Desarrollo seguro de aplicaciones con criptografía

¿cómo proteger los datos en nuestras aplicaciones?

Gunnar Wolf

ESITI • SEPI ESIMECU

2° Congreso de Seguridad de la Información, Instituto Politécnico Nacional

Punto de partida

- La industria del desarrollo por fin va comprendiendo la importancia de implementar criptografía
  - Escuchas / filtraciones / medios
  - Fallos criptográficos de alto perfil
- Planes de estudios relacionados con seguridad en cómputo
  - Fuerte enfoque en la criptografía
  - . . . Pero anclado en el aspecto algorítmico, teórico/científico
  - Muy poco en la parte *aplicada*

Desconexión academia/industria

- Consecuencia natural: **Desconexión** academia $\nleftrightarrow$ industria
- Nuestro papel como alumnos de posgrado
  - ¿Cómo nuestros estudios se enmarcan en el campo?
  - ¿Cómo contribuir para mejorar la sociedad con nuestro actuar especializado?
  - *Responsabilidad social* de buscar contribuir

Lo que busco cubrir

- Ejemplos de fallos de implementación
  - Fallos aparentemente triviales
  - Mecanismos criptográficos bien reconocidos
  - . . . Resultados catastróficos

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.

3ª de las Tres Leyes de la Seguridad (Adi Shamir, 2003)

¿Qué se aprende de estas vulnerabilidades?

- Los principales usos de criptografía se basan siempre en bibliotecas *bien conocidas*
- Pequeños *detalles* en la implementación
  - Resultan ser *devastadores* para la seguridad
- ¿Qué aprendemos?
  - Más ejemplos acerca de dónde ser cuidadosos
  - Prácticas comunes a evitar
  - Dónde buscar problemas similares (para bien o para mal)

Debilitamiento criptográfico

- *One-time pad*
  - Protección criptográfica perfecta y muy baja dificultad computacional
  - Mecanismo de transmisión del *pad*: Absolutamente impráctico
- Llaves de cifrado
  - Típicamente como semillas para generar un OTP
  - Espacios de búsqueda: Entre 2^{128} y 2^{256} para cifrado simétrico, entre 2^{1024} y 2^{4096} para llave pública
  - Perspectiva: El universo tiene 2^{80} átomos...
- Requieren de alta entropía
  - El *azar* es muy dificil para una computadora
  - Fuentes de entropía $\rightarrow$ interacción con el usuario, ruido en la red, etc.

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

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

```
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:

```
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) !=
    0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) !=
    0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) !=
    0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
```

Apple: *Too big to fail* (2)

- 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

Divulgación de información sensible

- Las llaves privadas deben *protegerse a conciencia*
- Divulgarlas equivale a perder la protección que nos *ha brindado* el cifrado — Ahora e históricamente
  - *Perfect Forward Secrecy*, pero muy poco utilizado aún
- Una vulnerabilidad que lleve a la divulgación de llaves privadas es necesariamente de alto perfil

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 económico de *Heartbleed*

¿Quién resultó beneficiado en primer lugar?
El *cártel* de las autoridades certificadoras (PKI)

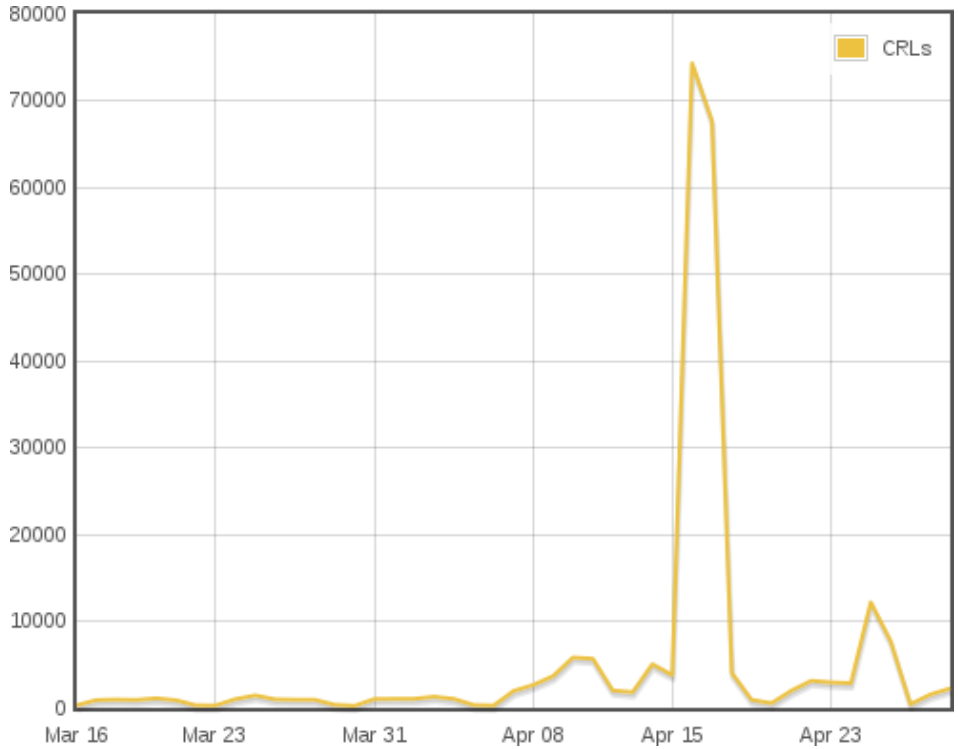

Figura: Revocaciones a certificados X.509 entre el 15 de marzo y 30 de abril del 2014 (SANS ISC)

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
  - 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
  - Google: BoringSSL
- *Mucha gente* buscando fallos de forma independiente en las implementaciones de criptografía
  - Obviamente, muchos nuevos fallos van apareciendo

Más allá del alto perfil

- Presenté tres ejemplos relativamente recientes, muy publicitados y circulados
  - Punto en común: Dentro de bibliotecas criptográficas ampliamente utilizadas
- Obviamente, sobran ejemplos de uso incorrecto de material criptográfico
- Siguen algunos ejemplos menos *notables*
  - Pero tal vez más cercanos a lo que enfrentarán los programadores *de a pie*.

WEP y el uso correcto de los modos de operación

- *Wired Equivalent Privacy*: Primer estándar de cifrado para redes inalámbricas 802.11 (1999)
  - Meta: Privacidad equivalente a la obtenida en una red cableada
  - No aísla a un usuario de otro dentro de la misma red
  - Evita a terceros no autorizados
- 2001: Ataques sobre WEP que hacen trivial obtener la llave por medio de una captura pasiva
  - RC4 con llaves de 64/128b

WEP y el uso correcto de los modos de operación

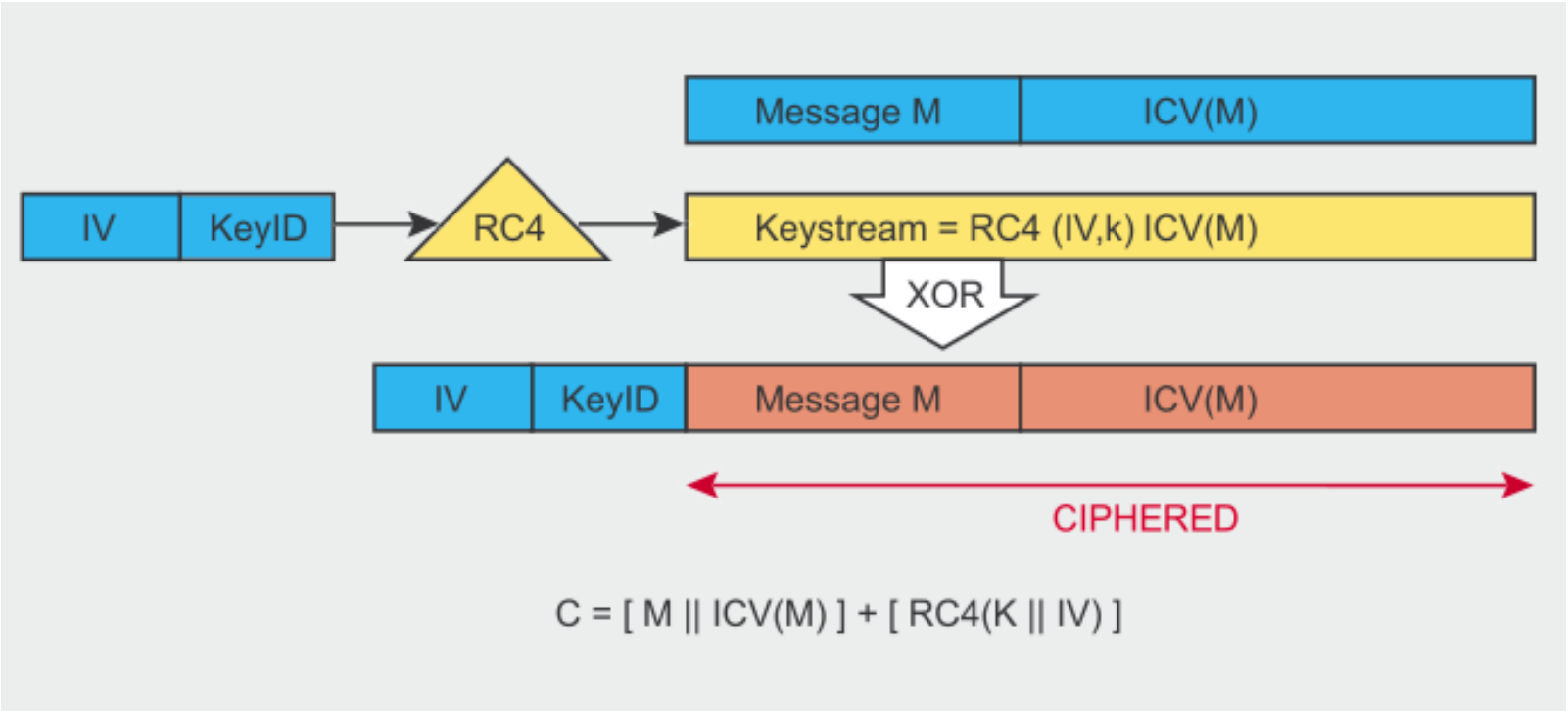

Figura: Esquema de operación de WEP

WEP y el uso correcto de los modos de operación

- Recortadas a 40/104, reservando 24 bits para el *vector de inicialización*
  - Problema: Espacio de vectores demasiado pequeño; repetición tras 16,777,216 paquetes
  - Se pueden inyectar paquetes mal-formados para obligar a retransmitir
  - $\approx$ 15 minutos de tráfico
- Mensaje claro con alta regularidad
  - Contiene una trama Ethernet y (casi siempre) un paquete IP $\Rightarrow$ Trivial reconocer el resultado por fuerza bruta
- Existen ya mejores implementaciones
  - 2004: WPA
  - ...Pero al día de hoy, $\approx$ 10 % de las redes inalámbricas (empírico) emplean aún WEP

Distribución de llaves: Cuando mis dispositivos no son realmente míos

- Los mecanismos de distribución de llaves públicas han sido largamente estudiados
  - 1976: Diffie y Hellman, *New directions on cryptography*
- Pero... ¿Qué pasa cuando la llave *privada* forma parte de un dispositivo?
  - ¿Y este dispositivo busca *controlar las acciones* de su usuario?

Distribución de llaves: Delineando el problema

- Estos dispositivos son, a fin de cuentas, computadoras de propósito general
- Autentican criptográficamente ante un proveedor de servicios ya sea al usuario o a los bibarios a ejecutar
- Cuando el usuario tiene acceso físico al hardware, tiene acceso potencial a todo lo que éste contiene
  - Incluyendo las llaves privadas
- Ejemplos de por qué esto no sirve... sobran.

Distribución de llaves y el *jailbreak*

Amazon Kindle Binarios firmados por una llave RSA/1024. El atacante puede *fabricar* una y reemplazarla.

Telefonía celular Los proveedores de telefonía y desarrolladores deben *aprobar* ciertas funcionalidades, pues pueden incorporarlas en paquetes más caros (p.ej. *tethering*, bloqueo de proveedor). Mismo mecanismo de ataque.

Consolas de videojuegos PS3, Wii, XBox. Ejecutables firmados, almacenamiento cifrado, cifrado de memoria, protección de integridad... Todos *rotos* encontrando cómo evadir las verificaciones.

Distribución de llaves: Proveniencia de imágenes

- Cámaras Canon y Nikon de gama alta: Incluían un sistema de autenticación de imágenes
  - Protegiendo propiedad e integridad
  - Hash HMAC (256) firmado por RSA (1024) única por cámara, como parte de un campo EXIF
- *Elcomsoft* (Rusia) logró extraer las llaves privadas de ambas cámaras (2011)
  - Publicó fotografías modificadas firmadas con llaves de ambos tipos de cámaras

Apple iOS: Brincando la verificación

- Apple requiere la firma de *todo binario* que se ejecute en iOS (desde el mismo bootloader)
- Promedio entre aparición de un iOS nuevo y su primer *jailbreak*: 56 días (Rango: 1..143 días)
  - Antes mucho más cortos; Apple reacciona dificultando el proceso
- En todos los casos: Resultado de explotar al software para *impedir que realice la comprobación*
  - Nunca resultado de obtener la llave firmante o romper la criptografía

Patrones emergentes: La red TOR

- Proyecto que busca crear una red privada superpuesta a Internet
  - Mantener el anonimato
  - Orientado a libertad de expresión bajo regímenes totalitarios, aunque también utilizado para fines criminales
- Oculta origen-destino y contenido de las comunicaciones por *ruteo cebolla*
- Vulnerable a que una entidad controle una *proporción significativa* de sus nodos
- Patrones de tráfico analizables pueden revelar origen-destino de ciertas conexiones

Patrones emergentes: Análisis de VoIP cifrado

- Casi todos los esquemas de voz sobre IP (VoIP) implementan cifrado
- . . . Pero emplean codecs *adaptativos*
  - Optimizan ancho de banda detectando silencios, cancelando retorno, comprimiendo diferenciadamente, etc.
- Análisis de *cantidad* de paquetes enviados (y conocimiento de patrones idiomáticos) $\rightarrow$ Descifrar conversaciones enteras
  - Sin atacar al cifrado
  - Sólo como una función sobre la probabilidad de datos sobre el tiempo

Conclusión

- La criptografía es un campo maduro y riguroso en sus métodos
- Los mecanismos criptográficos generalmente en uso son confiables
- La mayor parte de los errores vienen de *fallos en la implementación*
  - Fallos *alrededor* de la criptografía, no *dentro* de ella

Conclusión

- Los errores en el manejo de la criptografía llevan a *menor confiabilidad* que si no se hubiera utilizado
  - Falso sentido de la seguridad
- Misión *ante la sociedad* de los profesionales especializados en la seguridad (en particular: Alumnos de los posgrados de seguridad de la ESIME)
  - Quitar el velo de misterio o *magia negra* de la criptografía
  - Conocer los errores de implementación
  - Saber buscar y *prevenir* estos fallos
  - *Prevenir* el impacto social y económico de los fallos de implementación

A fin de cuentas. . .

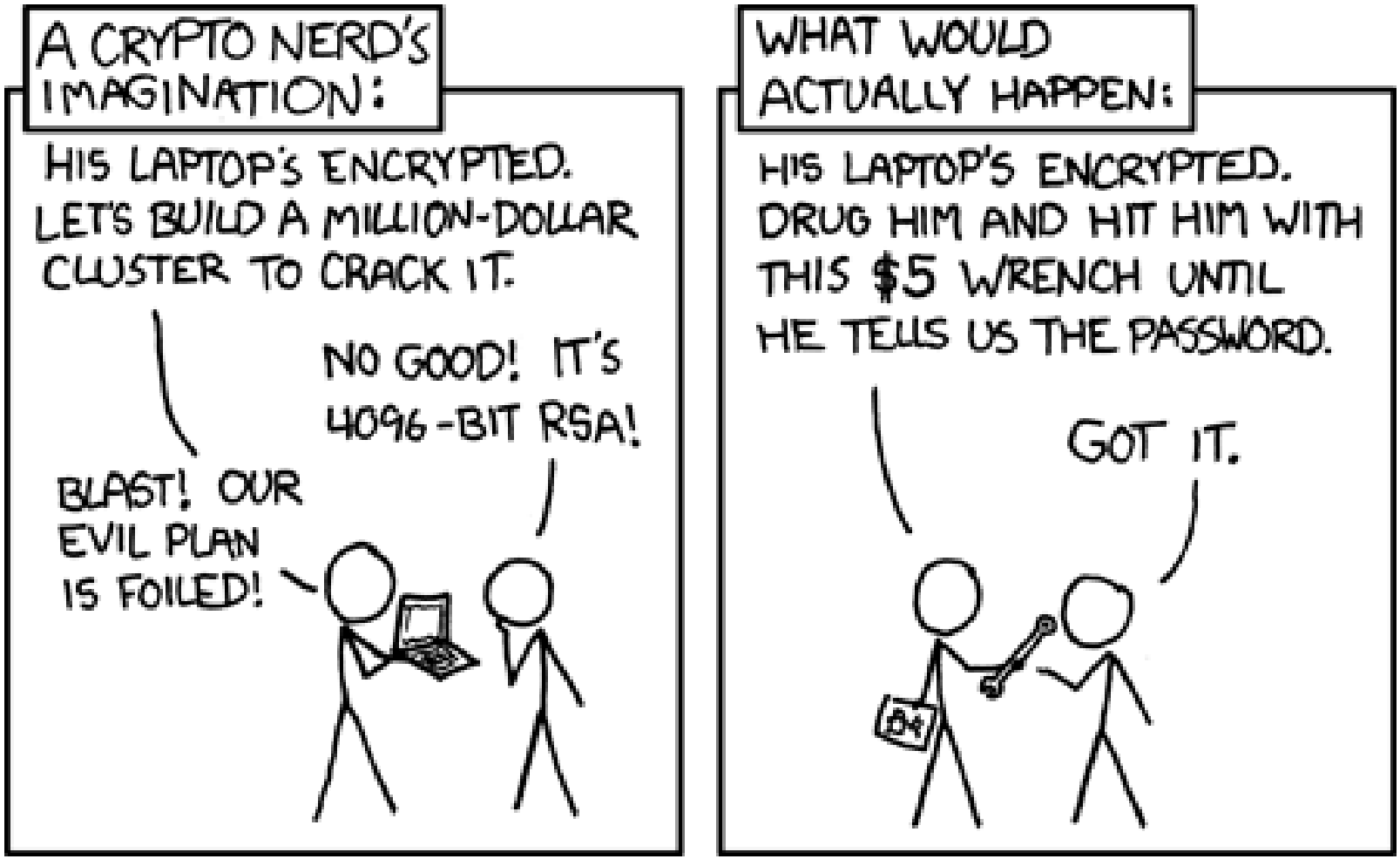

Figura: http://xkcd.com/538/

Fin

¿Preguntas?

Gunnar Wolf
gwolf@gwolf.org