Detección de anomalías en Oráculos tipo OpenSSL por medio del análisis de probabilidades.

Antonio Ricardo Castro Lechtaler[1], Marcelo Cipiano[1,2]

[1] Escuela Superior Técnica - IESE / Universidad Tecnológoca Nacional, C1426AAA, Buenos Aires, Argentina; [2]Instituto Fátima, Buenos Aires, Argentina
Antonio Castro Lechtaler, acastro@iese.edu.ar, Marcelo Cipriano marcelocipriano@iese.edu.ar

Abstract:

Los sistemas y programas pueden contener errores ocultos aún luego que se crea que se los han depurado y ofrecidos a los usuarios. Dichos errores pueden dejar al sistema vulnerable y expuesto a riesgos. Mucho más si pertenecen al Sistema de Gestión de la Seguridad de la Información como pueden ser las entidades que entregan certificados para la autenticación de usuarios, servicios para la generación de claves criptográficas, etc. A estos servicios los llamaremos Oráculos a lo largo de este artículo. Este trabajo estudia el funcionamiento general de estos Oráculos y propone una metodología de trabajo para la detección de errores que permita dejar al descubierto vulnerabilidades. Mediante el estudio del comportamiento de las 3-tuplas (módulo, clave pública, clave privada) se podrá comprobar un comportamiento anómalo o sesgado en los mismos. Evidencia de peso que permita descubrir Oráculos defectuosos.

1 Introducción

Desde que la Criptografía dejó de pertenecer a la esfera de los secretos militares y diplomáticos, para volcarse al ámbito civil -luego de la explosión informática de los años `80 y `90 no han dejado de crecer sus aplicaciones. Desde la consulta de mails, ingreso a redes sociales, home banking o compras online, son requeridos sus servicios para proteger la información de los usuarios y sistemas.

La conexión de los usuarios y sus equipos a diversas redes, desde las hogareñas hasta las públicas, requiere de servicios de autenticación y cifrado de datos.

Todo lo que implique intercambio de información entre dos equipos informáticos debería estar al resguardo de técnicas criptográficas. Pero esto al usuario le resulta transparente. Los sistemas se encargan de entablar todos los servicios requeridos por los estándares y protocolos de manera automática

Tickets de ingreso a sistemas, intercambios de claves, autenticación de usuarios y equipos, inicio de sesión, y muchos otros servicios son controlados en forma automática, a menos que el usuario tome el control y cambie las opciones por defecto.

¿Cómo comprobar si estos procesos y servicios contienen errores que pueden alterar la seguridad de lo que pretenden proteger?

Llamaremos Oráculo a un servicio que ofrezca a sus clientes claves públicas, privadas y módulos en un esquema de cifrado/descifrado RSA[1].

Estudiaremos los posibles escenarios de seguridad que estos Oráculos proponen y presentaremos una metodología probabilística para comprobar las fortalezas o debilidades de su diseño.

2 Código abierto no necesariamente aumenta la Seguridad de los Sistemas de Información.

En la industria del software conviven dos modalidades para el desarrollo de software: abiertos y cerrados. La diferencia, básicamente radica en la posibilidad que los usuarios puedan o no ver los códigos fuentes de los mismos.

La solución de software abierto (open source) Seguridad de los Sistemas de Información, es mucho más atractiva que la de software cerrado dado que se puede "ver" lo que el sistema hace y cómo lo hace. La ausencia de "puertas traseras" o claves no documentadas u otras técnicas como la inoculación de código virulento [01].

Aunque estén abiertos sus códigos no significa que se encuentren libres de errores.
No todos los usuarios pueden leer el código e interpretar su significado y de esa manera corroborar el correcto funcionamiento del servicio.
Luciano Bello ha descubierto un error en OpenSSL[2]. Que fue enmendado 20 meses después que la versión defectuosa. [02]

Según la Ley de Linus "dado un número muy grande de ojos, los errores se convierten en evidentes" [03]

Está claro que al tiempo que aumenta la sofisticación y complejidad de los códigos, la cantidad de ojos disponibles para evidenciar los errores decrece. Es por ello que

[1] La 3-tupla (*n*,*e*,*d*): n (módulo) es el producto de 2 primos, e (clave pública) y d (clave privada) son inversos entre sí mód ϕ(n).

[2] Una mala inicialización de una variable provocó una predictibilidad en el generador de números, abriendo una vulnerabilidad inimaginable.

estudios como el del presente trabajo y su aplicabilidad al contexto actual del software se hacen cada vez más necesarios.

3 Oráculo.

El presente trabajo muestra un procedimiento susceptible de codificar y controlar el funcionamiento de entidades que en forma general hemos llamado Oráculo[3]: una 3-tupla *(P, D, U)* donde:

P: conjunto de primos de Oráculo (q_i es primo $1 \leq i \leq p, p \in N$).

P = { q_1 ; q_2 ; q_3 ; . . . ; q_p }

D: Directiva. Criterio que administra los primos *q* para generar los módulos *n*. La directiva genera un contexto o escenario de seguridad que Oráculo ofrece. Las directivas determinan el cardinal del conjunto U.

U = { (n_1,e_1,d_1) ; (n_2,e_2,d_2) ; . . .; (n_u,e_u,d_u) }

U: conjunto universal de 3-tuplas que Oráculo calcula y administra. Los valores n se calculan a partir del producto de 2 números primos. Prescindimos de abordar la manera que tiene Oráculo de obtener dichos primos.

Sean entonces los siguientes Oráculos:

O_1= (P_1, D_1, U_1) ; O_2= (P_2, D_2, U_2) y O_3= (P_3, D_3, U_3)

3.1 Directiva y Escenario 1.

D_1: $\forall i \neq j \wedge 1 \leq i \leq u \wedge 1 \leq j \leq u$ mcd (n_i : n_j) =1
(no se repetirán los factores primos empleados en sistemas generados).

U_1 = { (n_1,e_1,d_1) ; (n_2,e_2,d_2) ; . . .; (n_u, e_u, d_u) } donde $u = p/2$.

3.2 Directiva y Escenario 2.

D_2: $\forall i \neq j \wedge 1 \leq i \leq u \wedge 1 \leq j \leq u$. Si $n_a = q_i q_j \in U \Rightarrow n_b = q_j q_i \notin U$
(no se aceptará los módulos con sus factores primos permutados)

U_2 = { (n_1,e_1,d_1) ; (n_2,e_2,d_2) ; . . .; (n_u, e_u, d_u) } donde $u = p\ (p-1)\ /2$.

[3] Por ejemplo un generador de claves RSA también puede ser defectuoso tal como se aprecia en [4].

3.3 Directiva y Escenario 3.

D_3: $1 \leq i \leq p\ n = q_i^2 \notin U$ (no se permitirán módulos cuadrados)

$U_3 = \{ (n_1,e_1,d_1) ; (n_2,e_2,d_2) ; \ldots ; (n_u, e_u, d_u) \}$ donde $u = p\ (p-1)$.

4 Vulnerabilidades de Oráculo.

Un Oráculo es vulnerable y por lo tanto el escenario de seguridad que le presenta al sistema al que sirve, cuando :

1) Oráculo genera un número relativamente pequeño de primos[4].

2) Oráculo está diseñado con una directiva errónea para la selección de los primos o la generación de los módulos.

En ambos casos Oráculo se distancia del comportamiento equiprobable, dentro de ciertos parámetros asumidos, a un comportamiento sesgado. Susceptible de ser estudiado por un atacante y vulnerar así el sistema.

3) Un atacante tiene forma de construir $P' \subset P$ tal que el conocimiento de P´ le permita vulnerar la factorización de RSA para una cantidad significativa de módulos.

5 Análisis del comportamiento de Oráculo.

Asumimos que no es posible leer el código fuente y/o no es fácil comprender el funcionamiento del mismo entonces los responsables del la seguridad deberán analizar el comportamiento de Oráculo para que cumpla con su función y no vulnere al sistema.

Simulando un entorno real de Oráculo se construye el conjunto N:

Sea $N = \{n_1, n_2, n_3, \ldots, n_m\}$ los módulos solicitados a Oráculo.

Calcularemos m=Card (N) tal que (el mayor número de módulos a extraer) para que al menos dos módulos compartan factores primos. En símbolos:

Sea m / $si\ i \neq j\ \exists n_i, \exists n_j, 1 \leq i \leq m \wedge 1 \leq j \leq m \Rightarrow mcd\ (n_i , n_j) \neq 1$

Se estudia el m para el caso que todos los elementos del conjunto sean coprimos de a dos y se esté por incorporar un nuevo módulo al conjunto.

[4] Vulnerabilidad de OpenSSL de Devian descubierta por L. Bello.

5.1 Oráculo es O_1.

Sea Oráculo=O_1 entonces es fácil ver que ni siquiera si m=Card (N)=Card (U)=u se podrá hallar un factor primo que compartan los módulos.

Entonces, si para valores mayores a u/2 peticiones de módulos no se producen "colisiones" de factores primos, entonces Oráculo está trabajando correctamente. Siempre y cuando u/2 esté dentro de los valores.

Sin embargo es muy poco probable que un Oráculo real sea O_1 pues la cantidad de recursos de almacenamiento para poder guardar los factores primos ya empleados (en caso que los genere a medida que se los piden) o la lista inicial de primos (predeterminada) para valores de primos de 512 o más bits sería enorme y no se aprecian dichos requerimientos en los sistemas de tipo OpenSSL y similares.

5.2 Oráculo es O_2.

Sea U distribuido de la siguiente manera. En cada celda se ubicará, cuando corresponda, el producto de un par de primos, dado el siguiente arreglo. Vemos que es un cuadrado de cuyos lados miden *(p-1):*

$\boldsymbol{n_1}=q_1q_2$				
q_1q_3	$q_2\ q_3$			
q_1q_4	$q_2\ q_4$	$\boldsymbol{n_2}=q_3\ q_4$		
…	…	…	…	
q_1q_p	$q_2\ q_p$	$q_3\ q_p$	…	$q_{p-1}q_p$

Tabla 1: Distribución geométrica de U tal que se pueden apreciar los valores primos y como se forma cada módulo

Sea n1=q_1q_2, luego A_1 ={ $x \in U / mcd\ (x:n_1) \neq 1$ }

El conjunto de todos los módulos del conjunto U tales que comparten alguno de sus factores primos con los factores primos de n_1.

Se puede comprobar que $Card(A_1)=2(p-2)+1$. De acuerdo a la tabla 1|, corresponde a la cantidad de módulos alojados en las dos primeras columnas.

Pedimos otro módulo. Sea N= $\{n_1, n_2\}$.

Si $mcd\ (n_1: n_2) \neq 1 \Rightarrow n_2 \in A_1$. Se encontró una colisión.

De lo contrario sea $n_2=q_2q_3$. Sea $A_2 =\{ x \in U / mcd(x:n_2) \neq 1 \}$ El conjunto de todos los módulos del conjunto U tales que comparten alguno de sus factores primos con los factores primos de n_2.

Se puede comprobar que $Card(A_2)=2(p-4)+1$. De acuerdo a la tabla 1, corresponde a la cantidad de módulos alojados en la tercera y cuarta columna.

Caso general:

$N = \{n_1, n_2, n_3, ..., n_{m-1}\}$

Sea n_m el nuevo módulo solicitado a Oráculo. Si comparte algún primo con los factores del conjunto N entonces:

$n_m \in A_1 \cup A_2 \cup A_3 \cup ... \cup A_{m-1}$.

Calculando:

$$Card(A_1 \cup A_2 \cup A_3 \cup ... \cup A_{m-1}) = \sum_{i=1}^{m-1} [2(k - 2i) + 1] \quad (01)$$

$$Card(A_1 \cup A_2 \cup A_3 \cup ... \cup A_{m-1}) = -2m^2 + (2k + 3)m - (2k + 1) \quad (02)$$

Si dividimos la expresión hallada por el cardinal de U. Entonces estaremos calculando con probabilidad h que n_m comparta algún factor primo con alguno de los $(m-1)$ módulos de N. Es decir:

$$\frac{-2m^2 + (2k + 3)m - (2k + 1)}{\frac{k(k - 1)}{2}} = h \quad (03)$$

$$-2m^2 + (2k + 3)m - (2k + 1) = \frac{h\,k(k - 1)}{2} \quad (04)$$

Igualando a cero y convirtiendo la expresión en un polinomio de manera de poder hallar los valores de m que satisfagan la expresión, se tiene:

$$-2x^2 + (2k + 3)x - \left[(2k + 1) + \frac{h\,k(k - 1)}{2}\right] = 0 \quad (05)$$

Desarrollando la fórmula de Bhaskara para hallar las raíces y tomando una aproximación en el discriminante se tiene que:

$$m \approx \frac{k + 1 \pm \sqrt{1 - h}}{2} \quad (06)$$

De los dos valores obtenidos por la fórmula el que representa al valor buscado es:

$$m < \frac{p}{2} \tag{07}$$

Pues la cantidad de primos es *2(m-1)* y este valor no puede ser mayor a *p= Card (P)*

5.3 Oráculo es O_3.

Análogo razonamiento se tiene que :

$$Card(A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_{m-1}) = 2\sum_{i=1}^{m-1}[2(k-2i)+1] \tag{08}$$

Y dado que $Card(U_3)=p(p-1)$

Pasando el factor 2 dividiendo al miembro de la derecha, se tiene:

$$-2m^2 + (2k+3)m - (2k+1) = \frac{h\,k(k-1)}{2} \tag{09}$$

Que es igual a la ecuación (04). Luego se tendrá el mismo resultado de la ecuación (06). Es por ello que en general los Oráculos trabaja con las directivas de O_3. Con el beneficio adicional que no deben consumir recursos de almacenamiento para tener "memoria" acerca de los módulos ya entregados a los usuarios.

6 Ejemplo de aplicación del método al OpenSSL defectuoso de Devian

Supongamos que la cantidad de primos que un OpenSSL hubiera generado son del orden de $p=Card(P)=2^{500}$

Calculamos que cantidad de módulos habrá que pedirle a Oráculo para tener una probabilidad de al menos 0,5 de tener un par de módulos que compartan alguno de sus factores primos.

Tomando el resultado anterior, se espera que $m \approx 2^{499}$ sobre un total de $Card(U) \approx 2^{999}$.

Dado que el bug provocó una disminución del rendimiento en la generación de primos, se tiene que $p=Card(P)=2^{15}$. Luego $Card(U) \approx 2^{30}$.
Pero para este oráculo sesgado la cantidad de módulos para una probabilidad igual a 0,5 es de $m=9597 \approx 2^{13}$.

7 Interpretación de las fórmulas calculadas.

Es común en Criptografía habituarse a trabajar con números enormes: las magnitudes son de tal tamaño que es fácil que oculten los errores de diseño o codificación.

Este procedimiento se asemeja a la "Paradoja del Cumpleaños" [05], pero queda escondida debajo del tamaño de los números.

Cabe aclarar que no es una verdadera "Paradoja": fue dada en llamar así porque los resultados que ella calcula difieren con la respuesta dada por nuestro sentido común. El enunciado de la paradoja pregunta ¿cuántas personas deben converger en una habitación para la que probabilidad de que al menos dos de ellas cumplan años el mismo día sea del 50%?. El resultado es 23.

Para hacer comparables los resultados aquí obtenidos con la mencionada paradoja, supongamos que un Oráculo generara 365 sistemas posibles, con p=27 primos aprox, entonces la cantidad de módulos que habría que extraer para que al menos 2 de ellos tuvieran una colisión en sus factores primos con probabilidad del 50% es de sólo 4 módulos, muy por debajo de los 23 de la propia paradoja y resulta extraño al sentido común este resultado, cuanto más lo es el de 4.

8 Conclusiones y futuros trabajos.

Ante el aumento en complejidad y sofisticación de los sistemas de información, se va haciendo más difícil depurar ciertos errores que pueden hallarse en dichos programas.

Es necesario delegar la tarea de auditar que los programas hagan "lo que prometen hacer" a algoritmos y así poder descubrirlos. Dada la magnitud de los números en cuestión usados en Criptografía, los procedimientos tienen que hacer acopio de métodos probabilísticos.

Este procedimiento permite auditar y descubrir el comportamiento incorrecto de una Entidad llamada Oráculo antes que sus debilidades permitan vulnerar los sistemas que pretendemos proteger.

Estamos desarrollando en el CriptoLab junto al Centro de Investigación y Desarrollo de Software (C.I.D.E.SO) una plataforma de computación distribuida para que podamos auditar distintos Oráculos y evaluar su funcionamiento. Queda entonces por delante la investigación que esta experiencia conllevará.

9 Referencias

[01] Young A and Yung M. An Elliptic Curve Asymmetric Backdoor in OpenSSL RSA Key Generation. Chapter 10. Cryptovirology. 2006. http://www.cryptovirology.com.

[02] Bello L, Bertacchini M. *"Generador de Números Pseudo-Aleatorios Predecible en Debian*". III Encuentro Internacional de Seguridad Informática. Manizales, Colombia. Octubre 2009.

[03] Glass, Robert "*Facts and Fallacies of Software Engineering"*. Addison-Wesley Professional, 2003.

[04] Young A and Yung M. YYGen: A Backdoor-Resistant RSA Key Generator. Chapter 10. Cryptovirology. 2006. http://www.cryptovirology.com.

[05] Rosen R. Matemática Discreta. Ed. Adisson Wesley. México. 2004.