Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem

Zane Ma

Postdoc Researcher
Georgia Institute of Technology
zanema@gatech.edu
https://zanema.com
TLS + Web PKI

Certificate Authority

Web Browser
Email Client

Web Server
Email Server
Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem

**TLS + Web PKI**

1. Clients Establish Trust in CAs

   - Certificate Authority
   - CA Certificate

   - Root store inclusion

- Web Browser
- Email Client

- Web Server
- Email Server
1. Clients Establish Trust in CAs

2. CAs Verify Server Identity

Certificate Authority

CA Certificate

Leaf Certificate

Web Server

Email Server

CA Certificates

Root store inclusion

Web Browser

Email Client

Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem • Zane Ma
TLS + Web PKI

1. Clients Establish Trust in CAs

2. CAs Verify Server Identity

3. TLS: Servers Send Proof of Identity
Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem

1. Clients Establish Trust in CAs
   - Root store inclusion
   - CA Certificate

2. CAs Verify Server Identity
   - CA Certificate
   - Signs
   - Leaf Certificate

3. TLS: Servers Send Proof of Identity
   - Leaf Certificate
   - Web Server
   - Email Server
   - Web Browser
   - Email Client
TLS user agents
TLS user agents
TLS user agents
TLS user agents
TLS user agents
TLS user agents

4 different root stores!
Research Questions

1. Which root store providers do TLS user agents rely on?

2. How do root store providers determine which CAs to trust?

3. Characterization of root store programs

4. How faithfully do providers copy root program trust?
Research Questions

1. Which root store providers do TLS user agents rely on?
2. How do root store providers determine which CAs to trust?
3. Characterization of root store programs
4. How faithfully do providers copy root program trust?
Data collection

Collected root stores for 77% of global CDN top 200 user agents
### Data collection

<table>
<thead>
<tr>
<th>Web Browsers</th>
<th>Other TLS Clients / Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>OpenSSL</td>
</tr>
<tr>
<td>Chrome Mobile</td>
<td>GnuTLS</td>
</tr>
<tr>
<td>Opera</td>
<td>BoringSSL</td>
</tr>
<tr>
<td>Firefox</td>
<td>Mbed TLS</td>
</tr>
<tr>
<td>Safari</td>
<td>curl</td>
</tr>
<tr>
<td>Mobile Safari</td>
<td>wget</td>
</tr>
<tr>
<td>Edge</td>
<td>okhttp</td>
</tr>
<tr>
<td>IE</td>
<td>LibreSSL</td>
</tr>
<tr>
<td>Chromium</td>
<td>+10 more</td>
</tr>
</tbody>
</table>

Collected root stores for 77% of global CDN top 200 user agents

Determined default root store for dozens of libraries / TLS clients
Root store providers

Web Browsers
- Chrome
- Chrome Mobile
- Opera
- Firefox
- Safari
- Mobile Safari
- Edge
- IE
- Chromium

OS
- Windows
- macOS
- Alpine
- iOS
- Android
- Ubuntu
- Debian
- Fedora
- Amazon Linux

User Agents

Other TLS Clients / Libraries
- OpenSSL
- GnuTLS
- BoringSSL
- Mbed TLS
- curl
- wget
- okhttp
- LibreSSL
- +10 more

Libraries / Frameworks
- NSS
- Electron
- NodeJS
- Java

Default / configured
Research Questions

1. Which root store providers do TLS user agents rely on?
2. How do root store providers determine which CAs to trust?
3. Characterization of root store programs
4. How faithfully do providers copy root program trust?
Clustering providers

![Metric MDS Plot]

- **NSS**
- **Android**
- **Alpine**
- **Debian**
- **Java**
- **Apple**
- **Microsoft**
- **Ubuntu**
- **AmazonLinux**
- **NodeJS**

The plot illustrates the clustering of various software providers based on a metric known as MDS (Multi-Dimensional Scaling). The date range from 2011 to 2021 is color-coded, with earlier years shown in warmer colors and later years in cooler colors, indicating changes over time in the clustering behavior of these providers.
Clustering providers

- Microsoft
- NSS+others
- Java
- Apple

Metric MDS

Date

2011

2021
Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem

Root store providers

User Agents
- Chrome
- Chrome Mobile
- Opera
- Firefox
- Safari
- Mobile Safari
- Edge
- IE
- Chromium

OS
- Windows
- macOS
- Alpine
- iOS
- Android
- Ubuntu
- Debian
- Fedora
- Amazon Linux

Default / configured

Web Browsers

Other TLS Clients / Libraries
- OpenSSL
- GnuTLS
- BoringSSL
- Mbed TLS
- curl
- wget
- okhttp
- LibreSSL
- +10 more

Libraries / Frameworks
- NSS
- Electron
- NodeJS
- Java

+10 more
Root store programs

Web Browsers
- Chrome
- Chrome Mobile
- Opera
- Firefox
- Safari
- Mobile Safari
- Edge
- IE
- Chromium

Other TLS Clients / Libraries
- OpenSSL
- GnuTLS
- BoringSSL
- Mbed TLS
- curl
- wget
- okhttp
- LibreSSL
- +10 more

OS
- Windows
- macOS
- Alpine
- iOS
- Android
- Ubuntu
- Debian
- Fedora
- Amazon Linux
- NodeJS
- Java

Default / configured

User Agents

Root Store Providers

Root Store Programs

Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem • Zane Ma
Research Questions

1. Which root store providers do TLS user agents rely on?

2. How do root store providers determine which CAs to trust?

3. Characterization of root store programs

4. How faithfully do providers copy root program trust?
Root program comparison

- **DigiNotar**
- **WoSign/StartCom**
- **Certinomis**
- **CNNIC**

Legend:
- ♣ NSS
- ■ Apple
- + Microsoft
- ♦ Java

Timeline: 2012 to 2021
Root program comparison

- **NSS**
- **Apple**
- **Microsoft**
- **Java**

- **DigiNotar**
- **WoSign/StartCom**
- **Certinomis**
- **CNNIC**

Still trusted

Root program comparison

- NSS
- Apple
- Microsoft
- Java

DigiNotar

WoSign/StartCom

Certinomis

CNNIC

1024-bit RSA

MD5 Signatures

Still trusted

Root program comparison

- **NSS**
- **Apple**
- **Microsoft**
- **Java**

<table>
<thead>
<tr>
<th>Year</th>
<th>DigiNotar</th>
<th>WoSign/StartCom</th>
<th>Certinomis</th>
<th>CNNIC</th>
<th>1024-bit RSA</th>
<th>MD5 Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Root program comparison

1. Mozilla responds quickly to CA distrust incidents; Microsoft relatively slow, Apple varies.

2. Apple/Mozilla operate relatively hygienic root stores.
Root program comparison

1. Mozilla responds quickly to CA distrust incidents; Microsoft relatively slow, Apple varies.

2. Apple/Mozilla operate relatively hygienic root stores.

3. Size: Mozilla < Apple < Microsoft; Mozilla most restrictive, Microsoft allows government super-CAs.
Root program comparison

1. Mozilla responds quickly to CA distrust incidents; Microsoft relatively slow, Apple varies.

2. Apple/Mozilla operate relatively hygienic root stores.

3. Size: Mozilla < Apple < Microsoft; Mozilla most restrictive, Microsoft allows government super-CAs.

4. Mozilla runs the most transparent root store program.
Mozilla derivatives
Research Questions

1. Which root store providers do TLS user agents rely on?

2. How do root store providers determine which CAs to trust?

3. Characterization of root store programs

4. How faithfully do providers copy root program trust?
Derivative delay
Derivative delay

Date

NSS

NodeJS

NSS version
Derivative delay
## Derivative delay

![Graph showing derivative delay over time](image)

<table>
<thead>
<tr>
<th>NSS</th>
<th>Debian/Ubuntu (1.96 versions behind)</th>
<th>Alpine (2.18 versions behind)</th>
<th>Node.js (2.46 versions behind)</th>
<th>Android (3.39 versions behind)</th>
<th>AmazonLinux (4.94 versions behind)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>166v1</td>
<td>166v1</td>
<td>166v1</td>
<td>166v1</td>
<td>166v1</td>
</tr>
<tr>
<td>2016</td>
<td>166v2</td>
<td>166v2</td>
<td>166v2</td>
<td>166v2</td>
<td>166v2</td>
</tr>
<tr>
<td>2017</td>
<td>166v3</td>
<td>166v3</td>
<td>166v3</td>
<td>166v3</td>
<td>166v3</td>
</tr>
<tr>
<td>2018</td>
<td>166v4</td>
<td>166v4</td>
<td>166v4</td>
<td>166v4</td>
<td>166v4</td>
</tr>
<tr>
<td>2019</td>
<td>166v5</td>
<td>166v5</td>
<td>166v5</td>
<td>166v5</td>
<td>166v5</td>
</tr>
<tr>
<td>2020</td>
<td>166v6</td>
<td>166v6</td>
<td>166v6</td>
<td>166v6</td>
<td>166v6</td>
</tr>
<tr>
<td>2021</td>
<td>166v7</td>
<td>166v7</td>
<td>166v7</td>
<td>166v7</td>
<td>166v7</td>
</tr>
</tbody>
</table>

**Date**

- 2015
- 2016
- 2017
- 2018
- 2019
- 2020
- 2021

**NSS version**

- 3.16.3
- 3.16.4
- 3.17.3
- 3.18
- 3.18.1
- 3.19.3
- 3.21
- 3.22.2
- 3.25
- 3.26
- 3.27
- 3.28.1
- 3.30.2
- 3.32
- 3.34
- 3.35
- 3.37
- 3.39
- 3.40
- 3.41
- 3.43
- 3.45
- 3.46
- 3.48
- 3.53
- 3.54
- 3.57
- 3.59
- 3.60
- 3.63
- 3.63.1
- 3.64
Derivative delay

- NSS
- Debian/Ubuntu (1.96 versions behind)
- Alpine (2.18 versions behind)
- NodeJS (2.46 versions behind)
- Android (3.39 versions behind)
- AmazonLinux (4.94 versions behind)
Trust deviations

- Trust purpose conflation: trusting non-TLS certificates
- Partial trust incapability: Symantec distrust dilemma
- Non-NSS trusted CAs: questionable trust
- App. developer confusion: trusting CAs for code signing, timestamping
Summary

Popular TLS user agents infrequently make their own TLS trust decisions and rely on the OS.

Apple, Microsoft run major root programs; all other root providers originate from Mozilla’s NSS root program.
Summary

Popular TLS user agents infrequently make their own TLS trust decisions and rely on the OS.

Apple, Microsoft run major root programs; all other root providers originate from Mozilla’s NSS root program.

NSS derivatives copy poorly: delayed updates, questionable bespoke trust, incompatible trust scope.
Summary

Popular TLS user agents infrequently make their own TLS trust decisions and rely on the OS.

Apple, Microsoft run major root programs; all other root providers originate from Mozilla’s NSS root program.

NSS derivatives copy poorly: delayed updates, questionable bespoke trust, incompatible trust scope.

Future TLS applications can avoid the rough edges of existing TLS root trust and adopt more modern root store practices.
Tracing Your Roots: Exploring the TLS Trust Anchor Ecosystem

Zane Ma

Postdoc Researcher
Georgia Institute of Technology
zanema@gatech.edu
https://zanema.com