Ejemplo de Ataque por Sustitución de Algoritmo

Antonio Castro Lechtaler, Marcelo Cipriano, Edith García, Julio Liporace, Ariel Maiorano, Eduardo Malvacio, María Eugenia Pazo Robles

Facultad de Ingeniería del Ejército (FIE), Universidad de la Defensa Nacional (UNDEF)
{acastro,marcelocipriano,egarcia,jliporace,maiorano, emalvacio,mpasorobles}@fie.undef.edu.ar

Resumen. El objetivo del presente artículo es presentar y describir la primera de una serie de implementaciones de diferentes esquemas, a manera de prueba experimental, para la validación del material estudiado y analizado en relación a la aplicación de los paradigmas y herramientas criptológicas modernas en la creación de software malicioso, como así también las técnicas de prevención, detección temprana y protección para ser considerados como un aspecto más de la Ciberdefensa. El código fuente de referencia ha sido publicado en Github.com.

Palabras clave: Software Malicioso, Criptología Maliciosa, Criptovirología, Kleptografía, Ataques por Sustitución de Algoritmos.

1 Introducción

Aunque comúnmente entendemos a la criptografía y a sus aplicaciones como herramientas de carácter defensivo, también pueden emplearse con usos ofensivos. Entre los usos maliciosos se encuentran los ataques basados en extorsión, software malicioso comúnmente denominado *Ransomware*, que causa la pérdida de acceso a la información. Por otra parte, la literatura también da cuenta de ataques en las etapas de diseño e implementación de en algoritmos criptográficos, comúnmente llamados *backdoors* o puertas traseras, que pueden vulnerar la privacidad, autenticidad y seguridad de sus usuarios.

Se resume a continuación, de manera muy breve y general, lo que podría considerarse el estado del arte y los alcances actuales de los conceptos de Criptovirología, Kleptografía, y, como un ejemplo particular de esta última, de los llamados ataques por sustitución de algoritmos –del inglés, *Algorithm-Substitution Attacks* (ASAs)-, haciendo foco principalmente en lo relativo al diseño e implementación de puertas traseras en algoritmos, protocolos y/o implementaciones de software criptográficos.

1.1 Contexto

Ya el año 1996, Adam Young y Moti Yung [1], introdujeron el concepto de "Criptovirología", exponiendo que aunque comúnmente entendemos a la criptografía y a sus aplicaciones como herramientas de carácter defensivo, que proporcionan privacidad,

autenticidad y seguridad a sus usuarios, también pueden emplearse con usos ofensivos. Entre otros, se ejemplifica el uso malicioso con el montaje de ataques basados en extorsión (para los cuales la criptografía de clave pública es esencial) que causan la pérdida de acceso a la información, pérdida de confidencialidad y fuga de información.

Al margen de su utilización para la implementación de un virus, en palabras de los autores en aquel entonces, del tipo actualmente categorizado como *Ramsonware*, o software malicioso de tipo extorsivo, se presenta luego el concepto de "Kleptografía", que abarca al diseño e implementación de *backdoors* o puertas traseras en algoritmos criptográficos, como fue expuesto sus trabajos posteriores [2,3,4].

En tales trabajos, se detallan diferentes metodologías de los llamados ataques kleptográficos, bajo el concepto de "*Secretly Embedded Trapdoor with Universal Protection*" (SETUP). Por citar un ejemplo particular, entre estos trabajos iniciales, se describe acabadamente un kleptograma para el algoritmo de intercambio de llaves Diffie-Hellman, pero también muestran cómo estos diseños podrían ser embebidos en otros sistemas, como los algoritmos de cifrado y de firma digital ElGamal, DSA, el algoritmo de firma Schnorr, y el PKCS de Menezes-Vanstone.

Luego aparecerían diseños de algoritmos SETUP, junto con otras técnicas o mecanismos kleptográficos, para el algoritmo RSA [5, 6, 8, 13].

Es importante destacar que estos diseños no se limitan a criptografía de llave pública. Se cuenta en la literatura con publicaciones que describen ejemplos aplicados a funciones de hashing donde se presentan colisiones para una versión de SHA-1 con constantes modificadas [7], como también alternativas para protegerse de esta primitiva criptográfica (potencialmente comprometida) en algoritmos de nivel superior, o que dependen de las primeras, como HMAC y HKDF [14].

Por supuesto tampoco los generadores de números pseudo-aleatorios serían inmunes a este tipo de ataques, como se explica en [10, 12]. Estas publicaciones describen mecanismos que afectan las propiedades estadísticas de un generador haciéndolo muy sensible a la entropía de entrada. Por ejemplo, cuando las entradas tienen una distribución correcta, este mecanismo no tiene efecto, pero cuando la distribución estuviera sesgada, el generador malicioso empeora considerablemente. Se destaca que además de su uso malicioso obvio, este mecanismo también se puede aplicar al testeo de generadores.

Por último podría resumirse, en línea con lo expresado en [14], que la seguridad de los esquemas criptográficos se mide tradicionalmente como la incapacidad de los adversarios con recursos limitados para violar un objetivo de seguridad deseado. El argumento de seguridad generalmente se basa en un diseño sólido de los componentes subyacentes. Podría decirse que uno de los fracasos más devastadores de este enfoque se puede observar al considerar adversarios con la capacidad de influir en el diseño, implementación y estandarización de primitivas criptográficas. El creer que esto no era posible se ha comprobado *naive*. Es entonces que considerando el impacto y la relevancia actual [16,17] de las técnicas y mecanismos mencionados se justificaría esta línea de investigación.

1.2 Ataques por Sustitución de Algoritmos

Más recientemente se han publicado una serie de trabajos enfocados en la aplicación de técnicas kleptográficas en algoritmos de criptografía simétrica, específicamente algoritmos de cifrado en bloques. Son ejemplos de tales trabajos [18, 19, 20].

Específicamente en [18], Bellare, Paterson y Rogaway abordan el concepto de ataques por sustitución de algoritmos, o *Algorithm-Substitution Attacks* (ASAs).

En ese trabajo, motivados por las revelaciones sobre la vigilancia masiva de las comunicaciones encriptadas, formalizan e investigan la resistencia de los esquemas de cifrado simétrico. El foco está puesto en ataques de sustitución de algoritmos (ASA), donde un algoritmo de cifrado subvertido reemplaza al real. Suponen que el objetivo del atacante –a quién llaman "*big brother"- es la subversión indetectable, lo que significa que los textos cifrados producidos por el algoritmo de cifrado subvertido deben revelar los textos en claro al atacante y, sin embargo, ser indistinguibles de los producidos por el esquema de cifrado real para los usuarios*.

Se formalizaron nociones de seguridad para capturar ese objetivo y luego ofrecen detalles acerca de ataques y defensas. Explicitan que los ASA se pueden montar en una gran clase de esquemas de cifrado simétrico.

Como recuerdan los autores, los ASA se han tratado antes bajo varios nombres, abarcados en el concepto de kleptografía. Mientras algunos criptógrafos parecen haber desestimado inicialmente a la kleptografía, revelaciones recientes sugieren que esta actitud resultó ser ingenua. Los ASA pueden estar sucediendo en la actualidad, posiblemente en una escala masiva.

1.3 Ejemplo de Ataque por Sustitución de Algoritmo

Como fuera adelantado en el resumen, este trabajo presenta además software experimental, como implementación de prueba de un esquema posible, con fines de investigación y testeo, para la implementación de una puerta trasera criptográfica.

Fue desarrollado por el GICSI (Grupo de Investigación en Criptografía y Seguridad Informática) de la FIE (Facultad de Ingeniería del Ejército), perteneciente a la UNDEF (Universidad de la Defensa Nacional).

Continuando en línea con otras publicaciones de realizadas por el GICSI, relacionados a la seguridad informática en general y a la criptografía en particular, se trata de software libre y de código abierto, y se encuentra publicado y disponible en la plataforma de alojamiento de proyectos de software libre Github.com [22], bajo la Licencia Pública General de GNU versión 3, o GPLv3 por sus siglas en inglés.

2 Sustitución de Algotimo AES en *engine* OpenSSL

De acuerdo con el primero de los dos tipos de ataque descriptos en [18], donde además se cita como ejemplo la aplicación al algoritmo AES con 128 bits de llave en modo de operación CBC (Cipher Block Chaining), los autores muestran que los esquemas de cifrado sin estado son típicamente subvertibles.

El tipo de ataque mencionado aplica a algoritmos de cifrados simétrico utilizando modos de operación que evidencian o exponen su vector de inicialización, o IV, por sus siglas del inglés, *Initialization Vector*.

Siguiendo a Young y Yung en los conceptos ya mencionados en apartados anteriores, y luego a Goh, Boneh, Pinkas y Golle [21], los autores consideran el problema de agregar un mecanismo oculto, subrepticio, de recuperación de la llave utilizada en esto algoritmos para diferentes protocolos.

Sugieren que cuando el servidor necesitara un valor aleatorio para ser utilizado como *nonce*, sería posible utilizar, en su lugar, un derivado de la llave en cuestión, que permtiera al atacante recuperar la llave original. De esta forma, teniendo en cuenta a los esquemas de cifrado simétrico en bloques que "exponen" el vector de inicialización en modo CBC, en lugar del IV explícito, se utilizaría el resultado de cifrar la llave de interés, mediante otra llave, que se denomina “de subversión”, en poder del atacante.

En este trabajo describiremos la implementación de una versión kleptográfica del algoritmo AES en modo CBC, dentro, o en la forma, de un motor o *engine* de OpenSSL. Consideraremos el ataque más simple, en el que el IV simplemente se reemplaza por el resultado de K $\oplus$ K’, siendo K la llave de cifrado del AES, y K’ la llave de subversión. El ataque se probará utilizando una conexión segura (TLS versión 1.2).

2.1 Concepto de motor o “*engine*” en ibrería OpenSSL

Desde su sitio Web oficial [23], OpenSSL se define como un conjunto de herramientas robusto, de grado comercial y con todas las funciones para los protocolos de seguridad de la capa de transporte (*Transport Layer Security*, o TLS) y la capa de sockets seguros (*Secure Sockets Layer*, o SSL). A la vez, también es una librería de criptografía de propósito general.

En lo que respecta los motores o *engines* [24, 25], desde la versión 0.9.6 de OpenSSL, se agregó un nuevo componente para admitir implementaciones alternativas de funcionalidades criptográficas, más comúnmente utilizado para interactuar con dispositivos criptográficos externos (por ejemplo, tarjetas aceleradoras). Este componente, también entendido como objeto, se denominó motor o *engine*. Estos objetos actúan como contenedores para implementaciones de algoritmos criptográficos y admiten un mecanismo para permitir que se carguen dinámicamente en la aplicación en ejecución.

2.2 Detalles de la implementación

En el ejemplo presentado en este trabajo, se implemntó, mediante un motor o *engine* OpenSSL, una versión “kleptográfica” del alogirtimo AES 128 en modo CBC. Los archivos fuentes, junto con la configuración y script decodificador pueden descargase del repositorio en Github del GICSI [22].

Como fuera adelantado más arriba, nos basamos en el primer ataque presentado por Bellare, Paterson y Rogaway en [18], quienes a su vez siguen a Young y Yung,

entre otros, considerando el problema de agregar una llave de recuperación oculta, planteando que cuando el servidor necesita un número aleatorio o nonce, podría utilizarse en su lugar el resultado del cifrado de la llave original, mediante una llave kleptográfica o de custodia. En [18] la llave de custodia es la llave de subversión, y muestran cómo subvertir los esquemas de cifrado que "exponen" el vector de inicialización (IV, por sus siglas en inglés -Initialization Vector-) en algoritmos de cifrado en bloque simétricos operados en modo CBC.

En el caso del *engine* OpenSSL presentado en este trabajo, si bien “reemplaza” la implementación original de AES 128 en modo CBC, OpenSSL en su implementación de los protocolos TLS versiones 1.1 y 1.2, siguiendo lo indicado en la sección 6.2.3.2 de la RFC 4346 [26], paricularmente lo descripto como alternativa (2)(b), generará el IV explícito cifrando un primer bloque a descartar por el destinatario. Por tal razón, esta implementación no realiza, estrictamente, un reemplazo de IV sino del primer bloque de texto cifrado generado.

Como fuera mencionado más arriba, este primer bloque será reemplazado por el resultado de $K \oplus K'$, siendo K la llave de cifrado del AES, y K’ la llave de subversión, que se encuentra fijada en el código fuente del *engine* y en el *script* para realizar la decodificación.

En [18], para simplificar la presentación, los autores asumieron que la longitud del IV y la longitud de la llave son las mismas, aunque a la vez presentaron alternativas para cuando éste no fuera el caso. Para nuestro ejemplo, con AES-128 en modo de operación CBC, esto no representa un problema.

> Por otro lado, como es advertido también por los autores, el evitar IVs repetidos, evidenciados a través de textos cifrados, requeriría limitar la sustitución de IV a un texto cifrado. Esto requiere el uso de un esquema de subversión con estado. En el presente trabajo no se intentará evitar esta repetición ya que se considerará que las llaves no se repetirán en el escenario planteado, dado que OpenSSL en su implementación de los protocolos TLS versiones 1.1 y 1.2, deriva una llave de 16 bytes que no se podrá distinguir de 16 bytes aleatorios, y por ende, tampoco el resultado de la suma xor con el valor -fijo- de la llave de subversión. Téngase en cuenta además que el reemplazo se realiza sólo inicialmente, es decir, sólo una vez.
>
> Por último, tambíen en relación a lo anterior, como se ha comentado, el primer requisito de un ataque es que no sea detectable (al utilizar la funcionalidad o funcionalidades criptográficas compometidas). En este caso, el IV explícito, si no se usara el *engine* kleptográfico, correspondería al resultado del cifrado AES de un bloque aleatorio (provisto por OpenSSL al incializar el cifrador), por lo tanto, tampoco podría distinguirse.

3 Resultados experimentales

3.1 Captura de conexión segura

La captura de paquetes de red de una conexión segura se realizó utilizando el utilitario s_client de openssl. Se trata básicamente de un comando permite la realización de pruebas y debug sobre conexiones TLS.

Al margen de lo anteor, réngase en cuenta que todo software del sistema que se encuentre enlazado dinámicamente con la librería OpenSSL utilizaría el engine kleptográfico.

El comando ejecutado fue el siguiente:

```
$ echo "GET /" | openssl s_client -ign_eof -tls1_2 -cipher \
AES128-SHA -connect www.google.com:443 -servername www.google.com
```

Listado 1. Ejemplo de ejecución de s_client para generación de conexión segura.

Habiendo configurado previamente el sistema para la utilización de nuestro engine para todo caso en que se requiera a OpenSSL utilizar AES 128 en modo CBC, de esta forma, se realiza una petición al sitio Web de google para obtener su página de incio.

La captura se registra en un archivo .pcap, que luego alimentará al script decodidicador, como se muestra en el apartado siguiente.

3.2 Decodificación de captura

Como fuera adelantado, el otro componente necesario para probar el ataque, es el script decodificador. Este script tomara por entrada la captura generada de acuerdo a lo descripto en el apartado anterior, y recuperará el IV explícito enviado.

Luego aplicará la operación XOR sobre éste y la clave de subversión fija para obtener así la llave que fue utilizada por el algoritmo AES.

Hecho esto, identificará los mensajes TLS de tipo "*Application Data*" para descifrarlos y presentarlos como salida.

```
$ python decodificar.py
IV explícito: b'\xdf\xbeQ\xfd\xcd\xc41\xae9L\xf0\xc7\rk\x15\t'
llave: b' A\xae\x022;\xceQ\xc6\xb3\x0f8\xf2\x94\xea\xf6'
data: b"\xbc)Q\xe7\\\xab\x17`…
decifrado: b'GET /…\x05\x05\x05\x05\x05'
```

Listado 2. Salida de script decodificar.py con ejemplo de datos decifrados.

Como puede apreciarse en este listado, se encuentra descifrada la petición HTTP "GET /" que fuera enviada en el ejemplo. También puede identidicarse el *padding* de cinco bytes, que corresponden al final del bloque en texto en claro.

4 Trabajo a futuro

Se planea la continuación del desarrollo de los diferentes esquemas kleptográficos para la experimentación y evaluación de factibilidad y eficacia.

En particular, se intentará tratar otras funcionalidades criptográficas como por ejemplo: generación de números aleatorios, funciones de *hashing*, y algoritmos de cifrado asimétrico.

5 Conclusiones

Este artículo presentó las generalidades de un ejemplo de implementación de kleptografía. Aunque específico, este primer ejemplo representa un caso posible que se confirma aplicabable para un algoritmo específico de la una de las *ciphersuites* aceptadas para TLS 1.2 (inclusive, es la única definida como obligatoria).

Aunque el escenario implica o requiere el compromiso inicial del sistema, el hecho de que un *backdoor* de este tipo luego no sea detectable *"over the wire"* y que persista aún frente actualizaciones del software del sistema operativo (incluyendo las de la librería OpenSSL) evidencia las capacidades de este tipo de *malware*.

Referencias

1. Young, Adam L. and Moti Yung. "Cryptovirology: extortion-based security threats and countermeasures." Proceedings 1996 IEEE Symposium on Security and Privacy (1996): 129-140.
2. Young, Adam L. and Moti Yung. "The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems." CRYPTO (1997).
3. Young, Adam L. and Moti Yung. "Kleptography: Using Cryptography Against Cryptography." EUROCRYPT (1997).
4. Young, Adam L. and Moti Yung. "Malicious cryptography - exposing cryptovirology." (2004).
5. Young, Adam L. and Moti Yung. "A Space Efficient Backdoor in RSA and Its Applications." Selected Areas in Cryptography (2005).
6. Young, Adam L. and Moti Yung. "An Elliptic Curve Backdoor Algorithm for RSASSA." Information Hiding (2006).
7. Albertini, Ange, Jean-Philippe Aumasson, Maria Eichlseder, Florian Mendel and Martin Schläffer. "Malicious Hashing: Eve's Variant of SHA-1." Selected Areas in Cryptography (2014).
8. Young, Adam L. and Moti Yung. "Cryptography as an Attack Technology: Proving the RSA/Factoring Kleptographic Attack." The New Codebreakers (2015).
9. Russell, Alexander, Qiang Tang, Moti Yung and Hong-Sheng Zhou. "Cliptography: Clipping the Power of Kleptographic Attacks." ASIACRYPT (2015).
10. Indarjani, Santi. Sugeng, Kiki. Widjaja, Belawati. "Modification Attack Effects on PRNGs: Empirical Studies and Theoretical Proofs." (2015).
11. Young, Adam L. and Moti Yung. "Cryptovirology: the birth, neglect, and explosion of ransomware." Commun. ACM 60 (2017): 24-26.

12. Teseleanu, George. "Random Number Generators Can Be Fooled to Behave Badly." IACR Cryptology ePrint Archive (2018).
13. Markelova, A. V. "Vulnerability of RSA Algorithm." (2018).
14. Fischlin, Marc. Janson, Christian. Mazaheri, Sogol. "Backdoored Hash Functions: Immunizing HMAC and HKDF." (2018): 105-118.
15. Xiao, Dianyan and Yang Yu. "Klepto for Ring-LWE Encryption." Comput. J. 61 (2018): 1228-1239.
16. Yogi, Manas. Aparna, S.. "Novel insights into Cryptovirology A Comprehensive Study." International Journal of Computer Sciences and Engineering. 6. (2018): 1252-1255.
17. Zimba, Aaron. Chishimba, Mumbi. "On the Economic Impact of Crypto-ransomware Attacks: The State of the Art on Enterprise Systems." European Journal for Security Research. (2019).
18. Bellare M., Paterson K.G., Rogaway P. "Security of Symmetric Encryption against Mass Surveillance." Advances in Cryptology. CRYPTO 2014. Lecture Notes in Computer Science, vol 8616. Springer, Berlin, Heidelberg (2014).
19. Mihir Bellare, Joseph Jaeger, and Daniel Kane. "Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks." In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). Association for Computing Machinery, New York, NY, USA, 1431–1440. (2015).
20. Dunkelman, Orr and Léo Perrin. "Adapting Rigidity to Symmetric Cryptography: Towards 'Unswerving' Designs." Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop. (2019).
21. E.-J. Goh, D. Boneh, B. Pinkas, and P. Golle. "The design and implementation of protocol-based hidden key recovery". In C. Boyd and W. Mao, editors, ISC 2003, volume 2851 of LNCS, pages 165-179. Springer. (2003).
22. Proyecto keplto-openssl-engine. GICSI. Repositorio en Github. En línea: https://github.com/gicsi/keplto-openssl-engine. Consultado en junio de 2020.
23. Home page. OpenSSL. Sitio oficial. En línea: https://www.openssl.org/. Consultado en mayo de 2020.
24. ENGINE cryptographic module support. OpenSSL. Sitio oficial. En línea: https://www.openssl.org/docs/man1.0.2/man3/engine.html. Consultado en mayo de 2020.
25. README.ENGINE. OpenSSL. Repositorio en Github. En línea: https://github.com/openssl/openssl/blob/master/README.ENGINE. Consultado en mayo de 2020.
26. RFC 4346. The Transport Layer Security (TLS) Protocol Version 1.1. IETF. En línea: https://tools.ietf.org/html/rfc4346. Consultado en mayo de 2020.
27. RFC 5246. The Transport Layer Security (TLS) Protocol Version 1.2. IETF. En línea: https://tools.ietf.org/html/rfc5246. Consultado en mayo de 2020.