Desarrollo seguro de aplicaciones con criptografía

Gunnar Wolf

Congreso de Seguridad de la Información del Instituto Politécnico Nacional

Resumen

Uno de los ejes principales en torno al cual están estructurados los planes de estudios relacionados con la seguridad en cómputo es el estudio de la criptografía. Este campo, si bien es fundamental para la formación de expertos en seguridad informática, debe presentarse en conjunto con la parte *aplicada*: ¿Cómo lo deben integrar los desarrolladores en las aplicaciones?

En el presente texto presento varios ejemplos de cómo los esquemas criptográficos más seguros han resultado *burlados* por fallas en la implementación, abordando brevemente cómo, por medio de una *desmitificación* de la criptografía, podría contribuirse a evitar estas situaciones.

1. Introducción

El uso de mecanismos criptográficos por fin está arraigándose en la industria del desarrollo de software.

Ante el eco que han dado los medios en general a situaciones de índole técnico-político, tales como las filtraciones de documentos confidenciales de los Estados Unidos (Wikileaks, Edward Snowden) o las vulnerabilidades detectadas en mecanismos criptográficos de alto perfil (SSH de Debian/Ubuntu en 2008, `goto fail` en 2014, Heartbleed de OpenSSL en 2014; estas tres vulnerabilidades son descritas en la sección 2), no sólo los desarrolladores sino que cada vez más los usuarios de software de todo tipo han ido cobrando conciencia de la importancia de proteger a sus datos de los ojos de terceros por medio de software *confiable*, que implemente mecanismos criptográficos fuertes.

Partiendo del caso específico de la ESIME Culhuacán, donde se dictan la *Especialidad en Seguridad Informática y Tecnologías de la Información* y la *Maestría en Ingeniería en Seguridad y Tecnologías de la Información*, el enfoque primario de los planes de estudio en temas de criptografía resulta claro.

Y si bien la comprensión de los mecanismos criptográficos desde un punto de vista formal, desde su descripción algorítmica, resulta fundamental para una formación integral en seguridad, no puede dejarse de lado a una de las quejas a los programas orientados a la investigación: El que van llevando a una desconexión entre la academia y la industria, en vez de potenciar una sinergia entre ellas. Esto se pone manifiesto incluso al revisar el listado preliminar de ponencias del congreso en el cual se presentará este texto: Si bien no puede hablarse de una correlación perfecta, con sólo ver el título de una ponencia es fácil aventurar si su ponente viene del entorno de la industria o de la academia.

Como alumnos de posgrado, tenemos que preguntarnos cómo nuestros estudios se enmarcan en la realidad del campo, y cómo podemos contribuir para mejorar la sociedad a través de nuestro actuar especializado. Tenemos la misión de especializarnos y profundizar, sí, pero tenemos también que hacerlo con la meta clara de contribuir a la sociedad.

A lo largo de esta presentación, se presentarán ejemplos de cómo, si bien los desarrolladores de aplicación conocen mecanismos de cifrado diseñados para cubrir las necesidades con que se enfrentan, fallos aparentemente triviales en su implementación ha limitado severamente o directamente revertido los efectos que se esperaría recibir de su utilización.

Para esto, cabe citar a la tercera de las *Tres Le-*

yes de la Seguridad, enunciadas por Adi Shamir en el discurso de recepción del Premio Turing 2002 [11]:

> 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.

2. Vulnerabilidades de alto perfil

Las vulnerabilidades descritas en esta sección han saltado a la notoriedad y se han insertado en el inconsciente colectivo de la comunidad *hacker*. Puede ser por lo obvio del fallo que llevó a su aparición o por la profundidad de su impacto — Estas se presentan en primer término, asumiendo que los lectores tienen ya familiaridad con sus detalles técnicos, pero buscando clasificarlas de forma que se puedan alinear con un análisis de *qué es* lo que exponen y cómo van demostrando el cumplimiento de la tercera Ley de la Seguridad de Shamir.

2.1. Debilitamiento criptográfico

Si bien se sabe que cifrar un mensaje empleando la *libreta de un solo uso* (*one-time pad*) brinda una protección criptográfica perfecta, dado que con diferentes llaves cualquier cadena puede corresponder con cualquier texto de la misma longitud, y con una muy baja dificultad computacional, la dificultad para transmitir de forma confiable la llave lo convierten en un mecanismo impracticable. Todos los esquemas criptográficos actualmente en uso basan su operación en el uso de *llaves*, cadenas de una longitud determinada y acorde al algoritmo empleado.

El univeso de llaves existentes es necesariamente muy grande; obtener (o *romper*) una llave equivale a poder descifrar todos los mensajes cifrados por o para ella. Es por este motivo que las llaves se "esconden" en espacios de búsqueda enormes — Al día de hoy se sugiere emplear entre 2^{128} y 2^{256} para algoritmos de cifrado simétrico, y entre 2^{1024} y 2^{4096} para los de llave pública. Para poner a estos números en perspectiva, en el universo se estima que hay *sólo* unos 2^{80} átomos.

Para generar una llave suficientemente fuerte es necesario que sea poco predecible — Que contenga un *alto nivel de entropía*. Esto es un problema importante para las computadoras, diseñadas para generar resultados predecibles y replicables. Generar números verdaderamente aleatorios es imposible, y hasta muy recientemente, las computadoras recurrían a leer señales en los puertos de entrada (como el movimiento del *mouse* o el ritmo de paquetes recibidos en una interfaz de red) y emplearlo como semilla para un *generador pseudoaleatorio*.

2.1.1. OpenSSL y Debian

La bilbioteca criptográfica *OpenSSL* es la implementación más completa y ampliamente utilizada de funciones criptográficas. Desafortunadamente, su código sufre de fuertes deficiencias de estilo: Es muy dificil de leer, y emplea varias construcciones no estándar o contrarias a las recomendaciones. En buena medida, estas deficiencias se deben a necesidades específicas de alguna de las plataformas que emplean a esta biblioteca.

En el año de 2006, el *mantenedor* de OpenSSL para la distribución Debian GNU/Linux notó que el compilador le enviaba advertencias acerca del uso de memoria no utilizada. Por un fallo de comunicación, la pregunta que hizo al respecto en la lista de desarrollo de OpenSSL fue vista como relativa a la depuración (y no al uso en producción). El mantenedor comentó una línea de código:

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

OpenSSL empleaba, sin embargo, la memoria no inicializada ocmo parte del proceso de obtención de entropía (contrario a lo que marca el estándar del lenguaje C). Este cambio anuló prácitcamente toda la recolección de entropía, y redujo el espacio de llaves generadas de 2^{128} a sólo 2^{32}. Pasaron más de dos años para que en 2008 otro desarrollador de Debian, Luciano Bello, se diera cuenta de este problema y el equipo de seguridad del proyecto lo corregira [15]; el

número de llaves que fueron generadas dentro de este espacio de 2^{32} se estima en varios millones, en prácticamente todas las computadoras basadas en Debian o alguno de sus derivados durante más de dos años. No se sabe que atacantes conocieran de esta vulnerabilidad previa a su publicación (aunque hay reportes de personas que detectaron *colisiones*, dos llaves generadas independientemente correspondientes a la misma semilla), pero su impacto puede ser la divulgación de cualquier material criptográfico hasta la necesaria recisión de *todas* las llaves implicadas.

2.1.2. Apple: `goto fail`

Hay errores mucho más sencillos que el anteriormente presentado, y que sí muestran sin lugar a dudas el efecto de un simple descuido. En febrero de 2014, se descubrió un error al hacer la validación de diversos aspectos de un certificado SSL en los productos Apple [14, 6]:

```
/* (...) */
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;
```

A cualquier programador hoy en día, lo primero que le llama la atención en este código es el uso de la instrucción `goto`, mal vista de forma universal a partir del artículo *Go To statement considered harmful* [5]. Esta instrucción es utilizada frecuentemente en código de *kernel* y en código que sea reiteradamente empleado (particularmente empleando recursión) pues no hace una llamada formal a una subrutina, y por tanto, no tiene que asignar memoria del *stack* ni copiar operandos — Sólo transfiere la ejecución.

El error en este código deriva de la segunda verificación presentada. Por la forma en que está indentado y por estar dentro de un conjunto grande de validaciones, el segundo `goto fail` fue ignorado, con lo cual el flujo nunca llegaba a las últimas validaciones. Esta validación incompleta permite que un atacante presente un certificado fingiendo ser la contraparte de la comunicación — Un ataque de *hombre en el medio*. Este ataque afectó a todos los productos Apple — Millones de equipos MacOS e iOS.

2.2. Divulgación de información sensible

Dado que la posesión de la llave correcta implica para efectos prácticos el conocimiento de cualquier mensaje cifrado con ella, resulta fundamental cuidar de su correcto manejo. Es por eso que una vulnerabilidad que lleva a la divulgación de llaves privadas resulta de tan alto perfil.

2.2.1. Heartbleed

En abril de 2014, el equipo de investigadores de seguridad *Codenomicon* publicó la vulnerabilidad conocida como *Heartbleed* [3]. Esta vulnerabilidad afecta también a la biblioteca OpenSSL, pero en este caso afecta a su línea principal de desarrollo (y no únicamente a una de sus distribuciones). Este problema también se presentó en las principales versiones de OpenSSL por espacio de dos años, desde marzo del 2012 y hasta el momento de su divulgación. Siendo OpenSSL la biblioteca criptográfica más ampliamente difundida del mundo, su impacto abarca no sólo a prácticamente todos los sistemas operativos libres, sino que a una gran cantidad de dispositivos embebidos que fueron producidos durante el periodo en cuestión.

A diferencia de los dos fallos descritos anteriormente, Heartbleed no es causado únicamente por un error fácilmente corregible, sino por una técnica de programación derivada de la naturaleza multiplataforma y de altos requisitos de tiempo de cómputo bajo los cuales fue diseñado OpenSSL (hace casi 20 años): Para evitar las demoras en que se incurre por reiteradas llamadas al sistema operativo para gestionar la memoria, OpenSSL implementa un gestor de memoria *internamente*. Las localidades de memoria que ya no están siendo utilizadas no se devuelven al sistema operativo, sino que se mantienen asignadas y en espera.

Bastó con que se aceptara código carente de verificación de límites en la funcionalidad *Heartbeat* para posibilitar a un atacante solicitar una mayor cantidad de memoria de la que debería recibir. Si OpenSSL utilizara las llamadas de asignación y liberación de memoria del sistema operativo (de la familia `malloc` y `free`), este fallo se hubiera limitado a una posible negación de servicio: Un atacante podría causar una violación de segmento (*segmentation fault*) e interrumpir al programa. Pero dada la naturaleza de OpenSSL, la respuesta incluye el contenido de la memoria que siguiera a la solicitud. Esta puede incluir todo tipo de material sensible — La información en claro que algún otro usuario envió a través de OpenSSL, contraseñas, e incluso la llave privada de los certificados SSL. Y si bien la llave misma al seguir, como todo material criptográfico, características de alta entropía y ser aparentemente aleatorio, sería muy dificil de reconocer en memoria, la manera en que OpenSSL lo almacena es como un certificado X.509 — Una codificación muy fácil de reconocer.

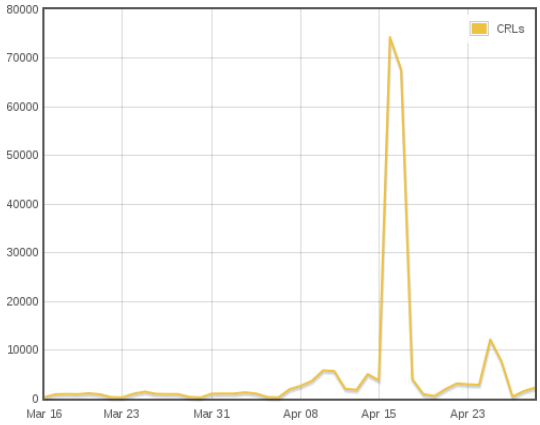

Figura 1: Revocaciones a certificados X.509 entre el 15 de marzo y el 30 de abril del 2014, según datos del *Internet Storm Center* de SANS [12]

El impacto económico directo de Heartbleed tiene una cota mínima fácil de estimar: Como lo muestra la figura 1, en la semana siguiente a la divulgación de Heartbleed, de un promedio de unos mil certificados al día, hubo un pico a más de 80,000 revocaciones diarias [12]. Cada uno de estos certificados tuvo que ser reemplazado con uno nuevo para que los proveedores de servicios Web siguieran operando regularmente, con un costo mínimo de cien dólares cada uno. Esto, claro, adicionalmente a los costos de reemplazo de contraseñas de millones de usuarios en millones de sitios Web en todo el mundo — Y, lo que es peor, la certeza de que no todos los afectados reaccionaron a la amenaza con la seriedad y prontitud necesarias.

3. La larga cola

Las tres vulnerabilidades recién descritas obtuvieron una gran cobertura de prensa e indudablemente cambiaron –para bien– muchas prácticas de desarrollo de software. Y las tres tienen un punto en común: Ocurrieron dentro de las grandes bibliotecas de implementación criptográfica ampliamente empleadas. Falta abordar lo que nosotros como *programadores de a pie* vamos a enfrentar.

La lista y severidad de aparentes *pequeños fallos* derivados de un uso incorrecto de material criptográfico es inacabable. En esta sección presentaré sólo algunos ejemplos, con la esperanza de que sirvan de aprendizaje.

3.1. Uso correcto de los *modos de operación*

Los esquemas simétricos de cifrado orientados a bloques operan sobre pedazos de información de tamaño fijo, típicamente relativamente pequeños de cara a un mensaje completo (entre 64 y 256 bits, o lo que es lo mismo, entre 8 y 32 caracteres). Para cifrar un mensaje completo pueden emplearse de diferentes maneras, repitiendo el cifrado con la misma llave y (en todos salvo uno de los modos estándar) algún factor que vaya alterando al resultado de forma predecible [7].

Cada uno de los modos de operación funciona bajo ciertos supuestos, y la realidad del sistema a implementar determina qué modo de operación empleará. Desafortunadamente, muchas veces la elección del modo de operación resulta inadecuada.

3.1.1. Wired Equivalent Privacy (WEP)

Tan pronto comenzó a popularizarse el uso de redes inalámbricas para el cómputo en general se hizo obvia la importancia de cifrar los mensajes enviados por la red. Este cifrado tenía que ser implementado a nivel componentes electrónicos (sin intervención expresa del procesador o del sistema operativo), lo cual limitó la complejidad del algoritmo criptográfico a utilizar.

La familia de estándares criptográficos 802.11 introdujo a WEP en 1999, presentándolo como un cifrado que brindaría una privacidad similar a la que se tiene en unared cableada: No aísla a un usuario a otro dentro de la misma red, pero sí evita que terceros no autorizados descifrar el tráfico capturado o emplear los servicios de la red.

Apenas dos años después de aprobado el estándar, para 2001 se publicaron ataques que hacen trivial obtener la llave de una red protegida por WEP con sólo hacer una captura pasiva [2, 10]. El algoritmo de cifrado que emplea WEP, RC4, puede operar con llaves de 64 o 128 bits; como lo ilustra la figura 2 WEP *recorta* este espacio a 40 o 104 bits, reservando 24 bits para ser utilizados como *vector de inicialización* (IV), evitando que la misma llave sea empleada literalmente de forma repetida.

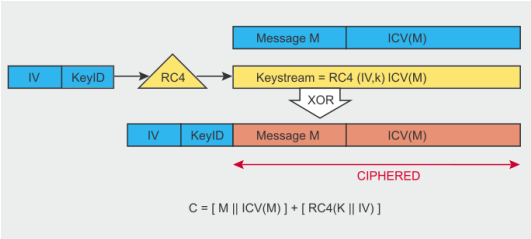

Figura 2: Esquema de operación de WEP [10]

Sin embargo, un espacio de 24 bits es demasiado corto para emplear como IV; la misma *secuencia cifrante* $k + IV$ será empleada repetidamente. Conociendo además la estructura de las tramas Ethernet, resulta trivial prever el contenido de los primeros 20 bytes de los paquetes transmitidos.

Estos ataques se fueron perfeccionando, y al día de hoy basta capturar tráfico de una red activa por unos 15 minutos (induciendo el tráfico inyectando paquetes mal formados) para obtener suficientes datos para romper su cifrado. Y si bien desde hace el 2004 existe un mecanismo de cifrado mejor implementado en la misma familia de estándares (WPA, *WiFi Protected Access*), experiencia empírica apunta a que cerca del 10 % de las redes siguen empleando WEP.

3.2. Distribución de llaves

El problema de cómo distribuir las llaves criptográficas ha sido largamente estudiado. La criptografía simétrica requiere de un canal seguro para el intercambio de la llave, y la gran contribución de la criptografía de llave pública fue la posibilidad de distribuir libremente una llave pública, y de establecer un secreto compartido sobre un canal público sin tener más información que la públicamente disponible [4].

3.2.1. Cuando mis dispositivos no son realmente míos

A pesar de lo anterior, si la llave privada forma parte de un dispositivo que debe ser físicamente distribuído a sus usuarios finales (y su principal función es evitar que los usuarios empleen a dicho dispositivo de formas ajenas a las previstas), ¿cómo evitar que el usuario tenga acceso a esta información?

Este problema es común a una gran cantidad de dispositivos orientados a la distribución de contenido, así como en plataformas móviles donde los desarrolladores buscan lider controlar centralmente las aplicaciones que *tienen permiso* de emplearse en lo que es, a fin de cuentas, una computadora de propósito general. Estos dispositivos basan su seguridad ya sea en identificar criptográficamente al usuario (en el primer caso) o validar los binarios a ejecutar (en el segundo) ante el proveedor de servicios.

Cuando el usuario tiene acceso al dispositivo, tiene acceso a lo que éste contiene — Incluso por medios *oficiales*, sin tener que manipular el hardware y limitándose sólo a las interfaces disponibles públicamente. Algunos de entre muchos ejemplos de esto son [8]:

- El sistema operativo del *Kindle*, de Amazon, requiere que todos los binarios estén firmados por una llave RSA de 1024 bits. Esta llave puede ser reemplazada por una *fabricada* por el atacante, logrando un *jailbreak*.

- Las empresas relacionadas con la telefonía celular –tanto los desarrolladores de los principales sistemas operativos como los proveedores de red– emplean esquemas muy similares al descrito anteriormente. Los primeros, buscando controlar qué aplicaciones permiten ser ejecutadas en un dispositivo, formalmente orientadas a *proteger al usuario* de correr código inseguro pero también dirigidas a limitar el uso de ciertas funciones no aprobadas (como el *tethering*, compartir la red TCP/IP con otros dispositivos, funcionalidad básica pero activada sólo en ciertos modelos y bajo determinados planes de datos), los segundos, para dificultar la migración de sus clientes a otras empresas proveedoras de servicio.

  En particular, el sistema operativo iOS de Apple, empleado en toda su gama de dispositivos móviles, busca controlar todos los programas que puede ejecutar un usuario requiriendo –desde el mismo gestor de arranque– binarios firmados.

  El artículo *«iOS*$_{\text{jailbreaking}}$*»* de la Wikipedia en inglés muestra una tabla de todas las versiones de dispositivo y versión de sistema que ha publicado Apple, así como el tiempo entre su salida a mercado y el primer *jailbreak* — El promedio se sitúa en 56 días, con una tendencia a la alza, demostrando que Apple busca dificultar esta tarea a los desarrolladores; en todos estos casos, el *jailbreak* fue resultado de explotar al software para impedir que realice la comprobación criptográfica, y nunca resultado de obtener la llave firmante o subvertir la criptografía.

- Las cámaras Nikon y Canon de gama alta incluían un sistema de autenticación de imágenes, orientado a asegurar tanto la propiedad de las imágenes como su integridad, incluyendo una firma en los campos EXIF de las fotografías.

  En el caso de Canon, este sistema empleaba una llave RSA de 1024 bits única por usuario, cuyo componente privado residía en el *firmware* de la cámara; en el caso de Canon, la firma era realizada por medio de una llave HMAC (simétrica) de 256 bits junto con el número de serie de la camara.

  En ambos casos, el grupo de investigadores en seguridad informática *Elcomsoft* logró recuperar las llaves y firmar imágenes modificadas [9]. El mecanismo empleado para obtener las llaves hoy está disponible públicamente, por lo que estos esquemas ya no pueden considerarse seguros.

3.2.2. Patrones emergentes

Por último, y sin presumir de que esta revisión sea exhaustiva, es importante recalcar también en que el uso de criptografía no evita la recuperación de la información codificada en los datos que un atacante puede interceptar por medio de los patrones emergentes en los datos. Presento dos casos:

- El sistema y red *TOR* (*The Onion Router* busca crear una red superpuesta a Internet que permita a sus usuarios emplearla manteniendo el anonimato. Los objetivos expresos de sus desarrolladores son ayudar a quienes viven bajo diversos regimenes opresores o amenazas criminales.

  La red TOR basa su seguridad en que todo paquete que la utiliza es reenviado a través de una serie de servidores a través del mundo. Una de las razones de ser una red distribuída por todo el mundo es dificultar el control de una *proporción significativa* de sus nodos por parte de una sola agencia de seguridad.

  Si bien hay reportes de que esto ya ha ocurrido [1], TOR sigue siendo una herramienta fundamental para una gran cantidad de activistas políticos — Así como, probablemente, también de criminales.

  Sin embargo, incluso sin romper la confianza que provee TOR, si lo que busca una entidad es demostrar el intercambio de información entre determinado usuario y servidor, y tiene la capacidad de ordenar (u obtener) un monitoreo de tráfico de ambos lados, si bien la identidad del

usuario que emplea TOR no será revelada, el patrón de tráfico entrante y saliente será consistente: Un paquete de datos de determinado tamaño saliendo del equipo del usuario corresponderá con uno entrando al servidor muy cerca en el tiempo, y uno enviado por el servidor corresponderá con otro enviado al cliente. Dado que las sesiones de TOR tienen una duración predeterminada, el usuario empleará a los mismos nodos de entrada y salida para llegar al mismo servidor por un tiempo también observable. Las direcciones de entrada y salida de la red TOR están documentadas, y pueden ayudar para filtrar la información facilitando más su análisis.

- El uso de aplicaciones de voz sobre IP (VoIP) registra un crecimiento dramático [13], incluso descontando que buena parte de las conversaciones telefónicas (tanto en líneas fijas como celulares) hoy en día se transmite en forma de paquetes de datos. Las razones para cifrar las conversaciones a estas alturas resultan obvias.

  Ahora bien, los *codecs* orientados a la comunicación VoIP son *adaptativos*: buscan optimizar el uso de ancho de banda y mejorar la calidad de la señal resultante detectando silencios, cancelando el retorno, comprimiendo los periodos de menor cambio, etc.

  Un equipo multidisciplinario de científicos computacionales y lingüistas demostró en 2011 [**Fon-ics**] cómo, analizando la *cantidad* de paquetes de datos enviados y con un conocimiento preciso de los patrones idiomáticos, es posible descifrar conversaciones enteras sin romper el cifrado, sólo como una función sobre la probabilidad de la cantidad de datos transmitidos por unidad de tiempo.

4. Conclusión

Los desarrolladores de software y administradores de redes están cada vez más concientes de la importancia de incorporar la criptografía en los servicios que ofrecen a sus usuarios. Esto es claramente positivo, particularmente a la luz de las revelaciones que han ocurrido relativas a las *escuchas* y espionaje generalizado. Sin embargo, la criptografía es un campo en el que es muy fácil cometer pequeños errores... Con graves consecuencias.

A lo largo de este trabajo se presentaron varios ejemplos de fallos relacionados con la criptografía, enfatizando en que estos fallos siempre se han centrado sobre la *implementación*, y no sobre los algoritmos criptográficos. Un algoritmo criptográfico que siguió los pasos de revisión formales y públicos entre pares puede con bastante certeza tomarse como el elemento más fuerte de un diseño de sistema. Sin embargo, un implementador enfocado a las prácticas más seguras debe poner especial cuidado en lo que pasa *alrededor* del uso de dicho algoritmo, y en que el uso del mismo obedezca a las principales buenas prácticas, dado que en caso contrario, más que contribuir a la seguridad de la información, se estará brindando un *falso sentido de la seguridad* que demeritará a la implementación completa.

Los participantes de este congreso, muchos de nosotros alumnos de los posgrados en Seguridad Informática de la ESIME Culhucán, otros muchos expertos interesados en el tema, tenemos la misión ante la sociedad (y en particular ante nuestros colegas desarrolladores) de quitar el velo de misterio, de *magia negra*, que rodea a la criptografía. Siendo como lo es hoy una herramienta fundamental para el funcionamiento de la red, con el impacto social y económico que tiene, recae en nosotros el conocer los principales requisitos para una operación correcta, divulgar las ventajas y riesgos de las distintas opciones, banderas y modos.

Referencias

[1] *Breaking: Half of TOR sites compromised, including TORmail*. 2013. URL: http://www.thehiddenwiki.net/breaking-half-of-tor-sites-compromised-including-tormail/ (vid. pág. 6).

[2] Nancy Cam-winget y col. "Security Flaws in 802.11 Data Link Protocols". En: *Communications of the ACM* 46.5 (2003), págs. 35-39. URL:

http://www.cs.berkeley.edu/~daw/papers/wireless-cacm.pdf (vid. pág. 5).

[3] Codenomicon Defensics. *Heartbleed Bug*. 2014. URL: http://heartbleed.com/ (vid. pág. 3).

[4] Whitfield Diffie y Martin Hellman. "New Directions in Cryptography". En: *IEEE Transactions on Information Theory* 22.6 (1976), 644–654 (vid. pág. 5).

[5] Edsger W. Dijkstra. "Go To statement considered harmful". En: *Communications of the ACM* 11.3 (1968). letter to the Editor, págs. 147-148 (vid. pág. 3).

[6] Paul Ducklin. *Anatomy of a "goto fail Apple's SSL bug explained, plus an unofficial patch for OS X*. 2014. URL: http://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/ (vid. pág. 3).

[7] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation*. NIST Special Publication SP 800-38A. 2001. URL: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf (vid. pág. 4).

[8] Peter Gutmann. *Crypto won't save you either*. 2014. URL: http://regmedia.co.uk/2014/05/16/0955_peter_gutmann.pdf (vid. pág. 5).

[9] Vladimir Katalov. *Nikon Image Authentication System: Compromised*. 2011. URL: http://blog.crackpassword.com/2011/04/nikon-image-authentication-system-compromised/ (vid. pág. 6).

[10] Guillaume Lehembre. "Wi-Fi security — WEP, WPA and WPA2". En: *hackin9* 06/2005 (2005). URL: http://www.peotta.com/arquivos/wifi/wifi_wpa-wep.pdf (vid. pág. 5).

[11] Adi Shamir. *Turing Lecture on Cryptology: a Status Report*. World-Wide Web slide presentation, video, and audio. 2003. URL: http://www.acm.org/turingawardlecture/RSA/ (vid. pág. 2).

[12] *SSL CRL Activity: Certificates Revoked per Day*. 2014. URL: https://isc.sans.edu/crls.html?startdate=2014-03-15&enddate=2014-04-30 (vid. pág. 4).

[13] Michael Ventimiglia. *What do the VoIP industry's projected growth rates mean for users?* 2013. URL: http://getvoip.com/blog/2013/03/25/what-do-the-voip-industrys-projected-growth-rates-mean-for-users (vid. pág. 7).

[14] Imperial Violet. *Apple's SSL/TLS bug*. 2014. URL: https://www.imperialviolet.org/2014/02/22/applebug.html (vid. pág. 3).

[15] Florian Weimer. *DSA-1571-1 OpenSSL — Predictable random number generator*. Inf. téc. Proyecto Debian, 2008. URL: https://www.debian.org/security/2008/dsa-1571 (vid. pág. 2).