Aufsatz über

Certificate Pinning mit NGINX

Dritte, überarbeitete Fassung

Jörg Kastning
joerg.kastning@my-it-brain.de

09.01.2016

Dieser Aufsatz beschreibt Certificate Pinning, wie es hilft TLS/SSL-Verbindungen zu schützen und wie es für den Webserver NGINX konfiguriert wird. Der Aufsatz beginnt mit einer kurzen Einleitung, welche die grundsätzliche Schwachstelle von TLS/SSL-Verbindungen beschreibt, mit dem Man-In-The-Middle-Angriff eine reale Bedrohung erläutert und Beispiele für Missbrauch nennt. Anschließend wird mit Certificate Pinning für TLS/SSL im zweiten Abschnitt der in RFC 7469 definierte Standard *Public Key Pinning Extension for HTTP* betrachtet und das TOFU-Prinzip erklärt. Der dritte Abschnitt widmet sich der Konfiguration von Certificate Pinning auf dem eigenen Server. Dabei werden notwendige Vorüberlegungen und eine konkrete Backupstrategie erläutert. Anschließend folgt die Berechnung der PINs und die beispielhafte Konfiguration für einen NGINX-Webserver.

Um die Funktionalität des Certificate Pinning zu prüfen wird die Backupstrategie validiert und eine Gegenprobe mit einem Zertifikat durchgeführt, dessen öffentlicher Schlüssel nicht gepinnt wurde. Der Aufsatz endet mit einem kurzen Fazit.

Inhaltsverzeichnis

1 Einleitung

Um zu verstehen, wie Certificate Pinning hilft, TLS/SSL-Verbindungen zu schützen, muss man sich zuerst der Schwachstelle der TLS/SSL-Verschlüsselung bewusst sein.

TLS/SSL-Verbindungen sollen die vertrauliche Kommunikation zwischen zwei Kommunikationspartnern sicherstellen. Bei den Kommunikationspartnern handelt es sich typischerweise um einen Client und einen Server. Die Sicherheit der Vertraulichkeit beruht darauf, dass der Client auch tatsächlich mit dem richtigen Server verbunden ist. Dazu weist sich der Server mit einem Zertifikat aus, welches von einer Zertifizierungsstelle (Certificate Authority, CA) ausgestellt wurde. Die Zertifizierungsstelle prüft die Identität des Dienstbetreibers und beglaubigt mit ihrer digitalen Signatur das Zertifikat.

Die c't schreibt in ihrem Artikel [2, S. 118, Spalte 2-3]: "Das Problem dabei: Es gibt weit über hundert solcher Zertifizierungsstellen, denen die gängigen Internet-Programme wie Browser und E-Mail-Client vertrauen. Als wäre das nicht unübersichtlich genug, haben die dann auch noch zahllose Unter-CAs, die berechtigt und in der Lage sind, im Namen dieser CAs zu unterschreiben."

Das Problem besteht nun nicht in der Anzahl der Zertifizierungsstellen, sondern darin, dass jede dieser Zertifizierungsstellen Zertifikate für beliebige Domains ausstellen darf. So können sich Dritte ein Zertifikat auf eine bereits existierende Domain ausstellen lassen und dieses zum Beispiel für einen Man-In-The-Middle-Angriff (MITM-Angriff)[1] verwen-

[1] `https://de.wikipedia.org/wiki/Man-in-the-middle`

den.

Das genannte Angriffsszenario existiert nicht nur in der Theorie. Wie die c't in [2, S. 119, Spalte 1] berichtet nutzte die chinesische Regierung gefälschte Google-Zertifikate auf ihrer großen Firewall, um den Google-Mail-Verkehr ihrer Bevölkerung überwachen zu können. Darüber hinaus werden weitere Fälle wie der Hack der niederländischen Zertifizierungsstelle DigiNotar und TrustWave aufgeführt. Im ersten Fall wurden gefälschte Zertifikate für GMail und Facebook ausgestellt. Im zweiten Fall wurden Zertifikate für Firmen ausgestellt, welche diese Zertifikate nutzten, um den verschlüsselten Internet-Verkehr ihrer Mitarbeiter zu überwachen.

Die Man-In-The-Middle-Angriffe funktionieren, da die Zertifizierungsstellen, welche die gefälschten Zertifikate ausgestellt haben, in der Liste der vertrauenswürdigen Zertifizierungsstellen der gängigen Browser geführt werden. Für den Browser ist damit auch das gefälschte Zertifikat gültig. Der Benutzer kann den MITM-Angriff nicht erkennen und nimmt fälschlicherweise an direkt mit dem gewünschten Server zu kommunizieren.

Um einen MITM-Angriff erkennen zu können, muss der Client überprüfen können, ob das ihm vorliegende Zertifikat von der CA des Dienstbetreibers oder von einer anderen CA ausgestellt wurde. Hier kommt Certificate Pinning ins Spiel.

2 Certificate Pinning für TLS/SSL

Certificate Pinning wurde im RFC 7469 [1] spezifiziert. Es handelt sich dabei um einen Mechanismus, bestimmte Eigenschaften eines Zertifikats „festzunageln“. Der Browser erhält dabei mit dem Zertifikat zusätzliche Informationen, mit denen er die Authentizität des Zertifikats überprüfen kann.

In der Praxis wird dem Browser ein Hash-Wert übermittelt, welcher über den öffentlichen Schlüssel des Zertifikats gebildet wurde. Dieser Hash-Wert wird in einem HTTP-Header-Field an den Browser übertragen und von diesem gespeichert. Wie dieses Verfahren genau funktioniert beschreibt [1, Abschnitt 2.1.1]. Da nur der Dienstbetreiber über den dazugehörigen privaten Schlüssel verfügt, ist ein Missbrauch durch Dritte ausgeschlossen. Übermittelt nun nämlich ein Angreifer bei einem MITM-Angriff den zu seinem Zertifikat gehörenden öffentlichen Schlüssel, kann der Browser mit Hilfe des PINs erkennen, dass dieser Schlüssel nicht dem Dienstbetreiber gehört und den Angriff damit erkennen.

Eine Schwachstelle bleibt jedoch. Damit das beschriebene Verfahren funktioniert, muss die erste Verbindung, die ein Client zum Server aufbaut, integer und vertrauenswürdig sein. Findet bereits beim ersten Verbindungsaufbau ein MITM-Angriff statt, kann der Angreifer einen zu seinem öffentlichen Schlüssel passenden Hash übermitteln und das beschriebene Verfahren damit aushebeln.

Es existiert also ein gewisses Henne-Ei-Problem, welches im folgenden Abschnitt kurz angerissen wird.

2.1 TOFU: Trust On First Use

TOFU steht in diesem Kontext für das Trust-On-First-Use-Prinzip, welches die c't in [2, S. 121, Kasten „TOFU: Trust On First Use“] erläutert.

Das bereits beschriebene Henne-Ei-Problem besteht darin, dass man zuerst eine sichere Verbindung benötigt, um die erforderlichen Informationen zu erhalten, mit denen zukünftige Verbindungen gesichert werden können.

Mittlerweile sollte jedoch bekannt sein, dass es keine hundertprozentige Sicherheit gibt. Damit ist der beschriebene Schutz besser als gar keiner. In der Praxis kann Certificate Pinning daher in sehr vielen Fällen helfen, die Sicherheit vertraulicher Kommunikation zu steigern.

3 Certificate Pinning auf dem eigenen Server

Dieser Abschnitt beschreibt, wie man Certificate Pinning auf dem eigenen Server konfigurieren kann. Es wird dabei auf notwendige Vorüberlegungen eingegangen und die Berechnung der PINs beschrieben. Im Anschluss wird das Certificate Pinning beispielhaft an einem NGINX-Server demonstriert.

3.1 Vorüberlegungen

Zu Beginn stellt sich die Frage, welchen öffentlichen Schlüssel man „pinnen“ möchte. Denn Certificate Pinning kann auf jeden öffentlichen Schlüssel der gesamten Zertifizierungskette angewendet werden. So kommen sowohl das eigene Serverzertifikat, als auch das der Intermediate CAs und das der Root CA in Frage [vgl. 3, S. 122, Spalte 1].

Pinnt man den öffentlichen Schlüssel einer Intermediate CA oder gar einer Root CA, so werden alle von diesen Zertifizierungsstellen ausgestellten Zertifikate vom Browser als gültig und sicher akzeptiert. Daher erscheint es am sichersten, wenn man den öffentlichen Schlüssel des eigenen Serverzertifikats festnagelt. Hierbei muss allerdings folgendes Risiko beachtet werden.

Wird der Schlüssel durch Kompromittierung unbrauchbar, oder geht durch Hardwaredefekt verloren, können Benutzer die gesicherten Dienste eventuell nicht mehr nutzen. Denn die Browser haben den PIN gespeichert und werden vor Ablauf der Lebensdauer keinen neuen PIN akzeptieren. Die Lebensdauer kann je nach Konfiguration mehrere Wochen bis Monate betragen [siehe 3, S. 122, Spalte 2].

Um das im vorigen Absatz beschriebene Risiko zu minimieren, definiert RFC 7469 die zusätzliche Verwendung eines Backup-PINs. Dabei wird ein Hash-Wert über den öffentlichen Schlüssel eines Schlüsselpaares gebildet, welches aktuell noch nicht genutzt wird

[1, Abschnitt 4.3]. Dies kann zum Beispiel dadurch erreicht werden, dass ein Hash-Wert für den öffentlichen Schlüssel eines Certificate Signing Request (CSR)[2] generiert wird.

Der Backup-PIN wird ebenfalls über ein HTTP-Header-Feld ausgeliefert und vom Browser eines Clients gespeichert. Der erzeugte Backup-CSR sollte sicher offline aufbewahrt werden. Er kann genutzt werden, um ein neues Zertifikat für die entsprechende Domain ausstellen zu lassen. Auf diese Weise können Zertifikate verlängert bzw. erneuert werden, ohne dass der Zugriff auf die entsprechende Domain unterbrochen wird.

Der für den Backup-PIN verwendete CSR ist sicher aufzubewahren. Denn mit ihm kann ein Zertifikat für eine Domain bzw. einen Host ausgestellt werden, welches von Clients als gültig anerkannt wird, da sie den dazugehörigen Backup-PIN gespeichert haben.

Ich persönlich speichere den CSR und den dazugehörigen privaten Schlüssel in KeePassX[3]. Auf diese Weise werden die Informationen sicher in einer Datenbank mit einer 256 Bit AES-Verschlüsselung abgelegt. Die KeePassX-Datenbank selbst bewahre ich in einem TeamDrive[4]-Space auf. Dadurch wird die Datenbank sicher auf meine Endgeräte synchronisiert, so dass die Daten auch bei Ausfall eines Endgeräts erhalten bleiben.

Die lokalen Datenträger der zur TeamDrive-Synchronisation verwendeten Endgeräte sind ebenfalls verschlüsselt. So sind die Daten im Falle eines Diebstahls eines Endgeräts gleich doppelt geschützt. Erstens durch die Verschlüsselung des lokalen Datenträgers und zweitens durch die verschlüsselte KeePassX-Datenbank.

3.2 Die PINs berechnen

Die benötigten PINs werden auf der Kommandozeile mit *openssl* generiert [siehe 1, Appendix A]. Für ein existierendes SSL-Zertifikat kann die PIN mit folgendem Code erzeugt werden:

```
openssl x509 -noout -in certificate.pem -pubkey | openssl \
asn1parse -noout -inform pem -out public.key
openssl dgst -sha256 -binary public.key | openssl enc -base64
```

Der erzeugte PIN wird auf der Standardausgabe ausgegeben und endet immer auf das Zeichen „=“.

Für den Backup-PIN wird zunächst ein neuer Certificate Signing Request erstellt:

```
openssl genrsa -out example.com.key 2048
openssl req -new -key example.com.key -out example.com.csr
```

[2] RFC 2986 https://tools.ietf.org/html/rfc2986
[3] The Official KeePassX Homepage – https://www.keepassx.org/
[4] TeamDrive Homepage – https://www.teamdrive.com/de/

Über den so erzeugten Certificate Signing Request (example.com.csr) wird nun ebenfalls ein PIN mit folgendem Code erzeugt:

```
openssl req -noout -in example.com.csr -pubkey | openssl \
asn1parse -noout -inform pem -out example.com_csr.key
openssl dgst -sha256 -binary example.com_csr.key | openssl enc -base64
```

Der erzeugte Backup-PIN wird ebenfalls auf der Standardausgabe ausgegeben und muss ebenfalls auf das Zeichen „=“ enden.

3.3 Konfiguration von NGINX

Sind die PINs erzeugt, kann anschließend der Webserver konfiguriert werden, um diese auszuliefern. Neben der PIN-Direktive [1, Abschnitt 2.1.1] ist hier die Max-Age-Direktive [1, Abschnitt 2.1.2] von Bedeutung. Letztere gibt die Zeit in Sekunden an, für die ein PIN durch den Browser gespeichert wird.

Für die ersten Tests sollte die Max-Age-Direktive auf wenige Minuten gesetzt werden. Dieser Wert sollte erst bei reibungslosem Betrieb auf mehrere Wochen erhöht werden. Dadurch wird das Risiko minimiert, bei etwaigen Fehlern für längere Zeit nicht mehr auf durch „Pinning“ geschützte Seite zugreifen zu können.

Das zusätzliche Header-Feld muss im SSL-Block der NGINX-Konfiguration hinzugefügt werden. Dazu wird folgender Code in den SSL-Block eingefügt:

```
add_header Public-Key-PINs 'pin-sha256="PRIMARY-PIN"; \
pin-sha256="BACKUP-PIN"; max-age=300; includeSubDomains';
```

Dabei sind PRIMARY-PIN und BACKUP-PIN durch die entsprechenden PINs zu ersetzen. Die optionale includeSubDomains-Direktive [1, Abschnitt 2.1.3] gibt an, dass die übermittelten PINs auch für alle Subdomains des Hosts gelten.

Anschließend muss die Konfiguration neu geladen werden, um diese zu aktivieren:

```
sudo service nginx reload
```

Soweit zur Theorie. Im folgenden Abschnitt wird die Backupstrategie validiert. Damit soll gezeigt werden, dass die in Abschnitt 3.1 auf der vorherigen Seite formulierte Backupstrategie funktional ist.

4 Validierung der Backupstrategie

Die in Abschnitt 3.1 auf Seite 5 aufgestellte Backupstrategie basiert auf der Annahme, dass ein PIN über einen CSR gebildet wird. Mit diesem soll es möglich sein, sich im Bedarfsfall ein neues Zertifikat von einer Zertifizierungsstelle ausstellen zu lassen. In diesem Abschnitt möchte ich zeigen, dass die genannte Backupstrategie funktioniert. Die Validierung erfolgt in einer Laborumgebung, welche im folgenden beschrieben wird.

4.1 Laborumgebung

Als Laborumgebung kommt ein Ubuntu Server 14.04 LTS zum Einsatz. Auf diesem wird ein NGINX Webserver und eine PHP-FPM-Umgebung installiert. Es wird eine Testkonfiguration erstellt und mit einem SSL-Zertifikat von CAcert[5] abgesichert.

Der private Schlüssel und der CSR werden dabei mit dem folgenden Befehl erzeugt:

```
openssl genrsa -out test.my-it-brain.de.key 2048
openssl req -new -key test.my-it-brain.de.key -out test.my-it-brain.de.csr
```

Für den Backup-CSR wird analog verfahren:

```
openssl genrsa -out test.my-it-brain.de.bak.key 2048
openssl req -new -key test.my-it-brain.de.key -out \
  test.my-it-brain.de.bak.csr
```

Das Certificate Pinning wird wie im Abschnitt 3.2 und 3.3 auf der vorherigen Seite beschrieben durchgeführt. Dabei wird lediglich darauf verzichtet auch sämtliche Subdomains mit einzuschließen. Konkret bedeutet dies, dass die Angabe von `includeSubDomains` in der NGINX-Konfiguration entfällt. Die maximale Gültigkeitsdauer wird auf 1 Stunde gesetzt.

Das Zertifikat wird anschließend unter dem Namen `test.my-it-brain.de.crt` gespeichert. Die Konfiguration der Testseite im NGINX sieht im konkreten Fall wie folgt aus. Es werden nur die relevanten Zeilen wiedergegeben:

```
## BEGIN Testlabor CONFIGURATION ##########################################
server {
listen 80;
listen [::]:80;
server_name test.my-it-brain.de;
```

[5] `http://cacert.org`

```
# Path to the root of your installation
     root /var/www/test.my-it-brain.de/public;

}
server {
    # Listen on Port 443
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name test.my-it-brain.de;

    ssl_certificate /var/www/test.my-it-brain.de/ssl/ \
  test.my-it-brain.de.crt;
    ssl_certificate_key /var/www/test.my-it-brain.de/ssl/ \
  test.my-it-brain.de.key;

# Add headers to serve security related headers
add_header Public-Key-Pins 'pin-sha256="<HIER STEHT DER PIN"; \
  pin-sha256="HIER STEHT DER BACKUP-PIN"; max-age=3600';

    # Path to the root of your installation
    root /var/www/$host/public;

    error_page 403 /core/templates/403.php;
    error_page 404 /core/templates/404.php;
}
## END Testlabor CONFIGURATION ##########################################
```

4.2 Test der Backupstrategie

Sind PIN und Backup-PIN eingerichtet und ist die Testseite über HTTPS abrufbar, so kann die Backupstrategie getestet werden. Dazu wird mit dem Backup-CSR ein Zertifikat beantragt. Für den hier geschilderten konkreten Fall wird dazu das Zertifikat bei CAcert widerrufen und mit dem Backup-CSR ein neues Zertifikat beantragt. Das neue Zertifikat wird unter dem Namen `test.my-it-brain.de.bak.crt` auf dem Server gespeichert. Es wird parallel zum alten Zertifikat abgelegt.

Um nun die Validierung durchzuführen, wird das neue Zertifikat in die NGINX-Konfiguration eingetragen und der PIN des alten Zertifikats gelöscht. Jetzt ist nur noch ein PIN in der Konfiguration hinterlegt. Dies verstößt zwar formal gegen den RFC 7469, da die Laborumgebung anschließend jedoch wieder abgerissen wird, geht dies in Ordnung. Die Konfiguration sollte nun wie folgt aussehen:

```
## BEGIN Testlabor CONFIGURATION ########################################
```

```
server {
listen 80;
listen [::]:80;
server_name test.my-it-brain.de;

# Path to the root of your installation
     root /var/www/test.my-it-brain.de/public;

}
server {
    # Listen on Port 443
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name test.my-it-brain.de;

    ssl_certificate /var/www/test.my-it-brain.de/ssl/ \
  test.my-it-brain.de.bak.crt;
    ssl_certificate_key /var/www/test.my-it-brain.de/ssl/ \
  test.my-it-brain.de.bak.key;

# Add headers to serve security related headers
add_header Public-Key-Pins 'pin-sha256="HIER STEHT DER BACKUP-PIN"; \
  max-age=3600';

    # Path to the root of your installation
    root /var/www/$host/public;

    error_page 403 /core/templates/403.php;
    error_page 404 /core/templates/404.php;
}
## END Testlabor CONFIGURATION ##########################################
```

Nach Neuladen der Konfiguration zeigt eine Prüfung, dass weiterhin auf die Webseite zugegriffen werden kann. Damit wurde die Backupstrategie erfolgreich validiert.

4.3 Gegenprobe

Ich gehe noch einen Schritt weiter und führe auch die Gegenprobe durch, indem ein Man-In-The-Middle-Angriff simuliert wird. Dazu erstelle ich einen weiteren privaten Schlüssel und einen neuen CSR, mit welchem ich ein weiteres Zertifikat auf den FQDN[6] `test.my-it-brain.de` ausstellen lasse. In der Theorie sollte der Webbrowser, welcher

[6] Fully Qualified Domain Name

den bzw. die PINs gespeichert hat, den Zugriff auf die Webseite verweigern, wenn dieses neue Zertifikat in die Seite eingebunden wird. Denn schließlich passt der im Webbrowser gespeicherte PIN nicht zu dem, der über das neue Zertifikat gebildet werden kann.

Die Erstellung des CSR für das „böse“ Zertifikat erfolgt mit folgenden Befehlen:

```
openssl genrsa -out trudy.key 2048
openssl req -new -key trudy.key -out trudy.csr
```

Das damit erzeugte „böse“ Zertifikat wird in die Laborumgebung eingebaut:

```
## BEGIN Testlabor CONFIGURATION ##########################################
server {
listen 80;
listen [::]:80;
server_name test.my-it-brain.de;

# Path to the root of your installation
     root /var/www/test.my-it-brain.de/public;

}
server {
    # Listen on Port 443
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name test.my-it-brain.de;

    ssl_certificate /var/www/test.my-it-brain.de/ssl/trudy.crt;
    ssl_certificate_key /var/www/test.my-it-brain.de/ssl/trudy.key;

    # Path to the root of your installation
    root /var/www/$host/public;

    error_page 403 /core/templates/403.php;
    error_page 404 /core/templates/404.php;
}
## END Testlabor CONFIGURATION ############################################
```

Nachdem die NGINX-Konfiguration neu geladen wurde, verweigert der Webbrowser einen erneuten Aufruf der Testseite mit dem Hinweis, dass der Server keine Zertifikatskette verwendet, die mit dem PIN-Set übereinstimmt.

Damit wurde erfolgreich gezeigt, dass der Webbrowser den Zugriff auf die Webseite verweigert, wenn die Zertifikatskette nicht zum „gepinnten“ Schlüssel passt.

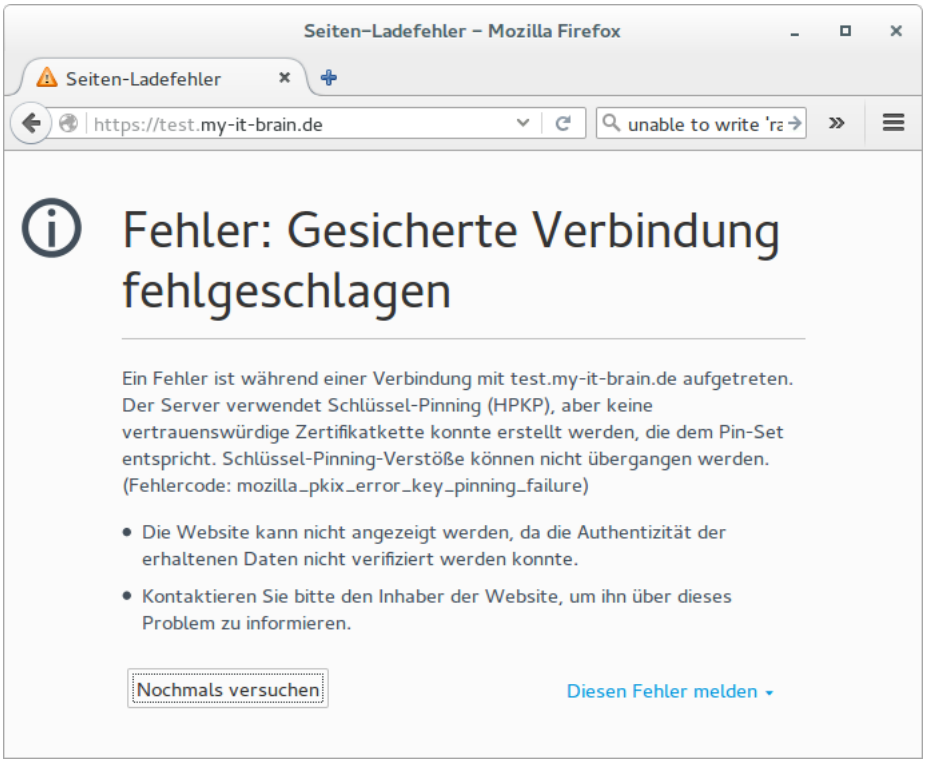

Abbildung 1: Key Pinning Failure

5 Fazit

Certificate Pinning ist ein im RFC 7469 spezifizierter Standard, welcher heute bereits von den weit verbreiteten Browsern Chrome und Firefox unterstützt wird. Auch die Konfiguration weit verbreiteter Webserver wie Apache, lighttpd und NGINX ist mit geringem Aufwand möglich, wie hier am Beispiel von NGINX gezeigt wurde.

Die sichere Aufbewahrung des Backup-CSR ist sicherzustellen, um eine längere Nichterreichbarkeit einer Domain vermeiden zu können. Denn mit diesem lässt sich im Bedarfsfall ein neues Zertifikat erzeugen und in die betroffene Webseite einbinden. Im Abschnitt 4 auf Seite 7 konnte die Backupstrategie für das Certificate Pinning validiert werden. Darüber hinaus konnte mit der Gegenprobe in Abschnitt 4.3 auf Seite 9 auch die generelle Funktionalität des Verfahrens nachgewiesen werden.

Damit ist Certificate Pinning grundsätzlich geeignet, die verschlüsselte Kommunikation im Internet noch sicherer zu machen.

Quellen

[1] C. Evans u. a. *Public Key Pinning Extension for HTTP (IETF RFC 7469)*. Internet Engineering Task Force (IETF). Apr. 2015. URL: `https://tools.ietf.org/html/rfc7469` (besucht am 15. 11. 2015).

[2] Jürgen Schmidt. „Festgenagelte Zertifikate: TLS wird sicherer durch Certificate Pinning“. In: *c't magazin für computer technik* 23 (2015), S. 118–121.

[3] Jürgen Schmidt. „Sicher mit Pin: Zertifikats-Pinning auf dem eigenen Server“. In: *c't magazin für computer technik* 23 (2015), S. 122–125.