Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

TrustScript: Language Support for Partitioning Trusted Web Applications

David Goltzsche, Tim Siebels, Rüdiger Kapitza | TU Braunschweig, Germany
goltzsche@ibr.cs.tu-bs.de, siebels@ibr.cs.tu-bs.de, rrkapitz@ibr.cs.tu-bs.de

Problem Statement

- Previous work **TrustJS** achieves trusted client-side execution of JavaScript using **trusted execution environments** (TEEs)
- **Partionining** of JavaScript code is necessary
- ! No existing development tools for TrustJS
- ! Time-consuming and error-prone development
- **Goal:** first-class language support for partitioning web applications
- ↳ **Approach**: Extend **TypeScript language** to support **partitioning**

TypeScript

- **Syntactical superset** of JavaScript
- Added features: types, namespaces, interfaces, ...
- **Type checking** in compilation step
- **Transcompiles** to pure JavaScript
- Compiler itself written in TypeScript
- **Type definitions** for interfacing TypeScript and JavaScript

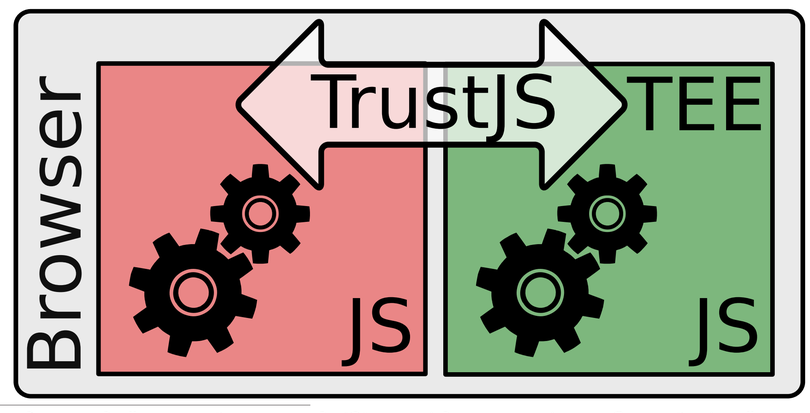
TrustScript Features

- **Single keyword** added to TypeScript language: `trusted`
- New namespace type: `trusted namespace`
  - Other possible approaches: annotations, `trusted` functions
- Compilation from a single file into **separate files**: trusted and untrusted
- Existing `export` keyword used for **exposing functions** to untrusted side
  - **Only explicitly exposed** functions are callable from untrusted side
- **Name mangling** for elements in trusted namespaces
  - Preventing name clashes due to different trusted namespaces
- **Diagnostics:** compiler warnings and errors
  - Exporting other elements than functions from trusted namespaces
  - DOM access from trusted side
  - Calling an untrusted function from within trusted namespace
- **IDE support** for Visual Studio Code

TrustJS

published[1] at EuroSec'17

- TrustJS enables **trusted**, client-side execution of JavaScript
- **Protected JavaScript engine** integrated into web browsers for securely offloading **JS applications**
- Enclave implementation based on **Intel SGX**
- ! **Partitioning** of JavaScript code is necessary



[1] Goltzsche et al. "TrustJS: Trusted Client-side Execution of JavaScript." Proceedings of the 10th European Workshop on Systems Security. ACM, 2017.

Written Code

```
// File: counter.ts
trusted namespace inside {
  let count = 0;
  export function counter(): number
  {
    return ++count;
  }
}

namespace outside {
  async function printCounter()
  {
    console.log("Counter: " +
      (await inside.counter())
    );
  }
}
```

Emitted Code

```
// File: counter_trusted.js
/* @exposed
__tsNSinsideFcounter 0;*/
var __tsNSinsideFcount = 0;
function __tsNSinsideFcounter() {
  return ++__tsNSinsideFcount;
}
```

```
// File: counter.js
var outside;
(function (outside) {
  async function printCounter() {
    console.log("Counter: " +
      (await __tsNSinsideFcounter())
    );
  }
})(outside || (outside = {}));
```

Future Work

- Currently, only **local TEEs** possible
- Extend our approach for enclaves in **remote browsers**
- Based on **WebRTC**