXXXXXXXXXXX-XXXXXXXXXXXXXXX
Private Key: XXXXXXXXXXXXXXXXXXXXXXXX-XXXXXXXXXXXXXXX

También nos hemos descargar desde el apartado “Resources” un zip con la librería y desde php incluimos el archivo recaptchalib.php con:

```
requiere_once(‘recaptchalib.php’);
```

Para mostrar el captcha podemos utilizar una función llamada recaptcha_get_html.

```
<?php
        requiere_once(‘recaptchalib.php’);
¿>
<form method=”post” action=”#”>
        <input type=”text” name=”input_login”/>
        <input type=”password” name=”input_pass”/>
        <input type=”submit” />
        <?php
                recaptcha_get_html($captcha_publickey, $error_captcha);
        ¿>
</form>
```

Así de fácil se integraría y para validar la información del formulario tenemos la función recaptcha_check_answer(). Esta función recibe también varios parámetros: La llave privada, la IP del usuario, y dos campos que contienen los valores que envía la captcha dentro del formulario $_POST["recaptcha_challenge_field"] y $_POST["recaptcha_response_field"].

Para validar podríamos hacer algo como esto:

```
$captcha = recaptcha_check_answer ($captcha_privatekey,
$_SERVER["REMOTE_ADDR"],
$_POST["recaptcha_challenge_field"],
$_POST["recaptcha_response_field"]);
if ($captcha -> is_valid) {
    // Captcha correcto
    // Ejecutamos nuestro código
} else {
    // Error al introducir el captcha.
    $error_captcha = $captcha -> error;
}
```

De esta manera podríamos evitar muchos de los ataques por fuerza bruta a los usuarios de nuestra aplicación de tal manera que si quisieran hacer este proceso, no les cabría otra más que introducir las combinaciones a mano, pero recordemos, que un atacante con experiencia podría sacar una contraseña en apenas 5-10 minutos, por lo que esta medida realmente no nos previene de todos los ataques.

La otra forma que tenemos de prevenir ataques de fuerza bruta es mediante los “max login attempts” para esto se pueden utilizar numerosos métodos, desde almacenar los registros en el .htaccess, almacenarlos y comprobarlos desde una base de datos MySQL o incluso utilizar Cookies o directamente usar bases de datos NoSQL para almacenar los registros durante “x” tiempo antes de volver a permitir el acceso a esa IP.

Protegiendo al usuario de sí mismo

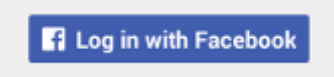
Como ya bien he comentado antes, la mayor amenaza para nuestros usuarios muy a menudo son ellos mismos.

No os confundáis, a la gente no le importa la seguridad, hasta que les hackean, he tratado de primera mano con empresas, no precisamente pequeñas, a las que les daba igual tener un SQL Injection en su web, y hablado-encuestado a gente que sencillamente piensan que a ellos nadie les iba a robar la contraseña de twitter, Facebook, de su correo electrónico o de su cartilla bancaria, para un usuario si tuviese que elegir, elegiría no utilizar contraseña.

En parte es algo malo, y bueno al mismo tiempo, ya que una contraseña es muy sencilla de averiguar, por ende, no hay que pensar en, tengo que hacer que mis usuarios tengan una contraseña cada vez más segura.

En seguridad, siempre se busca tener lo que se denomina como "Mínima Superficie de Exposición", dicho de otro modo, cuanta menos información expongas sobre el usuario más seguro estará, esto también se puede extrapolar a las contraseñas y sistemas de login y registro en general, ya habíamos hablado antes de SQL Injection y del riesgo que supone para la seguridad de nuestros usuarios, en este caso, y pensando desde el punto de vista de la seguridad del usuario que se registra en nuestra aplicación, cuanta menos información almacenemos sobre él, mejor.

Así es como funciona a groso modo el sistema de inicio de sesión de Google o Facebook en donde funciona mediante una key para el usuario el cual te sirve para identificar a ese usuario y el proceso que realiza a lo largo y ancho de nuestra aplicación web.



Estos sistemas, presentan un problema, y es que no solucionan el problema, sino que simplemente confías la seguridad de tus usuarios a otra aplicación, en este caso a Facebook o Google que tal y como ya sabemos salgo en contadas ocasiones confían su seguridad a un usuario y contraseña, no obstante hacen bien una cosa que más adelante veremos.

No obstante, y a pesar de esto, creo que el concepto de mínima exposición se entiende, pese a no servirnos de una gran ayuda.

Lo que si nos sirve de ayuda son establecer lo que conoce como **segundo factor de autenticación**.

Un segundo factor de autenticación, es un sistema mediante el cual, una vez puesto el usuario y la contraseña, nuestro usuario ha de introducir otra clave u otro método que le permite, ahora sí, acceder a nuestro sitio, y esta, es hoy por hoy, la única manera relativamente fiable de proteger la cuenta de uno de nuestros usuarios, incluso, de hacerlo más seguro podríamos integrarle la opción de que se hayan de introducir ambos valores al mismo tiempo, y ahora, la pregunta del millón.

¿Qué utilizamos como segundo factor de autenticación?

Lo más habitual sería pensar en utilizar otra contraseña para que tuviese que introducirla el usuario, lo que sería un completo error ya que la mayoría de los usuarios utilizarían la misma contraseña o una muy similar a esta.

Pese a esto, teóricamente, y teniendo en cuenta el último cálculo, supondrían 303^303 combinaciones de las más posibles, eso equivales a 91.809 combinaciones posibles, en teoría me reitero, luego utilizan la misma contraseña en ambos sitios y se cargan la estadística los propios usuarios, es por ello por lo que a menudo, estadística y psicología terminan dando resultados muy distintos frente al comportamiento de los usuarios (hecho que se trata de resolver con la llegada del big data).

Por suerte, existen otras muchas formas de establecer un segundo factor de autenticación para nuestra plataforma web.

Por ejemplo, contamos con:

- Google Authenticator
- Latch

De los que son APIs propias para implementar en nuestras aplicaciones.

Junto a estas, también podemos optar por otras alternativas, como por ejemplo la de enviar un mensaje al terminal móvil de nuestro usuario con un código temporal (un token), generalmente se suele usar un código numérico de 6 caracteres, este método, también se puede llevar a cabo mediante una aplicación propia que se haya de instalar en el dispositivo de nuestro usuario, es importante que este token caduque en un periodo corto de tiempo, por ejemplo, 15s, 30s o 1m.

Cabe recalcar que esto es justo lo que hace Google Authenticator.

Tampoco es recomendable utilizar un sistema de preguntas secretas o similares puesto que estos suelen ser datos que se suelen obtener mediante el paso de footprinting.

"El método de las preguntas secretas, suele ser usado para la recuperación de la contraseña, no como segundo factor de autenticación"

Y en el caso de estar estableciendo un sistema de autenticación físico, es especialmente importante el pensar en un doble factor de autenticación, en dicho caso es un factor biométrico como por ejemplo, un lector de huellas dactilares o un lector de retina, dicho esto cabe recalcar que los lectores de huellas digitales no son completamente seguros puesto que ya se ha demostrado poder ser vulnerados copiando la huella digital mediante una foto de la misma, la regla para establecer cualquier sistema de autenticación en 2 pasos es:

"Algo que sabes + algo que tienes"

Y pese a todo esto, no será completamente seguro puesto que por ejemplo si tuviera acceso un atacante a la contraseña y al dispositivo móvil de forma simultánea podría acceder, no obstante, el propósito del hardering en sistemas de login como el segundo factor de autenticación, no son más que una forma de intentar hacer más seguros los sistemas de acceso.

Métodos alternativos de login

Pese a ser inseguras, las contraseñas no siempre son la forma más insegura de proporcionar acceso a un sistema informático.

Existen alternativas más y menos seguras que las nombraremos a continuación, no obstante, hay que recordar de que cualquier sistema basado en un solo sistema de acceso de por sí, es más inseguro que otro que se componga de ese mismo método, más un segundo factor para acceder, siempre y cuando ninguno de ellos dos sean vulnerables y supongan un riesgo de seguridad para el otro factor de autenticación.

Factores biométricos:

Los factores biométricos son, tal vez, los más conocidos a la par que los más caros y difíciles de implementar ya que requieren de una instalación física, de igual manera, si trabajan sobre una API que se conecte con un servidores vez, son susceptibles de ser interceptados y modificados en tiempo real o incluso ser suplantados por una aplicación que emule las mismas peticiones que este realice, incluso si la conexión es cifrada.

Mediante USB:

No es raro toparse con personas que utilicen una llave USB como contraseña o clave de acceso para conectarse, algo similar al siguiente método que veremos a continuación, el problema de este tipo de dispositivos es que pueden ser clonados, sustraídos o aún peor, ser destruidos de manera accidental y privarte de acceso.

Mediante firma digital:

Esta es una forma bastante común en entornos profesionales donde se utiliza una firma digital para identificarse como que eres tú realmente, combinada junto a una contraseña por ejemplo, puede resultar una buena medida de seguridad para cualquier sistema de autenticación, no obstante es importante cuidar de esa firma digital dado que de ser sustraída, puede resultar un serio problema puesto que sería el equivalente a que obtuvieran tu DNI, de hecho, a efectos legales no existe diferencia alguna entre identificarte con tu DNI como con tu firma digital.

Mediante tarjeta magnética:

Con las tarjetas magnéticas sucede igual que con los 3 anteriores, al mismo tiempo, quizás incluso con las tarjetas magnéticas exista más información al respecto sobre como clonarlas, de hecho, pese a estar muy extendidas en el mundo de las tarjetas de crédito y debido, estas ya están siendo sustituidas por tarjetas de chip, otro problema con las tarjetas magnéticas es que resulta común que se desimanten y dejen de funcionar.

Y estas entre casi una infinidad de distintos métodos de identificación, no obstante, lo más seguro siempre será un sistema de control de accesos en 2 pasos también conocidos como 2FA.

Créditos

Herramientas relacionadas

- BrutiFramework:
  http://www.diegoceldran.es/brutiframework-alfa/

- Módulo generador de diccionarios avanzados:
  http://www.diegoceldran.es/producto/generador-diccionarios-avanzados/

- NMAP:
  https://nmap.org/download.html

- Foca:
  https://www.elevenpaths.com/es/labstools/foca-2/index.html

- Maltego:
  https://www.paterva.com/web7/downloads.php

Enlaces relacionados

- http://www.diegoceldran.es/
- http://www.elladodelmal.com/2014/06/sql-injection-minima-superficie-de.html
- http://www.hackplayers.com/
- https://webdesign.tutsplus.com/es/tutorials/how-to-integrate-no-captcha-recaptcha-in-your-website--cms-23024
- http://www.desarrolloweb.com/articulos/poner-captcha-en-3-pasos.html
- https://www.exploit-db.com/google-hacking-database/
- https://www.exploit-db.com/papers/

Información sobre el autor

Autor: Diego Celdrán Morell

Web del autor: http://www.diegoceldran.es/

Fecha de publicación: 31 de diciembre de 2016