TrustScript: Language Support for Partitioning Trusted Web Applications

David Goltzsche
TU Braunschweig, Germany
goltzsche@ibr.cs.tu-bs.de

Tim Siebels
TU Braunschweig, Germany
siebels@ibr.cs.tu-bs.de

Rüdiger Kapitza
TU Braunschweig, Germany
rrkapitz@ibr.cs.tu-bs.de

ABSTRACT

JavaScript is ubiquitous for client-side computing in web applications. However, application providers cannot trust the client and therefore need to verify or recompute the client's computations.

TrustJS enables trustworthy computation on untrusted clients using JavaScript. Employing TrustJS, the service provider is able to offload computation to its clients in trusted fashion, paving the way for lower infrastructure costs. This is achieved by executing JavaScript inside a trusted execution environment (TEE). However, the development approach of TrustJS is error-prone and time-consuming, as partitioning of JavaScript code is necessary. Furthermore, no development tools such as linters are available.

We present TrustScript, a programming language that extends *TypeScript*. TrustScript offers language support for partitioning an application into trusted and untrusted parts, as well as various diagnostics that help the developer to detect errors at compile time. Our compiler translates TrustScript to JavaScript, emitting different files for trusted and untrusted parts of the application.

1 TRUSTJS

TrustJS [2] is a browser add-on using the *WebExtensions API* which enables execution of JavaScript inside an Intel SGX enclave. The web-application developer needs to provide a partitioned application by writing separate JavaScript files for trusted and untrusted code These files are marked in HTML as trusted or untrusted.

In order for TrustJS to expose certain functions to the untrusted side, the trusted script must contain a special comment at the start of the file indicating these functions. For these functions, a proxy with the same name is injected into the untrusted side, which invokes the trusted function inside an enclave and returns a *Promise* containing the function's return value.

2 TRUSTSCRIPT

We add one keyword to the TypeScript [1] language, which can be prepended to a `namespace` declaration: `trusted`. This allows partitioning an application into a trusted and untrusted side containing arbitrary TypeScript code. Furthermore, the existing TypeScript keyword `export` for making an element accessible outside of the namespace is analogous to *exposing* a trusted function and therefore is used for this purpose.

```
trusted namespace inside {
  let count = 0;
  export function counter(): number {
    return ++count;
  }
}
async function printCounter() {
  console.log("Counter: " +
    (await inside.counter())
  );
}
```

Figure 1: Trusted counter in TrustScript.

Figure 1 shows valid TrustScript code containing a `counter` function in the trusted side. That function is exposed to the untrusted side, increasing a variable that is not accessible outside of the trusted namespace on every invocation. When called from the trusted side, it simply returns a `number`. When called from the untrusted side, as shown in the function `printCounter`, it returns a `Promise` containing a `number`. The `printCounter` function makes use of `async`/`await` to wait until the Promise is resolved and use the resolved value.

The TrustScript compiler emits code from trusted namespaces into a designated file (or multiple files depending on configuration) that can be later encrypted or signed and needs to be marked as trusted in the HTML. Code that does not reside in a trusted namespace will be compiled into different files which will be executed in the untrusted part of the application.

Diagnostics. The trusted side does not have the same capabilities as the untrusted side. To help the developer detect mistakes in the code as early as possible, we implement various diagnostics. For instance, the trusted side does not have access to the Document Object Model (DOM) and cannot access elements defined in the untrusted part of the application. Furthermore, it is invalid to define an untrusted namespace nested into a trusted namespace or export elements other than functions and interfaces from a trusted namespace. We issue an error during compilation if the code contains these cases.

When calling into the trusted side, the trusted part runs inside a different runtime environment. Therefore, arguments and return values need to be serializable using JSON. Whenever the developer uses a type that is not serializable, we issue an error. When a type that is `any` or contains `any` is used, we issue a warning during compilation.

REFERENCES

[1] Gavin Bierman, Martín Abadi, and Mads Torgersen. 2014. Understanding typescript. In *European Conference on Object-Oriented Programming*. Springer, 257–281.

[2] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter Pietzuch, and Rüdiger Kapitza. 2017. TrustJS: Trusted Client-side Execution of JavaScript. In *Proceedings of the 10th European Workshop on Systems Security*. ACM, 7.