Ingeniería social en informática.

La ingeniería social, aplicada al mundo de la informática, se basa en la premisa de que el usuario, ser humano al fin, es el eslabón más débil de la cadena, por lo tanto el más fácil de atacar.

https://twitter.com/SUSCERTE/status/733092312591392768

Desde hace muchísimos años ya sabíamos esto pero, a nuestro juicio, quien popularizó el término gracias al éxito de la normalización (esto es, expresarlo en reglas, pocas por demás, y aplicarlas al pie de la letra y masivamente) fue Kevin Mitnick.

- Todos queremos ayudar.
- El primer movimiento es siempre de confianza hacia el otro.
- No nos gusta decir "no".
- A todos nos gusta que nos alaben.

Aunque vosotros no me lo estaís preguntando, igual os lo digo: la letra ka de nuestro dominio web, donde están alojadas estas humildes lineas, es en honor a dicho cracker. Ah, y la fotografía que encabeza este artículo es Mister Kevin Mitnick. Dicho esto imaginad la cantidad de visitas que recibirá este entrada para divulgar nuestra idea (si alguien anteriormente la ha pubicado -la idea que van a leer-, comunicadlo de inmediato por favor ;-) , arriba tenéis nuestra dirección en Twitter.).

https://twitter.com/ks7000/status/731605936020787202
https://twitter.com/tecnologiapure/status/736601592661696512

Introducción.

Así pues, arrancamos el 2016, con el enfoque a la seguridad donde más falla. ¡Sabíais que en los accidentes de aviación un alto porcentaje de casos ocurrieron por impericia y/o confusión y/o conducta natural del ser humano! Y estamos hablando de la tripulación, no hablemos ya del personal en tierra: mecánicos e incluso los que pesan el equipaje que llevamos, o los que montan la carga o equipaje en los aviones (que para todos los efectos es lo mismo, en este caso).

Pero la Administración Federal de Aviación (la FAA por sus siglas en inglés, precisamente del país que desarrolló masiva y comercialmente la aviación) dedica mucho más tiempo en encontrar la solución para prevenir futuros accidentes que tiempo en investigar las causas de cada accidente. *Es decir, descubrir qué y cómo se estrella un avión toma tiempo y trabajo, pero prevenir dicho error a futuro es aún más trabajoso.* **La razón es simple: cuesta dinero implementar políticas de seguridad. Añada el hecho, además, que no se debe eliminar ni interferir en ninguna de las políticas de seguridad implementadas anteriormente, pues veís entonces que el asunto**

no va nada fácil, en lo absoluto. Tal como dice el hacker español Chema Alonso (ahora ejecutivo en Telefónica España) : "*la gente quiere estar segura PERO NO QUIERE PREOCUPARSE POR LA SEGURIDAD*"

https://www.youtube.com/watch?v=3tgQsZSCj0k&t=424

Por estos lares hay refrán que ilustra lo anterior: "estar bien con Dios y con el diablo"**.**

Un poco de historia.

El ejemplo de los aviones os lo colocamos para entrar de lleno en la seguridad informática, pero primero debemos describir, lo más breve y resumida posible, las "políticas de seguridad" implementadas anteriormente.

Por allá en el siglo pasado, a principios de la década de los noventa, utilizábamos el protocolo IPX/SPX de Novell Netware, un sistema operativo de red muy adelantado a su tiempo. No tuvimos acceso a internet de manera masiva hasta 1997 así que esas pequeñas red de área local con cable coaxial de 10 mbps eran verdaderas islas de datos. Vale decir que al desconectar un solo nodo por cualquier motivo, "colgaba" la red completa debido al uso de token, así que si alguien quería conectarse a la red pues no pasaba desapercibido, nos dábamos cuenta de inmediato. Y cuando por fin llegó el internet implementamos el TCP/IP para ciertas máquinas "avanzadas" en la red para repartir la conexión de marcado telefónico de los flamantes y nuevos modem PCI de 33 kbps (y luego los de 56 kbps) y todos tan felices y tan contentos, los datos del sistema de facturación e inventario corrían con cierto nivel de seguridad. Incluso por allá por 2004 ó 2005 hubo un incidente mundial del protocolo TCP/IP que nos dejó incomunicados por 2 días pero nunca se cayó el IPX/SPX.

Incluso hoy en día, en negocios pequeños, y a pesar que solo usamos TCP/IP con el fast ethernet o gigabit ethernet (100 ó 1000 mbps) con RJ45 (que detecta colisiones y que permite que alguien dentro de la empresa coloque una "T" para hacer un "ataque de hombre en el medio" por hardware o, más fácil aún, conectar un ordenador portátil a cualquier conector empotrado en pared para hacerlo por software) este sistema basado en "isla de información" sigue siendo relativamente seguro. Agreguemos el factor de red inalámbrica "wifi" y veremos que un atacante, sin necesidad de estar dentro de la empresa pero dentro del alcance de nuestra antena de red descartaría el método primero ya que le sería más fácil y cómodo "hackear" nuestro enrutador inalámbrico. Pero la tecnología llegó para quedarse, el inalámbrico tiene claramente sus ventajas que podemos observar acá:

Pueden intentar forzar la conexión inalámbrica (si usan WEP pues de hecho se conectarán indefectiblemente) o de irse de lleno y aplicar ingeniería social a nuestros empleados, más fácil

aún.

Claro está, todo lo descrito anteriormente, conectarse físicamente a nuestra red de área local, sería el primer paso para un atacante, lo siguiente sería tener las contraseñas necesarias para lograr escalar privilegios ya sea en el mismo enrutador, en los sistemas operativos de cada computadora o en las bases de datos de los servidores. De nuevo la modernidad facilita las cosas: un atacante en cualquier parte del mundo intentará entrar por medio de nuestro proveedor de internet, pero a favor nuestro están los "muros de fuego" tanto de los enrutadores -aparatos o computadoras que hagan las veces de enrutador(es)- como en los sistemas operativos modernos de cada computadora. Si queréis ver un método sencillo de protección con un muro de fuego "firewall" escrito con IPTABLES visitad este enlace, es un ejemplo sencillo para proteger nuestra red de área local.

Como siempre hay quien puede explicar mejor que nosotros, consideramos que el artículo de Alejandro Alcalde en su página web "El Baúl del Programador" bajo el título de "Cómo se almacenan tus contraseñas en internet (y cuando la longitud de la misma no importa)", os prometemos que es una explicación básica, sencilla y directo al grano.

A pesar de todos estos osbtáculos que hemos puesto para mantener a buen resguardo nuestros datos, he aquí en este mensaje o "tuit" que nos explican cómo obtienen las contraseñas hoy en día los "ciberdelincuentes":

https://twitter.com/GC_Illescas062/status/772173739496734720

Método que aspiramos combatir.

Como veís volvemos al principio de nuestro artículo: lo más fácil es lograr que los mismos empleados sean los que nos entreguen sus contraseñas, ***y si se niegan decirlas pues podemos hacer que nos las muestren.*** Esto se logra muchas veces con notar cual empleado de atención al público ha estado alejado de su computadora, generalmente para comer, y que llega a introducir su contraseña personal: allí lo abordamos con cualquier excusa, una consulta de precios, por ejemplo, mientras nos fijamos bien qué teclea, nos fijamos en sus manos ya qe estará entretenido viendo el monitor y no se percatará de qué observamos atentamente las teclas que presiona. **Aunque no lo creaís, hay personas que tiene una agudeza visual y excelente memoria para hacer esta pesca de contraseñas, así, "en vivo y en directo".** Es por ello que nuestros bancos, guardianes de nuestro dinero, lo repiten hasta el cansancio: **utilizemos contraseñas largas, amén de cambiarlas con cierta frecuencia.**

https://twitter.com/EFF/status/753731678816743425

¿Cómo almacenar información de manera segura?

Acá debemos volver al caso de las redes de área local de negocios pequeños, ya que los grandes utilizan el protocolo seguro https el cual encripta la información, previa implementación de claves públicas y privadas (ver anteriormente el ataque de "hombre en el medio"). Dichos sitios web pagan un alto costo por un certificado que garantiza, a nivel mundial, que uno está realmente conectado con el dominio web que uno introdujo para navegar y que, además, los datos enviados están encriptados, dando al traste con el plan que cualquiera haya implementado para interceptar los paquetes de datos que viajan mediante TCP/IP, el protocolo de red de facto.

En un negocio pequeño los servidores de datos los podemos implementar para que soporten https pero con la salvedad de que a cada ordenador le configuremos una excepción de seguridad -o instalemos el certificado generado por nosotros mismos con ayuda de SSH y scripts automatizados y masivos- *en cada navegador web que tengan instalado cada ordenador con GNU/Linux.* Todo sea por ahorrarnos un dinerillo en certificados digitales garantizados por terceros y servicios DNS.

Hypertext Transfer Protocol Secure (HTTPS).

Sin profundizar demasiado en este punto, solo vale recordar que el protoclo HTTPS es equivalente al protocolo HTTP pero establece una conexión segura, es decir, cifra la información para evitar que en el camino entre el usuario y el servidor cualquier tercera persona sea capaz de leer la información, siendo demasiado difícil su descifrado por no autorizados. En un principio utilizaba SSL pero luego se decantaron por un protocolo más seguro llamado TLS, sobre ambos pueden ahondar en su estudio en este enlace. *El punto clave a tener en cuenta es que contamos con una conexión privada entre el usuario y nuestro servidor con la cual se puede enviar información sensible como números de tarjetas de crédito, contraseñas, etc.*

Actualizado el sábado 30 de enero de 2016:

Leyendo las noticias acerca de las vulnerabilidades descubiertas por los *hackers* -Dios nos cuide NO sean *crackers*- nos enteramos que, bajo ciertas circunstancias muy especiales, el protocolo HTTPS puede ser vulnerado debido al uso de números primos "no seguros" -léase números primos "pequeños"- y el uso masivo de handshakes con el método Diffie-Hellman para reunir suficientes datos y aplicar un algoritmo antiguo muy conocido se puede llegar a descrifar los datos que transporta dicho servidor web. ***Por supuesto que estamos simplificando al máximo, allí teneís los enlaces web en inglés con los detalles a fondo.***

Guardando datos a buen resguardo.

Pero, aunque hayamos logrado resolver por alguna de las dos vías mencionadas en la sección anterior nos quedaría otro problema: ***¿Cómo se guardan las contraseñas de los usuarios en las bases de datos -o en los sistemas operativos- para garantizar que estén a salvo de ojos maliciosos?***

Volviendo al siglo veinte, por esa época las computadoras tenían muy poco poder de cálculo y debíamos arreglarnos con pocas líneas de código. Por 1994 ó 95, a petición de los usuarios que querían cuidar sus nuevos monitores CRT a colores, cuya pantalla se "quemaba" con la misma imagen fija (por ejemplo la consulta de precios, aún hoy es siempre la más usada) y dejaba marcas "fantasma" en ellas, tuvimos que programar un protector de pantalla basado en la teoría de fractales.

Para guardar los datos en disco duro de "manera segura", tampoco sobraba el poder de cálculo, así hicimos una función sencilla (y también muy fácil de descifrar) muy parecida a la que aún utiliza el lenguaje PHP con el función str_rot13 (podeís leer acerca de los métodos de encriptación desde lo más fácil hasta lo más difícil en este enlace en inglés). Para poder entender dicha función hay que tomar en cuenta que esos datos debían encriptarse y desencriptarse rápidamente y sin mayor consumo de ciclos de reloj del CPU. **Y aquí viene un concepto fundamental: debe funcionar rápidamente en ambos sentidos.**

Aquí ustedes se estarán preguntando ¿porqué aún el lengaje PHP ofrece esa función de encriptado siendo tan fácil de descifrar? **De nuevo la respuesta es el costo. Dicho método sigue siendo barato en cuanto a uso de hardware se refiere y es utilizada para guardar extensos párrafos como éste, que vois disfrutaís, en una base de datos.**

Suponiendo que, aunque obtuvieran las contraseñas para entrar a esa base de datos, pues no podrían modificar nada del texto pues resulta ininteligible a primera vista al ser humano y si llegaren a modificar algo, por puro vandalismo, nos percatariamos del acto ya que la entrada de blog mostraría texto extraño al ser descifrada (también podríamos realizar una función que copie -"respalde"- los textos y cada cierto tiempo -usando crontab- los compare y detecte alteraciones).

Si al párrafo anterior le aplicamos la función **str_rot13** veríamos lo siguiente (como será guardada en la base de datos):

```
Fhcbavraqb dhr, nhadhr boghivrena ynf pbagenfrÃ±nf cnen ragene n rfn onfr
 qr qngbf, chrf ab cbqeÃna zbqvsvpne anqn qry grkgb cbedhr erfhygn vavagr
yvtvoyr n cevzren ivfgn ny fre uhznab l fv yyrtnera n zbqvsvpne nytb cbe
cheb inaqnyvfzb abf crepngnevnzbf qry npgb ln dhr yn ragenqn qr oybt zbfg
eneÃn grkgb rkgenÃ±b ny fre qrfpvsenqn.
```

Debo hacer la observación que dicha función NO soporta los caracteres extendidos (a partir de la versión 7 dará soporte a caracteres UNICODE), lo cual no debe ser problema para el lenguaje html que tiene sus códigos especiales para representarlos, si deseaís saber más podeís acceder a nuestro tutorial sobre HTML.

Las funciones "hash".

Una función "hash" toma un valor, aplica un algoritmo predeterminado y produce un resultado único, aunque ALGUNAS VECES dicho resultado puede también ser obtenido con otro(s) dato(s) diferente(s), lo cual es llamado **colisión.** ¿Sencillo, no? Si quereís saber más pues aquí teneís el enlace a la Wikipedia sobre el tema.

Como podreis observar, la función del lenguaje PHP que presentamos anteriormente es una función "hash" y si analizamos como funciona (restando 13 de cada valor decimal que describe a cada caracter ASCII) dejando intactos los caracteres no alfabéticos -del alfabeto inglés- podemos llegar a la conclusión que no tiene colisión alguna.

Funciones "hash" criptográficas.

Pues bien, las funciones "hash" criptográficas en realidad NO existen. Lo que siempre ha existido es la criptografía casi desde el mismo invento de la escritura: a pesar de que antes del siglo XX un porcentaje muy pequeño de la población sabía leer y escribir siempre hubo la necesidad de ocultar la información, por ejemplo las órdenes dadas en el ejército. En la Segunda Guerra Mundial se utilizó una computadora especialmente diseñada para descifrar los mensajes del Tercer Reich Alemán, para lo cual fue muy exitosa y fue cuando se valoró enormemente el uso de los ordenadores en el área de la criptografía.

> Muy importante destacar que la en la criptografía el proceso es reversible, es decir, podemos obtener el texto original. De aquí en adelante hablaremos de **cifrado en un solo sentido; pudieramos llamarlo -en la práctica- como "cifrado irreversible".**

Para esta labor se escogieron unas funciones "hash" que tienen características bien adecuadas para el trabajo:

- Que no tengan colisiones -ya hablamos de la función PHP **str_rot13**-.
- Que tenga bajo coste en recursos de hardware y tiempo.
- Que no se pueda inferir o detectar un patrón por medio del análisis y comparación de

varios resultados con el mismo método -más adelante lo veréis con Winzip-.

En este último punto es cuando la función analizada, PHP **str_rot13**, falla y es principalmente porque NO encripta los espacios, los cuales a su vez delimitan las palabras y uno de los pilares fundamentales para el descifrado es eso: dónde comienza y donde termina cada palabra. Además tampoco cifra los demás caracteres, con lo que, si quiere, pudieramos decir que en realidad cifra a la mitad (y por eso es que es de tan bajo coste para cifrar párrafos y hojas enteras de información).

Winzip.

Esta entrada no puede estar completa sin mencionar al software de compresión llamado Winzip, un estandard o norma *de facto* en computación. Aunque lo usamos desde 1997 no fue hasta el año 2004 en su versión 9.0 que trajo la novedad de encriptación de AES 128 bits para máquinas de bajo rendimiento y AES 256 bits para máquinas de alto rendimiento (es decir, lo hicieron así para que uno pudiera ponderar nivel de seguridad contra tiempo usado para encriptación; con las máquinas de hoy en día de 16 núcleos y 32 gigabytes de RAM pues es imperceptible la diferencia de tiempo entre ambas).

He aquí un panorama de la estructura de archivo de un fichero con extensión .zip normal:

https://twitter.com/jonathansampson/status/816550384688918528

Como usábamos en aquella época el correo electrónico para compartir el código fuente, decidimos encriptar con AES 256 bits y un módulo pequeño, pero importante, y perdimos su contraseña. Lo tuvimos que reescribir y, ***hasta el sol de hoy aún no hemos podido descifrarlo.***

"Advanced Encryption Standard"

Advanced Encryption Standard fue promovida por el Instituto Nacional de Normas y Tecnología ("National Institute of Standards and Technology" o NIST por sus siglas en inglés) en el año 2001, y fue propuesta por dos criptógrafos belgas Joan Daemen y Vincet Rijmen con un bloque de 128 bits y longitudes de clave de 128, 192 y 256 bits convirtiendose en una norma de encriptación a nivel mundial. El gobierno federal de los Estado Unidos de Norteamérica considera la clave de 256 bits lo bastante segura para encriptar documentos de muy alto nivel de importancia, información muy secreta. Es posible descifrar incluso hasta las claves de 256 bits pero el esfuerzo y consumo de hardware consume muchos recursos, sin embargo sigue siendo la manera más viable de cifrar y descifrar información. **Hemos de recalcar que funciona en ambos sentidos: una cadena de caracteres (información a proteger) se encripta y se puede enviar a otra persona. Si la otra persona no conoce la clave tardará muchísimo tiempo y esfuerzo en "descubrir" la cave utilizada. Pero para el que encriptó los datos puede revertir la operación y recuperar la información.**

Si quieren leer un detalle sobre cómo se genera una llave AES notarán que hay una pequeñísima posibilidad de que comprimiendo y encriptando miles de archivos individuales con la misma contraseña, en algún momento se producirán 2 iguales que si se tiempo y paciencia podrán conducir a una pista sobre la contraseña, pero que se puede prevenir con usar llaves de 256 bits. Información para desarrolladores de software interesados en usar las últimas versiones de Winzip id a este enlace web. En ese último enlace, la mar de especificaciones, podemos leer que dicho software de compresión almacena un CRC (comprobación de redundancia cíclica, inventada por W. Wesley Peterson en 1961) *del archivo original -sin comprimir, sin encriptar-* en la cabecera del archivo comprimido para que después de desencriptar y descomprimir puedan ser comparados y verificar que la información anhelada se transmitió correctamente (recuerden nuestro caso de envio de código fuente por correo electrónico). Dicho CRC es ampliamente utilizado desde que se inventaron los CD, y simplemente es otro método de cifrado que exhibe un resultado de longitud fija *pero con muchas, muchísimas colisiones,* lo cual lo descarta de plano para el cifrado y sirve más bien como herramienta auxiliar. Cuando utilizábamos discos flexibles de 5,25" (y luego 3,5") los errores de hardware en la escritura, o luego la manipulación física por parte de nosotros (para llevarlos al trabajo, a la universidad) hacían muy necesario el uso del CRC para tener al menos una certeza que quedó bien grabado {y era sacarlo de la disquetera A y comprobarlo en la disquetera B, por eso el disco duro en los sitemas Windows se llama "C", -amigos linuxeros, no más chistes al respecto, por favor- ;-) }

En este otro enlace recomiendan aplicar "salt" a las rutinas de encriptación para Winzip (desarrolladores de software) pues ya hemos advertido de las posibles debilidadades del algoritmo de encriptación, así que se pueden "fortalecer" con nuestras propias "claves", si son aleatorias, pues mejor. **Por último, y de nuevo, nuestros bancos tienen la razón:** *usemos contraseñas largas (y vamos a cambiarlas con cierta frecuencia)*.

CRC.

Acá hacemos una pequeña pausa para recomendar la lectura de un artículo *patrocinado* para una famosa página web -vereís quién hace publicidad, lo veréis- donde explican, en idioma inglés, de manera fácil y amena el porqué se debe hacer la verificación de integridad de los datos transmitidos, y rápidamente colocan ejemplo en lenguaje C#. Para que veaís que todos los programadores tenemos nuestros momentos de ir a las raíces, a las bases y principios de los ordenadores (y si podéis ayudar al autor en su programación, hazlo, por favor).

Cifrado en un solo sentido.

Sin embargo hay casos donde en realidad no se necesita recuperar la información cifrada *pero necesitamos que dicha función no tenga colisiones, es decir, que la COMPARACIÓN de dos datos cifrados, en máquinas independientes, produzcan exactamente el mismo resultado o "hash".* En este caso no se busca encontrar el texto original, ya que es prácticamente imposible revertir el procesado de cifrado para conocer la información.

Para entender este proceso debemos siempre tener en cuenta que el "hash" generado por determinado método de encriptación conocido solo producirá un resultado único **que será el que guardemos en nuestra base de datos**, o en nuestro disco duro y sólamente el que conoce la información podrá repetir de nuevo el "hash" y así podremos comparar con el "hash" generado anteriormente; si son iguales pues se valida el proceso de identificación. Esto garantiza que la persona es quien dice ser, es dueño de la información, el "hash" es como una "huella" inconfundible y por eso lo hace adecuado para garantizar el acceso a sistema de cómputo por medio de contraseñas.

De hecho es tan efectiva que si nosotros olvidaramos nuestra contraseña para acceder a nuestra cuenta bancaria, por ejemplo, la única manera que el banco nos pueda ayudar es borrando el "hash" de la contraseña almacenada y permitirnos generar una nueva, ya que descifrar el "hash" es, como dijimos, practicamente inalcanzable. Por demás está decir que quienes llevan sistemas bancarios cobran buen dinero al banco por "reseteos" de contraseña, gasto que a la final, de una u otra manera, terminaremos pagando todos los clientes. *Pero el almacenamiento seguro de nuestras contraseñas, incluso bancarias, es otro proceso de ingeniería social que no discutiremos en esta entrada, allí hay bastante tela que cortar y sirve para una entrada en nuestro blog totalmente nueva sobre ese tema.*

En este punto volvemos a insistir que siempre hay gente que "sabe más que uno" ya sea por sagacidad *o por la fuerza bruta ¿Por qué decimos esto?* Existe el proyecto Ophcrack en el cual se han dado a la tarea de utilizar "diccionarios" de contraseñas comunes (en diversos idiomas) haciéndoles hash con los diversos métodos que enunciaremos para así comparar una contraseña obtenida ilegalmente con las almacenadas (y conocidas) en una base de datos monstruosa (algunas tabals llegan a alcanzar los 2 Terabytes, es decir, 2000 gigabytes). En este artículo (en idioma inglés) dan cuenta de las llamadas "tablas arcoiris" ("rainbow tables"), cómo funcionan y cómo trabajar con ellas. Si nos ponemos a ver la tarea hasta parece estúpida *sino fuera porque debemos considerar que en este siglo XXI la información es sinónimo de dinero* eso hace que esta gente trabaje tanto en estos proyectos (luego veremos que le podemos difucltar aún más ese trabajo que ellos tienen por medio de un método adicional de *hashing*).

MD5.

Desde los años noventa usamos el algoritmo MD5 aunque no recordamos exactamente cual año. Lo más probable es que haya sido con Windows 95, pero no estamos totalmente seguros. Lo que si es cierto es que ya entrado el siglo XXI se consideraba seguro y muy capaz de sustituir al vetusto CRC comentado enteriormente. De hecho aún se utliza por su bajo coste (ver párrafos anteriores) para publicarlos en las descargas de las páginas web: si aplicamos el MD5 al archivo descargado y comparamos dicho resultado al publicado en la página web de donde lo descargamos **es muy altamente probable que tengamos una copia idéntica a la que reposa en la páginas web.** Establecido como norma desde su invención, devuelve como resultado una número de 32 dígitos en hexadecimal y funciona "rellenando" con bits -mínimo 1, máximo 512-

(*atentos que más adelante tomaremos prestado este concepto*) y agregando hasta completar a un múltiplo de 512. Allí realmente comienza a aplicarse al algoritmo, que al final redunda en un dígito de longitud fija que es fácilmente manejable en una base de datos.

En el año 2004 se demostró una colisión y hubo la necesidad de buscar un método de cifrado mejor. Antes de continuar, comentamos que existió un algoritmo MD4 con el cual no tuvimos ni tenemos experiencia. También es justo comentar que fue el profesor Ronald Rivest su autor (y padre de toda la serie MD del 1 al 5, ea, ¡prolijo el profesor!).

Si revisamos el lenguaje PHP, mayormente utilizado para generar páginas web dinámicas, tenemos la función llamada -qué original- **md5()**, he aquí un ejemplo:

SHA1

Otro algoritmo soportado de manera nativa por el lenguaje PHP es la función SHA1 el cual produce 40 dígitos en hexadecimal, de nuevo dos ventajas, longitud fija y numérico, dos características muy apreciadas en el manejo de base de datos y tablas indexadas. También fue logrado su ruptura por gente muy voluntariosa. En fin, que los algoritmos fueron hechos para romperse, más ahora que se avecina la computación cuántica donde una "variable" en "memoria RAM" puede tener diferentes valores según "el ángulo" de donde se mire. Lo poco que podemos agregar con respecto a MD5 es que es un poco más seguro.

Tipos de cifrados en PHP (MD5, SHA1 y Salt).

En realidad esta idea no es nuestra, para nada, tuvimos la oportunidad de conocerla hace varios años divulgada por medio del Twitter:

https://twitter.com/codejobs/status/468896556104830976

El artículo completo lo podeís leer en este enlace. En esencia es combinar ambos cifrados pero con la inclusión de una "firma" adicional -en este caso la página web escrita en PHP- que solamente el programador conoce y que es llamada "salt" cuya traducción al castellano es "sal" ya que, si a ver vamos, le da un "sabor" distinto a las contraseñas. *La idea es ayudar al usuario agregándole longitud -y complejidad- a la misma a la hora de ser "hasheada".* Si lo pensamos bien, tiene la "debilidad" de que dicha "salt" (así la llamaremos) no debe ser extraviada, porque la usaremos cuando el usuario, en otra oportunidad de ingreso, se compare con la almacenada en nuestra base de datos. Más adelante veréis que dicha "salt" viene ahora incrustada por defecto en el lenguaje PHP.

Otros tipos de cifrado.

Ya casi llegamos al punto de nuestra propuesta, pero antes debemos enseñar que, como era de imaginarse, existen muchos otros tipos de cifrado. Volviendo al nuestro querido PHP existe la función avanzada llamada (si, están leyendo bien) "**hash()**". Para ser más precisos dicha función necesita tres parámetros: primero el tipo de encriptación, segundo, el mensaje a ser cifrado (en nuestro caso, la contraseña del usuario) y tercero en que formato se entrega, ya se en binario o hexadecimal.

El primer parámetro acepta los siguientes tipos de cifrado (versiones PHP 5 >= 5.1.2, PHP 7, PECL hash >= 1.1):

- MD2.
- MD4.
- MD5.
- SHA (cinco variaciones).
- RIPEMD (cuatro variaciones).
- WHIRLPOOL.
- TIGER (seis variaciones).
- SNEFRU (dos variaciones).
- GOST (dos variaciones).
- ADLER.
- CRC (dos variaciones).
- FNV (cuatro variaciones).
- JOAAT
- HAVA (quince variaciones).

Como podeís observar, las opciones de combinación son muchas, así que no os limiteís en utilizar solo MD5 y SHA, hay muchas otras opciones.

Actualizado el sábado 23 de enero de 2016:

Un estudiante paquistaní que utiliza C# en Ubuntu (y además no enseña sobre cómo manejar ese entorno de programación, eso merece otra entrada completa en nuestro blog -a pesar de los detractores de "C sharp"-) nos indica que es mejor utilizar SHA512 + SALT, claro vosotros estad claros en que plataforma estáis programando y si lo permite, siempre es bueno tener muchas alternativas porque no se sabe de cual lenguaje uno va a echar mano a la hora de asegurar nuestro trabajo en este mundo globalizado y con tanto "hacker" suelto:

https://twitter.com/afzaalvirgoboy/status/689880755229458432

Como nota adicional nos enteramos que la opción "SALT" agregada por el usuario será descontinuada en su uso en la versión 7 del lenguaje PHP y será sustituida por la ya soportada generación automática de la misma (muchas gracias sr. Ángel):

https://twitter.com/abr4xas/status/688828485792591874

Más adelante vovleremos a tocar el tema, porque ¿cómo recordar la "SALT" utilizada si fue generada aleatoriamente -con ayuda de la hora del servidor- para luego recuperar la contraseña? La solución es ingeniosa, ya lo vereís.

Combinaciones normalizadas.

El lenguaje PHP ofrece unas funciones, extensas, para el tema de la encriptación de manera nativa, faciltándonos la vida a nosotros los programadores pero manteniendo a raya el coste para el servidor donde esté alojada la página web.

Una terna de ellas es muy interesante, avalada con otra adicional, todas ellas siempre pensadas a un proceso y respuesta rápida a la página web solicitada lo que evita ciertos ataques maliciosos. POR EJEMPLO un atacante puede intentar hacer "login" -a sabiendas de ser errado- a determinado sitio web y por herramientas tomar el tiempo que tarda en responder el servidor. Si tarda 1 ó 2 segundos (una eternidad en tiempo de ordenadores) pueden iniciar ataques desde varios sitios para hacer colapsar el poder de procesamiento cuando realiza el desencriptamiento de dichos envios. Más información en el siguiente "tuit" ("tweet"):

https://twitter.com/chemaalonso/status/755238655002804224

Las tres funciones son:

1. hash_init(algoritmo_deseado).
2. hash_update(respuesta_obtenida_de_hash_init , cadena_a_encriptar).
3. hash_final(respuesta_obtenida_de_hash_update).

La función **hash_update()** puede ser repetida tantas veces se desee o sea necesario para agregarle la o las "SALT" comentada anteriormente. Como vemos es una solución normalizada, "estandarizada" si queremos usar el anglicismo. He aquí un código de prueba:

Como pueden observar es estructurada la propuesta, son cuatro líneas mínimo. La función que la

complementa es hash_equals(), solución nativa en este lenguaje y que devuelve verdadero o falso cuando compara dos "hash": el hash que leemos de la base de datos (guardada cuando el usuario se registró) y el "hash" calculado al momento de hacer "login" o autenticación cuando el usuario regresa en otro día y/o momento.

Repetimos: hay muchos otros algoritmos y muchas otras funciones para encriptar en un solo sentido, veamos la función crypt() trabajando con la función hash_equals():