Mining your P's and Q's

Detection of Widespread Weak Keys in Network Devices

Paperinformationen

Autoren: Nadia Heninger, Zakir Durumeric, Eric Wustrow, J. Alex Halderman
Veröffentlicht: 21 USENIX Security Symposium (2012)
In diesem Paper geht es um den negativen Einfluss von mangelhafter Entropie, bei der Generierung von Zufallszahlen, auf Verschlüsselungsalgorithmen.

RSA Schwachstelle

Beim RSA Algorithmus werden zwei Primzahlen *p* und *q* genutzt, um einen Wert *n* und daraus wiederum sowohl den privaten als auch den öffentlichen Schlüssel zu berechnen.

Eine gewöhnliche Faktorisierung einer solchen Zahl n einer Größe von 768 Bit wurde 2009 mit 1000 Kernen über zwei Jahre vorgenommen.

p und *q* werden zufällig ermittelt. Ist die Entropie bei der Ermittlung der Zufallszahl mangelhaft, kann es dazu kommen, dass zwei *n* sich den gleichen Faktor *p* teilen.

In dem Fall. dass sich n_1 und n_2 einen Faktor *p* teilen, kann dieser über den $ggt(n_1, n_2)$ bestimmt werden und somit auch die verbleibenden Faktoren berechnet werden. Dies funktioniert in unter einer Sekunde.

DSA Schwachstelle

Der DSA Algorithmus basiert auf mehr zufällig gewählten Werten, die jedoch zum größten Teil öffentlich sind. Ein DSA Zertifikat besteht aus den beiden Werten (r, s), welche auch öffentlich sind.

Es gibt einen festen privaten Schlüssel x und einen vergänglichen privaten Schlüssel *k*, welcher nach jeder Nachricht zufällig gewählt wird.

Aus zwei Nachrichten, die mit demselben *k* verschlüsselt wurden, dann mithilfe der öffentlichen Informationen der feste private Schlüssel *x* errechnet werden.

TLS Angriffsszenario

Mangelhafte Entropie kann Angreifern dabei helfen den privaten Schlüssel zu berechnen. Ein passiver Angreifer kann bei einer TLS Session, die den RSA Algorithmus nutzt, mithilfe des privaten Schlüssels den Session Key ermitteln und die gesamte Session mitlesen. Dies kann durch den Diffie-Hellman-Austausch verhindert werden.

Ein aktiver Angreifer kann sowohl bei der RSA Verschlüsselung als auch bei einem Diffie-Hellman-Austausch als man-in-the-middle die Kommunikation entschlüsseln und verändern.

SSH Angriffsszenario

Bei SSH-1 wird der öffentliche Schlüssel eines Servers genutzt, um die Session zu verschlüsseln. Bei SSH-2 wird der Diffie-Hellmann-Austausch genutzt.

Wie auch bei TLS, kann sowohl ein passiver als auch ein aktiver Angreifer bei SSH-1 mitlesen. SSH-2 nutzt jedoch den Diffie-Hellman-Austausch, wodurch SSH-2 auf diese Weise nur von einem aktiven Angreifer ausgenutzt werden kann.

Hierbei ist noch wichtig zu erwähnen, dass bei SSH das Passwort im Klartext geschickt wird, wodurch ein Zuhörer Zugriff auf den Server bekommt.

Methodik

2010(TLS) und 2012(SSH) wurden Portscans des gesamten IPv4 Adressraumes durchgeführt. Es wurden mit NMap5 SYN-Anfragen produziert und ein SYN-ACK wurde als aktiver Host interpretiert. Das Ergebnis:

- Port 443: 28.923.800 aktive Hosts
- Port 22: 23.237.081 aktive Hosts

Mit den aktiven Hosts wurde ein Handshake durchgeführt um die Zertifikate bzw. SSH Host Keys abzugreifen.

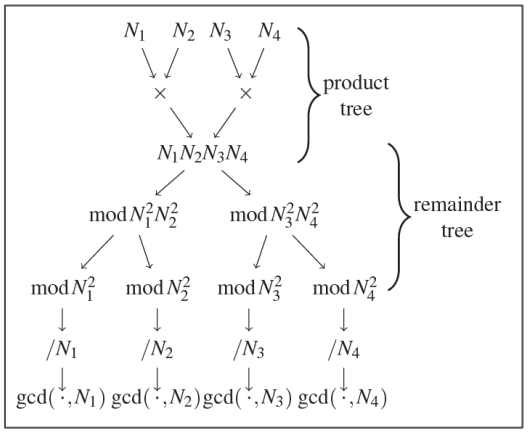
- 12.828.613 TLS Hosts hatten eine Certificate Chain
- 10.216.363 SSH Hosts lieferten einen SSH Host Key aus
- Beides entspricht in etwa 44% der aktiven Hosts

Die Hosts wurden im Falle von TLS zum Großteil durch das Subject- und Issuer-Feld in den X.509 Feldern des Zertifikats gruppiert. SSH Hosts wurden durch eine Kombination von TCP/IP Fingerprinting und vergleichen der ausgelieferten Informationen auf HTTP/HTTPS Ports gruppiert.

Bernstein Algorithmus

Algorithm 1 Quasilinear GCD finding

Input: $N_1, \ldots, N_m$ RSA moduli
1: Compute $P = \prod N_i$ using a product tree.
2: Compute $z_i = (P \bmod N_i^2)$ for all i using a remainder tree.
Output: $\gcd(N_i, z_i/N_i)$ for all i.



Der Bernstein Algorithmus vergleicht Primfaktorprodukte auf sich wiederholende Primfaktoren mit einer Laufzeit von $O(mn \log(m) \log(mn) \log(\log(mn))$ für den „Product Tree“ und $O(mn \log(n)^2 \log(\log(n)))$ für den „Remainder Tree“. m ist hierbei die Anzahl der N’s und n die Bitlänge der Primfaktorprodukte.

Funde und Ursachen

- 7.770.232 (61%) TLS Hosts 6.642.222 (65%) SSH Hosts und haben den gleichen Key ausgeliefert wie mindestens ein weiterer Host
- 1.013 (0,02%) der RSA SSH Host Keys und 23.576 (0,4%) der TLS Privatschlüssel konnten faktorisiert werden.
- /dev/urandom wird leider häufiger verwendet als /dev/random. /dev/urandom wartet jedoch nicht bis genügend Entropie erzeugt wurde (non-blocking). Vor allem auf Headless Geräten mangelt es an Entropie zur Bootzeit

Gegenmaßnahmen (Auswahl)

- Betriebssysteme sollten den Applikationen die Entropie mitteilen
- Bibliotheken sollten die sicherste Konfiguration der RNG’s als Default verwenden
- Applikationen sollten Schlüssel erst bei erster Verwendung generieren
- Hersteller sollten die Entropie-Pools ihrer Geräte seeden und die Wirksamkeit in Tests prüfen