Trusted Volunteer Computing

Michele Ianni[1] and Elio Masciari[2]

[1] DIMES, Università della Calabria mianni@dimes.unical.it
[2] ICAR CNR elio.masciari@icar.cnr.it

Abstract. Technology becomes more and more advanced everyday, both from the software and from the hardware perspective. Brand new devices, more powerful and capable of the generation preceding them, are steadily released. Everybody owns laptops, smartphones and many other devices with great compute capabilities, able to easily solve problems a few years ago considered almost impossible. These devices are, however, most of the time underused, resulting in an incredible waste of computational resources. The needs of professional people and scientists are evolving too, alongside the advances of technology. In many fields, from financial and biomedical simulation to insurance predictions, from 3D rendering to mathematical computations etc huge compute capabilities are still required. Despite the always growing presence of powerful devices and their increasingly cheap prices, in many situations using only our own devices is not enough. The contrast between underused devices and needs of computational resources led to the birth of a new approach: volunteer computing. This approach brings a user to be part of a network, sharing his idle CPU (or GPU) cycles in order to solve a subtask of a problem thus contributing, along with many other users, to the solution of a bigger task. In a very large number of cases, however, volunteer computing needs to tackle some serious security concerns. Many tasks, in fact, deal with sensitive information whose disclosure, even in a minimal part, must be avoided. For this reason still many problems are usually not handled by collaborative networks, since it is not possible, in many cases, to hide confidential data from inputs of every subtask. In this paper we present a way to solve the problem described. Through the use of trusted computing we are able to manage complex problems in a distributed network of volunteer computing devices without the risks related to the spread of sensitive data.

1 Introduction

Nowadays we are constantly surrounded by an always growing number of so called smart devices [18]. Everything we use seems to need great compute capabilities and Internet connection to work properly. Many everyday life objects, from fridges to cars, from light bulbs to toasters are integrating computers and providing functionality that several years ago where almost unthinkable. We are facing a new era, the era of Internet of Things [15], everything is always connected thus accessible from everywhere. The object that best represents this revolution is the smartphone. Everybody owns at least one of them, they are always connected and capable to perform tasks with remarkable performances. The hardware vendors are unceasingly releasing new generations every season, with new devices that easily overwhelm their predecessors, both for compute capabilities and new functionality added. The industry of personal computers follows an analogous rhythm, releasing more and more powerful devices with incredible constancy. These devices are becoming affordable and everybody, due to the hard rules imposed by our consumer society, have an induced need not only to buy them, but even upgrading to newer models as soon as the pressure related to the menace of losing their social position becomes unbearable. However this then begs the question: do we really need all those compute capabilities? Or at least, do we need them continuously? The answer is pretty obvious: our devices are, most of the time, underused. The average user does not take advantage of all the power embedded in his devices and the waste of resources is huge. On the other side there exist a lot of people who really need a huge amount of compute capabilities, that goes far beyond the considerable power of a brand new laptop. Computer scientists, 3D artists, financial analysts to cite a few, most of the time are involved in problems that require a lot of time to be solved and they are forced to spend time and money to obtain results. From this contrast between compute resources need and waste a new paradigm has born: volunteer computing. It is a type of distributed computing in which a user is part of a larger network. He donates idle CPU (or GPU) cycles, or even storage sometimes, in order to contribute to one or more "projects". Volunteer computing frameworks usually have a master/slave structure. When a problem (called job) have to be solved, one server is responsible of splitting the job in subtasks which are distributed across the nodes of the network (the volunteer computers). A client application runs on the volunteer's computer and receives the subtask from the remote server. After completing the assigned subtask the result is sent back to the server who merges the partial results to provide a solution for the initial job. In many cases the system keeps track of the volunteers work in order to implement a reward policy. Volunteer computing is a well known paradigm that spans across many different application areas. The first volunteer computing project was the Great Internet Mersenne Prime Search [23], which was started in January 1996. From that date many other projects arose [20, 19, 1, 9]. A special mention, due to the several hundred thousand volunteers involved, goes to SETI@home project [7], whose purpose is to analyze radio signals, searching for signs of extraterrestrial intelligence, and Folding@home [17] which aims to determine the mechanisms of protein folding, a subject of significant academic interest with major implications for medical research into many types of disease. One of the biggest steps in volunteer computing history has been made in 2002 with the born of BOINC [4], the Berkeley Open Infrastructure for Network

Computing. It is a project, founded at University of California, Berkeley Space Sciences Laboratory, whose aim is to provide a complete middleware system for volunteer computing. BOINC includes everything needed to start a project using volunteer computing, from the client application to the server software.
Although it is easy, thanks to the middleware available, to start a new project, volunteer computing is still not expressing its full potential. This is because this computing paradigm is based on the distribution of problem inputs among users and there is a serious problem related to data distribution: *privacy*. Many projects perfectly suitable for volunteer computing: from financial simulations to insurance predictions, from health databases scraping or biomedical modeling to marketing analysis and so forth most of the time deal with sensitive information that must remain secret. In many cases the sensitive data is so tightly connected to the rest of the information that is impossible to hide the former without altering significantly the result of the computation. Most of the times it is not possible to distribute even a small subset (the input for a subtask) of the data, because sensitive information can be inferred by its observation or because the solution of the subtask itself cannot be computed without the rest of the dataset.
Example: In our past experience with distributed computing we faced the problem of distributed rendering. In order to solve the rendering equation a ray tracing algorithm (or variants of it) is used [13]. The algorithm correctly computes the value of luminosity of every pixel in the scene by taking into account all the objects and lights that can affect the color of the given pixel. In distributed rendering, even if the rendering of a single tile (a subset of the entire scene) is requested to a volunteer, he needs to know all the information present in the entire scene (objects, lights) in order to solve the rendering problem for the pixels in the tile. It is easy to figure many cases in which the scene to be rendered must be kept secret, we just need to think about the work of architects, engineers, graphics, 3D artists and so on.
All these security concerns are crucially narrowing the potential of volunteer computing. The current middleware projects, in fact, are not suitable to deal properly with sensitive data. Our contribution in this paper is to introduce the use of trusted computing to extend the power of volunteer computing by making possible to deal with sensitive data in a safe way. The proposed solution described in this paper is related to the use of Intel® Software Guard Extensions [3], but the same concepts are applicable for other environments (e.g. ARM TrustZone [2]) with slight modifications on which we are currently working on.

2 Intel® Software Guard Extensions

It is very common to deal with software applications that need to work with sensitive data. Operating system's security policy based on permissions are often not enough to protect secrets. Despite they prevent a user from accessing other user's files they cannot avoid the access to the sensitive information to processes running with higher privileges, including the operating system itself and a plethora of various types of malicious software. To protect the secret data even in the presence of privileged malware, Intel® designed Software Guard Extensions (SGX): a security technology introduced in autumn 2015 in their CPUs based on the Skylake x86 microarchitecture. SGX is a set of CPU instructions that permit the allocation of private regions of usermode address space, called *enclaves* by leveraging trusted hardware. These regions are protected from processes running even at higher privilege levels and provide confidentiality and integrity [12]. Several protection are guaranteed by Intel® SGX:

– Enclave memory is encrypted and cannot be read or written from outside even by processes with high privileges and independently by the CPU mode. The encryption key, stored in an inaccessible region within the CPU, is changed every power cycle.
– The only way to enter in an enclave is through a dedicated instruction, there is no other way to jump inside an enclave.
– Enclaves cannot be debugged by software or hardware debuggers.
– Data within enclaves can only by accessed by code in the same enclave.

One of the main drawbacks of this technology is that the applications running in SGX enclaves are severely restricted compared to normal programs. The Trusted Computing Base (TCB) of SGX is minimal, it consists of the CPU (hardware logic, microcode, registers, cache memory) and the software components used for attestation (see section 4). More in detail the operating system is not part of the TCB, this means that is not possible to use kernel services or system calls directly. This is because the operating system is untrusted and a syscall involves the transfer of control flow from trusted code running in an enclave to untrusted OS code. The latter could be a malicious kernel exposing crafted system calls capable of compromising the security of the enclave. As a workaround SGX includes instructions, called OCALLs, that offer the opportunity to temporarily exit the enclave and calling untrusted code to perform general purpose operations like system calls, I/O etc. Several studies (Graphene [14], SCONE [8], Haven [10], Panoply [22]) have been released on this topic and many of them aim at overcoming the burden of switching from enclaves to untrusted OS. The solutions proposed in those projects are based on a syscall forwarding mechanism that handle the calls to system services inside the enclave and execute them outside taking care of switch between two different context and the transfer of the data needed to perform the operation requested. This allows the shielded execution, within a SGX enclave, of unmodified legacy applications written for traditional operating systems. In our paper we are assuming that, if required by the algorithm that runs inside the enclave and handles the sensitive data, one of this syscall forwarding mechanism is implemented and we can run arbitrary code within the enclave.

3 Trusted volunteer computing system architecture

The proposed architecture is based on a Client-Server model. The clients of the system are all the users sharing their compute capabilities in the network. Their work is coordinated by one or more servers which are in charge of splitting the task to be executed in subtasks and distributing them among the clients. Usually in volunteer computing networks the server is responsible of many other actions: collecting the results from the clients and merging them to obtain the result of the main task, verifying the correctness of the results provided, enforcing client management policies based on several factors, such that performances or reliability, of the volunteer devices, just to cite a few. All these server duties are well known topics fully covered in literature [21, 6, 5].
The volunteers of the network, who wish to share their computational resources, can achieve this goal by using a client application on their devices. We focus our attention on protecting the client application, since they tend to run on platforms that may not have the same degree of control and security as servers. Nevertheless, the techniques described here can also be applied in order to protect assets on the server.
The client application consists of two different parts: a trusted and an untrusted module. The trusted part of the application runs within the SGX enclave and handles the sensitive data. It is kept as small as possible, it is composed only by the algorithms which need to deal directly with the secret assets. The untrusted part of the application is made up of all the rest: graphical user interface, libraries and all the functionality needed to communicate with the rest of the network performing tasks such as signaling presence in the network, handling connections, managing the amount and the type of the resources shared with the volunteer network. The trusted and untrusted components of the client application communicate to each other through well-defined entry-points. These are designed to ensure that no secrets are allowed to leak out from the enclave. The integrity and confidentiality of the data inside the enclave is guaranteed by the the hardware/software protections of SGX.
The subtasks to be resolved are sent by the remote server to the client application. The sensitive data is encrypted (see section 4) and is forwarded by the untrusted part of the client application to the trusted part. After the completion of the subtask the results, encrypted, are sent back to the remote server which merges them with the other results coming from other clients to finally get the result of the main task.

4 Provisioning and attestation issues

Since our goal is to provide a way to deal with sensitive information in volunteer computing networks, we need to tackle some security issues related to the

transfer of this data between server and client. Our threat model is based on the assumption that every client on the network is untrusted, thus we need to provide a way to secure the communication among all the parties involved. The SGX enclave must have the assurance of communicating with a legit remote server and vice versa (this process is called *attestation*). They also need to establish a secure channel to talk each other and to transfer the sensitive data (*provisioning* phase). To address the issues described we make use of the services provided by Intel® SGX [3, 16]. On the client machine is running, in fact, the SGX Platform Software (PSW) that includes also a Quoting Enclave containing an asymmetric attestation key representing the SGX TCB.
In response to the first request of connection from a client application to the remote server the latter will send to the client an attestation request containing a nonce for liveness purposes. This request aim at verifying that the software runs on a legit SGX enclave as well as ensuring the validity of the enclave and collecting information about the identity of software being attested. It is the untrusted part of the client, as explained in section 3 that handles the connection with the remote server. This part of the application receives the attestation request and forwards it to the trusted part, running inside the enclave. At this point the trusted part sends a local attestation to the Quoting Enclave through the untrusted part. The local attestation consists of a report structure along with a manifest that include the nonce and a temporary public key C_P that is used later by the remote server. The Quoting Enclave creates a remote-attestation by verifying, and signing (with the asymmetric attestation key) the local one. The result of this process is called *Quote* and is sent (always through the untrusted part) to the remote server. A verification server is used by the remote server to verify the received Quote. To avoid some privacy concerns related to this scheme [24] Intel® adopts an extension called Intel® Enhanced Privacy ID (EPID) [11]. EPID signatures are anonymous and thanks to a group signature scheme, nobody can uniquely identify the platform where the application is running by looking at objects signed by the application itself. At the end of the attestation process the remote server is sure to communicate with a legit secure enclave. The entire client application running alongside the secure enclave is now part of the volunteer computing network and it is ready to share its computational resources to solve the subtasks provided by the remote server. It is still mandatory, however, to ensure that the communication between client and server is conducted through a secure channel and is necessary to create this channel along the remote attestation process to avoid the provisioning of sensitive data to not attested enclaves. To create this channel the temporary public key C_P, created by the trusted enclave and delivered to the server inside the Quote is used. The remote server generates an encryption key E, encrypts it with C_P and sends the result of this operation to the application. Only the trusted part of the client, running within the enclave, knows the private key associated with C_P and can obtain the encryption key E. The sensitive data, as well as the results of

the subtasks, can be encrypted using E, permitting the secure communication between the client and the remote server.

5 Conclusions and future work

In this paper we described a new way to further enhance the power of volunteer computing networks. By the leverage of Intel® Software Guard Extension we provided a way to build trusted volunteer computing systems without the problems related to the spread of secret information. Using the system proposed it is possible to use computational resources of untrusted devices for solving tasks that deal with sensitive data. As a future development of the approach described we are working on extending these ideas to other Trusted Execution Environments (e.g. ARM TrustZone), since they are integrated in most of the new smartphones.

References

[1] Albert D Alexandrov et al. "Superweb: Towards a global web-based parallel computing infrastructure". In: *Parallel Processing Symposium, 1997. Proceedings., 11th International.* IEEE. 1997, pp. 100–106.

[2] Tiago Alves, Don Felton, et al. "Trustzone: Integrated hardware and software security". In: *ARM white paper* 3.4 (2004), pp. 18–24.

[3] Ittai Anati et al. "Innovative technology for CPU based attestation and sealing". In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy.* Vol. 13. 2013.

[4] David P Anderson. "Boinc: A system for public-resource computing and storage". In: *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on.* IEEE. 2004, pp. 4–10.

[5] David P Anderson, Carl Christensen, and Bruce Allen. "Designing a runtime system for volunteer computing". In: *SC 2006 Conference, Proceedings of the ACM/IEEE.* IEEE. 2006, pp. 33–33.

[6] David P Anderson, Eric Korpela, and Rom Walton. "High-performance task distribution for volunteer computing". In: *e-Science and Grid Computing, 2005. First International Conference on.* IEEE. 2005, 8–pp.

[7] David P Anderson et al. "SETI@ home: an experiment in public-resource computing". In: *Communications of the ACM* 45.11 (2002), pp. 56–61.

[8] Sergei Arnautov et al. "SCONE: Secure linux containers with Intel SGX". In: *12th USENIX Symp. Operating Systems Design and Implementation.* 2016.

[9] Arash Baratloo et al. "Charlotte: Metacomputing on the web". In: *Future Generation Computer Systems* 15.5 (1999), pp. 559–570.

[10] Andrew Baumann, Marcus Peinado, and Galen Hunt. "Shielding applications from an untrusted cloud with haven". In: *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015), p. 8.

[11] Ernie Brickell and Jiangtao Li. "Enhanced privacy ID from bilinear pairing for hardware authentication and attestation". In: *International Journal of Information Privacy, Security and Integrity 2* 1.1 (2011), pp. 3–33.

[12] Victor Costan and Srinivas Devadas. "Intel SGX Explained." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86.

[13] Andrew S Glassner. *An introduction to ray tracing.* Elsevier, 1989.

[14] *Graphene Library OS for Intel SGX.* `https://github.com/oscarlab/graphene/wiki/Introduction-to-Intel-SGX-Support`. Accessed: 2017-03-12.

[15] Jayavardhana Gubbi et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.

[16] Simon Johnson et al. "Intel software guard extensions: EPID provisioning and attestation services". In: *White Paper* (2016).

[17] Stefan M Larson et al. "Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology". In: *arXiv preprint arXiv:0901.0866* (2009).

[18] Stefan Poslad. *Ubiquitous computing: smart devices, environments and interactions.* John Wiley & Sons, 2011.

[19] Ori Regev and Noam Nisan. "The popcorn market. online markets for computational resources". In: *Decision Support Systems* 28.1 (2000), pp. 177–189.

[20] Luis FG Sarmenta. "Bayanihan: Web-based volunteer computing using Java". In: *International Conference on Worldwide Computing and Its Applications.* Springer. 1998, pp. 444–461.

[21] Luis FG Sarmenta. "Volunteer computing". PhD thesis. Massachusetts Institute of Technology, 2001.

[22] Shweta Shinde et al. "Panoply: Low-TCB Linux Applications With SGX Enclaves". In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26-March 1, 2017.* 2017.

[23] George Woltman, Scott Kurowski, et al. "The great internet mersenne prime search". In: *Online],(1997, March 23) available http://www. mersenne. org* (2004).

[24] *Working Document on Trusted Computing Platforms and in particular on the work done by the Trusting Computing Group (TCG group).* Article 29 Data Protection Working Party. European Commission, Brussels, 2004.