# Trust Through Origin and Integrity: Protection of Client Code for Improved Cloud Security

**dr. Anders Fongen**, prof. Kirsi Helkala, prof. Mass Soldal Lund, nov 2020
*Norwegian Military University College, Cyber Defence Academy, Lillehammer*
email: anders@fongen.no

SECURWARE 2020, Valencia, Spain

## Presenter's bio

**Anders Fongen**

- Associate Professor, Norwegian Military University College
- Field of research: Software Defined Networking, Networking security
- PhD in Distributed Systems, Univ. of Sunderland, UK, 2004
- Career history
    - 4 years in military engineering education
    - 10 years research in military science (Chief Scientist)
    - 8 years in civilian college (Associate professor)
    - 11 years in oil industry
    - 6 years in electronics industry

## Introduction

How can we protect a service interface against abuse and fraud?

- Allow only **approved** client software to access that interface
- The contribution of the presented paper is to offer cloud application services a protection from rogue client code to access its interface in a *harmful* or *unintended* manner.

## Technology elements considered:

- Platform integrity protection
- Hardware bound keys and certificates
- The Cross Origin Resource Sharing (CORS) protocol
- *Indication of Origin* in the HTTP headers
- Device Security Policy Management
- Mutually authenticated TLS connections

## Platform integrity protection

Trusted Platform Module (TPM) can validate the software platform through

- a chain of protected hash values.
- digital signature of the software stack, created by sw supplier

Reservations:

- Integrity is checked only at boot time
- Application software is not checked, only the platform
    - exception: **Chrome OS** (since no application can be installed)

## Certificates and keys bound to hardware

An essential property of device attestation is that certificates and keys can be **bound to hardware.**

- Windows 10, Android, ChromeOS allows this

Furthermore, local user authentication can bind certificates to users in order to

- Authenticate **both the user and the device!**

## Cross Origin Resource Sharing

Normally a browser will block **XMLHttpRequest** call made to a different origin than the web page containing the Javascript.

This restriction can be relaxed if the target of the call explicitly allows pages with other origins to make calls to it. It does so through HTTP Response headers.

Originally designed for protection against cross-site scripting attacks, the protocol will now be used for the purpose of client-side integrity protection:

**A service can through CORS  indicate that it only allows calls to be made to its interface from specific Javascript origins. (And these origins is presumed to only contain approved and inspected code.)**

## *Indication of origin* in the HTTP Header

Another mechanism is the **Origin:** HTTP Header element. Its value is the URL of the page making the call. It is included in service requests and allows a service provider to deny or permit a service depending on its value.

Useful where the CORS protocol is not implemented, as in **WebSocket**.

## Integrity of browser code

How can the service provider **safely assume** that the browser obey the rules of CORS and the Origin-header?

- **Device management system** can apply a whitelist of allowed applications, including only browsers which are known to operate correctly
- **Chrome OS** has only one browser, which is a part of a sealed (digitally signed) software stack (platform+applications)

## Only approved *devices* are given access to service

Through **mutually authenticated** TLS connections, service providers may allow only devices/users with approved certificates to access the service.

- Approved certificates are issued to devices with integrity protection (e.g., Chrome OS)
- Certificates are bound to their assigned devices

## CORS protocol at work

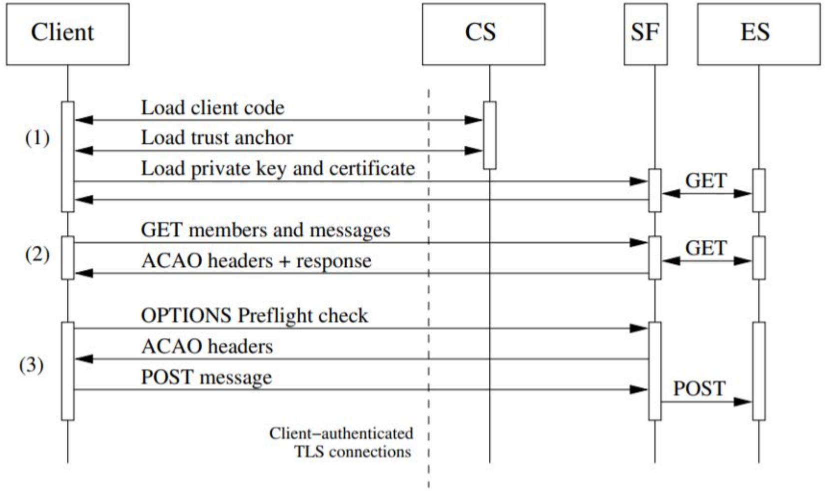


Fig. 1. The protocol elements of the Forum application, CS is the Code Service, ES the Execution Service and SF the Servlet Filter. Note how the GET and POST methods uses the ACAO headers in different ways, and how the SF component shields the CORS protocol headers from the business logic in ES.

Only code loaded by CS will be allowed to access services in ES

## The trust chain is

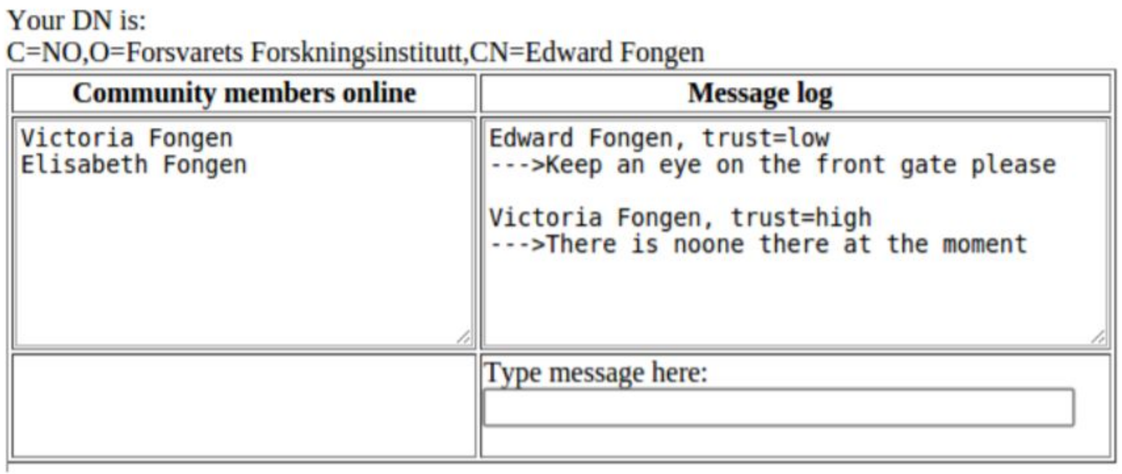
- The client code is approved because it is loaded from an approved code server
  - Code management ensures that the code server only contains approved software
  - The code is protected from modifications during transport by the TLS protocol
  - Code from any other source will not be able to invoke the service
    - Because the CORS rules will make the browser to stop the invocation
      - The CORS rules is obeyed by the browser in use by the client
        - Because the browser code is integrity protected
          - Indicated by the certificate used by TLS

- Se article for Java servlet code for this arrangement to work

## Demo application - chat service

- Messages are signed by sender, relayed by service, validated by receiver
  - using the “Web Cryptography API” (WCA) for key generation and crypto operations
- Employed the described integrity protection arrangement based on CORS and “Indication of Origin” mechanisms
- Testing both WebSocket and HTTP protocols between service and clients
  - Tomcat was the best suited server for WebSocket protocol
- Difference between Web Browsers were revealed
  - WCA was implemented with different level of maturity
  - Firefox and Chrome ok, Safari not working

## Screenshot of chat service



Your DN is:
C=NO,O=Forsvarets Forskningsinstitutt,CN=Edward Fongen

| Community members online | Message log |
|---|---|
| Victoria Fongen<br>Elisabeth Fongen | Edward Fongen, trust=low<br>--->Keep an eye on the front gate please<br><br>Victoria Fongen, trust=high<br>--->There is noone there at the moment |
| | Type message here: |

Fig. 2. A screen shot from the chat forum application. Messages from "Victoria" are sent from a ChromeOS device and are marked `trust=high`. Messages from "Edward" are sent from a MacBook.

## Conclusion

Existing APIs and protocols can be used to build new security service in order to improve cloud-based services. Principles and feasibility have been demonstrated.

Behavior of client code cannot be controlled by security protocols alone.

Programming code available on request

Comments, questions: **anders@fongen.no**

*Thank you for your attention!*