Desarrollo de software y criptografía

¿cómo proteger los datos en nuestras aplicaciones?

Gunnar Wolf

Debian • IIEc-UNAM • FI-UNAM

SG Conferencia y Expo 2014

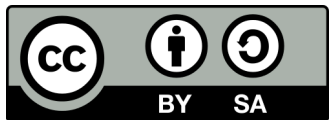
Contenidos

Acerca de mí

- Desarrollador, administrador de sistemas, entusiasta...
  - Como todos ustedes (¡espero!)
- *Mantenedor* del *llavero de confianza* OpenPGP en el *Proyecto Debian*
- No soy *experto* en criptografía
  - Me resulta interesante
  - Estoy convencido de la *importancia* de que los desarrolladores la comprendan y utilicen.

Acerca de la ponencia

- Mencionaré algunos (no demasiados) datos y referencias
- La presentación está disponible en http://gwolf.org/desarrollo_y_criptografia
- Con *ligas vivas* a la fuente de información donde haga falta



La criptografía y el *mundo real*

- ¿Para qué desarrollamos sistemas?
- ¿Qué sabemos de nuestra información?
- ¿Cómo podemos *proteger* nuestra información?
- ¿Y en qué nos podemos equivocar?

La criptografía y el *mundo real*

- ¿Para qué desarrollamos sistemas?
  - Invariablemente, para gestionar información
- ¿Qué sabemos de nuestra información?
  - En ella radica el valor de nuestro trabajo
- ¿Cómo podemos *proteger* nuestra información?
- ¿Y en qué nos podemos equivocar?

La criptografía y el *mundo real*

- ¿Para qué desarrollamos sistemas?
  - Invariablemente, para gestionar información
- ¿Qué sabemos de nuestra información?
  - En ella radica el valor de nuestro trabajo
- ¿Cómo podemos *proteger* nuestra información?
  - ...¿Qué significa proteger?
- ¿Y en qué nos podemos equivocar?
  - ¡A eso vamos!

Propiedades a *defender* en un mensaje

¿Qué podemos *asegurar* empleando técnicas criptográficas?

- Confidencialidad
- Integridad
- Autenticación

Y varias otras propiedades que *derivan* de estas.

Un par de ejemplos

Veamos brevemente un ejemplo de cada uno.

Importante: Son ejemplos *basados* en nuestra realidad, aunque no absolutamente literales

Confidencialidad

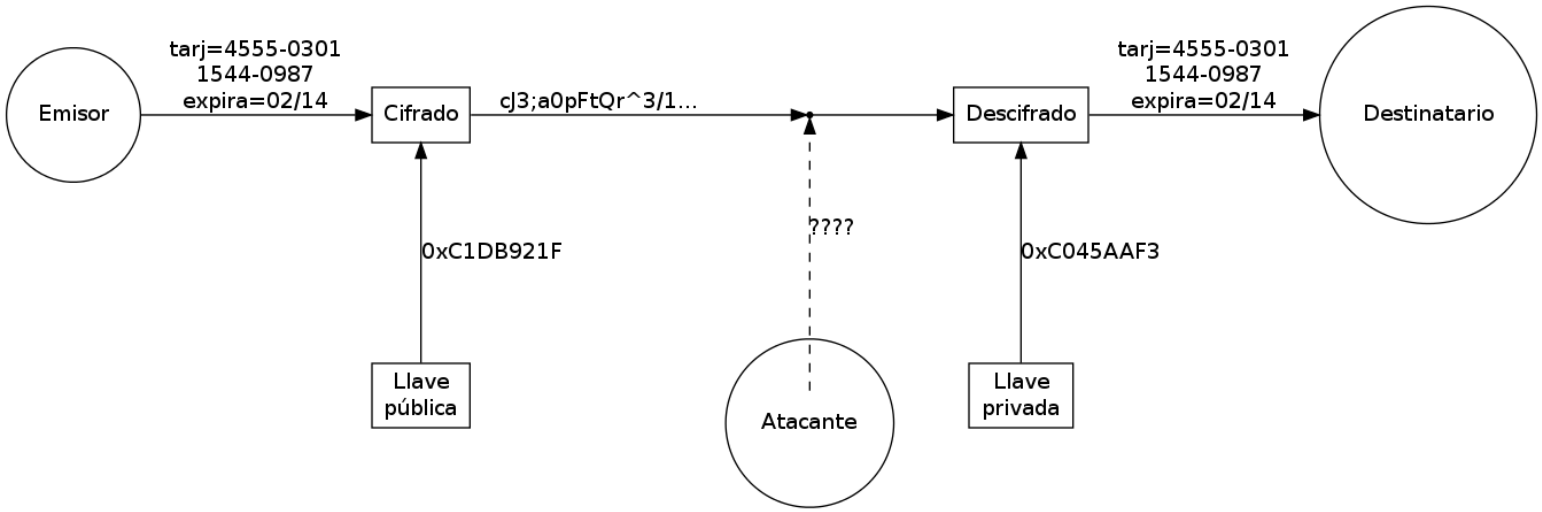


Figura: Ejemplo clásico de confidencialidad: Transmisión de datos para el comercio electrónico

Autenticación

- ¿Qué hacen para verificar usuario/contraseña en sus sistemas? ¿Y qué graban en la base de datos?

```
1 user = User.find_by_login(params[:login])
2 return false if user.blank?
3 passwd_hash = Digest::MD5.hexdigest(user.pw_salt +
      params[:passwd])
4 return (user.passwd != passwd_hash) ? false : true
```

```
1 sistema=> select login, passwd, pw_salt from users
      where login='gwolf';
2  login |                passwd                | pw_salt
3 -------+--------------------------------------+----------
4  gwolf | ce0f7176bd13fd657770cefb55923b7f | {BwHnnL?
5 (1 row)
```

- *Nunca* se guarda la contraseña, sino una *prueba criptográfica* de su posesión.

Autenticación

- ¿Qué hacen para verificar usuario/contraseña en sus sistemas? ¿Y qué graban en la base de datos?

```
1 user = User.find_by_login(params[:login])
2 return false if user.blank?
3 passwd_hash = Digest::MD5.hexdigest(user.pw_salt +
      params[:passwd])
4 return (user.passwd != passwd_hash) ? false : true
```

```
1 sistema=> select login, passwd, pw_salt from users
      where login='gwolf';
2  login |                passwd                | pw_salt
3 -------+--------------------------------------+----------
4  gwolf | ce0f7176bd13fd657770cefb55923b7f | {BwHnnL?
5 (1 row)
```

- *Nunca* se guarda la contraseña, sino una *prueba criptográfica* de su posesión.

Autenticación

- ¿Qué hacen para verificar usuario/contraseña en sus sistemas? ¿Y qué graban en la base de datos?

```
1 user = User.find_by_login(params[:login])
2 return false if user.blank?
3 passwd_hash = Digest::MD5.hexdigest(user.pw_salt +
      params[:passwd])
4 return (user.passwd != passwd_hash) ? false : true
```

```
1 sistema=> select login, passwd, pw_salt from users
      where login='gwolf';
2  login |              passwd              | pw_salt
3 -------+----------------------------------+----------
4  gwolf | ce0f7176bd13fd657770cefb55923b7f | {BwHnnL?
5 (1 row)
```

- *Nunca* se guarda la contraseña, sino una *prueba criptográfica* de su posesión.

Integridad

- Al bajar un archivo de Internet, quiero confirmar que sea *exactamente el mismo* que el que subió el autor
  - Muchos autores y depósitos de software lo publican junto con su *hash* (MD5, SHA1, SHA-256)
- No se comprueba la identidad del autor, sino únicamente la integridad del documento

```
$ apt-cache show linux-image-3.14-1-amd64
Package: linux-image-3.14-1-amd64
  (...)
Size: 30780396
MD5sum: 251351c12ed891abf3659514f62d05c6
SHA1: 8bbf040135253e96b4624ad0e141672015b0394a
SHA256:
    072815c82ebd18f7998fffb441faea571518a6e1502652687d33
```

- Importante: ¿Estos datos se transmitieron por un *canal confiable*?

Integridad

- Al bajar un archivo de Internet, quiero confirmar que sea *exactamente el mismo* que el que subió el autor
  - Muchos autores y depósitos de software lo publican junto con su *hash* (MD5, SHA1, SHA-256)
- No se comprueba la identidad del autor, sino únicamente la integridad del documento

```
$ apt-cache show linux-image-3.14-1-amd64
Package: linux-image-3.14-1-amd64
  (...)
Size: 30780396
MD5sum: 251351c12ed891abf3659514f62d05c6
SHA1: 8bbf040135253e96b4624ad0e141672015b0394a
SHA256:
    072815c82ebd18f7998fffb441faea571518a6e1502652687d33
```

- Importante: ¿Estos datos se transmitieron por un *canal confiable*?

Algunas otras propiedades *derivadas*

- Vinculación / *no-repudio*
- Certificación
- Control de acceso
- Tiempo de validez: Validación / expiración
- Sello de tiempo
- Revocación
- Recibo / confirmación
- Firma anónima o ciega

Ojo: Varias de estas dependen de la existencia de un *tercero confiable* (autoridad certificadora).

Y con esto, ¿a qué quiero llamar la atención?

- Los esquemas antes descritos *son seguros* y *están comprobados al 100 %*.
  - Módulo *teoría de la complejidad* → Más al respecto en un minuto
- Pero... ¿Y la implementación que los rodea / llama / invoca?

Contenidos

¿Cómo voy a implementar mi criptografía?

- Los diferentes algoritmos son ampliamente conocidos y están públicamente documentados
  - Criptografía simétrica / asimétrica; diferentes longitudes de llave, diferentes *modos de operación*
  - ¿Por qué?

- Muchos de estos algoritmos son –aparentemente– fáciles de comprender e implementar. ¿Por qué no hacerlo?

¿Cómo voy a implementar mi criptografía?

- Los diferentes algoritmos son ampliamente conocidos y están públicamente documentados
  - Criptografía simétrica / asimétrica; diferentes longitudes de llave, diferentes *modos de operación*
  - ¿Por qué? Cada uno presenta ciertas ventajas y desventajas en situaciones específicas; algunos ejemplos en breve
- Muchos de estos algoritmos son –aparentemente– fáciles de comprender e implementar. ¿Por qué no hacerlo?

¿Cómo voy a implementar mi criptografía?

- Los diferentes algoritmos son ampliamente conocidos y están públicamente documentados
  - Criptografía simétrica / asimétrica; diferentes longitudes de llave, diferentes *modos de operación*
  - ¿Por qué? Cada uno presenta ciertas ventajas y desventajas en situaciones específicas; algunos ejemplos en breve
- Muchos de estos algoritmos son –aparentemente– fáciles de comprender e implementar. ¿Por qué no hacerlo?

¡Nunca lo hagan!

Regla de oro de la criptografía

No *haces* tu propia criptografía

Regla de oro de la criptografía

No *haces* tu propia criptografía

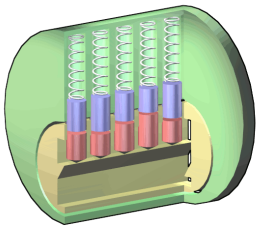
...Pero asegúrate de entender la que existe

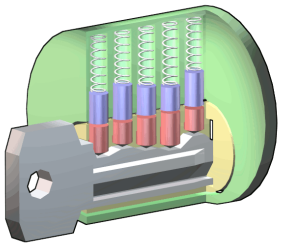
Las mejores soluciones: Las más probadas

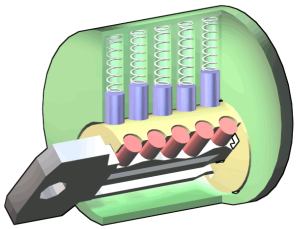
- Mecanismo existente por más de 4,000 años
- Con adecuaciones / actualizaciones
- Pero el mecanismo con el que aún hoy estamos más familiarizados

Imagen: Wikipedia, *Cerradura*

Las mejores soluciones: Las más probadas







Imágenes: Wikipedia — *Cerradura de tambor de pines*

- Largo proceso de estandarización y revisión de un algoritmo criptográfico
  - ¡No se usa criptografía nueva!
- Importancia de la *revisión por pares*
  - Adecuaciones necesarias a mecanismos existentes
  - Y, en contados casos, la *demostración* de su ineficacia
  - p.ej. Knapsacks (Merkle-Hellman, 1978) → Shamir, 1982

Y va más allá del mero símil

«La seguridad se logra a través de la apertura. Desarmar las cosas y jugar con ellas (...) exponiendo a las fallas de seguridad es lo que nos protege a todos nosotros.»

–Deviant Ollam, http://toool.us/
Introduction to Lockpicking and Physical Security
DEFCON 13, 2005

Volviendo a la regla de oro: Los criptoanalistas

Cualquiera, sin importar qué tan pocas habilidades tenga, puede diseñar un algoritmo que él mismo no pueda romper

Bruce Schneier, 1999

Leyes de Shamir: La criptografía es *fuerte*, pero. . .

Tres Leyes de la Seguridad, de Adi Shamir

1. Los sistemas absolutamente seguros no existen
2. Para reducit tu vulnerabilidad a la mitad, debes duplicar tus gastos
3. La criptografía normalmente es evadida, no penetrada

La criptografía normalmente es evadida, no penetrada

La criptografía normalmente es evadida, no penetrada. No conozco ningún sistema importante y de uso mundial que emplee criptografía en el que los atacantes penetraran al sistema a través del criptoanálisis. (...) Normalmente hay maneras mucho más simples de penetrar el sistema de seguridad.

–Adi Shamir

Ejemplos de evasión de criptografía

- Consolas de videojuego (PS3, Wii, Xbox)
  - Ejecutables firmados, almacenamiento cifrado, cifrado de memoria, protección de integridad, coprocesadores de seguridad, ...
- Prácticamente todos los teléfonos, tabletas y dispositivos similares
  - SecureBoot, protección desde el cargador del sistema operativo
- Esquema antispam basado en llaves públicas (DKIM — *DomainKeys Identified Mail*)
  - 4,000 de 12,000 organizaciones investigadas en 2012 vulnerables por no usar llaves suficientemente fuertes
  - Incluyendo Amazon, Apple, Dell, eBay, HP, HSBC, LinkedIn, Paypal, Twitter...

¿Existen algoritmos *trucados*?

- La alteración de las *S-boxes* de DES
  - La NSA alteró el órden interno de las cajas de substitución en DES sin dar explicación
  - Muchos creyeron que debilitaban el mecanismo — en realidad, lo *blindaron* contra análisis diferencial (que no fue conocido públicamente hasta 1990)
- Sin embargo, el Dual_EC_DRBG (*generador determinístico de bit aleatorio por doble curva elíptica*)...
  - Diseñado por la NSA, forma parte del estándar NIST SP 800-90A
  - *Backdoor* intencional en su planteamiento
  - Documentado como parte de las *fugas* de Snowden (2013); NIST retira Dual_EC_DRBG del estándar en abril de 2014

Dual_EC_DRBG como caso atípico

RSA La NSA pagó US$10,000,000 a RSA para que elija Dual_EC_DRBG como default en la suite criptográfica *BSafe*...

Friday, September 20, 2013

RSA warns developers not to use RSA products

In today's news of the weird, RSA (a division of EMC) has recommended that developers desist from using the (allegedly) 'backdoored' Dual_EC_DRBG random number generator -- which happens to be the *default* in RSA's BSafe cryptographic toolkit. Youch.

RSA

The Security Division of EMC

Microsoft Microsoft incluyó este algoritmo en su sistema operativo porque un *cliente importante* lo solicitó. –Kim Zetter

OpenSSL Un patrocinador lo solicitó como uno de varios entregables. El razonamiento (...) que implementaríamos cualquier algoritmo basado en estándares publicados oficiales

En resumidas cuentas. . .

Los ataques no rompen el cifrado.

Aprovechan debilidades, sea en la implementación o en la interfaz con esta.

. . . Excepto en *muy contados casos*, en que las *puertas traseras* son introducidas a propósito *desde el diseño*

Pero no sólo referente al algoritmo. . .

- Complejidad de implementar *bien* un algoritmo
- Complejidad de interoperar con *todos los demás* que usan ese mismo cifrado
  - Nuevos programadores que se integren al equipo
- Incluso en un desarrollo 100 % propio: Mantenimiento *a futuro*

Empleo de implementaciones *estándar*

- Permitir a *quien sí sabe* dar un mantenimiento responsable a mi código
- Software ampliamente disponible
- En gran parte, bajo licenciamiento libre
- Disponibles en fuente $\rightarrow$ Auditables
- Amplias comunidades de usuarios *conocedores* y comprometidos

Aún así, mucho por aprender

- Incluso usando implementaciones estándar de algoritmos bien reconocidos como seguros, tenemos que comprender lo que hacemos
- O si no: *Falso sentido de seguridad*
  - Confiar ciegamente en un proceso *agujereado* $\rightarrow$ mayores riesgos que si no doy por sentado que mis datos están seguros

Ejemplo: *Modo de operación* ECB

- El estándar NIST Special Publication 800-38A define cinco *modos de operación* para el cifrado *de llave simétrica* para *algoritmos de cifrado en bloques*
- El primero de estos, *Electronic Code Book*, tiene algunas propiedades interesantes:
  - Eficientemente paralelizable
  - Mínimo daño al mensaje ante ruido
- Sin embargo, *nunca debe utilizarse* para mensajes largos, donde pueda haber *más de un bloque igual*

Implementación ejemplo: AES (ECB) con Ruby+OpenSSL


```ruby
require 'openssl'
require 'base64'

cifrador = OpenSSL::Cipher::Cipher.new('AES-128-ECB')
cifrador.encrypt
cifrador.key = 'Un4_ll4v3_b13n_53gUr4'
result = cifrador.update('Esto es todo.   Esto es todo.
    Esto es todo.   Esto es todo.   Esto es todo.   Esto
    es todo.   Esto es todo.   Esto es todo.   ')
result << cifrador.final
puts Base64.encode64(result)
```

Resultado:

```
cclaoTOpe/SjNJDEb67JynHJWqEzqXv0ozSQxG+uycpxyVqhM6l79KM0kMRv
rsnKcclaoTOpe/SjNJDEb67JyhWjrh7E34p6rEOYLvGpAJk=
```

¿Ubican el problema?

Ilustrándolo gráficamente

Figura: Imagen original

Figura: Cifrado empleando ECB

Figura: Cifrado empleando OFB

Modos de operación: ¿Y por qué?

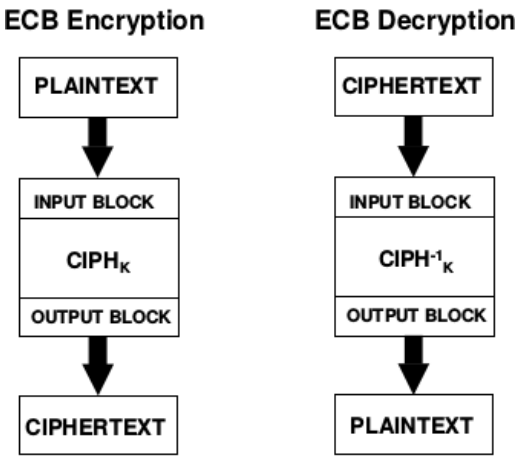


Figure 1: The ECB Mode

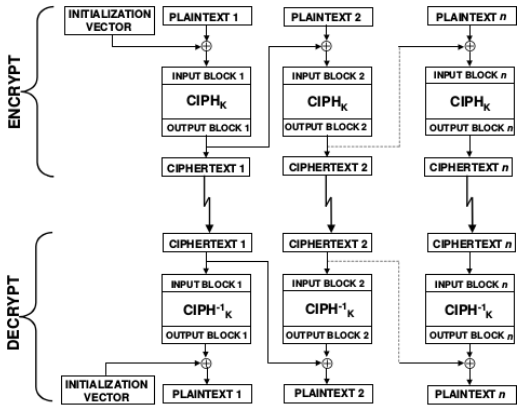


Figure 2: The CBC Mode

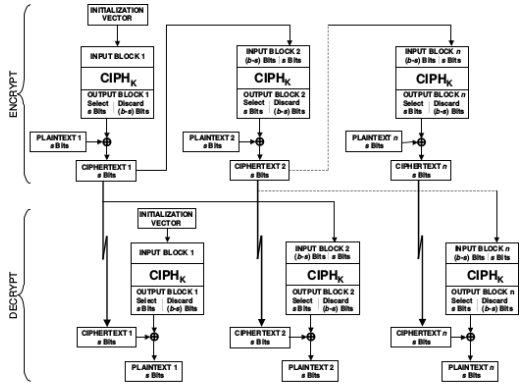


Figure 3: The CFB Mode

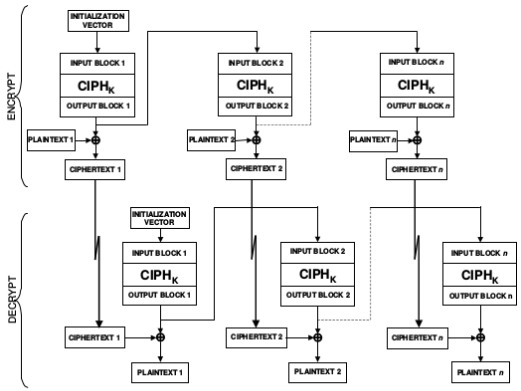


Figure 4: The OFB Mode

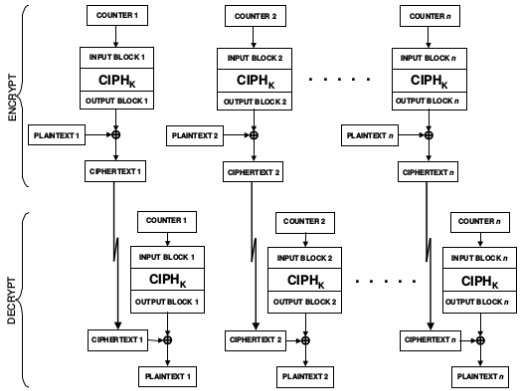


Figure 5: The CTR Mode

Esquemas tomados de NIST SP 800-38A

Contenidos

Cuando se presentan problemas

- El cifrado es difícil de implementar y mantener *bien* — Incluso para los expertos
- ¿Qué pasa cuando los expertos se equivocan?
  - ¿Cómo nos damos cuenta?
  - ¿Cómo podemos reaccionar?
  - ¿Qué aprendemos de estos errores?

A continuación, algunos ejemplos

Debian DSA-1571-1 openssl – predictable random number generator

- En mayo de 2006, el *mantenedor* de OpenSSL en Debian comentó una línea de código:

```
1  MD_Update(&m,buf,j);
```

  - Porque *Valgrind* marcaba *acceso a memoria no inicializada* (¡Recuerden esto!)
- Pero esto efectivamente *redujo la recolección de entropía a prácticamente cero*
  - En Debian y todas sus distribuciones derivadas
  - Hasta mayo del 2008, en que Luciano Bello (también de Debian) descubrió el problema
- ¿Resultado?
  - Millones de llaves (x.509, SSH) reducidas de un espacio 2^{128} a uno 2^{32}

Apple: *Too big to fail* (1)

En febrero de 2014, se descubrió este error en el código criptográfico de los sistemas operativos Apple:

```
1   if ((err = SSLFreeBuffer(&hashCtx)) != 0)
2       goto fail;
3   if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
4       goto fail;
5   if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom))
        != 0)
6       goto fail;
7   if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom))
        != 0)
8       goto fail;
9   if ((err = SSLHashSHA1.update(&hashCtx, &signedParams))
        != 0)
10      goto fail;
11      goto fail;
12  if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
13      goto fail;
```

Apple: *Too big to fail* (1)

- El segundo `goto fail` de la verificación con `&signedParams` lleva a *escapar* de la correcta verificación de la cadena de confianza
- Lo cual volvió básicamente inútil todo el código criptográfico de Apple
  - Facilitando ataques de *hombre en el medio* (*MitM*)
- ¿Impacto? Millones de dispositivos MacOS, iOS vulnerables

OpenSSL y *Heartbleed*

- En abril de 2014 se dio a conocer el fallo ampliamente publicitado bajo el nombre *Heartbleed*
- En resumen: Un mal manejo de límites en memoria y el uso repetido de memoria no correctamente asignada/liberada causa fugas de información arbitraria
  - Bloques de hasta 64K del espacio de memoria del proceso
  - Contiene *cualquier cosa*
- Causado no por una línea como los dos anteriormente mencionados
  - Sino por prácticas de programación *muy* cuestionables
  - En una biblioteca *muy ampliamente* utilizada

OpenSSL y *Heartbleed*

- En abril de 2014 se dio a conocer el fallo ampliamente publicitado bajo el nombre *Heartbleed*
- En resumen: Un mal manejo de límites en memoria y el uso repetido de memoria no correctamente asignada/liberada causa fugas de información arbitraria
  - Bloques de hasta 64K del espacio de memoria del proceso
  - Contiene *cualquier cosa*
  - ...¿Recuerdan lo que llevó al *DSA-1571-1*?
- Causado no por una línea como los dos anteriormente mencionados
  - Sino por prácticas de programación *muy* cuestionables
  - En una biblioteca *muy ampliamente* utilizada

Impacto de *Heartbleed*

- Este bug llevó a un amplio debate acerca de la falta de auditoría en infraestructura crítica y ampliamente empleada, como OpenSSL
- Al día de hoy, *tres* esfuerzos independientes de auditoría al código (resultando en dos *forks*)
  - Linux Foundation: Core Infrastructure Initiative
  - OpenBSD: LibReSSL → Reporte de avance (Bob Beck)
  - Google: BoringSSL
- *Mucha gente* buscando fallos de forma independiente en las implementaciones de criptografía

Impacto de *Heartbleed*

- Este bug llevó a un amplio debate acerca de la falta de auditoría en infraestructura crítica y ampliamente empleada, como OpenSSL
- Al día de hoy, *tres* esfuerzos independientes de auditoría al código (resultando en dos *forks*)
  - Linux Foundation: Core Infrastructure Initiative
  - OpenBSD: LibReSSL → Reporte de avance (Bob Beck)
  - Google: BoringSSL
- *Mucha gente* buscando fallos de forma independiente en las implementaciones de criptografía
  - Obviamente, muchos nuevos fallos van apareciendo

¿Google busca crear código *aburrido*?

(...) Fundamental security components like SSL/TLS should be very, very boring. They're not a place for innovation and experimentation, they're not a place for clever code that demonstrates the author's virtuosity (...). They're not a place for exploration of how the C preprocessor can be used to automatically generate much of the codebase (which is something that OpenSSL has done). They're where you want very simple, straightforward, boring implementations of industry best practice algorithms and protocols.
When it comes to security, boring is good.

–swillden; slashdot.org 21/06/2014

Lo rescatable. . .

- Estos fallos son únicamente los *más publicitados* recientemente
  - Cosa de comenzar a rascarle. . .
- Los fallos no sólo llevan a revisión del código
  - Van creando *cambios culturales* en el desarrollo de software
- En las culturas de desarrolladores de software *públicamente auditable* ($\rightarrow$ libre, necesariamente) cada uno de estos casos llevan a discusiones y cambios *muy positivos* de prácticas
  - p.ej. cambio del proceso de gestión de *parches* en Debian; Patch Tagging Guidelines

Lo rescatable. . .

- Cada vez hay más conciencia de la importancia del uso de mecanismos criptográficos *para todo*
  - Se va convirtiendo en la norma, no la excepción
  - p.ej. campaña+plugins HTTPS Everywhere
- Pero mucho camino por recorrer
  - . . . Y ni hablamos de los problemas derivados de certificados x.509 (SSL) comprometidos → EFF SSL Observatory
  - Ni de esquemas alternos (y *muy* probados) de verificación de identidad, como los *anillos de confianza* de PGP, en contraposición de la PKI de las Autoridades Certificadoras. . .

A fin de cuentas. . .

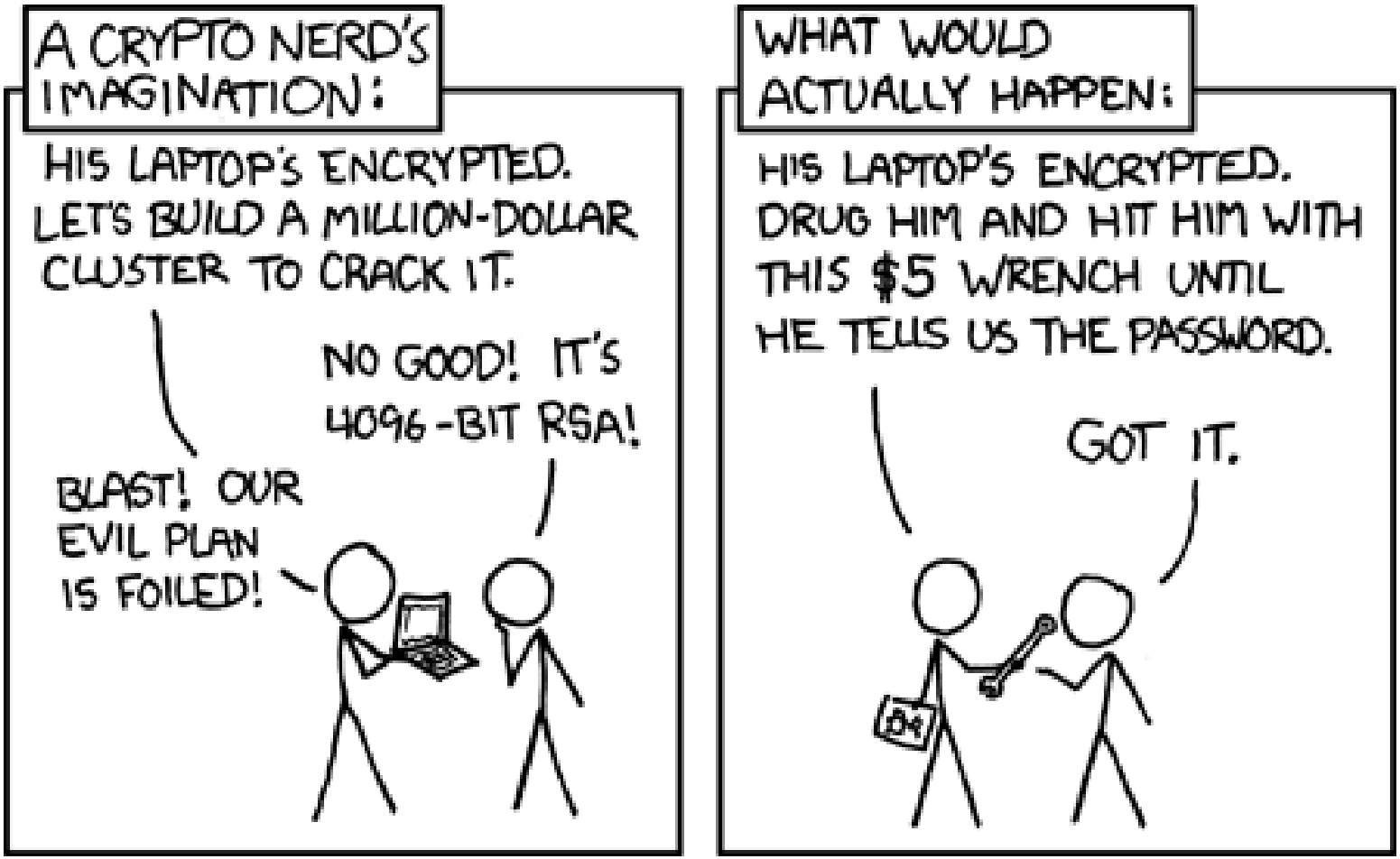


Figura: http://xkcd.com/538/

Recursos útiles

Buena parte del material aquí presentado proviene directa o indirectamente de:

- Crypto won't save you either (Peter Gutmann)
- LibreSSL — An OpenSSL replacement. The first 30 days, and where we go from here. (Bob Beck)

Fin

Aquí se termina mi material preparado...

Por mi parte,

Gracias.

Gunnar Wolf gwolf@gwolf.org
http://gwolf.org/desarrollo_y_criptografia