Ein Fehler im Pseudozufallszahlengenerator in Debian/OpenSSL

im Rahmen des Seminars „IT-Sicherheit“ bei Dr. Gerd Beuster im Wintersemester 2012/2013

Florian Meister
FH Wedel
inf9116

2. April 2013

Inhaltsverzeichnis

1 Einleitung **2**

2 Pseudozufallszahlengenerator in Debian / OpenSSL **2**

2.1 Grundlagen 2

2.1.1 Zufallszahlen und Entropie 2

2.2 Fehlerhafter Generator in Debian Systemen 4

2.2.1 OpenSSL 4

void ssleay_rand_add(const void *buf, int num, double add) 4

int ssleay_rand_bytes(unsigned char *buf, int num) 4

2.2.2 Problematik in Debian 5

Ursprung des Problems 5

Funktionsweise nach der Änderung 7

2.2.3 Auswirkungen und betroffene Dienste 8

Betroffene Dienste 8

Angriffe 9

2.2.4 Möglichkeiten zum Vorbeugen solcher Fehler 10

Kommunikation 10

Testen von Zufallszahlengeneratoren 11

3 Fazit **13**

Literatur **14**

Abbildungsverzeichnis **15**

1 Einleitung

Zufallszahlengeneratoren sind einer der wichtigsten Bestandteile moderner kryptografischer Verfahren. Ist ihre Funktionalität eingeschränkt, so kann auch die Sicherheit einer Verschlüsselung nicht mehr gewährleistet sein. Tritt ein solcher Fehler auf, so ergeben sich daraus zahlreiche Gefahren, sowohl für die alltägliche Kommunikation im Internet, als auch zum Beispiel für gesichert übertragene Firmen- und Regierungsdaten.

In dieser Ausarbeitung wird speziell auf einen Fehler im Pseudozufallszahlengenerator der Debian Implementierung von OpenSSL eingegangen. Er blieb für mehr als anderthalb Jahre (17. September 2006 – 13. Mai 2008 [OpenSSL Version 0.9.8c-1 bis 0.9.8g-9][1] unentdeckt, bis er von Luciano Bello gefunden wurde, als dieser eine große Anzahl an Schlüsseln erzeugte und sich über vorkommende Duplikate wunderte. Er betraf auch die von Debian abgeleiteten Derivate, wie etwa das weitverbreitete Ubuntu.

2 Pseudozufallszahlengenerator in Debian / OpenSSL

2.1 Grundlagen

2.1.1 Zufallszahlen und Entropie

Entropie ist in der Informatik ein Maß für die Zufälligkeit bzw. den Informationsgehalt einer Bitfolge. Betrachtet man die Sequenzen
00000000000000000000
und
11010001111100100000
bei einer gleichen Auftrittswahrscheinlichkeit für Nullen und Einsen, so wirkt die erste intuitiv weniger zufällig, obwohl die Wahrscheinlichkeit für beide identisch ist ($\frac{1}{2^{20}}$). Je geordneter ein System ist, desto weniger Zufall ist jedoch in ihm enthalten und so liegt es nahe, dass Sequenzen, die aus Mustern bestehen, weniger Zufälligkeit enthalten.
Die Entropie solch einer Zahlenfolge wird dadurch bestimmt, wie komplex sie zu beschreiben ist (Kompression). Für die erste Folge oben könnte man auch „zwanzig Mal Null"schreiben, in der Zweiten ist kein erkennbares Muster vorhanden und die oben stehende Notation ist die kürzeste mögliche. [2, S. 2]
Die Entropie wird in Bit gemessen und ist maximal, wenn sie der Länge einer Bitfolge entspricht. Für die Generation von Zufallszahlen werden Zeichenketten fester Länge und möglichst hoher Entropie benötigt, der Computer verfügt allerdings naturgemäß eher über lange, recht vorhersehbare Datenströme.
Da ein deterministisches Programm keine Entropie erzeugen kann, nutzen Pseudozufallszahlengeneratoren einen internen Entropiepool, aus welchem bei Bedarf Bitfolgen hoher Entropie (statistisch unabhängige und gleichverteilte Bits) entnommen werden können (vgl. Abbildung 1 auf Seite 3).

[1] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0166 (Stand: 23.02.2013)[1]

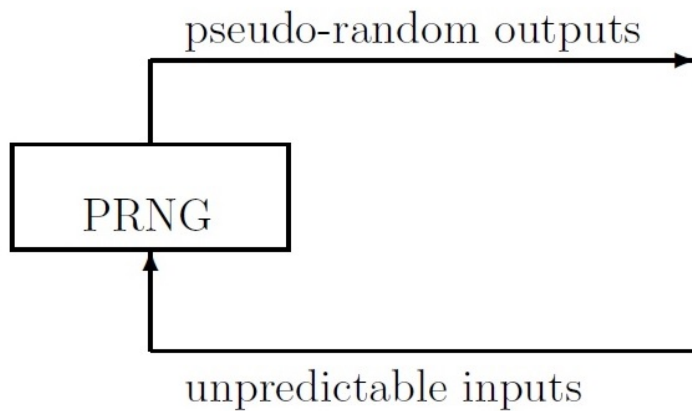

Abbildung 1: Funktionsweise Pseudozufallszahlengenerator [3]

Um diesen zu füllen, werden beliebige, meist wenig Zufälligkeit enthaltende, Zeichenketten verwendet und ihre minimale Entropie abgeschätzt.
Diese werden dann mithilfe einer Hash-Funktion (z.B. MD5 oder SHA1) in den Entropiepool übernommen. Durch die Funktionsweise der Hash-Algorithmen hat jedes der Eingangsbits Einfluss auf das Ergebnis, sodass eventuell enthaltene Entropie in jedem Fall übernommen wird.
Ist keine Zufälligkeit in den hinzuzufügenden Bits vorhanden, so bleibt die Entropie im Pool gleich, sodass es nie schadet, mehr Daten hinzuzufügen. Auf diese Weise kann Entropie „gesammelt"werden und die Ergebnisse lassen sich praktisch nicht mehr von denen eines echten Zufallszahlengenerators unterscheiden. [3]
Mögliche Quellen für Entropie bei Computersystemen sind z.B. das Umgebungsrauschen von Gerätetreibern, Äbstände zwischen Tastatureingaben und Werte aus dem Netzwerkverkehr. [5]

2.2 Fehlerhafter Generator in Debian Systemen

2.2.1 OpenSSL

OpenSSL ist eine Open Source Bibliothek, die die SSL (Secure Socket Layer) und TLS (Transport Layer Security) Protokolle implementiert. Auf dieser Bibliothek basiert die Übertragungssicherheit vieler wichtiger, im Open Source Bereich genutzter, Betriebssysteme.[6]
Hier soll es nur um den betroffenen Zufallszahlengenerator dieses Paketes (crypto/rand/md_rand.c) gehen. Dabei handelt es sich um einen Pseudozufallszahlengenerator, der einen Teil seines Initialwertes (Seed) in Unix-Systemen von /dev/random liest. Dies ist ein vom Betriebssystem bereitgestellter Zufallszahlengenerator, der für hohe kryptografische Ansprüche ausgelegt ist, allerdings aus Performance-Gründen nur für die Initialisierung genutzt wird.[2]
In kryptografischen Anwendungen wird eine große Menge an Zufallsdaten benötigt: für RSA zum Beispiel große zufällige Primzahlen, beim One-time Pad Verfahren lange Ketten von zufälligen Integer-Werten. Ausgehend von dem gelesenen Wert soll nun der Entropiepool vergrößert und die Entropie erhöht werden, um Zugriff auf mehr Zufallswerte zu bieten.

Hierzu betrachten wir die Funktionen *void ssleay_rand_add(const void *buf, int num, double add)* und *int ssleay_rand_bytes(unsigned char *buf, int num)* genauer.

void ssleay_rand_add(const void *buf, int num, double add)
Diese Funktion wird genutzt, um Daten, die mittels des `*buf`-Parameters übergeben werden, zum Entropiepool hinzuzufügen.
`Num` gibt die Anzahl der Bytes, die aus dem Buffer in den Pool übernommen werden soll, an. Es kann auch der komplette Speicher übernommen werden, auch wenn er zuvor nicht vollständig gefüllt wurde. Das heißt es wird auch der uninitialisierte Speicherplatz am Ende des Arrays übernommen. Dies kann, wie zuvor erläutert, nicht schaden, da sich hierdurch die enthaltene Entropie nicht verschlechtern kann und es kann die enthaltene Zufälligkeit erhöhen, wenn noch Werte im uninitialisierten Speicherplatz enthalten sind.
In `add` wird die untere Grenze für die erwartete Entropie übergeben, sodass eine Abschätzung möglich ist, wie viel Entropie momentan im Pool vorhanden ist.
Diese Schätzung wird bei Nutzung eines uninitialisierten Bereiches nicht höher angesetzt, da ungewiss ist, ob diese Bits Entropie enthalten.

int ssleay_rand_bytes(unsigned char *buf, int num)
Mithilfe dieser Funktion können Zufallszahlen bestimmter Länge und hoher Entropie angefordert werden. Sie wird von `RAND_bytes` aufgerufen, wenn man den Pseudozufallszahlengenerator nutzt.
Der Buffer wird leer übergeben und dann mit Zufallsbytes aus dem Entropiepool gefüllt.

[2] http://man7.org/linux/man-pages/man4/random.4.html (Stand: 20.03.2013)[7]

Der Parameter `num` gibt die Anzahl der benötigten Bytes an, die in den Buffer geschrieben werden sollen. Ist laut der internen Variable zu wenig Entropie im Pool vorhanden, so wird eine Fehlermeldung ausgegeben.
Nachdem Zufallszahlen entnommen wurden, wird die interne Entropieschätzung entsprechend heruntergesetzt.
Auch innerhalb dieser Funktion wird uninitialisierter Speicher genutzt, um eventuell die Entropie zu erhöhen. Dazu wird zu Beginn der noch komplett leere Speicher in den Entropiepool hinzugefügt.

2.2.2 Problematik in Debian

Ursprung des Problems

Ein Debian Maintainer (Kurt Roeckse) untersuchte 2006 die OpenSSL Bibliothek, die in Debian verwendet werden sollte. Hierfür nutzte er das Debugging-Tool Valgrind.
In diesem enthalten ist das memcheck-Tool, welches genutzt werden kann, um Speicherlecks, Lesen und Schreiben über Speichergrenzen hinaus und Lese- und Schreibzugriffe auf freigegebenen Speicher im Programm zu finden. [3]
Außerdem meldet es Zugriffe auf uninitialisierten Speicher, da es normalerweise nicht sinnvoll ist, Aktionen aufgrund von unbekannten Werten auszuführen. Wie im letzten Kapitel beschrieben, ist es im Fall von OpenSSL in Ordnung, da hier die Unbekanntheit der Werte gegebenenfalls die Entropie erhöhen kann - Valgrind meldet die entsprechenden Speicherzugriffe aber selbstverständlich dennoch.
Der Maintainer suchte daher nach einer Möglichkeit, die Fehlermeldungen wegen des uninitialisierten Speichers zu vermeiden, um gegebenenfalls vorhandene echte Fehler leichter finden zu können.
Nachdem er die entsprechenden Stellen lokalisiert hatte, wendete er sich an die OpenSSL-Mailingliste mit folgender Nachricht[4]:

> Subject: Random number generator, uninitialised data and valgrind. Date: 2006-05-01 19:14:00
>
> When debbuging applications that make use of openssl using valgrind, it can show alot of warnings about doing a conditional jump based on an unitialised value. Those unitialised values are generated in the random number generator. It's adding an unintialiased buffer to the pool.
>
> The code in question that has the problem are the following 2 pieces of code in crypto/rand/md_rand.c:
>
> 247: MD_Update(&m,buf,j);
>
> 467: #ifndef PURIFY MD_Update(&m,buf,j); /* purify complains */ #endif
>
> Because of the way valgrind works (and has to work), the place where the unitialised value is first used, and the place were the error is reported can be totaly different and it can be rather hard to find what the problem is.

[3]http://www.valgrind.org/info/tools.html#memcheck (Stand: 23.03.2013)[8]
[4]http://marc.info/?l=openssl-dev&m=114651085826293&w=2 (Stand: 15.03.2013)[9]

> ...
>
> What I currently see as best option is to actually comment out those 2 lines of code. But I have no idea what effect this really has on the RNG. The only effect I see is that the pool might receive less entropy. But on the other hand, I'm not even sure how much entropy some unitialised data has.
>
> What do you people think about removing those 2 lines of code?

Die beiden Zeilen sind identisch und die zweite ist als obsolet gekennzeichnet, indem im Kommentar erklärt wurde, dass auf die Codezeile verzichtet werden kann, wenn Purify (ein kommerzielles Tool, welches ähnlich arbeitet wie Valgrind) Meldungen dazu ausgibt. Für den Maintainer lag es nun nahe, dass das obere Vorkommen der Zeile auch von keiner großen Bedeutung war.
Er erhielt mehrere Antworten, Ulf Möller, ein OpenSSL Entwickler [5], schrieb[6]:

> Not much. If it helps with debugging, I'm in favor of removing them. (However the last time I checked, valgrind reported thousands of bogus error messages. Has that situation gotten better?)

Und Marco Roeland[7]:

> [...] So yes I think not using the uninitialized memory (it's only a single line, the other occurrence is already commented out) helps valgrind.

[5] http://www.openssl.org/about/ (Stand: 10.03.2013)[10]
[6] http://marc.info/?l=openssl-dev&m=114652287210110&w=2 (Stand: 15.03.2013)[11]
[7] http://marc.info/?l=openssl-dev&m=114655011201800&w=2 (Stand: 15.03.2013)[12]

#	revision 140 by *kroeckx*, Tue May 2 16:25:19 2006 UTC	revision 141 by *kroeckx*, Tue May 2 16:34:53 2006 UTC
#	**Line 271** static void ssleay_rand_add(const void *	**Line 271** static void ssleay_rand_add(const void *
271	else	else
272	MD_Update(&m,&(state[st_idx]),j);	MD_Update(&m,&(state[st_idx]),j);
273		
274		/*
275		* Don't add uninitialised data.
276	MD_Update(&m,buf,j);	MD_Update(&m,buf,j);
277		*/
278	MD_Update(&m,(unsigned char *)&(md_c[0]),sizeof(md_c));	MD_Update(&m,(unsigned char *)&(md_c[0]),sizeof(md_c));
279	MD_Final(&m,local_md);	MD_Final(&m,local_md);
280	md_c[1]++;	md_c[1]++;
#	**Line 465** static int ssleay_rand_bytes(unsigned ch	**Line 468** static int ssleay_rand_bytes(unsigned ch
468	MD_Update(&m,local_md,MD_DIGEST_LENGTH);	MD_Update(&m,local_md,MD_DIGEST_LENGTH);
469	MD_Update(&m,(unsigned char *)&(md_c[0]),sizeof(md_c));	MD_Update(&m,(unsigned char *)&(md_c[0]),sizeof(md_c));
470	#ifndef PURIFY	#ifndef PURIFY
471		/*
472		* Don't add uninitialised data.
473	MD_Update(&m,buf,j); /* purify complains */	MD_Update(&m,buf,j); /* purify complains */
474		*/
475	#endif	#endif
476	k=(st_idx+MD_DIGEST_LENGTH/2)-st_num;	k=(st_idx+MD_DIGEST_LENGTH/2)-st_num;
477	if (k > 0)	if (k > 0)

Abbildung 2: Diff der letzten funktionierenden Version und der ersten fehlerhaften [4]

Also kommentierte er die Codezeilen aus (siehe Abbildung 2).

Funktionsweise nach der Änderung

Das erste auskommentierte Vorkommen von `MD_Update` ist die einzige Stelle, wo der an `ssleay_rand_add` übergebe Buffer mit Zufallswerten von /dev/random zur Initialisierung des Pseudozufallszahlengenerators hinzugefügt wird. Dieser wird also in der neuen Version nicht in den Entropiepool übernommen, die übergebene erwartete Entropie wird jedoch zur bisher vorhandenen addiert.

Es gibt zwar noch weitere Aufrufe von `MD_Update` im Programmablauf, diese erzeugen jedoch keine echte Zufälligkeit. Der Grund, warum trotz der eingeschränkten Initialisierung noch 2^{15-1} (32.767) verschiedene Werte je Schlüsselgröße und Prozessorarchitektur erzeugt werden konnten, liegt in diesen Codezeilen:

Listing 1: Aufruf mit Process-ID

```
1 if (curr_pid) /* just in the first iteration to save time */
2 {
3     MD_Update(&m,(unsigned char*)&curr_pid,sizeof curr_pid);
4     curr_pid = 0;
5 }
```

Die aktuelle Prozess-ID wird hier in den Entropiepool übernommen, wobei die maximale ID in Linux Systemen standardmäßig 32.767 ist.

Da es sich um einen Pseudozufallszahlengenerator handelt, kann ein Angreifer, der den Anfangszustand und alle darauf folgenden Schritte in ihrer Reihenfolge kennt, den daraus folgenden Endzustand berechnen.

2.2.3 Auswirkungen und betroffene Dienste

Betroffene Dienste
Betroffen waren die folgenden Systeme:

- Ubuntu 7.04 (Feisty)
- Ubuntu 7.10 (Gutsy)
- Ubuntu 8.04 LTS (Hardy)
- Ubuntu „Intrepid Ibex "(development): libssl Versionen bis 0.9.8g-8
- Debian 4.0 (etch)

,wenn auf ihnen ein SSH-Server installiert war, SSH Schlüssel oder SSL-Zertifikate erzeugt wurden.[8]
Betroffen waren unter anderem SSH-, OpenVPN-, DNSSEC-Schlüssel und Material für die Verwendung in X.509-Zertifikaten und Sitzungsschlüssel, die in SSL/TLS-Verbindungen verwendet wurden. Mit GnuPG oder GNUTLS erzeugte Schlüssel waren jedoch nicht betroffen.[9]
Auch die Sicherheit des Webservers Apache, des Nameservers Bind, der E-Mail-Verschlüsselung S/MIME sowie die Vertrauenswürdigkeit digitaler Signaturen war gefährdet.
Die entsprechenden Schlüssel und Zertifikate mussten allesamt neu generiert werden, als der Fehler entdeckt wurde. Da Server häufig genutzt werden, um Schlüssel für ganze Netzwerke zu generieren, waren deutlich mehr als die oben genannten Systeme indirekt betroffen. [5].
Dies setzte voraus, dass die Systemadministratoren auf die Sicherheitsmeldung aufmerksam wurden und die Tragweite und Wichtigkeit einer schnellen Reaktion erkannten.
Dies untersuchten Ylek, Scott, et al. in der Arbeit „When private keys are public: results from the 2008 Debian OpenSSL vulnerability“. Aus der Grafik 3 auf Seite 9 lässt sich ablesen, dass viele Server noch recht lange Zeit nach der Veröffentlichung des Fehlers angreifbar blieben, manche sogar bis zum Ende des Beobachtungszeitraums. Untersucht wurden über 50.000 SSL/TLS Webserver.

[8] http://www.ubuntu.com/usn/usn-612-1/ (Stand: 15.03.2013)[13]
[9] http://www.debian.org/security/2008/dsa-1571 (Stand: 15.03.2013)[14]

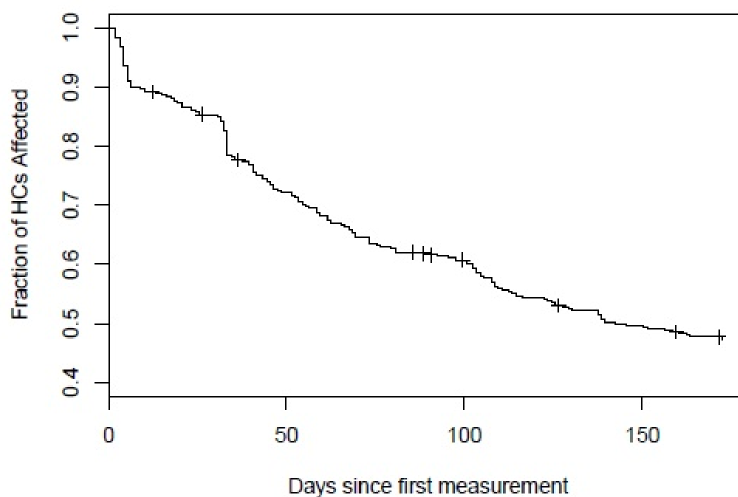

Abbildung 3: Betroffene Host-Server [15]

Angriffe
Angreifern war es möglich, beispielsweise SSL-Verbindungen abzuhören und zu manipulieren, sich unautorisierten Zugriff auf SSH-Server zu verschaffen oder den Cache von DNS-Servern zu verändern.[10]
Um ein betroffenes System anzugreifen, genügt meist ein einfacher Brute-Force-Angriff. Hierzu werden zunächst alle möglichen privaten Schlüssel zum Testen generiert und anschließend durchprobiert.
Da der Schlüssel abhängig von der Prozess-ID ist, liegt es nahe, die Brute-Force-Methode mithilfe dieses Wissens zu optimieren. Linuxsysteme vergeben Prozess-IDs aufsteigend ab dem Systemstart, bis alle benutzt wurden. Die Vermutung, dass Schlüssel häufig recht schnell nach dem Bootzeitpunkt generiert werden, bestätigt sich allerdings nur unwesentlich, siehe Grafik 4 auf Seite 10.

Auch die Prozessorarchitektur der Server spielt eine Rolle, nach ([15]) waren 80% der Server 32-Bit-Architekturen und nur 20% 64-Bit, sodass es sich eher lohnen würde, erst die 32-Bit-Schlüssel zu testen, falls die Architektur des angegriffenen Servers unbekannt ist.

Eine bekannte Stelle, die schwache Zertifikate nutzte, war z.B. der Dienstleister Akamai, der unter anderem Server für die deutsche elektronische Steuererklärung (ELSTER) und für den Download von ATI Treibern bereitstellte. [11]
Auch eine große deutsche Bank war angeblich betroffen.[12]

[10] http://www.heise.de/security/meldung/Schwache-Krypto-Schluessel-unter-Debian-Ubuntu-und-Co-207332.html (Stand: 15.03.2013)[16]
[11] http://blog.fefe.de/?ts=b6c9ec7e (Stand: 15.03.2013)[17]
[12] http://lists.grok.org.uk/pipermail/full-disclosure/2008-May/062537.html (Stand: 16.03.2013)[18]

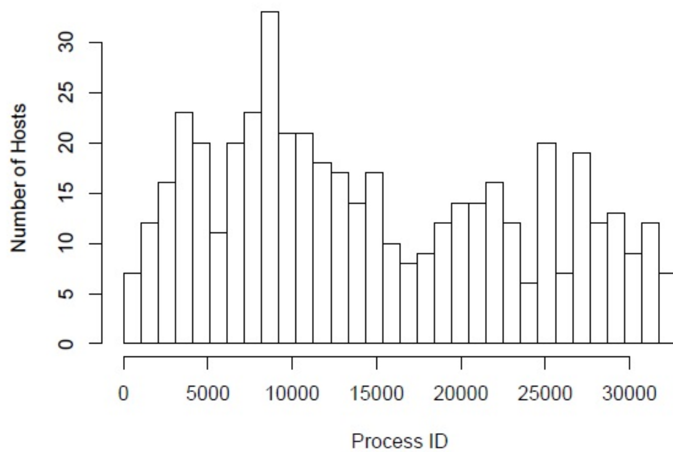

Abbildung 4: Process-IDs der betroffenen Systeme [15]

Das X.509-Zertifikat dieser Server ist öffentlich abrufbar und unter Ausnutzung des Fehlers kann auch der zugehörige private Schlüssel bestimmt werden. Dies ermöglicht einen Man-in-the-Middle Angriff, bei dem eine gefälschte Webseite mit dem echten Zertifikat ausgestattet wird, sodass die Besucher kaum einen Verdacht schöpfen, da sie sich auf die https-Verbindung mit gültigem Zertifikat verlassen.
Diese Zertifikate verfallen in der Standardeinstellung als Sicherheitsmaßnahme ein Jahr nach der Ausstellung, das der Bank war angeblich sogar für 3 Jahre gültig[18].
In solch einem Fall können Zertifikate zwar für ungültig erklärt werden, allerdings überprüften die meisten Browser dies nicht.
In Firefox kann eine Überprüfung beispielsweise unter Tools → Options → Advanced → Validation → Use the Online Certificate Status Protocol (OCSP) to confirm the current validity of certificates aktiviert werden.

2.2.4 Möglichkeiten zum Vorbeugen solcher Fehler

Kommunikation

Ulf Möller, der OpenSSL Entwickler, der in der Mailingliste geantwortet hatte, äußerte sich nachdem das Problem bekannt geworden war in einem Forum zu den Kommunikationsproblemen wie folgt[13]:

> Es gab zwei Probleme: Kurt Roeckx sprach vom Debuggen von Anwendungen, nicht davon, dass er Debian patchen wollte, und er behauptete, dass „buf “ in Zeile 247 ein uninitialisierter Buffer sei. Nun kenne ich tatsächlich nicht alle 625000 Zeilen auswendig, und da wirklich auch uninitialisierte Buffer in den Pool eingemischt werden und seine Mail wie eine allgemeine Diskussion zum Thema Entropie daherkam, habe ich keinen Grund gesehen, in

[13] http://www.c-plusplus.de/forum/p1510780#1510780 (Stand: 23.03.2013)[19]

> den Programmcode zu schauen - und mich dafür ausgesprochen, die uninitialisierten Buffer beim Debuggen wegzulassen. Seine Behauptung war aber leider falsch; Zeile 247 ist die einzige Stelle, an der buf, das Argument von ssleay_rand_add(), überhaupt dereferenziert wird! Um das zu merken, muss man mit dem Code auch nicht besonders vertraut sein.
>
> Wenn er den Grund seiner Anfrage erwähnt oder um eine Prüfung seines Patches gebeten hätte, hätte ich mir den Kontext angesehen und er hätte eine ganz andere Antwort bekommen. Dumm gelaufen. Das eigentliche Problem ist aber, dass Debian einzelne Personen sicherheitsrelevante Änderungen machen lässt, ohne dass die von irgendjemandem geprüft werden.

Bei derart sicherheitsrelevanten Paketen sollten Änderungen in jedem Fall durch die eigentlichen Entwickler der Bibliotheken geprüft werden. Dies sehen auch die Debian Entwicklerrichtlinien vor.[14]

Testen von Zufallszahlengeneratoren

Das Testen von Zufallszahlen ist generell recht schwierig, da man bei ungewöhnlichen Zufallsverteilungen nie wissen kann, ob ein Fehler vorliegt oder ob durch Zufall ein sehr unwahrscheinlicher Fall eingetreten ist. Dennoch wäre es wünschenswert testen zu können, ob Abhängigkeiten zwischen Abschnitten oder Elementen der Ausgabe bestehen und ob die erhaltenen Werte ungefähr gleich verteilt sind.
Zum Testen gibt es grundsätzlich zwei Ansätze: empirische Tests und theoretische Tests. Die Erstgenannten arbeiten mit den Ausgaben der Zufallszahlengeneratoren, während bei Letzteren auch die Struktur der Generatoren selbst betrachtet wird.[2, S. 6] Hier soll es ausschließlich um empirische Tests gehen.
Um die Initialisierung des Pseudozufallszahlengenerators zu testen, wäre es ein Anfang gewesen, eine große Anzahl an Schlüssel zu erzeugen und diese auf Duplikate zu überprüfen. Diese können zwar auch bei einem einwandfrei funktionierenden Generator auftreten, sobald sich diese allerdings häufen, sollte man dennoch in Betracht ziehen, dass man einen Fehler bei seinen Änderungen gemacht hat.
Es gibt außerdem einige systematischere Ansätze, um die Verteilung der erzeugten Zufallszahlen zu testen.

[14] http://www.debian.org/doc/debian-policy/ch-source.html#s4.3 (Stand: 17.02.2013)[20]

Chi-Quadrat-Test

Dies ist vermutlich der am häufigsten genutzte Test, um Zufallszahlengeneratoren zu überprüfen. Hierbei betrachten wir n verschiedene Testwerte, von denen jeder in eine von k Kategorien fällt. Beschreibt Y_s die Häufigkeit des Auftretens der Kategorie s, für die die Wahrscheinlichkeit p_s, erwarten wir, dass ungefähr

$$Y_s \approx p_s * n$$

gilt. Es soll nun geprüft werden, wie weit die Kategorien (jeweils gewichtet) von dem jeweiligen Idealwert abweichen. Dazu wird die Chi-Quadrat-Statistik V von Y_1 bis Y_k berechnet:

$$V = \frac{1}{n} \sum_{1 \leq s \leq k} \frac{Y_s^2}{p_s} - n$$

Mithilfe dieses Wertes und einer Chi-Quadrat-Verteilungstabelle, welche den Zusammenhang zwischen der Anzahl an Kategorien und den Wahrscheinlichkeiten für bestimmte V beschreibt, lässt sich bestimmen, wie wahrscheinlich das berechnete V ist. Um nun Zufallszahlenfolgen zu testen, berechnet man die Chi-Quadrat-Statistik mehrerer Teile davon und überprüft, ob die Mehrzahl dieser sehr unwahrscheinlich ist. In diesem Fall werden die Werte als nicht zufällig angesehen und der Generator sollte überprüft werden.[2, S. 7f.]

Kolmogorow-Smirnow-Test

Dieser Test kann, wie der Chi-Quadrat-Test, verwendet werden, um zu überprüfen, ob die erzeugten Zufallszahlen einer bestimmten Verteilung z.B. Gleichverteilung folgen.[15] Er ist genauer als der Chi-Quadrat-Test und kann auch für kurze Zufallszahlen eingesetzt werden, allerdings ist er auch aufwendiger in der Durchführung.

FIPS 140-2

Der FIPS 140-2 Test sammelt immerhin 20.000 Bytes für einfache statische Tests, was im Fall des hier behandelten Fehlers vermutlich ausgereicht hätte, um die Qualität der Entropiequelle zu überprüfen.[22]

Testsuites

Um viele solcher Tests leicht durchführen zu können gibt es einige Testsuites, z.B. „Diehard“[16] und dessen Weiterentwicklung „Dieharder“[24], die genutzt werden können, um die Verteilung in den Ausgaben des Pseudozufallszahlengenerators zu überprüfen.

Weitere Tests

15 weitere Tests und eine Beschreibung deren Funktionsweise finden sich unter [17]

[15] http://www.statistik.tuwien.ac.at/public/dutt/vorles/inf_bak/node61.html (Stand: 25.03.2013)[21]
[16] http://www.stat.fsu.edu/pub/diehard/ (Stand: 25.03.2013)[23]
[17] http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html (Stand 15.03.2013)[25]

3 Fazit

In dieser Arbeit wurde gezeigt, wie verhängnisvoll eine derart kleine Änderung in einer sicherheitsrelevanten Bibliothek sein kann und welche Angriffsmöglichkeiten daraus speziell in diesem Fall entstanden sind.
Es lässt sich festhalten, dass der Fehler eventuell durch bessere Zusammenarbeit mit den eigentlichen Entwicklern oder durch anschließendes Testen hätte vermieden werden können.
Da kryptografische Verfahren in hohem Maße von guten Zufallszahlen abhängen, sollte die Bereitstellung dieser ebenso gut durchdacht und überprüft werden, wie die Verschlüsselungsalgorithmen selbst.
Wie der Generator geprüft werden kann, wurde ebenfalls beschrieben.

Andere Beispiele für Sicherheitslücken durch schwache Zufallszahlen werden unter anderem in

- Luczaj, Michal. „Cryptographic software: vulnerabilities in implementations.“ Annales UMCS, Informatica. Vol. 11. No. 4. Versita, Warsaw, 2011.
- Stieber, Anthony J. „Enterprise Cryptographic Key Management Realities and Issues.“(2010).
- Heninger, Nadia, et al. „Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices.“(2012)

beschrieben.

Literatur

[1] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0166
(Stand: 23.02.2013)

[2] Biebighauser, Dan. „Testing Random Number Generators.“University of Minnesota (2000).

[3] Kelsey, John, et al. „Cryptanalytic attacks on pseudorandom number generators. “Fast Software Encryption. Springer Berlin/Heidelberg, 1998.

[4] http://anonscm.debian.org/viewvc/pkg-openssl/openssl/trunk/rand/md_rand.c?p2=%2Fopenssl%2Ftrunk%2Frand%2Fmd_rand.c&p1=openssl%2Ftrunk%2Frand%2Fmd_rand.c&r1=141&r2=140&view=diff&pathrev=141 (Stand: 02.04.2013)

[5] Radziszowski, S. P., Jeton Bacaj, and Adam Friedlander. „Randomness in Modern Cryptography.“
verfügbar unter http://www.cs.rit.edu/ ajf5570/crypto2.pdf (Stand: 02.04.2013)

[6] Ahmad, David. „Two years of broken crypto: debianś dress rehearsal for a global PKI compromise. “Security & Privacy, IEEE 6.5 (2008): 70-73

[7] http://man7.org/linux/man-pages/man4/random.4.html (Stand: 20.03.2013)

[8] http://www.valgrind.org/info/tools.html#memcheck (Stand: 23.03.2013)

[9] http://marc.info/?l=openssl-dev&m=114651085826293&w=2 (Stand: 15.03.2013

[10] http://www.openssl.org/about/ (Stand: 10.03.2013)

[11] http://marc.info/?l=openssl-dev&m=114652287210110&w=2 (Stand: 15.03.2013)

[12] http://marc.info/?l=openssl-dev&m=114655011201800&w=2 (Stand: 15.03.2013)

[13] http://www.ubuntu.com/usn/usn-612-1/ (Stand: 15.03.2013)

[14] http://www.debian.org/security/2008/dsa-1571 (Stand: 15.03.2013)

[15] Yilek, Scott, et al. „When private keys are public: results from the 2008 Debian OpenSSL vulnerability.“Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference. ACM, 2009.

[16] http://www.heise.de/security/meldung/Schwache-Krypto-Schluessel-unter-Debian-Ubuntu-und-Co-207332.html (Stand: 15.03.2013

[17] http://blog.fefe.de/?ts=b6c9ec7e (Stand: 15.03.2013)

[18] http://lists.grok.org.uk/pipermail/full-disclosure/2008-May/062537.html (Stand: 16.03.2013)

[19] http://www.c-plusplus.de/forum/p1510780#1510780 (Stand: 23.03.2013)

[20] http://www.debian.org/doc/debian-policy/ch-source.html#s4.3
(Stand: 17.02.2013)

[21] http://www.statistik.tuwien.ac.at/public/dutt/vorles/inf_bak/node61.html
(Stand: 25.03.2013)

[22] FIPS, PUB. „140-2: Security Requirements for Cryptographic Modules.“National Institute of Standards and Technology (2001).

[23] http://www.stat.fsu.edu/pub/diehard/ (Stand: 25.03.2013)

[24] Brown, Robert G., D. Eddelbuettel, and D. Bauer. „Dieharder: A random number test suite.“, 2009
verfügbar unter http://www.phy.duke.edu/ rgb/General/dieharder.php (Stand: 02.04.2013).

[25] http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html (Stand 15.03.2013)

[26] Luczaj, Michal. „Cryptographic software: vulnerabilities in implementations.“ Annales UMCS, Informatica. Vol. 11. No. 4. Versita, Warsaw, 2011.

[27] Stieber, Anthony J. „Enterprise Cryptographic Key Management Realities and Issues.“(2010).

[28] Heninger, Nadia, et al. „Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices.“(2012).

Abbildungsverzeichnis

1 Funktionsweise Pseudozufallszahlengenerator 3
2 Diff der letzten funktionierenden Version und der ersten fehlerhaften . . . 7
3 Betroffene Host-Server 9
4 Process-IDs der betroffenen Systeme 10