Client Side Trusted Web-server Modules using Intel SGX

Ajay Brahmakshatriya and **Alexandra Zytek**
{ajaybr, zyteka}@mit.edu

Abstract

Intel's SGX enclaves allow execution of code on a client computer without risk of external processes reading or saving data within the enclave. We propose using SGX to allow arbitrary code (such as for a browser game) from a server to be executed on a clients computer without allowing the client to unfairly change the code or its outputs. This makes it possible for a small server to handle thousands of clients without a decrease in performance.

1 Motivation

Web applications are usually divided into two parts – client side components and server side components. The client side components handle aspects such as UI, capturing user inputs and actions, basic sanitation of input, and relaying input to the server for further processing. The server components handle authentication, maintaining records in databases, and interacting with other services. These modules are usually run on the server because the computation they perform has to be correctly done for the application logic. For example, if `foo.com` started checking the passwords on the client side, a malicious user could bypass this logic and simply return to the server that the user has correctly authenticated.

This model is usually fine for applications that are not latency sensitive. But with the introduction of near native speed execution in browsers with NaCl or Web Assembly, many game developers are making their games available to the users to be played directly in the browser. This allows users to quickly demo games without having to install a separate game client. Even when playing against an AI, the game logic has to run on the server so that the client cannot unfairly change game state and arbitrarily award themselves rewards or advantages. But this separation between the server and client has two main issues – Firstly, the communication between the client and the server introduces communication latencies which can range all the way from 15 ms to a few hundred ms (depending on the network quality and the number of hops between the client and the server). While this may be okay for turn based games like chess, it is completely unsuitable for real-time fast-paced games. The second challenge is that since the computation of the game state happens on the server side, the server resources (both memory and compute) have to scale linearly to the number of clients connected. For some very popular games maintained by small companies this could incur significant costs.

We attempt to solve this problem by allowing the server to send over to the client some modules of the server side logic and provide a guarantee that the logic is run exactly as it is supposed to be. Our solution comes as an API extension to the JavaScript engine in the browser that allows the JavaScript loaded from the server to create Intel SGX enclaves inside the browser, load arbitrary code and remote attest the code with the server. The JavaScript can then communicate with the running module through a defined interface while the trusted module can communicate with the server as required. The JavaScript can terminate the execution of the enclave at any time.

The game developers can now offload the game state computation to the client machine while also reducing the latency between the browser and the game server to order of a few microseconds. The game server can also trust the final state (user score, player profile changes) relayed to it from the trusted enclave and update it when the game ends. This allows a very small server to handle thousands of clients at the same time.

2 Criteria for Success

We define the following three goals for the project:

1. Successfully design an API extension to the JavaScript engine that allows the browser to create an SGX enclave, load and execute arbitrary code, and attest the code with the server.
2. Demonstrate that latency, bandwith utilization, and server utilization are improved by the design.
3. Show that the system is secure for the client.

We believe that goal 1 is the most important for a project in this context; however, goals 2 and 3 would be necessary before considering publishing this project further.

3 Design

We developed a new API in JavaScript which browsers can choose to implement (depending on availability of SGX hardware and user preferences). The following types and function calls were added to the JavaScript API to facilitate creation

of the enclave module, communication with the module, and termination of the module.

First, we need to ensure the browser can create an SGX enclave:

```
@global
@retuns bool: True if browser supports
    creation of SGX enclaves, False otherwise
function
    browser_has_trusted_module_capability()
```

Next, we can create the enclave module, defined as an object type:

```
@global
@module
TrustedServerModule = {
    @params url - URL for the image of the
        trusted module to be loaded inside
        the enclave

    @returns the newly created
        TrustedServerModule or None if the
        creation failed
    create: function(url: string) :
        TrustedServerModule

    @returns a new TrustedModuleConnection
        object with the module which is the
        equivalent of the Web Socket over the
         network or an existing connection if
         created before
    get_connection: function() :
        TrustedModuleConnection
```

And we keep track of the connection to the module with an object type:

```
@global
@module
TrustedModuleConnection = {
    @params message - arbitrary JSON object
        to send to the module
    send: function(message: json),

    @params timeout - optional timeout in
        milliseconds to wait for response
        from the module
    @returns message response from the
        TrustedServerModule
    recv: function([timeout]: number): json

}
```

The trusted module has a C API that allows it to communicate with the JavaScript in the browser:

```
struct TrustedConnection;

// returns the currect trusted connection
    object between the JavaScript and the
    trusted module.
struct TrustedConnection*
    get_current_trusted_connection(void);

// params connection - TrustedConnection
    object
// message: string - a string encoded valid
    JSON object
void trustedSend(struct TrustedConnection *
    connection, char* message);

// params connection - TrustedConnection
    object
// buffer: string - a buffer to get a JSON
    encoded object in
// buffer_size: size_t - size of the buffer
// returns the amount of bytes received
int trustedRecv(struct TrustedConnection *
    connection, char* buffer, size_t
    buffer_size);
```

4 Implementation

The feature is added to the browser through a browser extension. On browser startup, a module server will begin running. This server will create, communicate with, and terminate enclaves when requested by the browser.

On the call to the `TrustedServerModule`'s `create` function, the server forks into a new process and create the SGX enclave by downloading the image from the given url. The new process then communicates with the main server using pipes and the communication is relayed back to the browser process through RPC again. The server can also forward the pipe fd to the browser to reduce RPC calls. The trusted module server maintains a separate communication object for each call of `create` from the browser, thus isolating each instance from each other and allowing an instance to only communicate with its calling JavaScript tab.

5 Testing Performance

In order to test performance, we ran a simple Kmeans++ algorithm [1] to cluster points, defined in a file inputted by the user. This application was designed only to put a strain on the CPU while also requiring user inputs.

Our trusted browser had a significantly better response time than the baseline, as shown in figure 1.

References

1. K-means++ clustering. (2019, April 28). Retrieved from https://rosettacode.org.

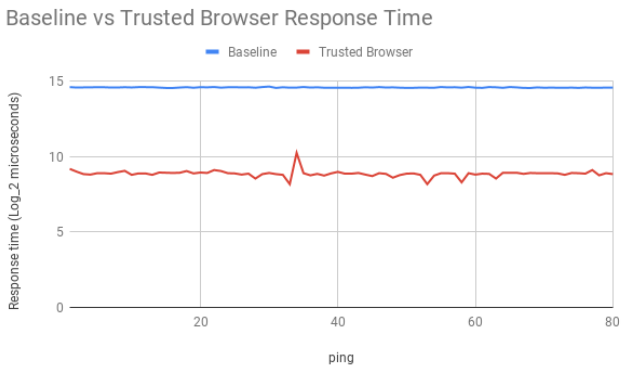

Figure 1: The response time across pings was significantly improved when using the trusted browser. Note the log scale of the y-axis.