Onna Bugeisha

Guerreras de la Ciencia

Hackers & Developers™ Press
2020

Onna Bugeisha – Guerreras de la Ciencia

ISBN 978-987-86-3950-5

Editado por Eugenia Bahit
1ª edición, marzo de 2020. Avellaneda, Buenos Aires. Rep. Argentina.
Hecho el depósito que marca la Ley 11.723.

Publicado por Hackers & Developers™ Press.
483 Green Lanes. N13 4BS, London. United Kingdom.
https://www.hdmag.press/

Compilación de glosario: Maiara Beatriz Cercasi
Revisión de artículos: A. Navarro, E. Bahit, F. Hernández, J. Monsegur y M. Mora.

Diseño de portada: © 2019 Eugenia Bahit. CC BY-NC-ND 4.0
Diseño y maquetación interior: © 2019 Eugenia Bahit. CC BY-NC-ND 4.0

A todas las mujeres que dedican su vida a la ciencia, luchando por la igualdad y la equidad en el mundo.

Eugenia Bahit

Autora y Editora

(Argentina, 1978)

Informática Teórica y Técnica Profesional en Neuropsicología.

Miembro de la *European Association for Theoretical Computer Science* (EATCS), se encuentra especializada en lógica matemática aplicada a la Seguridad Informática, Teoría de Conjuntos y Teoría de Objetos. Actualmente se encuentra trabajando en Modelos Axiomáticos para la postulación de una Teoría de Objetos en Estado Puro. Como docente se ha especializado en Neurociencias aplicadas al aprendizaje de la programación informática. En el campo de la Informática Aplicada, se especializó en Ingeniería Inversa de Código y Seguridad por Diseño sobre entornos GNU/Linux y OpenBSD.

INFORMÁTICA APLICADA

Sistema de Autenticación por Credenciales Criptográficas Disociadas (SACRED)

Eugenia Bahit

Resumen

Con el Sistema de Autenticación por Credenciales Disociadas (**SACRED**) se pretende generar un mecanismo de autenticación de usuarios, que aísle a estos de sus respectivas credenciales de acceso.

SACRED propone el uso concomitante de tres recursos informáticos, como artilugio para la generación y manejo seguro de credenciales criptográficas de alta entropía:

1. **Algoritmos *hash*** preexistentes, como medio para la ofuscación e irreversibilidad de datos sensibles.

2. **Sistemas de persistencia de objetos y Objetos en Estado Puro**, como medio para la lectura (recuperación) y escritura (almacenamiento) de usuarios.

3. **Generación de datos en tiempo de ejecución**, como único medio para relacionar usuarios con sus respectivas credenciales.

La conjunción de estos tres mecanismos genera como resultado, que incluso ganando acceso a los datos, se dificulte el reconocimiento y manipulación, de nombres de usuario y contraseñas, o poder deducir qué credenciales pertenecen a cuáles usuarios.

usuario_id	denominacion	nivel
bdb11458	Jane Doe	25
c334c51a	Administradora general	1
e15f9a89	Ficus Master	1
f936f26c	Juan Pérez	15
5de97892	John Doe	99
83993ec4	Ramona García	25

21. Tabla: Ejemplo de tabla de usuarios empleada en un sistema SACRED

Directorio: */path/to/.credentials*
.1b3c3dd18ec6dd33669727b63078e518d753de3d .5d0a3b698fecf59027a9c2547acd84d4760e6a61 .a011033e465aff4d44ca96ab65a3750e93af38eb .a9d30746053b546d41617f61e17c0d05acb4a46e .ad9f4c989d118103db43da25e188b37e4fa06e92 .ee1d6014143dd077facb2e7298c020ac441babeb

22. Tabla: Ejemplo de credenciales basadas en SACRED

Como se puede observar en el ejemplo anterior, no puede establecerse una relación fehaciente entre las credenciales almacenadas en un directorio oculto del sistema y sus respectivos propietarios, almacenados en la tabla de usuarios. Esto es debido a que los nombres de usuario y contraseñas empleados para acceder al sistema, no son almacenados ni siquiera de forma cifrada. En SACRED, los nombres de usuario y contraseña, se emplearían para generar tanto las ID de los usuarios (UID) como las credenciales, como partes complementarias de estos, pero no como piezas únicas.

De esta forma, **SACRED** podría estar garantizando que solo el propietario de los datos pueda manipularlos, y sin que esto suponga un cambio visible en la forma en la que el usuario final del Software, se autentica en el sistema.

No obstante, debe tenerse en cuenta el orden cronológico de creación de los archivos y registros de la base de datos (y de modificación y/o acceso en el caso de archivos). En los archivos de bases de datos, los registros quedarán almacenados en orden sucesivo de creación, mientras que los archivos, conservarán la fecha y hora de creación del *i-nodo*, acceso y modificación, que podrían evitarse, alterando dichos valores de forma pseudoaleatoria. La

modificación pseudoaleatorias de *i-nodos* será tratada en futuras investigaciones.

Definiciones previas

Sistema de autenticación

Mecanismo que permite validar la identidad de un usuario.

Credencial

Componente que interviene en la autenticación, el cual permite constatar que un usuario es quién afirma ser.

Credencial criptográfica

Credencial cuya información se encuentra cifrada mediante un conjunto de funciones *hash* combinadas, las cuáles no poseen un algoritmo derivado que permita que su valor *hash* sea revertido.

Funcionamiento del sistema

Proceso de autenticación

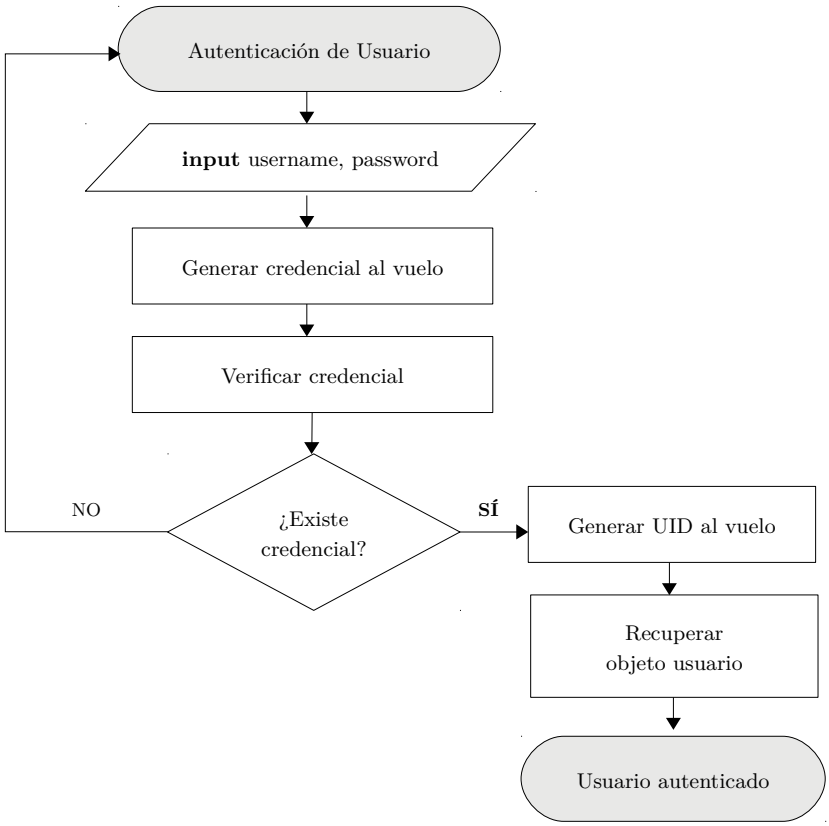
En el proceso de autenticación, el usuario aporta una combinación tradicional de *nombre de usuario* y *contraseña* de forma habitual.

A partir de dicha entrada, el sistema genera en tiempo de ejecución, la credencial correspondiente, mediante el empleo combinado de funciones *hash*.

A continuación, busca dicha credencial en el sistema de archivos (el nombre del archivo es el valor hash de la credencial).

Si la credencial existe, genera el UID (ID del usuario) correspondiente en tiempo de ejecución y utiliza dicho valor, para recuperar el objeto Usuario.

Si la credencial no existe, se vuelve al punto de inicio y el usuario deberá aportar nuevamente sus datos de acceso.



12. Ilustración: Proceso de autenticación

Generación de la credencial

Funciones necesarias y obtención de valores *hash*

Función *hash* para encriptado del nombre de usuario, $f(u) \rightarrow U$, donde u es el nombre de usuario en texto plano (valor de entrada) y U , el nombre de usuario encriptado (valor de salida).

Función *hash* para encriptado de la contraseña, $f(p) \rightarrow P$, donde p es la contraseña en texto plano (valor de entrada) y P , la contraseña encriptada (valor de salida).

Valor aleatorio reproducible de longitud fija (sal) de alta entropía. Dicho valor, se encontrará dado por una función *hash*, para cualquier orden de combinación de U , P y $x : f(U) \rightarrow x \vee f(P) \rightarrow x$. Dicha función se define como $f(U, x, P) \rightarrow s$, donde f es la función *hash* para la sal, x un valor aleatorio reproducible, obtenido a partir de U o P , y s , la sal resultante.

Valor hash para la generación de la credencial, determinado por una función *hash* $f(P, U, s) \rightarrow c$ para cualquier orden de combinación de U , P y s , donde c es la credencial (o valor *hash* resultante).

A continuación, se propone una implementación sencilla en lenguaje PHP, a modo de ejemplo.

Implementación a alto nivel (ejemplo en PHP)

```
function set_credencial() {
    $user_hash = hash('sha512', $_POST['usuario']]);  f(u) → U
    $pass_hash = hash('md4', $_POST['clave']]);       f(p) → P
    $x = substr($user_hash, 5, 12);                   f(U) → x
```

```
    $salt_hash = hash('crc32', "{$user_hash}{$x}{$pass_hash}");
    return hash('md5', "{$pass_hash}{$user_hash}{$salt_hash}");
}
```

$f(U,x,P) \rightarrow s$

$f(P,U,s) \rightarrow c$

Generación del UID

Funciones necesarias y obtención de valores *hash*

Función *hash* para encriptado del nombre de usuario, $f(u) \rightarrow U$.

Donde u es el nombre de usuario en texto plano (valor de entrada) y U , el nombre de usuario encriptado (valor de salida). Esta función puede (y debería) ser una función distinta a la función *hash* utilizada para encriptar el nombre de usuario en la generación de credenciales.

Valor aleatorio reproducible de longitud fija (sal) de alta entropía. Dicho valor, se encontrará dado por una función *hash* $f(x,U) \rightarrow s$, para cualquier orden de combinación de U y $x : f(U) \rightarrow x$, donde f es la función *hash* para la sal, x un valor aleatorio reproducible, obtenido a partir de U , y s es la sal resultante.

Valor hash para la generación del UID, determinado por una función *hash* $f(U,s) \rightarrow y$, para cualquier orden de combinación de U y s , donde y es el UID (o valor *hash* resultante).

A continuación, se propone una implementación sencilla en lenguaje PHP, a modo de ejemplo.

Implementación a alto nivel (ejemplo en PHP)

```
function set_uid() {
    $user_hash = hash('crc32', $_POST['usuario']]);
    $x = substr($user_hash, -5);
    $salt_hash = hash('md5', "{$x}{$user_hash}");
    return hash('crc32', "{$user_hash}{$salt_hash}");
}
```

$f(u) \rightarrow U$

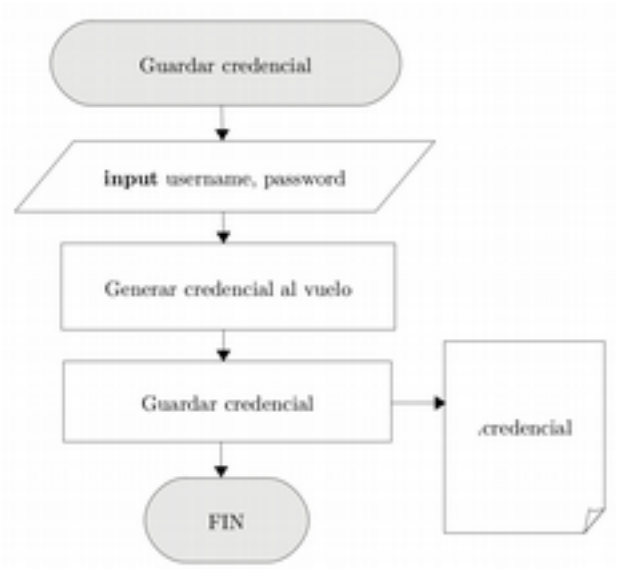
$f(U) \rightarrow x$

$f(x, U) \rightarrow s$

$f(U, s) \rightarrow y$

Persistencia de la credencial

Las credenciales podrán persistir como parte del sistema de archivos, utilizando el valor *hash* de la misma como nombre del fichero.



13. Ilustración: Persistencia de credenciales en el sistema de archivos

Cuestiones relativas a la seguridad del sistema de archivos

Se deberá destinar un directorio privado oculto, para el almacenaje de credenciales, el cual no pueda ser accesible por el usuario. En aplicaciones

Web, esto será un directorio no servido , con permisos 777, bajo la propiedad del usuario *root*.

Validación de credenciales

Siendo C el directorio destinado al almacenamiento de todas las credenciales c del sistema tal que $C=\{c\}$, y x el valor *hash* de una credencial obtenida en tiempo de ejecución mediante una función $f(P,U,s)\rightarrow x$, una credencial será válida sí y solo sí, se cumple:

$$f(P,U,s)\rightarrow x \text{ , } x\in C \Leftrightarrow x=c$$

A continuación, se propone una implementación sencilla en lenguaje PHP, a modo de ejemplo.

Implementación a alto nivel (ejemplo en PHP)

```
$credencial = set_credencial();      f(P,U,s)→x
$credencial_file = "/path/to/.credentials/.{$credencial}";
if(file_exists($credencial_file)) {  x∈C
    // credencial válida             x=c
}
```

El objeto Usuario

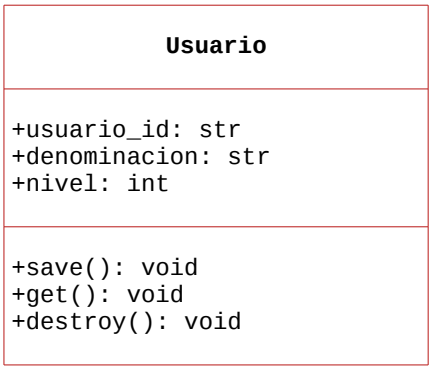
Para que pueda implementarse **SACRED** como método de autenticación en un sistema informático, un objeto *Usuario* debe tener, necesariamente, una propiedad ID (*UID*) y de tipo *string*, cuyo valor se encuentre determinado por $f(U,s)\rightarrow y$ (definido anteriormente), y no puede tener, bajo ningún concepto, propiedades destinadas a almacenar el nombre de usuario, la contraseña ni la credencial. De esta forma, cuando una denominación pública

(o identificación humanamente legible) sea requerida, una propiedad *denominacion* formará parte de los atributos del objeto.

Dado que **SACRED** como sistema de autenticación no aporta una forma concreta de prevenir nombres de usuarios duplicados, deberían tomarse los recaudos necesarios para respetar el *principio de unicidad* del valor de *UID*, que plantea que siendo $K^{\{O\}}$ el universo de todos los objetos O de una misma clase, y p una propiedad de identidad única (*UID*), debe cumplirse que:

$$\forall O_i^n \in K^{\{O\}}, (p \in O_i) \Rightarrow p \notin O_n$$

Un objeto *Usuario* típico podría verse como el siguiente:



Donde *usuario_id* es el UID, *denominación* la denominación pública del usuario (distinta al nombre de usuario de la credencial), y *nivel* el nivel de acceso (permiso) del usuario con respecto a su rol dentro del sistema.

Sobre la persistencia del objeto Usuario en bases de datos relacionales

Cuando se trabaje con persistencia en bases de datos relacionales, dado que el UID será generado a partir del nombre de usuario de la credencial, y que la modificación periódica de credenciales (tanto de la contraseña como del nombre de usuario) se debe considerar una buena práctica de seguridad, cuando el valor del UID sea clave primaria en una tabla, toda clave foránea que haga referencia a la misma, deberá establecerse por defecto con la opción de actualización en cascada.

Ejemplo en MariaDB:

```
usuario VARCHAR(12),
FOREIGN KEY(usuario)
      REFERENCES usuario(usuario_id)
      ON UPDATE CASCADE
```

Modificación de credenciales

La modificación de una credencial, supone una actualización tanto del UID como de la propia credencial.

Cuando un usuario previamente autenticado en el sistema, manteniendo una sesión válida abierta, pretende modificar sus datos de acceso, se hace necesario conocer el UID, para actualizar la propiedad del objeto usuario, y la credencial actual, bien sea para moverla (renombrarla), o bien para eliminarla y que no quede una credencial huérfana.

El UID puede mantenerse persistente en una variable de sesión, puesto que será necesario para identificar al usuario a lo largo de las diversas funcionalidades del sistema. Sin embargo, almacenar el valor *hash* de la credencial en una variable de sesión, supondría un riesgo de seguridad, puesto que sería contrario a la disociación, que es el principio en el que se basa **SACRED** para garantizar la seguridad. Dado que para obtener el valor *hash* de la credencial, el nombre de usuario es igual de necesario que la contraseña, hacer que persista en una variable de sesión, sería un riesgo aún superior que hacer persistir el valor *hash* de la credencial, puesto que dejaría expuesto un dato sensible.

Por lo tanto, la única forma de modificar una credencial, es pedir al usuario sus datos de acceso actuales en el mismo momento en el que intenta modificarlos. De esta forma, un descuido habitual del usuario que dejase una sesión abierta, no implicaría un riesgo para la manipulación de sus datos de acceso, puesto que su credencial continuaría estando a resguardo. En este sentido y mediante este artilugio, **SACRED** serviría para **evitar** uno de los **principales riesgos de la Ingeniería Social**: si una sesión queda abierta, con el usuario ausente, nadie que pretenda asumir su rol, podrá modificar sus datos de acceso.

Tras la solicitud de datos de acceso actuales y nuevos, y la generación de los valores *hash* correspondientes, el proceso de modificación de una credencial, consistirá en una sucesión de tres pasos:

1. Generar una nueva credencial

2. Actualizar el UID del usuario

3. Eliminar la credencial actual

Sobre la interfaz de entrada de los datos

Si bien la disociación es el mecanismo de seguridad primario de **SACRED**, se debe tener en cuenta que la ofuscación y el ocultamiento (o volatilidad) de los datos, también son parte esencial para la seguridad del sistema. Por consiguiente, tanto las interfaces gráficas (GUI) como las de texto (TUI), deben emplear (toda vez que sea posible) campos de contraseña tanto para la propia clave como para el nombre de usuario.

Mantenimiento del Sistema

Un sistema de autenticación por credenciales disociadas, requiere del establecimiento de ciertos parámetros constantes y mecanismos internos, que permitan ejercer un control de seguridad complementario para el mantenimiento del sistema. Estos parámetros y mecanismos se definen a continuación.

Parámetros constantes

Se definen dos valores constantes para establecer el *tiempo de vida* y el *período de inactividad* específicos, que afectarán a la validez de una credencial.

Tiempo de vida

Por tiempo de vida L, se entiende al período de validez máximo de una credencial desde el momento de su creación, expresado en segundos.

$L = 365 \cdot 24 \cdot 3600$ `; tiempo de vida equivalente a un año`

Siendo la constante L el tiempo de vida establecido para cualquier credencial; t_c los segundos transcurridos desde 1970-01-01 a las 00:00:00 HS UTC, hasta el momento de creación de la credencial; y t_a los segundos transcurridos desde 1970-01-01 a las 00:00:00 HS UTC, hasta la actualidad, una credencial será válida mientras se cumpla $t_a - t_c \leq L$ o $t_c + L \geq t_a$.

En sistemas GNU/Linux, se establecen tres tipos de fecha para los archivos: fecha de acceso, de modificación del archivo, y fecha de modificación de los atributos del archivo. Estas fechas pueden verse con el comando *stat* (o mediante *struct stat* en la biblioteca GNU C).

```
eugenia@bella:~$ echo "hola" > archivo
eugenia@bella:~$ stat archivo
  Fichero: archivo
  Tamaño: 5               Bloques: 8          Bloque E/S: 4096
fichero regular
Dispositivo: fe03h/65027d  Nodo-i: 1449828     Enlaces: 1
Acceso: (0644/-rw-r--r--)  Uid: ( 1000/ eugenia)   Gid: ( 1000/ eugenia)
      Acceso: 2018-06-20 20:17:04.772791949 -0300
Modificación: 2018-06-20 20:17:04.772791949 -0300
      Cambio: 2018-06-20 20:17:04.772791949 -0300
    Creación: -
eugenia@bella:~$ chmod +x archivo
eugenia@bella:~$ stat archivo
  Fichero: archivo
  Tamaño: 5               Bloques: 8          Bloque E/S: 4096
fichero regular
Dispositivo: fe03h/65027d  Nodo-i: 1449828     Enlaces: 1
Acceso: (0755/-rwxr-xr-x)  Uid: ( 1000/ eugenia)   Gid: ( 1000/ eugenia)
      Acceso: 2018-06-20 20:17:04.772791949 -0300
Modificación: 2018-06-20 20:17:04.772791949 -0300
      Cambio: 2018-06-20 20:17:35.505052227 -0300
    Creación: -
```

Sin embargo, la fecha de creación no es soportada directamente sobre el archivo. La obtención de la misma se hace a partir del *inodo* del fichero:

```
inodo=`stat -c %i archivo`
```

Conociendo el *inodo* del archivo y el dispositivo en el que se encuentra:

```
device=`df -P archivo | awk 'END{print $1}'`
```

Es posible obtener la fecha de creación del fichero mediante *debugfs* (aunque no directamente del propio archivo, sino, a partir del *inodo* en el dispositivo):

```
str_time=`debugfs -R "stat <$inodo>" $device 2>/dev/null | grep
-oP 'crtime.*--\s*\K.*'`
```

Dado que la fecha se obtiene como una cadena, se hace necesario convertirla a fin de obtener la cantidad de segundos transcurridos desde 1970-01-01 a las 00:00:00 HS UTC:

```
crtime=`date -d "$str_time" +%s`
```

Como **alternativa**, la fecha de creación podría grabarse dentro de la credencial en texto plano (siempre en segundos). En ese caso, siendo C la credencial, esta será válida si se cumple $t_a - (t_c \in C) \leq L$ y también $(t_c \in C) + L \geq t_a$.

Es importante aclarar que una credencial caducada, permanece en el sistema hasta su renovación, pero no debe ser eliminada del sistema antes de ser renovada, debido a este período. La excepción será una credencial inactiva, es decir, una credencial que estando caducada, supere el período de inactividad (el cuál es inferior al tiempo de vida). Este período se explica a continuación.

Período de inactividad

Se entiende por período de inactividad de una credencial, al tiempo máximo que puede trascurrir después de la fecha de último acceso del archivo al tiempo actual. Dicho período es una constante similar al tiempo de vida, denotada por P , y quedará determinada por: $P=\frac{L}{2}$.

Siendo la constante P el período de inactividad establecido para cualquier credencial; t_l los segundos transcurridos desde 1970-01-01 a las 00:00:00 HS UTC, hasta la fecha de último acceso de la credencial (arrojada por *stat*); y t_a los segundos transcurridos desde 1970-01-01 a las 00:00:00 HS UTC, hasta la actualidad, una credencial será válida mientras se cumpla $t_a - t_l \leq P$ o $t_l + P \geq t_a$.

Las credenciales inactivas podrán ser eliminadas siempre que se cumpla la negación de lo anterior. Es decir que para toda credencial c en el conjunto C de credenciales, se elimina una credencial si se cumple $\neg(t_l + P \geq t_a)$ o su equivalente $t_l + P \leq t_a$.

Siendo $f(c) \rightarrow t_l$ la función de retorno de fecha de último acceso de una credencial, la lógica sistemática de eliminación de credenciales inactivas quedaría determinada por el siguiente enunciado:

$$\forall c \in C : \neg(f(c) \rightarrow t_l + P \geq t_a) \Rightarrow \neg c$$

Mecanismos internos de control complementario

A fin de ejercer un control de seguridad complementario sobre un sistema de autenticación por credenciales disociadas, se deberán establecer los siguientes mecanismos:

- *Mecanismo de renovación de credenciales*: determinado por el tiempo de vida de una credencial, obliga al usuario a modificar sus datos de acceso al sistema cada cierto período de tiempo.

- *Mecanismo de recolección de basura*: determinado por el período de inactividad de una credencial, permite mantener el sistema libre de credenciales huérfanas o en desuso, que podrían ser explotadas con fines maliciosos.

Renovación de credenciales

La renovación de una credencial, es la acción de generar una nueva credencial para un usuario preexistente. La renovación de una credencial puede llevarse a cabo cuando una credencial alcanza el tiempo de vida establecido L , o bien, cuando la persona propietaria de la misma lo solicite como consecuencia del extravío de su credencial (olvido de su nombre de usuario y/o contraseña). En

este último caso, quedará una credencial huérfana que deberá ser eliminada a través de un mecanismo de limpieza o recolección de basura.

Renovación por caducidad

Se llevará a cabo en tiempo de autenticación.

En el momento de comprobar los datos de una credencial (ver «Validación de credenciales») y tras constatar que una credencial existe, el sistema deberá verificar el tiempo de vida de la credencial. Cuando no se cumpla $t_a - t_c \leq L$, deberá solicitarse al usuario que aporte nuevos datos de acceso.

El proceso de renovación por caducidad, debería llevarse a cabo de la misma forma que el proceso de modificación convencional descrito anteriormente:

1. Generación de nueva credencial
2. Actualización del UID del usuario
3. Eliminación de la credencial actual

Renovación por extravío

Por **extravío** de una credencial, se entiende a la dificultad práctica y cognitiva del propietario de una credencial, para recuperar sus datos de acceso al sistema.

En caso de que un usuario extravíe sus datos de acceso, el sistema le deberá ofrecer la posibilidad de establecer unos nuevos. Esta acción dejará una credencial huérfana (extraviada) en el sistema, la cual deberá ser purgada mediante un mecanismo de recolección de residuos (descrito más adelante).

Por lo tanto, la revocación por extravío se llevará a cabo mediante solicitud expresa del usuario.

Dicha renovación, deberá hacerse mediante la *generación de un token criptográficamente* seguro, enviado al usuario a través de un dato de contacto aportado previamente y almacenado en el sistema (antes del extravío). Dicho token deberá almacenarse de forma temporal, y tener una vida media no superior a las 24 horas.

Diferentes formas de crear un token criptográficamente seguro con PHP:

```
# Empleando random_bytes (desde PHP 7.0)
$token = bin2hex(random_bytes(128));

# Mediante la biblioteca OpenSSL
$token = bin2hex(openssl_random_pseudo_bytes(128));

# Utilizando OAuth (requiere el paquete php-oauth)
$token = OauthProvider::generateToken(128);
```

Definir un método de «recuperación de credenciales» va más allá de los objetivos de SACRED. No obstante, un mecanismo viable podría incluir una serie de pasos como los siguientes:

1 USUARIO solicita la generación de una nueva credencial por extravío

2 SISTEMA solicita al usuario aportar un dato de contacto (se espera que este dato exista en el sistema)

3 USUARIO aporta un dato de contacto

4 SISTEMA realiza una búsqueda inversa de la ID del usuario, a partir del dato de contacto (esto se hace en realidad, para constatar que el dato de contacto pertenezca a un usuario preexistente en el sistema).

4.1 Si encuentra coincidencia:

4.1.1 Genera un token criptográficamente seguro

4.1.2 Almacena el token temporalmente junto a la ID del usuario

4.1.3 Envía al usuario a través del dato de contacto indicado, el token junto a una URL en la cual intercambiar dicho token por una nueva credencial

5 USUARIO ingresa a la URL indicada y aporta el token recibido

6 SISTEMA verifica el token

6.1 Si encuentra coincidencia:

6.1.1 Elimina el token

6.1.2 Solicita al usuario un nuevo nombre de usuario y contraseña

6.1.3 Modifica la ID del usuario en la base de datos

6.1.4 Genera nueva credencial

Purga de credenciales y recolección de basura

La purga de credenciales es la acción sistemática y automatizada de *«recolección de basura»*, entendiéndose por tal —en el presente contexto—, a la eliminación de credenciales huérfanas (credenciales extraviadas) o en desuso (credenciales inactivas).

Una credencial solo podrá considerarse huérfana cuando no pueda ser asociada a un usuario existente del sistema. Dado que SACRED disocia las credenciales de los usuarios, solo podrá saberse que existe al menos una credencial huérfana en el sistema (pero no cuál), en el caso de que un usuario reporte su credencial como extraviada. Y esto solo sucederá, en el momento que el usuario solicite una renovación por extravío. Cuando esto suceda antes de cumplirse el plazo establecido para P, la credencial será considerada huérfana, aunque en un sentido práctico no puede determinarse cuál es esa credencial. Por ese motivo, una credencial huérfana solo podrá ser purgada cuando iguale o supere el período de inactividad P establecido en el sistema.

Este procedimiento de eliminación de credenciales huérfanas o inactivas, deberá llevarse a cabo de forma periódica (al menos una vez por día), eliminando toda credencial cuando la diferencia entre la fecha actual t_a y la fecha de último acceso t_l sea mayor o igual al período de inactividad P establecido en el sistema, tal que:

$$\forall c \in C : \neg(f(c) \rightarrow t_l + P \geq t_a) \Rightarrow \neg c$$