Practical Applications of Client-side Trusted Computing

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

ABSTRACT

Offloading computations from central infrastructure to client machines of end-users imposes restrictions because they are untrusted environments. However, with trusted computing technology becoming widely available on commodity hardware, the approach of designing distributed systems shifts: When it is possible to assume trusted clients, existing systems can be designed differently. Additionally, entirely new use cases are possible.

KEYWORDS

trusted computing, offloading, Intel SGX

1 INTRODUCTION

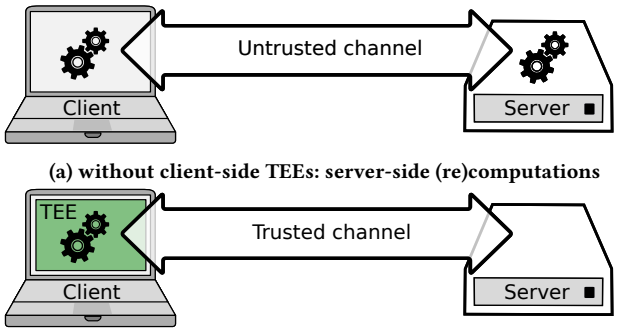
The general trend to offload server-side functionality to clients is ubiquitous: websites contain code that is executed by web browsers, volunteer computing systems use the computing power of private machines, and with fog computing various cloud services are offloaded to client machines. This trend is motivated by shorter latencies, thus achieving a better user experience or the ability to relieve central infrastructure by exploiting otherwise idle resources.

While this approach has become common practice, system designers are required to take precautions because client machines are not under central control and therefore cannot be trusted. Precautions include recomputations in the systems back-ends (e.g. in web applications) or replication of computations to other clients (e.g. blockchains or volunteer computing systems). In other cases, system designers refrain from offloading computations at all (e.g. centralised network functions).

However, the widespread availability of trusted execution environments (TEEs) for commodity hardware in the form of the Intel Software Guard Extensions (SGX) opens up new system design paradigms: remote parties can verify the TEE (called *enclaves*) on client machines in order to establish a trusted channel and gain trust in the computations performed in it, making offloading of arbitrary computations to untrusted clients even more attractive. Figure 1 shows a general client-server model with and without client-side TEEs.

In this research proposal, we will show how and why we can design web applications (Section 2) and volunteer computing systems (Section 3) differently when assuming trusted clients. Additionally, we propose the entirely new use case of client-side middlebox functions (Section 4), which has become feasible with trusted network clients.

EuroDW'18, April 23, 2018, Porto, Portugal
2018.



(a) without client-side TEEs: server-side (re)computations

(b) with client-side TEEs: no server-side (re)computations

Figure 1: General client-server model (a) with and (b) without client-side TEEs.

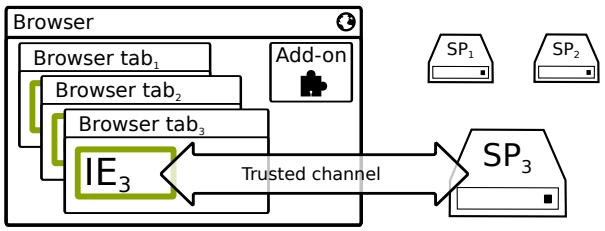


Figure 2: Architecture of TrustJS; example with three service providers (SPs), each associated with an interpreter enclave (IE).

2 TRUSTED WEB BROWSERS

Web applications increasingly replace traditional desktop applications, because developers gain instant deployability and platform independence. The scripting language *JavaScript* enables developers to offload computations to client-side browsers in a standardised way, resulting in minimised round trip times and less resource demand on server side.

However, offloading code to clients has its limitations: From the web service provider's views, clients are not trustworthy, thus computation results may be wrong, corrupted or incomplete. To solve this, client results are usually checked for consistency at server-side, which requires recomputations. Furthermore, clients cannot be entrusted with the intellectual property of confidential JavaScript code because it cannot be protected. Solutions like code obfuscation have been proven as insufficient [7].

With TrustJS [5], we proposed a solution for trusted client-side execution of JavaScript code using *interpreter enclaves* (IEs) based on Intel SGX and MuJS[2], an open-source JavaScript interpreter.

As shown in Figure 2, TrustJS integrates the interpreter enclaves into a browser and creates trusted channels between JavaScript runtime environments and service providers. Trust is established by combining in-enclave termination of encrypted data streams and remote attestation as provided by Intel SGX.

In the paper, we show how TrustJS enables trustworthy execution of JavaScript inside commodity browsers, allows service providers to save resources and achieves lower latencies, improving overall user experience. The main contributions of TrustJS are the transparent browser integration including SGX support and the creation of a new web development paradigm that considers trusted clients. Currently, we are improving TrustJS by enabling WebRTC and just-in-time (JIT) compilation by using Google V8 instead of MuJS. The support of WebRTC raises new challenges due to missing system support within enclaves. In terms of JIT compilation, security concerns of in-enclave generated code have to be evaluated and handled.

Furthermore, we are experimenting with use cases for TrustJS. So far, we investigated client-generated CAPTCHAs and anti-cheat mechanisms for browser-based massively multiplayer online games.

3 TRUSTED VOLUNTEER COMPUTING

Volunteer computing systems like BOINC [1] have become incredibly popular, attracting millions of participants to voluntarily donate computing power to a plethora of research projects. In addition to sheer altruism, volunteers are motivated by credit systems that these systems implement: users receive credit points for computations they perform and leader boards allow users to compete.

However, we identified three fundamental design flaws in today's volunteer computing systems: First, the resources of clients are wasted by recomputations of jobs to deal with cheaters or otherwise misbehaving clients. Second, both code and data of jobs are always disclosed to participants, which may not be desirable in every case. And third, most volunteers cannot be motivated for long-term commitments to the system: less than 4% of BOINC's 4.5 million users are actually active contributors [3].

We propose TruVC, a decentralised volunteer computing system that solves the issues stated above. Wasting of resources is solved by establishing trust in client machines using TEEs provided by Intel SGX. The combination of encrypted data channels and TEEs also allows the deployment of encrypted compute jobs. For isolated execution, TruVC uses a near-native-speed sandboxing environment and, for motivating volunteers, TruVC's participants are financially reimbursed by the task providers. The main challenges for TruVC are provisioning a work estimation measurement to fairly reimburse participants for equal workloads and the trusted management of job offers from task providers.

4 CLIENT-SIDE MIDDLEBOXES

So-called *middleboxes* are used in large managed networks to implement functions like firewalls, intrusion detection, caching or load balancing. However, middleboxes are typically deployed centrally, despite high infrastructure and management costs [9]. Alternatively, in defiance of high security and legal risks, research projects propose to outsource middleboxes to the cloud [8].

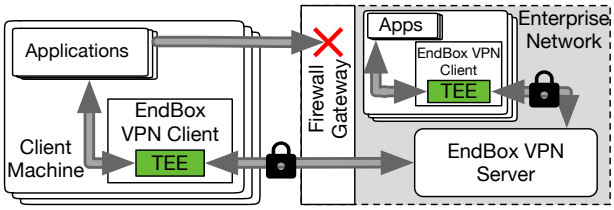


Figure 3: Architecture of EndBox in an enterprise scenario

To address these limitations, we suggest EndBox [4], a system with a *decentralised* deployment approach, placing middleboxes on client machines at the network edge exploiting idle resources. EndBox combines a VPN with hardware-protected middlebox functions to allow usually untrusted clients the execution of middlebox functions. Figure 3 shows the architecture of EndBox: internal and external network clients run an enhanced VPN client incorporating a TEE based on Intel SGX. The TEE contains security-sensitive parts of the VPN client and middlebox functions implemented using the software router *Click* [6]. The VPN server only accepts encrypted connections that originate from genuine enclaves, thus, it ensures middlebox functions are executed on clients. As EndBox distributes middlebox functions across clients, the main challenge is to maintain the manageability of those. We do this by implementing a mechanism for rapid and seamless middlebox function configuration for centralised administration.

EndBox is a scalable system that enables secure deployment and execution of middlebox functions on untrusted client machines. In the paper, we show that EndBox scales linearly with the number of clients and achieves an up to 4× higher throughput than traditional deployments. Additionally, we show the deployment of five use cases on EndBox including an intrusion detection and prevention system (IDPS) and the prevention of distributed denial of service attacks.

REFERENCES

[1] David P Anderson. "Boinc: A system for public-resource computing and storage". In: *Grid Computing*. IEEE. 2004.
[2] Artifex Software, Inc. http://mujs.com/. 2016.
[3] *BOINC statistics*. https://boincstats.com/en/stats/-1/project/detail/overview. 2018.
[4] David Goltzsche et al. "EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution". In: *Proceedings of the 48th International Conference on Dependable Systems and Networks*. DSN'18. Accepted for Publication. 2018.
[5] David Goltzsche et al. "TrustJS: Trusted Client-side Execution of JavaScript". In: *Proceedings of the 10th European Workshop on Systems Security*. ACM. 2017.
[6] Eddie Kohler et al. "The Click modular router". In: *ACM Transactions on Computer Systems*. TOCS (2000).
[7] Clemens Kolbitsch et al. "Rozzle: De-cloaking internet malware". In: SP. 2012.
[8] Chang Lan et al. "Embark: Securely Outsourcing Middleboxes to the Cloud". In: *13th USENIX Symposium on Networked Systems Design and Implementation*. NSDI '16. 2016.
[9] Justine Sherry et al. "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service". In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2012.