Kryptographie Praktikum: SSL Notaries

Technische Analyse und Design

EINFÜHRUNG

SSL Notaries stellen Dienste dar, welche als dritte Partei neben dem Client und der CA selbst die Korrektheit und Gültigkeit eines Zertifikats bezeugen sollen. Hierbei existieren mehrere Dienste, deren Interfaces im Folgenden kurz beschrieben werden sollen. Die Beschreibung der Interfaces soll als Grundlage für die Implementierung innerhalb des Kryptographie Praktikums dienen und eine zentrale Anlaufstelle für technische Fragestellungen darstellen. Weiterhin sollen Gemeinsamkeiten der Ansätze herausgearbeitet werden, um innerhalb des Praktikums den Code möglichst einfach und wiederverwendbar zu halten.

Darüber hinaus werden Libraries genannt, welche bei der Entwicklung zum Einsatz kamen. Die Auswahl der Libraries erfolgt auf Basis der Beschreibungen der SSL Notaries. Es wird versucht, die Abhängigkeiten so gering wie möglich zu halten, sodass eine einfache Nutzung der Lösung ermöglicht wird.

Im letzten Kapitel soll das Design der Applikation anhand der Anforderungen abgeleitet und kurz skizziert werden. Hierzu erfolgt im ersten Teil die Nennung der Anforderungen und deren Auswirkungen und Abhängigkeiten zueinander. Im zweiten Teil wird auf Basis der Anforderungen ein Design entworfen und mittels UML visualisiert.

Diese Dokumentation erfüllt bewusst nicht mögliche Ansprüche an eine wissenschaftliche Arbeit. Sie dient lediglich als Entwicklungsdokumentation im Rahmen des Praktikums.

SSL NOTARY IMPLEMENTIERUNGEN

Die ICSI Certificate Notary

Die ICSI Certificate Notary sammelt passiv Informationen über vorhandene Zertifikate aus dem „upstream Traffic“ mehrere Internetkontenpunkte und speichert diese in einer zentralen Datenbank. Die Abfrage von Zertifikatsinformationen erfolgt über die DNS-Server des Projekts, wobei die angefragte Domain folgenden Aufbau haben muss:

```
<sha1>.notary.icsi.berkeley.edu
```

`<sha1>` stellt hierbei den Hashwert des Zertifikats dar, welcher durch den Browser oder durch die hier entwickelte Applikation vom zu testenden Zertifikat berechnet werden muss:

```
checkNotary(sha1(cert))
```

Je nach abgefragtem Resource Record existieren unterschiedliche Rückgabewerte:

1. Existiert die abgefragte Domain nicht in der ICSI Datenbank, so wird `NXDOMAIN` zurückgegeben. Diese Antwort ist unabhängig vom abgefragten Resource Record.
2. Existiert die abgefragte Domain in der ICSI-DB und wird ein A-RR abgefragt, dann existieren zwei Rückgabewerte:

A. Der Rückgabewert `127.0.0.1` gibt an, dass das Zertifikat in der Datenbank vorhanden ist

B. Der Rückgabewert `127.0.0.2` gibt an, dass das Zertifikat innerhalb des hierarchischen Vertrauensmodells auf ein RootCA-Zertifikat aus dem „Mozilla root store“ zurückgeführt werden kann.

3. Existiert die abgefragte Domain in der ICSI-DB und wird ein TXT-RR abgefragt, dann enthält die Antwort fünf Attribute, denen mittels dem „=“-Zeichen Integer-Werte zugewiesen:

   - `version` Version der Antwort, Format
   - `first_seen` Unix Timestamp, zu dem das Zertifikat zum ersten Mal gesehen wurde
   - `last_seen` Unix Timestamp, zu dem das Zertifikat das letzte Mal gesehen wurde
   - `times_seen` Anzahl an Tagen zwischen first_seen und last_seen, an dem das Zertifikat mindestens einmal gesehen wurde.
   - `validated` Boolean Wert, welcher den Validierungsstatus des Zertifikats angibt. Der Zeitpunkt der Validierung ist nicht bekannt!
   - Beispiel:

     ```
     version=1 first_seen=15387 last_seen=15646 times_seen=260 validated=1
     ```

Es besteht Kompatibilität zu Google Certificate Catalogue. Hierbei werden die Versions- und Validated-Attribute unterdrückt und die Attributsnamen inkl. der „=“-Zeichen entfernt.

Mit dem Ansatz und den Rückgabewerten existieren mehrere Probleme:

1. Selbst-signierte Zertifikate oder Zertifikate unbekannter CAs können nicht validiert werden und enthalten daher immer `validated=0`, unabhängig ihrer Gültigkeit.

2. Es findet keine Überprüfung des Sperr-Status statt. Daher kann man trotz `validated=1` nicht sagen, ob das Zertifikat gesperrt wurde oder noch gültig ist.

3. Es wird keine Prüfung der Zuordnung zwischen Host und Zertifikat durchgeführt oder in der Datenbank gespeichert. Um ein Zertifikat bekannt zu machen und später als möglicherweise legitim darzustellen kann somit beispielsweise ein Testserver, welcher mit dem Angreiferzertifikat ausgestattet ist, einmal täglich aufgerufen werden. Zum Zeitpunkt des Angriffs ist das Zertifikat bereits lange bekannt, sodass eine Unterscheidung zum korrekten Zertifikat möglicherweise nicht mehr möglich ist.

4. Die im TXT-RR gespeicherten Werte können nur in Verbindung zueinander betrachtet werden und besitzen für sich selbst keine Aussagekraft. times_seen muss hierbei immer in Verbindung mit first_seen und last_seen gesehen werden, first_seen darf nicht zu weit in der Vergangenheit und last_seen muss möglichst nahe am aktuellen Datum liegen. Weiterhin muss der Abstand zwischen first_seen und last_seen groß genug sein, um eine Aussage darüber treffen zu können, ob das Zertifikat regelmäßig im Netzwerkverkehr gesehen wird.

5. first_seen bzw. last_seen haben keine Beziehung zur eigentlichen Gültigkeit des Zertifikats. first_seen kann daher zwar einen „optimalen“ Zeitpunkt des ersten Auftretens (z.B. vor einem Jahr) enthalten, jedoch kann das Zertifikat zu diesem Zeitpunkt bereits 20 Jahre gültig sein.

Da die Entwickler der Notary selbst keinerlei Score-Berechnung durchführen, soll im Folgenden eine beispielhafte Berechnung entwickelt werden. Die Berechnung des Scores wird dabei mittels Pseudocode gezeigt. Als Maximum-Score können 50 Punkte erreicht werden. Der niedrigste Score beträgt 0. Pro Fall können weiterhin bis zu 10 Punkte "erworben" werden.

Im "nullten" Fall wird überprüft, ob die Domain überhaupt der ICSI-Notary bekannt ist. Ist sie dies nicht, so beträgt der Score 0.

```
result = 0;
if(NXDOMAIN) return result;
```

Der erste Fall behandelt den Rückgabewert einer Abfrage des A-RR. 127.0.0.2 stellt hierbei den optimalen Wert da, durch den 10 Punkte dem Endscore hinzuaddiert werden. Für 127.0.0.1 werden lediglich 5 Punkte hinzuaddiert.

```
if(ip = 127.0.0.1) result += 5;
else if(ip = 127.0.0.2) result += 10;
```

Der zweite Fall behandelt die `times_seen`-Eigenschaft des TXT-RRs. Hierbei wird angenommen, dass die Wahrscheinlich, dass das Zertifikat valide ist, proportional mit der `times_seen`-Zahl ansteigt. Im Optimalfall kann `times_seen=last_seen-first_seen` erreichen. Wenn das Zertifikat vorhanden ist, so wurde es mindestens einmal gesehen. Dies stellt hierbei das Minimum dar.

```
max_seen = last_seen - first_seen
min_seen = 1
// [min_seen,max_seen] –> [1,10]
result_a = 10*(times_seen - min_seen)/(max_seen - min_seen)
if(result_a<0)result+=0
else result+=result_a
```

Der dritte Fall betrachtet das Problem, dass der Zeitpunkt, zu dem ein Zertifikat das letzte Mal gesehen wurde, nicht zu weit in der Vergangenheit liegen darf, um weiterhin als aktuell gelten zu können. Veraltete Zertifikate bekommen daher einen niedrigeren Score, als Zertifikate, welche erst kürzlich gesehen wurden. Im folgenden wurde beispielhaft t-30 Tage als minimaler Wert als minimaler `last_seen`-Wert gewählt.

```
// last_seen should be near today: „near“ ~ [today-30,today]
```

```
// If result_b == 0 -> Certificate has not been seen for a long time
// and should be treated as possibly invalid.
max_seen = today
min_seen = today - 30days
// [min_seen,max_seen] —> [0,10]
result_b = 10*(last_seen - min_seen)/(max_seen - min_seen)
if(result_b<0) result += 0;
else result+=result_b
```

Der vierte Fall beschäftigt sich mit der Gültigkeit der Zertifikate. Dieser Wert wird implizit durch `first_seen` repräsentiert, da zu diesem Zeitpunkt das Zertifikat das erste Mal gesehen (und möglicherweise genutzt) wurde. Dieser Wert sollte nicht zu weit in der Zukunft liegen. Analog zur Gültigkeit, welche für SSL Zertifikate empfohlen wird, wurden für diese t-2 Jahre gewählt. Jedoch sollte das Zertifikat auch nicht zu neu sein. Daher wurde als Optimalwert t-1 Jahr gewählt.

```
// first_seen should not be too far away from today
//    .. but also not too near..: Maybe use best practice from
// certificate validity periods: 2 years
// If result_c == 0 -> Certificate is too old
// and could already be compromised.
max_seen = today
min_seen = today - 2 years
optimal_seen = today - 1 year
if(first_seen == optimal_seen) result += 10;
if(first_seen > optimal_seen) result += 10 - 10*(x - optimal_seen)
                                          /(max_seen - optimal_seen)
if(first_seen < optimal_seen) result += 10*(x - min_seen)/(optimal_seen
                                              - min_seen)
```

Im fünften und letzten Fall werden 10 Punkte addiert, falls `validated=1`.

```
if(validated=1) result += 10
else result += 0
```

Quellen:

- http://notary.icsi.berkeley.edu

Convergence

Die Grundlage von Convergence stellt das **Perspectives** Projekt dar, welches bereits als Browser Addon bereit steht, jedoch einige Nachteile aufweist, die im Rahmen von Convergence beseitigt wurden. Bei Perspectives wird - neben einem kontinuierlichen Monitoring von SSL-Sites - eine Verbindung sowohl vom Client, als auch von der Notary aufgebaut und die erhalten Zertifikate lediglich auf Clientseite verglichen. Der Notary wird das zuvor erhaltene Zertifikat dabei nicht mitgeteilt:

```
if(checkNotary(url)==“ok“)
      // Everything is fine for now…
```

Bei der Implementierung entstanden dadurch einige Probleme hinsichtlich

- Completeness: Das Browser-Addon untersucht nur die initiale Verbindung, jedoch nicht die innerhalb der Website aufgebauten Verbindungen, z.B. für Bilder, Stylesheets, usw…
- Privacy: Tracking von Zugriffen, wodurch Bewegungsprofile erstellt werden können.
- Responsiveness: Der sog. „Notary Lag“ bezeichnet die Zeit, in der ein Zertifikat von der Notary zwischengespeichert wird und dadurch die Antwort z.B. im Falle einer Zertifikatserneuerung, falsch sein kann.

Bei **Convergence** wurden unterschiedliche Lösungen für die o.g. Probleme entwickelt:

- Notary Lag: Dieses Problem wurde dadurch gelöst, dass das Zertifikat an die Anfrage an die Notar gebunden wird. Dadurch muss die Notary lediglich zum Server verbinden und erhält Match/Mismatch des Zertifikats und kann dies den Client mitteilen.

```
checkNotary(url, cert)
```

- Privacy: Local Caching:

```
if(!checkLocalCache(url, cert)      // Cache invalid or
                                    // cert not avail.
   && checkNotary(url, cert)==“ok“)
      addCache(url, cert);
```

- Privacy: Notary Bounce: Zentrale Notary, die weitere Notaries anfragen kann und dadurch die Zurückverfolgbarkeit eliminiert.
- Erweiterbarkeit: Über Convergence lassen sich alle weiteren Notaries prinzipiell anbinden und abfragen („Collective Trust“). Damit stellt Convergence genau das dar, was im Rahmen dieses Projekts entwickelt werden soll.

Es wird ein RESTful Protokoll implementiert, wobei JSON zur Repräsentation der Daten eingesetzt wird. Die Notaries sind auf `Port 443` und `4242` erreichbar. Port 80 kann weiterhin als HTTP Proxy für `CONNECT` Requests an `Port 4242` genutzt werden. Es sind zwei Anfragen möglich, die Antwort setzt sich aus JSON-String und HTTP-Response-Code zusammen:

1. Eine POST-Anfrage, welche den Fingerprint des Zertifikats enthält. Hierbei findet eine Überprüfung bereits auf auf der Convergence Notary statt.

```
POST /target/<host to validate>+<port>
fingerprint=<sha1(cert)> (in der Form "AA:BB:CC:…")
```

Als Antwort kommen unterschiedliche HTTP-Status-Codes zum Einsatz.

- 200: Die Notary konnte das Zertifikat verifzieren. Im Body wird ein JSON String nach unten gezeigtem Aufbau übertragen.
- 409: Die Notary konnte das Zertifikat nicht verifizieren. Im Body wird ein JSON String nach unten gezeigtem Aufbau übertragen, welcher die (potentiell) korrekten FPs enthält.
- 303: Verifikation des Fingerprints ist nicht möglich (z.B. wenn der Host aufgrund einer fehlenden Eigenschaft nicht abgefragt werden konnte).
- 400: Fehler bei der Anfrage bzw. fehlender Parameter. Im Body wird eine lesbare Fehlermeldung mitgegeben.
- 503: Interner Fehler bei der Notary. Fehlermeldung im Body. Verifikation sollte als fehlgeschlagen angesehen werden.

2. Eine GET-Anfrage an die selbe URL. Hierbei wird lediglich die Liste der Fingerprints zurückgegeben, es findet jedoch keine Validierung auf der Notary statt.

```
GET /target/<host to validate>+<port>
```

Aus den Rückgabewerten der unterschiedlichen Notaries kann der Client einen Wert fürs „Collective Trust“ bestimmen. Hierbei können unterschiedliche Methoden zum Einsatz kommen. Ein Rückgabewert besteht dabei aus dem HTTP Status Code und einem JSON String, welcher im Body mitgeliefert wird und folgende Form besitzt:

```
{"fingerprintList":
      [{
            "timestamp": {
                  "start": "1363209680",
                  "finish": "1363209680"
            },
            "fingerprint": "2C:BE:35:EF:B3:FB…"
      }, …],
      "signature": "Uby+MrIl/1973M7…"
}
```

Der JSON-String kann mehrere Fingerprints enthalten.

Der HTTP Status Code gibt genauere Informationen über das Ergebnis der Notary an und wird daher als Hauptantwort genutzt:

- 200: Die Notary konnte den im Post-Request gesendeten Fingerprint verifizieren. Im Body der Antwort wird eine Liste von der Notary bekannten Fingerprints mitgeliefert, die vom Client zum Caching genutzt werden können.
- 409: Die Notary konnte den im Post-Request gesendeten Fingerprint nicht verifizieren. Wie beim Status Code 200 auch, wird eine Liste der bekannten Fingerprints mitgeliefert.
- 303: Die Notary konnte den Fingerprint nicht überprüfen, weil der Host nicht der Spezifikation der Notary entsprochen hat. Dies kann z.B. der Fall sein, wenn die Notary Eigenschaften überprüft, welche auf dem Zielhost nicht vorhanden sind. Die Meinung dieser Notary wird bei der Berechnung des Endscores außen vor gelassen.
- 400: Der Request war fehlerhaft. Im Body wird eine lesbare Fehlermeldung ausgegeben.
- 503: Interner Fehler beim Bearbeiten der Anforderung.

Wie zu sehen ist spielt der JSON-String bei der Berechnung des Scores keine Rolle und es werden lediglich die HTTP Status Codes herangezogen. Die Berechnung eines Endergebnis erfolgt im Client nach Abfrage aller konfigurierten Notaries. Im ersten Schritt werden dazu die erfolgreichen Antworten der Notaries gezählt, die nicht den HTTP Status Code 303 zurückgegeben haben:

```
for (var i in checkNotaries) {
      […]
      if (notaryResponse == ConvergenceResponseStatus.VERIFICATION_SUCCESS) {
            successCount++;
      } else if (this.isStandAsideResponse(notaryResponse)) {
            checkedNotaryCount--;
      }

      checkedNotaryCount++;
}
[…]
var aggregateStatus = this.calculateAggregateStatus(successCount,
checkedNotaryCount);
```

[quelle: https://github.com/moxie0/Convergence/blob/master/client/chrome/content/ssl/ActiveNotaries.js, Zeile 87ff]

Im zweiten Schritt wird die Entscheidung getroffen, ob das Zertifikat als gültig (verifiziert) angesehen werden kann. Hierfür gibt es mehrere Ansätze, welche in folgender Reihenfolge überprüft werden:

1. Minderheitsentscheidung

```
if (this.isThresholdMinority() && (successCount > 0)) {
      return true;
}
```

2. Konsensentscheidung

```
if (successCount <= 0
        || (this.isThresholdConsensus()
        && (successCount < checkedNotaryCount))) {
        return false;
}
```

3. Mehrheitsentscheidung

```
var majority = Math.floor(checkedNotaryCount / 2);
if ((checkedNotaryCount % 2) != 0)
        majority++;
return (successCount >= majority);
```

[quelle: https://github.com/moxie0/Convergence/blob/master/client/chrome/content/ssl/ActiveNotaries.js, Zeile 172ff.]

Die Methode zur Entscheidungsfindung kann in der Konfiguration gesetzt werden und wird in der Variablen "verificationThreshold" gespeichert.

Es existieren mehrere Probleme bei diesem Ansatz. Zwei davon werden im Vortrag von Moxie Marlinspike selbst genannt:

- Citybank Problem: Es wurden mehrere Verbindungen zur Website der Citybank aufgebaut. Dabei wurden jedoch unterschiedliche Zertifikate zurückgegeben und nicht, wie erwartet, nur eins.
- Captive Portals: Steht kein Internetzugang zur Verfügung, so kann auch keine Überprüfung der Zertifikate durchgeführt werden.

Es ist leicht zu sehen, dass sich diese nur auf spezielle Umgebungen beziehen, in denen entweder keine Notary verfügbar ist oder welche abseits der Standards und Best Practices arbeiten.

Es existieren jedoch weitere Probleme mit diesem Ansatz:

- Es ist essentiell für die Aussagekraft des Ergebnisses, dass mehrere Notaries konfiguriert werden, da die Antwort einer Notary möglicherweise falsch sein kann.
- Das Protokoll nutzt selbst HTTPS zur Übertragung der Anfragen und Antworten. Daraus ergibt sich das Problem, dass das Zertifikat der Notary nicht automatisch werden kann, sondern die Validierung manuell vorgenommen werden muss.
- Wenn der abgefragte Host selbst falsche Zertifikate ausliefert, kann dies nicht erkannt werden, da sowohl die Notaries, als auch der Client die (falschen) Zertifikate sehen.
- Es ist nicht klar, welcher der Zertifikatshashes, die vom Convergence-Server zurückgeliefert werden, korrekt ist oder woher diese Hashes stammen. Dieses Thema wird darüber hinaus auch nicht in der Dokumentation angeschnitten. Hierdurch entsteht die Situation, dass der Convergence-Server möglicherweise bereits vor dem Erstkontakt mit dem Client falsche Zertifikate kennt. Das Zertifikat wird dem Client also schon beim

Erstkontakt vom Convergence-Server als gültig gemeldet, was weitere Mechanismen (z.B. Certificate Pinning) außer Kraft setzen kann.

Weiterhin existieren einige praktische Probleme:

- Der Code ist stark veraltet und wird nicht mehr aktiv gepflegt. Es gibt zwar einige Forks auf github, diese sind jedoch meist nicht mehr kompatibel zur ursprünglichen Version.
- Die Dokumentation behandelt nur die rudimentäre Funktionsweise und geht nicht auf Protokolldetails ein.
- Der Server kann lediglich als Proof-of-Concept bezeichnet werden. Während der Implementierung konnten über falsche Parameter Abstürze provoziert werden.
- Es gibt nur sehr wenige aktive, hochverfügbare Convergence-Server.

Quellen

- http://sarwiki.informatik.hu-berlin.de/Convergence
- https://github.com/moxie0/Convergence/wiki/Notary-Protocol

Crossbear

Crossbear ist ein auf Convergence aufbauendes Konzept der TU München zur Erkennung von MitM-Angriffen. Sowohl Server, als auch Clients sind über Github als OpenSource erhältlich. Die Datensammlung erfolgt analog zu Perspectives/Convergence über aktives Monitoring auf Basis von Client-Anfragen. Um weitere Aktionen durchzuführen, wurden sog. „Hunter“ entwickelt, welche Aufgaben bzgl. Erkennung übernehmen. Hierbei gibt es zwei Arten von Huntern:

- Add-on für Mozilla Firefox
- Eigener Dienst.

Hunter „jagen“ einen MitM und versuchen ihn mittels verschiedener Methoden zu identifizieren und zu lokalisieren. Diese Informationen werden zum zentralen Crossbear-Server gesendet und dort zur Analyse gespeichert:

1. Zertifikate der Verbindungen, welche entweder selbst hergestellt oder mitgeschnitten wurden. Dies ermöglicht die einfache Erkennung falscher Zertifikaten auf Basis bereits bekannter Zertifikate.
2. Traceroutes auf Basis von ICMP zum Ziel, um alle Hosts auf dem Weg zum Ziel in die Analyse einschliessen zu können. Dies ermöglicht die Lokalisierung eines möglichen Angreifers anhand der Schnittpunkt von „guten“ (es wurde ein gültiges Zertifikat geliefert) und „schlechten“ (es wurde ein falsches Zertifikat geliefert) Verbindungen.
3. Weitere Informationen werden über die Anbindung von auf Convergence basierenden Notaries (siehe https://github.com/crossbear/Crossbear/blob/master/server/fourhundredfourtythree/src/crossbear/convergence/ConvergenceConnector.java) gesammelt und zur Anreicherung der eigenen Daten genutzt, beispielsweise Häufigkeit oder last_seen-Werte (siehe Convergence-Kapitel).
4. Out-of-Band-Informationen: Genutzte CAs, WHOIS-Informationen, Geo-IP-Mapping

Zwischen Hunter/Client und Crossbear-Server wird immer eine `TLS`-Verbindung aufgebaut. In der aktuelle Implementierung ist das Zertifikat im Client hardcodiert und bei einem falschen Zertifikat wird die Verbindung automatisch unterbrochen.

Beim Verbindungsaufbau schickt der Client dem Server einen `CertVerifyRequest`, welcher die Zertitikatskette, den Hostnamen und die aufgelöste IP-Adresse enthält. Der Server speichert diese Anfragen zusammen mit dem Zeitpunkt der Anfrage und stellt ebenfalls eine Verbindung zum Server her, um das Zertifikat zur überprüfen. Gleichzeitig werden die Ergebnisse mit anderen Notaries überprüft und korreliert. Als Ergebnis sendet der Server eine `CertVerifiyResult` zurück an den Client. Optional kann diese Nachricht eine Anfrage an einen Traceroute enthalten, sofern der Verdacht überprüft werden soll, ob eine MitM vorliegt. Optional können ebenfalls eine `PublicIPNotification` und der `Timestamp` mitgesendet werden. Weiterhin erstellt der Crossbear-Server einen Score zwischen 0 (nicht vertrauenswürdig) und 255 (vollständig vertrauenswürdig) für das Zertifikat auf Basis einiger Eigenschaften des Zertifikats. Dieser wird dem Client ebenfalls mitgeteilt. Der Standardwert im Client, ab dem eine Warnung erfolgt, liegt bei 100.

Hunter holen sich vom zentralen Crossbear-Server Hunt-Aufträge ab oder bekommen diese in Form von `Traceroute`-Anfragen bei einer eigenen Anfrage zugewiesen. Der Hunt-Auftrag enthält den abzufragenden Host und eine ServerTime-Nachricht. Der Hunter baut eine Verbindung zum Host auf und speichert dabei sowohl `Traceroute`, als auch die Zertifikatskette und gibt dies an den Crossbear-Server zurück.s
Der Hunter muss einen `PublicIPRequest` durchführen.
Da Crossbear lediglich als Notary dienen soll, wird auf eine Implementierung der Hunter-Funktionalität verzichtet. Es wird im folgenden daher nur auf die Pakete `CertVerifyRequest` und `CertVerifyResult` eingegangen, sowie auf die Methode zur Ergebnisberechnung durch den Crossbear-Server. Auf die Vorgehensweise bei der Validierung von SSH-Fingerprints wurde ebenfalls verzichtet, da dies nicht im Fokus dieser Arbeit steht.

Ein `CertVerifyRequest` wird vom Client an den Server geschickt. Er enthält folgende Informationen:

- Hostname, IP und Port des abgefragten Servers
- Optionen: Hier wurde bisher lediglich die Angebe implementiert, ob sich der Client hinter einem SSL-Proxy verbirgt.
- Die vom Server zurückgelieferte Zertifikatskette in base64-Kodierung.

Der Request, welcher aus den Verkettung der o.g. Parameter entsteht, wird über eine https-Verbindung (jedoch ohne Nutzung des HTTP-Protokolls) an einen Server des Projekts (https://crossbear-host/certVerify.jsp) übergeben. Das Zertifikat des Servers wird dazu im Client hardcodiert, da sonst eine Überprüfung nicht möglich wäre. Inzwischen (Stand: März 2014) wurde auch Certificate Pinning am Server implementiert. Zum Status der Implementierung im Client liegen noch keine weiteren Informationen vor.

Als Antwort erwartet der Client vom Server eine `CertVerifyResult`-Nachricht. Diese enthält sowohl einen Wert zwischen 0 und 255 (erstes Byte), als auch eine Begründung, die Aufschluss darüber geben können, wie dieser Wert zustande gekommen ist.

Folgende Eigenschaften des Zertifikats fliessen dabei in die Berechnung des Wertes mit ein:

1. Hat der Server dasselbe Zertifikat erhalten, das auch der Client erhalten hat?
   - Server hat kein Zertifikat erhalten: -100
   - Erhaltene Zertifikate sind gleich: 80
   - Erhaltene Zertifikate sind nicht gleich: 0
2. Wann hat der Server das Zertifikat das letzte Mal gesehen?
   - Die Bewertung erfolgt nach folgendem Pseudocode:

```
int observationdays = Anzahl Tage zwischen erster und letzter Sichtung
if(war letzte Sichtung erst vor kurzem?) {
        int rating = observationdays / 3 * 2;
}else{
        int rating = observationdays /3;
}
```

3. Wie oft wurde das Zertifikat vom Server gesehen?
   - Die Bewertung erfolgt nach folgendem Pseudocode:

```
long nomOfObservations = Anzahl von Sichtungen
int rating = (int)(nomOfObservations/30);
```

4. In welchen Zeitabständen wurde das Zertifikat von Convergence gesehen?
   - Die Berechnung erfolgt analog zu Punkt 2, wobei die Werte vom Convergence-Server genutzt werden.
5. Wurde das Zertifikat für den abgefragten Host ausgestellt?
   - Das Zertifikat wurde für den abgefragten Host ausgestellt: 50
   - Das Zertifikat wurde nicht für den abgefragten Host ausgestellt: -70
6. Ist das Zertifikat zum aktuelle Zeitpunkt gültig?
   - Valide: 20
   - Nicht Valide: -20
7. Werden die aktuell genutzten Algorithmen im Zertifikat noch als sicher betrachtet?
   - Werden veraltete Algorithmen genutzt: -60
   - Werden aktuelle Algorithmen genutzt: 0
8. Wird die aktuelle Schlüssellänge noch als sicher betrachtet?
   - Folgender Java-Code wird genutzt:

```
int keylengthRating = (keylength - 2048 <0)
        ? ((2048 - keylength)*(keylength - 2048))/30000
        : (keylength - 2048)/100;
```

[Quelle: https://github.com/crossbear/Crossbear/blob/master/server/fourhundredfourtythree/src/crossbear/CVRProcessor.java; Zeile 698ff.]

Jeder dieser Fragen ergibt einen Wert, der sowohl negativ, als auch positiv sein kann. Die Berechnung des Endwertes erfolgt folgendermaßen:

```
int rating = 0;
Iterator<CertJudgment> itr = judgments.iterator();
while(itr.hasNext()){
      rating += itr.next().getRating();
}
if(rating<0) return 0;
if(rating>255) return 255;
```

[Quelle: https://github.com/crossbear/Crossbear/blob/master/server/fourhundredfourtythree/src/crossbear/messaging/CertVerifyResult.java; Zeile: 70ff.]

Es werden somit alle Bewertungen addiert, woraus sich schlussendlich der Score ergibt.

Da zum Zeitpunkt der Implementierung kein Crossbear-Server zur Verfügung stand, konnte die Entwicklung dieses Notary-Connectors nicht beendet werden.

Quellen:

- https://pki.net.in.tum.de/
- https://github.com/crossbear/Crossbear/tree/master/server/fourhundredfourtythree/src/crossbear
- https://github.com/crossbear/Crossbear/blob/master/ffplugin/chrome/content/CBMessages.js

SSLObservatory

Das SSLObservatory war ein Projekt der Electronic Freedom Frontier (EFF), bei dem das gesamte Internet gescannt und alle gefundenen Zertifikate inkl. deren Adressen gespeichert wurden. Das Projekt fand Im Jahr 2010 statt und weist dementsprechend schon ein vergleichsweise hohes Alter auf.

Das SSLObservatory stellt keine Notary im klassischen Sinne dar. Stattdessen ist es lediglich möglich, ein Zertifikat der Datenbank hinzuzufügen. Dies ist über einen POST-Request an https://observatory.eff.org/submit_cert möglich, wobei folgende Parameter mitgeliefert werden müssen:

- `domain` - Die abgefragte Domain.
- `server_ip` - Die IP des abgefragten Hosts
- `certlist` - Eine Liste von Zertifikaten in der Form `{ base64(Zertifikat_1), base64(Zertifikat_2), …}`
- `fplist` - Eine Liste von Fingerprints der mitgelieferten Zertifikate. Die Fingerprints haben die Form md5fingerpriint+sha1fingerprint (siehe https://github.com/EFForg/https-everywhere/blob/master/src/components/ssl-observatory.js, Zeile 314). Die Fingerprints könnten von Server auch selbst berechnen werden. Daher ist dieser Parameter optional.

- `client_asn` - In diesem Parameter kann die Nummer des autonomen Netzwerks mitgeliefert werden, in dem sich der Client befindet.
- `private_opt_in` - Bestimmt, ob der Client der Speicherung der Anfrage bei nicht-öffentlichen Domainnamen zustimmt (1 -> Zustimmung; 0 -> Ablehnung).
- `padding` - Zufallszahlen, mit denen der POST-Request gepadded wird.

Das Resultat der Aktion wird in Form eines HTTP Status Codes und einer 1 bzw. 0 im Body zurückgeliefert und kann dazu genutzt werden, die Gültigkeit des Zertifikats zu überprüfen:

- HTTP-Status-Code `200`
  - Im Body steht eine "1": Der SHA256-Hash des Zertifikats war nicht in der Tabelle "certs" des Projekts vorhanden und das Zertifikat wurde nun hinzugefügt.
  - Im Body steht eine "0": Das Zertifikat wurde nicht hinzugefügt.
- HTTP-Status-Code `403`
  - fplist enthält ein Zertifikat, welches vom Observatory als gefährlich (z.B. widerrufen, "broken key") eingestuft wird. Im Body wird nun eine Fehlerbeschreibung zurückgeliefert.

Bei der Entwicklung eines Moduls ist aufgefallen, dass die Notary selbst bei falschen Zertifikaten den HTTP-Status 200 zurück liefert. Dies lässt darauf schliessen, dass Zertifikate nur manuell als ungültig bzw. gefährlich "markiert" werden müssen. Somit ist eine Live-Erkennung von MitM-Angriffen nicht möglich.

Für das SSL Observatory wurde der Konfigurationsparameter "countable" eingeführt. Dieser gibt an, ob das Ergebnis einer Berechnung mit in die Berechnung eines Endscores einbezogen werden soll. Dieses ist aufgrund o.g. Einschränkung bei der SSLObservatory abgeschaltet.

Quellen:
- https://trac.torproject.org/projects/tor/wiki/doc/HTTPSEverywhere/SSLObservatorySubmission
- https://github.com/EFForg/https-everywhere/blob/master/src/components/ssl-observatory.js

BESCHREIBUNG DER IMPLEMENTIERUNG

Architektur

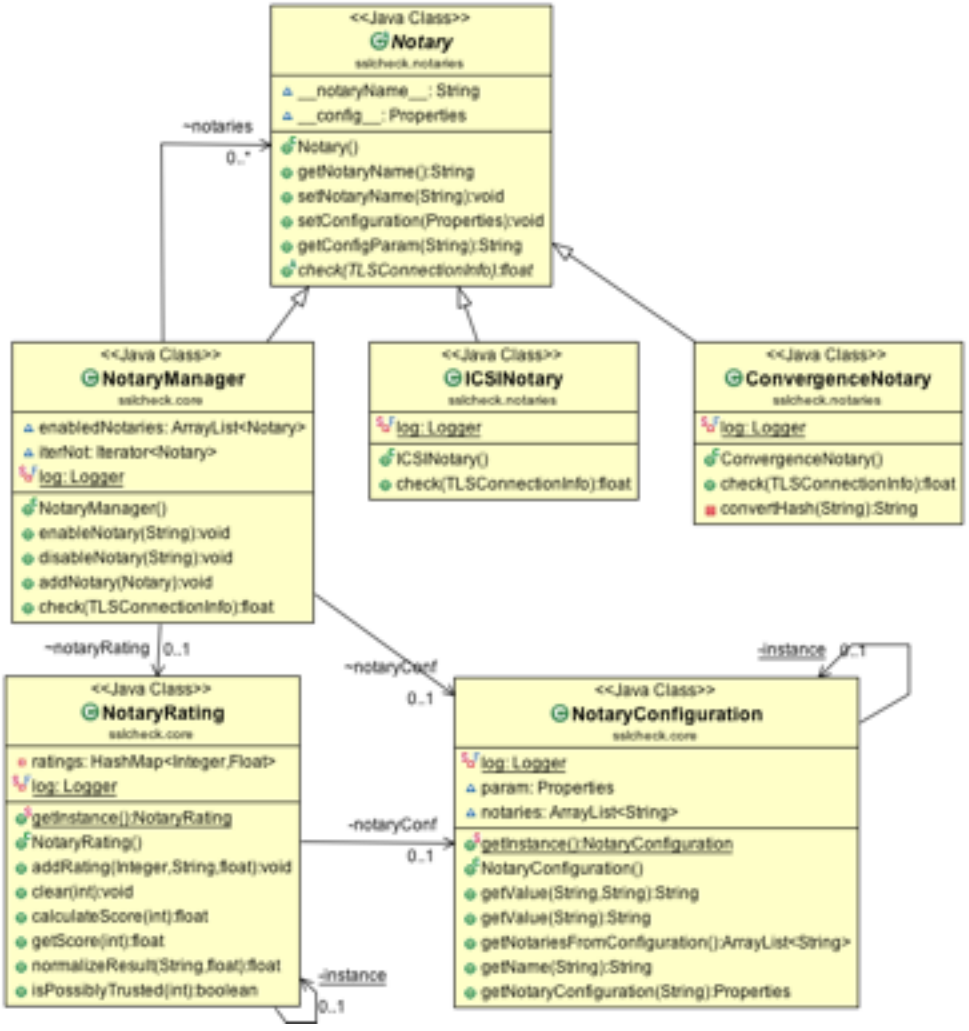
Die Applikation wird getreu den Anforderungen in einer Client/Server-Architektur realisiert. Dadurch teilt sich die Applikation in drei Teile:

1. Den Core, welcher die Logik zur Anbindung der Notaries und der Berechnung eines Scores enthält. Der Core steht lediglich dem Server zur Verfügung. Die Steuerung erfolgt mit Hilfe einer Konfigurationsdatei.
2. Den Client, welcher ein Zertifikat auf seine Gültigkeit hin untersuchen will. Der Client teilt dem Server hier mit, welche URL er überprüft haben möchte, welches SSL Zertifikat er möglicherweise bereits erhalten hat und welche Notaries zu dieser Überprüfung genutzt werden sollen. Die Anfrage wird weiterhin noch mit

einer Signatur über die abgefragten Informationen und das Ergebnis vor Veränderung geschützt. Der Client wird im Rahmen der Veranstaltung von einer weiteren Gruppe in Form eines Firefox-Addons entwickelt.

3. Der Server, welcher den Core anbindet und nach außen eine Schnittstelle zur Abfrage von Zertifikaten anbietet. Der Server kann hierbei beliebige Interfaces nach außen bereit stellen und so potentiell beliebige Clients anbinden. Das Ergebnis einer Anfrage stellt hierbei ein float-Wert dar. Dieser Wert ist liegt in einem festgelegten Bereich und wird vom Core berechnet.

Erweiterbarkeit um neue Notaries spielt eine große Rolle. Aus diesem Grund wurde besonderen Wert auf den Core gelegt, welcher im folgenden beschrieben werden soll.



Core-Architektur

Der Kern der Applikation wird durch den NotaryManager gebildet. Dieser bildet durch ein abgewandeltes Proxy-Entwurfsmuster eine Abstraktionsebene zu den konkreten Notaries und steuert den Zugriff auf diese.

Ebenfalls wird durch ihn die Komposition der Ergebnisse und die Berechnung eines Endergebnisses mit Hilfe der NotaryRating-Klasse angestoßen.

Unterstützt wird der NotaryManager durch die Singletons NotaryConfiguration und NotaryRating, welche die Konfiguration bzw. die Berechnungslogik der Bewertung bereitstellen. Sie werden vom NotaryManager zur Anbindung der Notaries und Verarbeitung derer Ergebnisse benötigt.

Die Implementation der Notaries erfolgt für jede Notary in ihrer eigenen Klasse. Für den Aufruf der Methode check() der Notary-Klasse ist der NotaryManager verantwortlich. Der Server greift wie bereits beschrieben dadurch nicht direkt auf die Notaries zu, sondern überlässt diese Aufgabe dem NotaryManager, wodurch eine einfache Erweiterbarkeit erreicht wird.

Diese Architektur ermöglicht so dem Entwickler eines Server die einfache Integration, da er sich lediglich um die Anbindung einer Notary (d.h. dem NotaryManager) kümmern muss. Nichtsdestotrotz können Aufgaben des NotaryManagers auch vom Server selbst übernommen werden. Dies kann z.B. die Auswahl, die Konfiguration oder die Berechnung eines Scores betreffen.

Notaries können hinzugefügt werden, in dem die Logik in einer Klasse implementiert wird und die Eigenschaften in der zentralen Konfigurationsdatei hinzugefügt werden. Eine Änderung am bestehenden Code ist hierbei nicht nötig. Die Notary bekommt vom NotaryManager einen Namen zugewiesen (siehe Konfigurationsdatei), sowie ihre eigene Konfiguration übergeben. Auf Basis dieser Informationen kann sich ein Entwickler vollständig auf die Funktionalität der Notary konzentrieren.

Anbindung

Ein implementierender Server baut auf dem Core auf und soll dessen Funktionalität einfach dem Client zur Verfügung stellen können. Hierzu bindet eine Serverimplementierung die Klasse sslcheck.core.NotaryManager an, da diese die volle Funktionalität zur Verfügung stellt und hierbei als Adapter zum Zugriff auf die Notaries, die Konfiguration und die Berechnungen dient.

Eine Beispielimplementation ist in der Klasse "sslcheck.test.SimpleServerPreconfiguredCertificates" gegeben.

Hinzufügen neuer Notaries

Eine neue Notary wird als eigene Klasse, welche von der abstrakten Klasse `sslcheck.notaries.Notary` erbt, implementiert. Im Rahmen dieser Klasse muss die Methode `float check(TLSConnectionInfo)` überschrieben und somit implementiert werden. In dieser Methode muss die Überprüfung der Zertifikate auf Basis der Informationen im TLSConnectioninfo-Objekt (d.h. der aufgerufene Host inkl. Port und die zurückgelieferten Zertifikate) implementiert werden.

Zur Unterstützung stehen mehrere Methoden zur Verfügung:

- `sslcheck.notaries.Notary.getParam(String)` - Zugriff auf Konfigurationsparameter aus der `notaries.properties`-Datei. Es besteht jedoch nur Zugriff auf die eigenen Konfigurationsparameter. Zugriff auf alle Parameter kann über folgende Codezeile erlangt werden.

```
NotaryConfiguration nc = NotaryConfiguration.getInstance();
```

- `sslcheck.notaries.Notary.setTrustManager(X509TrustManager)` - Setzen eines TrustManager-Objekts bei Zugriffen auf SSL-geschützte URLs. Dies wird üblicherweise innerhalb der nächsten Methode erledigt, welche automatisch beim Start der Notary aufgerufen wird.
- `sslcheck.notaries.Notary.initialize()` - Diese Methode kann überschrieben werden und wird beim Start der Notary aufgerufen. Ihr stehen alle Konfigurationsparameter zur Verfügung. Wie eben beschrieben kann hier der TrustManager gesetzt werden.
- Logging: Das Logging im Core wird mittels Log4j realisiert. Es ist daher auch möglich, Log4j auch in den Notaries einzusetzen. Log4j kann beispielsweise folgendermaßen integriert werden.

```
private final static Logger
      log = LogManager.getLogger("notaries.NotaryName");
```

- Die Methode `float check(TLSConnectionInfo)` kann bei internen Fehler eine `NotaryException` werfen. Die Notary wird dann von der Berechnung eines Ratings ausgenommen.

Konfiguration

Für die zentrale Konfiguration existiert die Konfigurationsdatei notaries.properties. Hier können notary-spezifische Parameter angepasst und gesetzt werden. Weiterhin existieren einige grundlegende Einstellungen vorgenommen werden. Diese grundlegenden Einstellungen beginnen mit "DefaultNotary".

- `DefaultNotary.{ratingFactor, minRating, maxRating}`: Parameter für die Berechnung des End-Ratings. RatingFactor wird dabei mit dem Ergebnis einer Notary multipliziert, um ihr Gewicht in der gesamten Berechnung zu bestimmen. min- bzw. maxRating geben die Grenzen an, nach denen ein Ergebnis normalisiert wird.
- `DefaultNotary.{enabled, countable}`: Parameter zur Einbeziehung bzw. zum Ausschluss von Notaries in Ausführung bzw. Ergebnisberechnung ("countable").
- `DefaultNotary.{trustMode, trustLimit}`: Der TrustMode bestimmt den Modus der Entscheidungsfindung für die Frage, ob ein Zertifikat vertrauenswürdig ist oder nicht. Hierbei gibt es drei Modi: minority (es gibt eine Notary, die das Zertifikat als gültig validiert hat), majority (die Mehrheit der Notaries hält das Zertifikat für gültig) oder consensus (Alle Notaries halten das Zertifikat für valide). Zur Berechnung der Gültigkeit wird der Parameter trustLimit hinzugezogen. Ein Zertifikat wird dann als gültig betrachtet, wenn das Ergebnis der Notary größer ist, als trustLimit * maxRating für diese Notary.

LIBRARIES UND TOOLS FÜR DIE REALISIERUNG

Libraries

Da die Notaries unterschiedliche Interfaces aufweisen, wurden bei der Entwicklung der Notaryklassen unterschiedliche Libraries zur Implementation genutzt.

- ICSINotary: dnsjava (http://www.dnsjava.org/)

- ConvergenceNotary: Jersey
- CrossbearNotary: Guava (Google Common Libraries)

Um Logging zu ermöglichen wurde eine Minimalkonfiguration der Library log4j (https://logging.apache.org/log4j/2.x/) integriert. Sie dient sowohl in den Notaries, als auch im Core dazu, Info, Trace, Debug und Fehlermeldungen in stdout auszugeben. Ein Dateilogging findet nicht statt, kann aber in der Datei log4j.properties konfiguriert werden.

Weiterhin wurde für das Testing JUnit verwendet.

Tools

Als Buildtool wird Maven3 (https://maven.apache.org/ref/3.0/) mit einer zu Maven2 kompatiblen pom.xml eingesetzt. Das Ausführen von

```
~$ mvn3 install
```

führt dazu, dass alle benötigten Abhängigkeiten aus den Maven-Repositories heruntergeladen werden und mehrere JAR-Dateien (mit und ohne Abhängigkeiten) im Ordner target/ erstellt werden. Diese JAR-Dateien können nun direkt in die Clients eingebunden werden. Alternativ ist das Ausführen eines Testfalls mittels

```
~$ java -jar sslcheck.jar
```

möglich. Dieser Server muss in der pom.xml in Zeile 34 und 58 angegeben werden.

Als Entwicklungsumgebung wurde Eclipse (https://www.eclipse.org/) mit Maven-Plugin eingesetzt.

ANFORDERUNGEN AN DAS DESIGN

Nicht-funktionale Anforderungen

- Leichte Erweiterbarkeit, modularer Aufbau
- Programmiersprache: Java

Funktionale Anforderungen

- Abfrage von Zertifikatsgültigkeitsinformationen mittels unterschiedlicher Interfaces
- Server/Client-Design, Client only Console Application
- Auswahl der abzufragenden SSL Notaries und Module beim Aufruf
- Rückgabe einer Bewertung der Antworten und Berechnung eines End-Scores

SPEZIFIKATION DER ANGEBOTENEN SCHNITTSTELLEN

Auswahl der Design Patterns

- Anbindung einzelner Module zur Abfrage der Notaries: Proxy, Factory Method
- Abfrage aller Module und Rückgabe der Werte: Iterator
- Anbindung Core<>Server: Adapter, Singleton

AUSBLICK

Probleme

Wie auch bei der ConvergenceNotary besteht das Problem, dass ein Ergebnis nur dann überhaupt aussagekräftig ist, wenn mindestens zwei Notaries eingesetzt werden.

Es existiert aktuell keine dynamische (d.h. zertifikatsabhängige) Anpassung der Parameter bei der Bewertung. Dies ist jedoch nötig, um auf spezielle Policies reagieren zu können. Als Beispiel kann hierbei das Zertifikat von Google betrachtet werden. Dieses ist max. 3 Monate gültig und wird nach 2 Monaten erneuert. Durch die kurze Sichtbarkeitsspanne fällt die Bewertung somit recht schlecht aus.

Steigerung der Sicherheit

Die Korrektheit und Integrität der Antworten kann hierbei noch weiter gesteigert werden, indem statt dem Namen einer Notary die Signatur über die Notary in die Berechnung des Hashs einbezogen wird. Hierdurch wird die Integrität der Implementierung der Notary-Klasse bzw. je nach Anbindung auch der Notary selbst in die Antwort mit einbezogen und kann leicht überprüft werden.