Criptografía: debilidades y ataques

Contenidos

1. Introducción.
2. Autenticación.
3. Password hashing.
4. Cómo almacenan las contraseñas los SO.
    1. Contraseñas en GNU/Linux.
    2. Contraseñas en MS Windows.
5. Atacando funciones hash.
    1. Colisiones.
    2. Cracking.

Introducción

- **Criptografía aplicada**:
  - Cifrado de las comunicaciones.
  - Cifrado de dispositivos.
  - Almacenamiento de contraseñas – Autenticación.
- **Criptología de empleo en el ENS** (Guía CCN-STIC 807)
- **Errores frecuentes** (mejores prácticas criptográficas).
  - Crear tu propio algoritmo o implementar uno existente.
  - Mal uso de librerías o algoritmos (Ej: ***Zerologon***).
    - Introducing the Tink cryptographic software library (Librería criptográfica multilenguaje y multiplataforma desarrollada por Google).
  - Incorrecta protección de claves criptográficas.
  - Aleatoriedad que no es aleatoria.
  - La criptografía se evita (canal lateral) (Air Gap).
  - Criptografía no aislada (Ej: SSL/TLS).
    - Seguridad del protocolo SSL/TLS. Ataques criptoanalíticos modernos

Introducción

- **Zerologon** (CVE-2020-1470)
  - Escalada de privilegios (*domain admin*) aprovechando una implementación criptográfica pobre en el protocolo **Netlogon**:
    - PoC
    - Exploit
  - Hacking Windows con Zerologon
    Entrada en el blog Un informático en el lado del mal donde se explica esta vulnerabilidad.

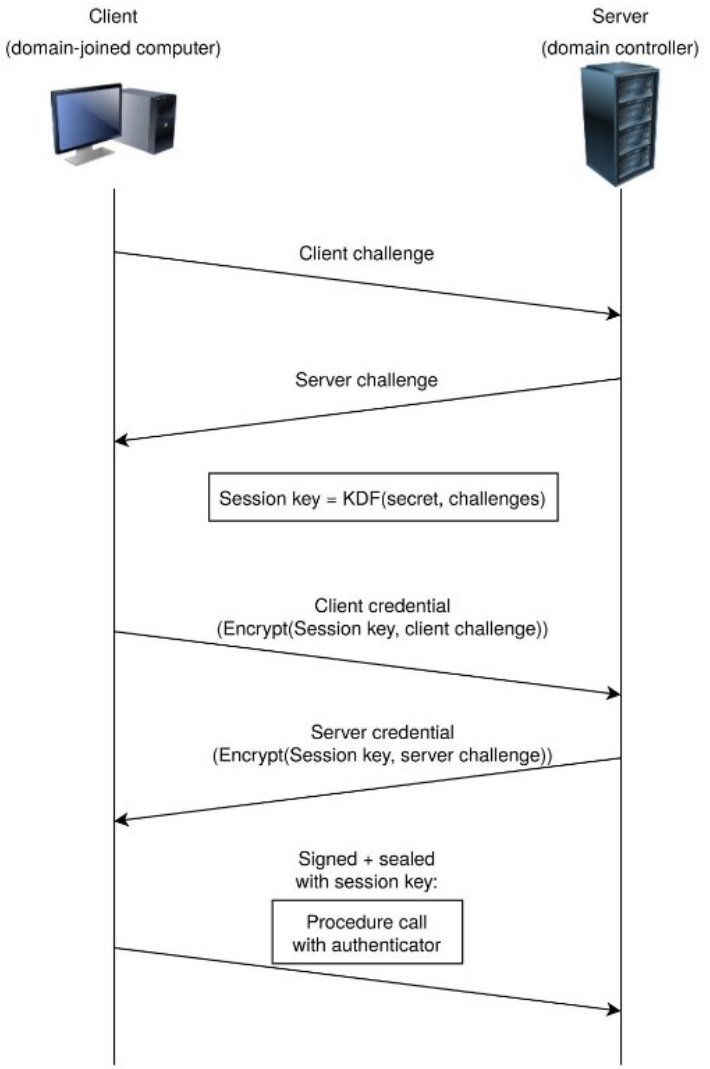


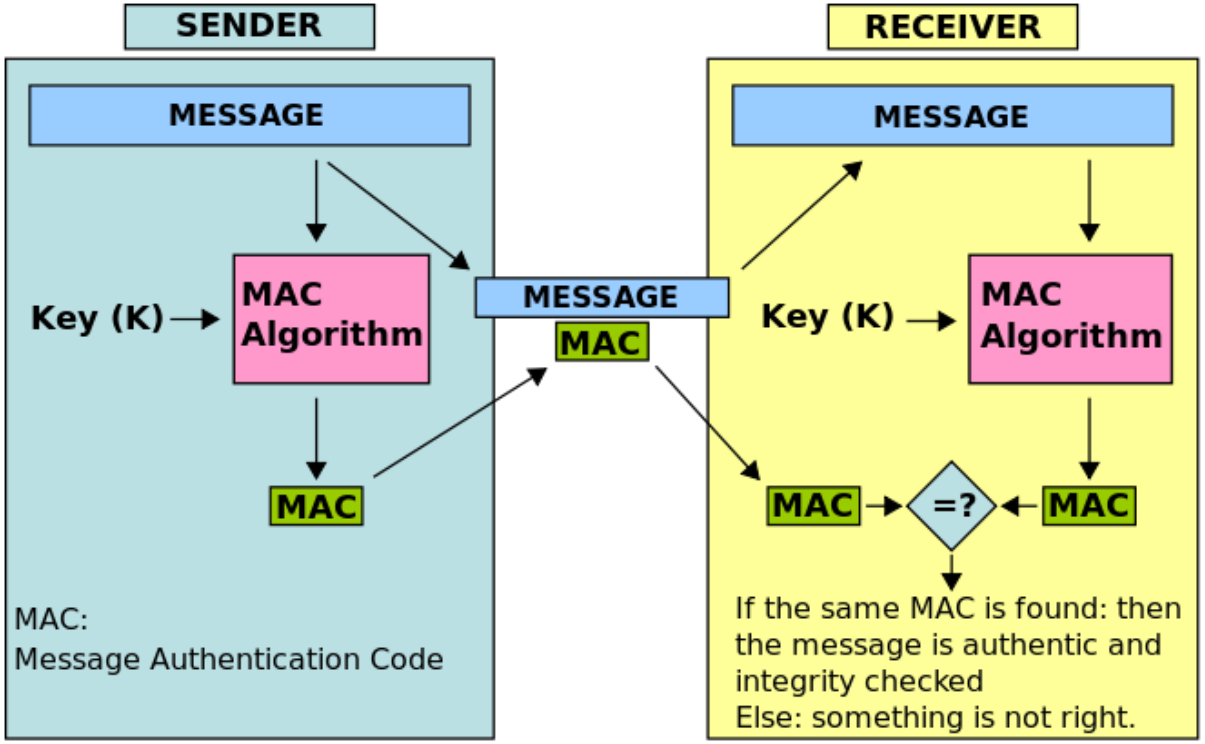
Figure 1: Simplified Netlogon authentication handshake

Autenticación

- El método más empleado para **almacenar contraseñas** es mediante **funciones hash**.
- **Propiedades** de las funciones hash:
  - Facilidad de cálculo.
  - Unidireccionalidad.
  - Compresión.
  - No predictibilidad.
  - Resistencia a colisiones.

Autenticación

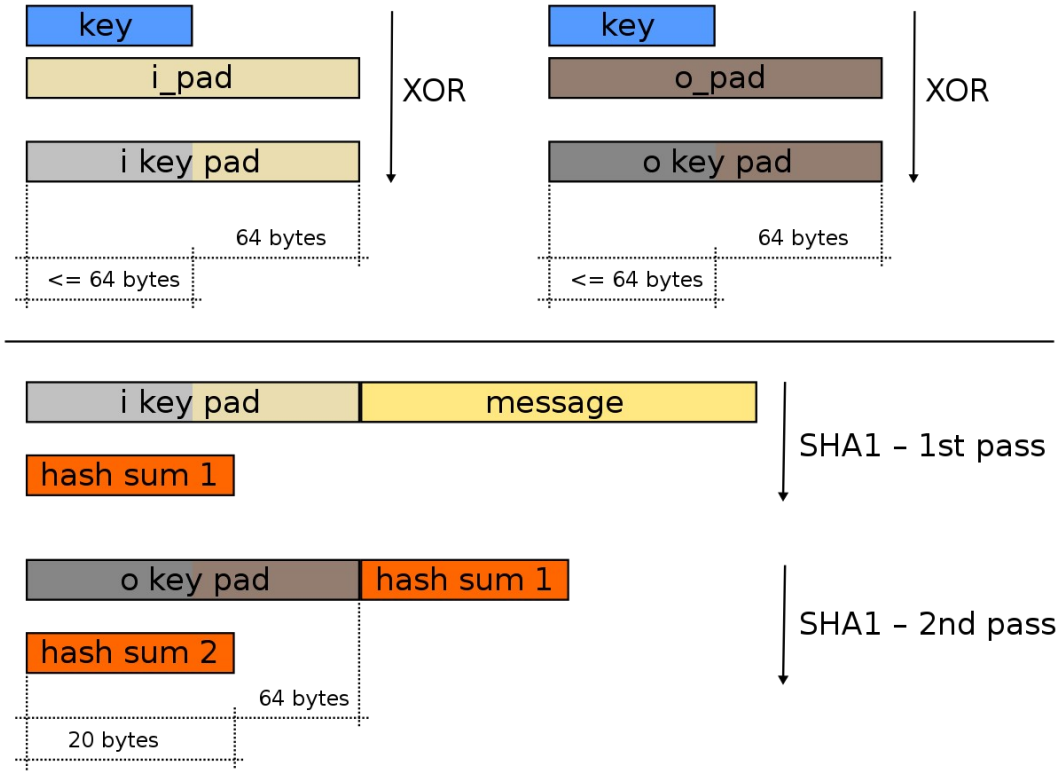
- **Esquemas típicos de autenticación**. Utilizan los **códigos MAC** (*Message Authentication Code*) para autenticar y verificar la integridad de los mensajes.



https://en.wikipedia.org/wiki/Message_authentication_code

Autenticación

- Algoritmos más usados para la creación de MACs.
  - HMAC (*Hash-based* MAC, RFC-2104, ej: HMAC-SHA256).



https://en.wikipedia.org/wiki/HMAC

Autenticación

- Otros algoritmos para la creación de MACs.
  - CBC-MAC. Emplea cifradores de bloque.
  - GMAC. Basado en *Galois Counter Message Authentication Code.*
  - KMAC. Emplea la función Keccak.
  - Poly1305 creado por Daniel J. Bernstein.
  - UMAC. Basado en Universal Hashing (se elige una función hash aleatoriamente).
  - VMAC. Basado en cifradores de bloques y diseñado para alta eficiencia.
  - SipHash. Una función simple, rápida y segura para computar MACs.

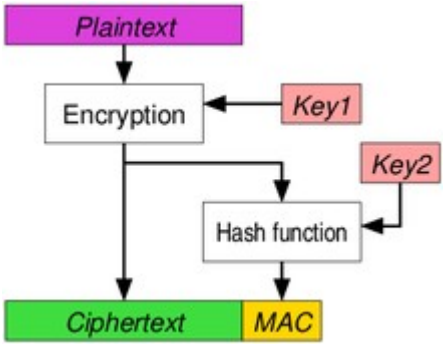
Autenticación

- Los códigos HMAC se suelen emplear en conjunción con técnicas de confidencialidad (cifrado autenticado).
- **Cifrado autenticado** (*Authenticated Encryption*-AE). Propuestas criptográficas en las que se garantiza simultáneamente la confidencialidad y autenticidad de la información.
  - Proporcionan seguridad contra ataques de texto cifrado (chosen-ciphertext atack-CCA).
- **Cifrado autenticado con datos asociados** (*Authenticated Encryption with Assicated Data* AEAD). Una variante del anterior que permite incluir datos no encriptados, garantizando la integridad de datos encriptados y no encriptados. Ej: paquetes de red que necesitan una cabecera visible.

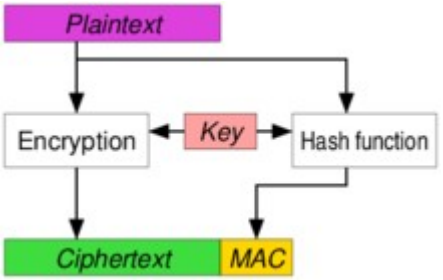
Autenticación

- Los esquemas posibles de cifrado autenticado son:
  - ***Encrypt-then-MAC***. El MAC se genera del texto cifrado (ISO/IEC 19772:2009). Ej: IPSec. ÚNICA OPCIÓN RECOMENDADA.
  - ***Encrypt-and-MAC***. El MAC se genera del texto en claro y, a continuación se obtiene el texto en claro cifrado sin el MAC.
  - ***MAC-then-Encrypt***. El MAC se produce del texto en claro, y MAC y texto se cifran conjuntamente.

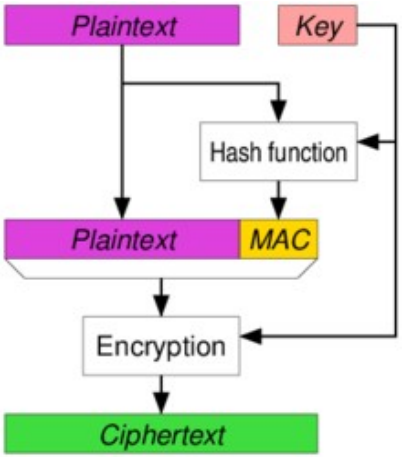
Esquema EtM



Esquema E&M



Esquema MtE



https://en.wikipedia.org/wiki/Authenticated_encryption

Autenticación

- **Cuidado con el padding**.
  - Consiste en añadir cierta información al principio, al final o en mitad de los mensajes.
  - A menudo es necesario que la información a proteger tenga un tamaño múltiplo de un cierto valor.
  - Estándares para crear paddings: ANSI X.923, ISO 10.126, ISO/IEC 7816-4, …
  - La predictibilidad del padding puede facilitar el criptoanálisis.
    - **Padding Oracle Attacks**.
      - HackTricks – Padding Oracle. https://book.hacktricks.xyz/cryptography/padding-oracle-priv
      - Ataques SSL/TLS (Lucky13, New Bleichenbacher side Channels and attacks, Poodle, Drown, Robot, Zombie Poodle, GoldenPoodle, …).
        - Charla Navaja Negra 2019: "*Reversing Cryptographic attacks over SSL/TLS*".

Autenticación

- **Cuidado con el padding**.
  - No siempre es necesario recurrir al padding. Existen alternativas:
    - Cifradores de flujo: cifran la información bit a bit.
    - Uso de **técnicas Ciphertext-Stealing**. En determinados escenarios es interesante no recurrir al padding, por ejemplo para no desperdiciar espacio de almacenamiento o mejorar la transmisión.
      - The Security of Ciphertext-Stealing.
      - Recommedations for Block Cipher Modes of Operation.

Password Hashing

- ***Password Hashing***.
  - Clave criptográfica vs password.
  - En criptografía simétrica: 128-256 bits.
  - En criptografía asimétrica: 512-8192 bits.
  - Fortaleza de contraseñas:
    - [1] Esfuerzo computacional en ataques de fuerza bruta.
    - [2] Test de fortaleza de contraseñas.

Password Hashing

- **Protección y almacenamiento de claves** criptográficas en función de:
  - Se almacenan en disco.
    - Almacenamiento software: KeePass, LastPass, 1Password.
    - Almacenamiento online: GuardedBox.
    - Almacenamiento hardware: HSM, YubiKey, etc.
  - Se utilizan en tiempo de ejecución (RAM).
    - Whitebox cryptography.

Password Hashing

- **Derivación de claves criptográficas**. Una función KDF (***Key Derivation Function***) es una función matemática que transforma una clave, normalmente de tamaño variable, a una clave de tamaño fijo.
  - **HKDF** (***Hash-based KDF***). Las funciones hash se pueden emplear como KDF. Hay que emplearlas adecuadamente ya que puede facilitar ataques de fuerza bruta, diccionario, rainbow tables, etc.
  - El ***key stretching*** son técnicas para hacer más seguras las claves. Algunas de ellas son:
    - Usar un ***salt*** para almacenar contraseñas de manera más segura frente a ataques de fuerza bruta o diccionario. Ej: /etc/shadow (sistemas Unix).
    - ***Pepper***, similar al anterior pero se almacena en un lugar distinto a la clave.
    - Añadir un número de iteraciones elevado de modo que la adivinación de contraseñas sea computacionalmente costoso. Ej: PBKDF2.
    - Emplear requerimientos de memoria elevados.
  - Funciones KDF recomendadas hoy en día para el almacenamiento de contraseñas (*password hashing*): PBKDF2, Bcrypt, Scrypt, Argon2.
    - OWASP Password Storage Cheat Sheet.
    - CrackStation – Salted Password Hashing. Doing it right

Password Hashing

- **Derivación de claves criptográficas**.
  - **PBKDF2** (*Password-Based Key Derivation Function 2*). Función pseudoaleatoria de derivación de claves de hasta 160 bits. Definida en el RFC-2898.
    - Usa como entrada una contraseña o frase de paso unido a un valor aleatorio (*salt*) y una función hash.
    - El proceso se repetirá varias veces para producir la clave derivada. El NIST recomienda un rango de 1000 a 10 millones.
      - Recomendaciones del NIST para PBKDF .
      - Ejemplo de uso con CyberChef
    - Empleado en redes WiFi (WPA y WPA2), software de cifrado (Filevault, Luks, Truecrypt, Veracrypt), gestores de arranque (grub2), Android, etc.

Password Hashing

- **Derivación de claves criptográficas**.
  - **Scrypt PBKDF**. Creada en 2016 para minimizar algunos inconvenientes de funciones KDF anteriores.
    - Propuestas como PBKDF2, bcrypt, crypt, etc., basadas en *key stretching* por uso de iteraciones son viables de atacar mediante hardware específico (ASICS o FPGA).
    - El diseño de ***scrypt*** fuerza el **uso de gran cantidad de memoria** comparada con sus antecesores. Esto dificulta, por coste y tamaño, implementaciones hardware de ataque y su paralelización.
    - Implementado en el RFC7914.
    - Se ha usado como *Proof-of-Work* en diversas criptodivisas.

Password Hashing

- **Derivación de claves criptográficas**.
  - **Argon2**. Función de derivación de claves (KDF) ganadora del *Password Hashing Competition* (2015).
    - Se considera una de las propuestas más robustas para la protección de claves.
    - Es liberado en 3 versiones que gestionan parámetros de memoria requerida, tiempo de ejecución (salt, iteraciones, …) y grado de paralelismo.
      - **Argon2d**: maximiza la resistencia a ataques de cracking con GPUs.
      - **Argon2i**: optimiza la resistencia a ataques de canal lateral.
      - **Argon2id**: versión híbrida. Recomendada salvo alguna excepción que haga preferir las otras versiones.

Almacenando contraseñas

- Contraseñas en GNU/Linux.
  - Fichero ***/etc/passwd***. Contiene información de usuarios y grupos pero con el hash de la contraseña oculto. Es accesible por cualquier usuario.
  - Fichero /etc/shadow. Similar al anterior pero el hash de la contraseña es visible. Se necesitan permisos de root para acceder.
  - El formato del hash es: *$n$salt$hash*
    - 1 para MD5.
    - 5 para SHA-256.
    - 6 para SHA-512.

Almacenando contraseñas

- Contraseñas en GNU/Linux.
  - El algoritmo para generar las contraseñas en Linux se establece en la variable **ENCRYPT_METHOD** del fichero /etc/login.defs. La opción por defecto es SHA-512.
  - El comando ***mkpasswd*** se emplea para generar los hashes de cadenas. Ejemplo:
    - mkpasswd --method=sha-512 --salt=se.Ed3lK pass
  - Internamente el SO emplea la llamada al sistema ***crypt()***.

Almacenando contraseñas

- Contraseñas en Windows.
  - Clientes: Se almacenan en el **registro SAM** (*Security Account Manager*).
    - El acceso puede protegerse opcionalmente con la aplicación SYSKEY que cifra la BD con una contraseña.
  - Servidores:
    - Se almacenan el fichero ***ntds.dit*** en el controlador de dominio.
    - Si se emplea autenticación LDAP mediante Kerberos, con los tickets TGS (***Ticket Granting Service***) y TGT (***Ticket Granting Ticket***).

Almacenando contraseñas

- Contraseñas en Windows.
  - Dos tipos de cifrado.
    - **LM** (**Lan Manager**). Es el más antiguo. Sigue estando presente aunque deshabilitado por defecto desde Windows Vista/Server 2008. Usa el algoritmo ***LMHash***.
    - **NTHash** o NTLM (**Lan Manager de Windows NT**). Es el método empleado actualmente. Utiliza el algoritmo de cifrado ***NTHash***
    - **Net-NTLMv1/v2** son protocolos de autenticación en entornos de red Windows que utilizan las contraseñas NTHash / LM Hash.

https://en.wikipedia.org/wiki/LAN_Manager
https://en.wikipedia.org/wiki/NT_LAN_Manager

Almacenando contraseñas

- Contraseñas en Windows.
  - LM (**Lan Manager**). Algoritmo:
    - La contraseña se convierte a mayúsculas y puede tener un máximo de 14 caracteres.
    - La contraseña se divide en dos elementos de 7 caracteres.
    - Las contraseñas de menos de 14 caracteres se completan con ceros.
    - El hash de cada parte se calcula por separado utilizando un algoritmo basado en DES.
    - Los dos hash concatenados forman el hash LM.

Almacenando contraseñas

- Contraseñas en Windows.
  - **NtHash** (NTLM). Es el método en el que se almacenan las contraseñas locales en Windows en la base de datos SAM o en los controladores de dominio en el **fichero NTDS.dit**. Son los hashes que se emplean para realizar el **ataque *pass-the-hash***.

MD4(UTF-16-LE(password))

Almacenando contraseñas

- Contraseñas en Windows.
  - **NTLMv1/v2** (Net-NTLMv1/v2). Se utilizan como desafío de autenticación entre cliente y servidor.
    - NTLMv1 está obsoleto pero puede aparecer en redes y sistemas antiguos.
    - NTLMv2 es una versión mejorada del anterior. Es la versión por defecto desde Windows 2000.
    - Ambos métodos son vulnerables a ataques relay.

https://book.hacktricks.xyz/windows/ntlm

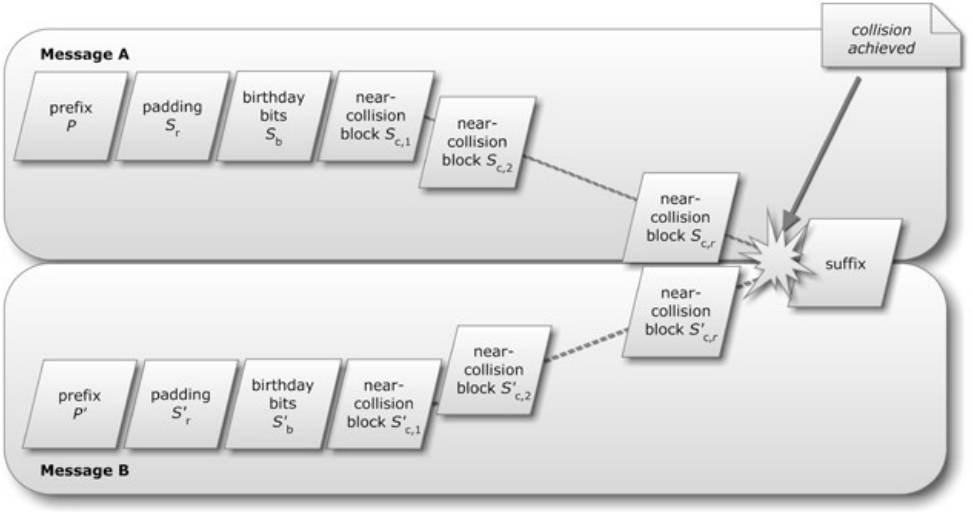
Almacenando contraseñas

- Contraseñas en Windows.
  - **Kerberos.** Desarrollado por el MIT. En Windows, se emplea como mecanismo por defecto para autenticar clientes en el controlador de dominio. Utiliza los puertos TCP/UDP 88.
    - TGS (*Ticket Granting Service*). Es el ticket que se presenta ante un servicio para poder acceder a sus recursos.
    - TGT (*Ticket Granting Ticket*). Es el ticket empleado para autenticarse ante Kerberos y obtener los TGS que dan acceso a diferentes recursos.
  - Ataques a Kerberos.
    - ***Overpass The Hash*** o ***Pass The Key*** (PTK).
    - ***Pass the Ticket*** (PTT).
    - ***Golden Ticket*** y ***Silver Ticket***. Se intentan construir los tickets TGT y TGS respectivamente.
    - ***Kerberoasting***. Emplear los TGS para crackear las contraseñas offline.
    - ***ASREPRoast***. Similar al anterior

https://www.tarlogic.com/es/blog/como-funciona-kerberos/

Atacando funciones hash

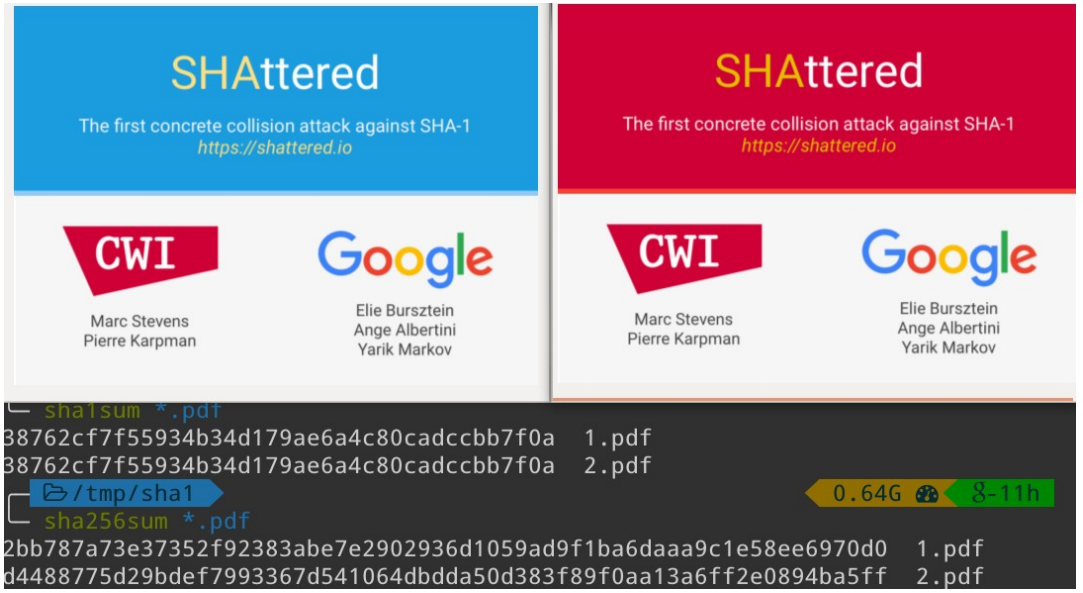
- **Colisiones**.
  - **MD5**. Vulnerable a un ataque conocido como *Collision Attack*.
  - Primer ejemplo práctico: **malware Flame**.
  - MD5 Collision Attack Lab. (Guide)



https://arstechnica.com/information-technology/2012/06/flame-crypto-breakthrough/

Atacando funciones hash

- **Colisiones**.
  - **SHA1**. CWI y Google anuncian el ataque **SHAttered** (febrero de 2017).
  - Ejercicio práctico colisiones MD5 y SHA1: CLCript 13
  - Ejemplos y explotación de colisiones hash.



Atacando funciones hash

- ***Hash length extension attack***.
  - Este ataque se emplea cuando el cliente envía datos al servidor mediante autenticado mediante un código MAC.
  - Si no se hace correctamente, un atacante situado como intermediario podría modificar los datos y generar una signatura válida para el mensaje.
  - Ejercicio práctico: Laboratorio SEED Lab.

https://github.com/iagox86/hash_extender

https://github.com/marcelo140/length-extension

Atacando funciones hash

- **Cracking**.
  - **Fuerza bruta**. Se computan todas las posibles combinaciones
  - **Ataques de diccionario**. Se utiliza una lista de palabras comunes utilizadas como contraseñas. Se debe hacer el proceso de generar el hash.
  - ***Rainbow tables*** (tablas arco iris). Son tablas precompiladas de hashes correspondientes a contraseñas comunes.

Atacando funciones hash

- **Cracking**. Fuerza bruta.

Estimated Password Recovery Times — 1x Terahash Brutalis, 44x Terahash Inmanis (448x Nvidia RTX 2080)
Full US keyboard mask attack with Terahash Hashstack

	Speed	Length 4	Length 5	Length 6	Length 7	Length 8	Length 9	Length 10	Length 11	Length 12	Length 13
NTLM	31.82 TH/s	Instant	Instant	Instant	Instant	3 mins 29 secs	5 hrs 30 mins	3 wks 0 day	5 yrs 7 mos	538 yrs 1 mo	51.2 mil
MD5	17.77 TH/s	Instant	Instant	Instant	Instant	6 mins 14 secs	9 hrs 50 mins	1 mo 1 wk	10 yrs 1 mo	963 yrs 4 mos	91.6 mil
NetNTLMv1 / NetNTLMv1+ESS	16.82 TH/s	Instant	Instant	Instant	Instant	6 mins 35 secs	10 hrs 24 mins	1 mo 1 wk	10 yrs 8 mos	1 mil	96.8 mil
LM	15.81 TH/s	Instant	Instant	Instant	Instant						
SHA1	5.89 TH/s	Instant	Instant	Instant	Instant	18 mins 47 secs	1 day 5 hrs	3 mos 3 wks	30 yrs 7 mos	2.9 mil	276.3 mil
SHA2-256	2.42 TH/s	Instant	Instant	Instant	Instant	45 mins 39 secs	3 days 0 hr	9 mos 1 wk	74 yrs 4 mos	7.1 mil	671.9 mil
NetNTLMv2	1.22 TH/s	Instant	Instant	Instant	Instant	1 hr 30 mins	5 days 23 hrs	1 yr 6 mos	147 yrs 10 mos	14.1 mil	1335.5 mil
SHA2-512	801.9 GH/s	Instant	Instant	Instant	1 min 28 secs	2 hrs 17 mins	1 wk 2 days	2 yrs 4 mos	224 yrs 9 mos	21.4 mil	2029.7 mil
descrypt, DES (Unix), Traditional DES	647.59 GH/s	Instant	Instant	Instant	1 min 48 secs	2 hrs 50 mins	1 wk 4 days	2 yrs 11 mos	278 yrs 3 mos	26.5 mil	2513.3 mil
Kerberos 5, etype 23, TGS-REP	206.97 GH/s	Instant	Instant	Instant	5 mins 38 secs	8 hrs 54 mins	1 mo 0 wk	9 yrs 2 mos	870 yrs 10 mos	82.8 mil	7864 mil
Kerberos 5, etype 23, AS-REQ Pre-Auth	206.78 GH/s	Instant	Instant	Instant	5 mins 38 secs	8 hrs 54 mins	1 mo 0 wk	9 yrs 2 mos	871 yrs 8 mos	82.9 mil	7871.2 mil
md5crypt, MD5 (Unix), Cisco-IOS 1 (MD5)	7.61 GH/s	Instant	Instant	1 min 37 secs	2 hrs 33 mins	1 wk 3 days	2 yrs 7 mos	249 yrs 5 mos	23.7 mil	2252.6 mil	213995.1 mil
LastPass + LastPass sniffed	1.78 GH/s	Instant	Instant	6 mins 52 secs	10 hrs 52 mins	1 mo 1 wk	11 yrs 2 mos	1.1 mil	101.1 mil	9600.8 mil	912079.6 mil
macOS v10.8+ (PBKDF2-SHA512)	335.09 MH/s	Instant	Instant	36 mins 34 secs	2 days 9 hrs	7 mos 2 wks	59 yrs 7 mos	5.7 mil	538.2 mil	51127.7 mil	4857134 mil
WPA-EAPOL-PBKDF2	277.23 MH/s					9 mos 0 wk	72 yrs 0 mo	6.8 mil	650.5 mil	61799.3 mil	5870931.8 mil
TrueCrypt RIPEMD160 + XTS 512 bit	211.78 MH/s	Instant	Instant	57 mins 52 secs	3 days 19 hrs	11 mos 3 wks	94 yrs 3 mos	9 mil	851.6 mil	80899.5 mil	7685455.6 mil
7-Zip	181.51 MH/s	Instant	Instant	1 hr 7 mins	4 days 10 hrs	1 yr 1 mo	110 yrs 0 mo	10.5 mil	993.6 mil	94389.2 mil	8966975.1 mil
sha512crypt 6, SHA512 (Unix)	119.46 MH/s	Instant	1 min 5 secs	1 hr 42 mins	6 days 18 hrs	1 yr 9 mos	167 yrs 2 mos	15.9 mil	1509.7 mil	143419.6 mil	13624861.4 mil
DPAPI masterkey file v1	47.23 MH/s	Instant	2 mins 44 secs	4 hrs 19 mins	2 wks 3 days	4 yrs 5 mos	422 yrs 10 mos	40.2 mil	3818.1 mil	362723.1 mil	34458696.1 mil
RAR5	28.15 MH/s	Instant	4 mins 35 secs	7 hrs 15 mins	4 wks 0 day	7 yrs 5 mos	709 yrs 7 mos	67.4 mil	6407.6 mil	608720.6 mil	57828453.9 mil
DPAPI masterkey file v2	27.82 MH/s	Instant	4 mins 39 secs	7 hrs 20 mins	4 wks 1 day	7 yrs 6 mos	717 yrs 10 mos	68.2 mil	6482.1 mil	615797.6 mil	58500769.5 mil
RAR3-hp	20.84 MH/s	Instant	6 mins 12 secs	9 hrs 47 mins	1 mo 1 wk	10 yrs 1 mo	958 yrs 2 mos	91.1 mil	8652.3 mil	821972.3 mil	78087367.8 mil
KeePass 1 (AES/Twofish) and KeePass 2 (AES)	17.8 MH/s	Instant	7 mins 15 secs	11 hrs 28 mins	1 mo 2 wks	11 yrs 9 mos	1.1 mil	106.7 mil	10131.9 mil	962529.5 mil	91440305.8 mil
bcrypt $2*$, Blowfish (Unix)	11.37 MH/s	Instant	11 mins 21 secs	17 hrs 57 mins	2 mos 1 wk	18 yrs 5 mos	1.8 mil	167 mil	15860.3 mil	1506727.9 mil	143139150.9 mil
Bitcoin/Litecoin wallet.dat	3.55 MH/s	Instant	36 mins 18 secs	2 days 9 hrs	7 mos 2 wks	59 yrs 1 mo	5.6 mil	534.1 mil	50743.7 mil	4820655.6 mil	457962282.7 mil

https://twitter.com/TerahashCorp/status/1155128018156892160

Atacando funciones hash

- **Cracking**. Ataques de diccionario
  - Listas de contraseñas. En Kali: ***/usr/share/wordlists***
    - **Diccionario Rockyou**. La empresa del mismo nombre sufrió una brecha de seguridad que dejó a la vista más de 32 millones de cuentas (las contraseñas se almacenaban en claro).
    - ***Seclists***. Colección de listas de palabras agrupadas por categorias, elaborada por el investigador Daniel Miessler. Se puede clonar el repositorio de GitHub o instalar mediante CLI: ***apt install seclists***.
    - **Listas de aplicaciones de fuzzing**. Aplicaciones como *dirb*, *dirbuster*, *wfuzz*, etc. disponen de listas de palabras que se pueden emplear en cualquier otra herramienta.
    - Wordlists for Pentester. Artículo de HackingArticles recopilatorio de listas y herramientas para crear diccionarios.
    - SkullSecurity. Listado de ficheros procedentes de leaks.
    - **Kaonashi Project**. Presentado en la RootedCON 2019. Dispone de diccionarios, máscaras y reglas para hashcat.
    - Default password list. Lista de contraseñas por defecto de numerosos dispositivos.
    - WordList-Compendium. Recopilación de enlaces interesantes relacionados con diccionarios.

Atacando funciones hash

- **Cracking**. Ataques de diccionario
  - Crear diccionarios de contraseñas:
    - **Crunch**: creación de listas dado un alfabeto y longitud.
    - CeWL: creación de diccionario basado en web.
  - Ataques online:
    - **Hydra**. Permite realizar ataques de fuerza bruta/diccionario a servicios activos (FTP, HTTP, SSH, etc.).
    - **Medusa**. Similar a Hydra.
    - **Ncrack**. Desarrollada por *nmap* con una sintaxis similar a esta herramienta.

Atacando funciones hash

- **Cracking**. Rainbow tables.
  - Software y tablas.
    - Rainbow-crack.
    - 0phcrack.
    - Free rainbow tables.
    - Cain&Abel (sin mantenimiento).
  - **Ataques offline** (herramientas gratuitas/comerciales).
    - Hashcat. (GitHub).
    - John The Ripper. (GitHub). Comprehensive guide to John the Ripper.
    - Terahash.
    - Passware.
    - Hashsuite.

FIN