UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATHEUS DA CUNHA MELO ALMEIDA

A STUDY OF CONFIDENTIAL COMPUTING AS A WAY TO PREVENT SENSITIVE INFORMATION EXPOSURE ON INFORMATION SYSTEMS

CAMPINA GRANDE - PB

2021

MATHEUS DA CUNHA MELO ALMEIDA

A STUDY OF CONFIDENTIAL COMPUTING AS A WAY TO PREVENT SENSITIVE INFORMATION EXPOSURE ON INFORMATION SYSTEMS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Andrey Brito.

CAMPINA GRANDE - PB

2021

A447s Almeida, Matheus da Cunha Melo.
A study of confidential computing as a way to prevent sensitive information exposure on information systems. / Matheus da Cunha Melo Almeida. – 2021.

9 f.

Orientador: Prof. Dr. Andrey Elísio Monteiro Brito.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. confidential computing. 2. Privacy. 3. Application security. 4. SCONE. 5. Sensitive information exposure - prevent. I. Brito, Andrey Elísio Monteiro. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

MATHEUS DA CUNHA MELO ALMEIDA

A STUDY OF CONFIDENTIAL COMPUTING AS A WAY TO PREVENT SENSITIVE INFORMATION EXPOSURE ON INFORMATION SYSTEMS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Andrey Brito

Orientador – UASC/CEEI/UFCG

Professor Dr. Rohit Gheyi

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de Outubro de 2021.

CAMPINA GRANDE - PB

RESUMO (ABSTRACT)

A privacidade do usuário é uma das maiores preocupações dos desenvolvedores de aplicações hoje em dia. Com o advento de novas regulamentações e ciber ataques se tornando mais comuns e caros, a demanda por novas tecnologias que podem ajudar a reduzir ou mitigar o risco de exposição da informação está aumentando. Como os humanos são os mais suscetíveis à falha na maioria dos sistemas, é importante procurar um método para reduzir potenciais erros humanos ou exposição intencional de informações. Neste artigo, a computação confidencial é estudada como uma forma de prevenir tais vazamentos de dados executando aplicações dentro de um ambiente de execução confiável.. Neste contexto, um ambiente de execução confiável é definido como uma área segura de um processador principal, o que garante que o código e os dados carregados internamente são protegidos com respeito à confidencialidade e integridade. Para avaliação, um sistema de informação usando SCONE foi implementado e uma série de testes de segurança e desempenho em uma aplicação genérica foram executados. Os resultados mostraram uma melhoria considerável na segurança do aplicativo e uma deterioração considerável no desempenho da aplicação. Os resultados sugerem que a computação confidencial pode proteger os aplicativos contra os ataques de nível de administrador mencionados, mas seu uso deve se adequar a certos casos de uso onde o desempenho não é fundamental para o aplicativo comportamento ou o fato de que são necessários mais recursos para funcionar no mesmo nível de desempenho.

A study of confidential computing as a way to prevent sensitive information exposure on information systems

Matheus da Cunha Melo Almeida
Federal University of Campina Grande
Campina Grande, Brazil
matheus.almeida@ccc.ufcg.edu.br

Andrey Brito
Federal University of Campina Grande
Campina Grande, Brazil
andrey@computacao.ufcg.edu.br

ABSTRACT

User privacy is one of the biggest concerns of application developers nowadays. With the advent of new regulations and cyber attacks becoming more common and expensive, the demand for new technology that can help reduce or mitigate the risk of sensitive information exposure is rising. As humans are the most substantial liability in most systems, it's important to search for a method to reduce potential human errors or intentional information exposure. In this article, confidential computing is studied as a way to prevent such data leaks by running applications inside a trusted execution environment. In this context, a trusted execution environment is defined as a secure area of a main processor, which guarantees code and data loaded inside to be protected with respect to confidentiality and integrity. For evaluation, a information system using the SCONE runtime was implemented and a series of security and performance tests against a sample application were performed. The results showed a considerable improvement in application security and a considerable deterioration in application performance. The results suggest that confidential computing can protect applications against the mentioned admin-level attacks, but its use must fit certain use cases where performance is not key to the application's behaviour or the fact that it needs more resources to run on the same performance level is acceptable.

1 INTRODUCTION

Information security is one of the biggest challenges of today's software development research and Computer Science in general. This concern is evidenced as new data regulations start to be implemented, such as the European Union's General Data Protection Law and Brazilian government's Lei Geral de Proteção de Dados Pessoais (general personal data protection law). With the number, severity, and cost of data breaches increasing over the years, and the fact that most of them are caused by human error, some questions are raised more and more often: how to ensure a system is trustworthy? How to reduce human liability?

In standard applications, some level of human trust is needed in order to make it viable. Companies need to trust their developers, code reviewers, testers and, mostly, system administrators, will not insert potential attack targets in their software or infrastructure, leak sensitive credentials or even breach it themselves. According to recent studies, human errors are the cause of the majority of cyber-attacks[1].

The admin-level attack model is a much common and easy way of sabotaging standard applications and breach user privacy by changing application code or running database queries, for example. It can be caused by a malicious admin-level user themselves or by a malicious attacker with stolen admin-level credentials.

Although some data breaches are caused by human error while programming the application, many are caused by insufficient protection of key parts of the system, such as credentials or network traffic. To prevent some of such data breaches caused by human liability, one possible solution is to use confidential computing technologies, such as Intel SGX and SCONE, where the application runs in a safe environment and needs to be attested before every run, and transfer the trust to hardware and software rather than people.

In this paper, a use case of an application which stores sensitive user data and how secure this application becomes by using Intel SGX and SCONE in order to prevent admin-level attacks. This application has served as one of the pillars for what today is the AMTS project, which targets the identification of possibly infected people by using thermal camera footage. This article also evaluates the solution's security and performance, and compares it to a non secure version, in order to show potential trade-offs of the trusted computing solution.

1.1 Intel SGX and SCONE

Intel Software Guard Extensions (SGX)[3] is a set of instructions and changes to memory access added to the Intel X86 architecture. They allow user level code to define enclaves, which are private regions of memory, whose contents are protected and unable to be read by any process outside of the enclave. It does that by encrypting a portion of memory, which is decrypted on the fly only within the CPU, and only for code and data running from within the enclave itself.

SCONE[2] is a toolset, including runtime, compiler, and curated images that aim to provide secure container mechanisms based on SGX to protect container processes from outside attacks. SCONE provides a user-level threading implementation which maximizes the time threads spend inside the enclave. It also maps OS threads to logical application threads in the enclave, scheduling OS threads between application threads when they are blocked due to thread synchronization.

SCONE provides new concepts that were used in the context of this paper. For the presented use case, SCONE is capable of encrypting disk and memory content, while attesting that the code was not modified prior to its execution. Along with other known techniques, such as HTTPS and mutual TLS, it is possible to make applications much more resilient to admin-level attacks.

Before running the application for the first time, it is needed to create a session in a CAS instance. A session is a set of application security policies which are described in a session file. In the session file, it is possible to configure TLS certificate generation, environment variables, disk protection, and much more. All confidential configurations are stored in the CAS instance and are

provided to applications after the CAS instance has verified their integrity and authenticity. A LAS instance is responsible for being the middleware between the CAS instance and the application to be attested.

1.2 Kubernetes

Kubernetes is currently the industry standard for container orchestration. It was launched in 2014 by Google, and was heavily inspired by the company's internal container orchestration solution, Borg [5]. A Kubernetes cluster is a set of master and worker nodes, where the former runs control plane components, such as the API server and the controller manager, and the latter runs user applications. Kubernetes creates an abstraction layer between application containers and infrastructure, enabling users to focus on new features and how their applications behave, rather than how the applications will run.

2 OUR SOLUTION

The application consists of an Angular 9 frontend, a Python 3.7 with Flask backend, and MariaDB as the database. The backend handles sensitive user data, such as pictures, addresses, phone numbers, and interactions the user had with the application. It can also reset passwords and send emails.

With all that said, there is a huge risk of admin-level attacks for this use case, since in standard application development and deployment infrastructure professionals such as system admins, DevOps engineers, and software developers in some cases, have access to databases, network traffic, and virtual machines or containers where sensitive data is exposed to them.

The idea is to prevent system admins to analyze the code using encryption at rest, to look into the application's memory using RAM encryption, and to modify the application using remote attestation. The database is also protected by creating and storing a new password for it. This password is a random string generated by the backend in the first run, then encrypted and stored in the container's storage volume. By creating the password inside the SGX enclave, it will never be exposed to the operating system and, consequently, also not to the system administrator. The application also uses JWT tokens as authentication. SCONE is responsible for creating certificates for HTTPS, encrypting disk and memory, and attesting that the application was not modified. The architecture for the final solution can be found on Figure 1.

When accessing the application for the first time, the user sees a login page. There, the user can fill their login information if they already have an approved registration. In case they do not, there is a link which leads to the registration page. There is also another link to reset their password in case they have forgotten it.

On the registration page, information such as email, phone number, address, professional email, job title, account type, and picture, needs to be filled. Images are stored as BLOB types in the database. There are two account types: users and operators. Users and operators can add their picture and edit their registered information. Operators can approve or deny a user registration.

2.1 Architectural downsides

Because the database password is stored in the container volume, at first it is not possible to scale the application over one replica. This could be fixed by mounting the encrypted code inside each container at run time, which is possible using various techniques such as simply copying it from a secure storage or using NFS.

Another downside of the database password local storage is that the code cannot be modified after it is first run, meaning it is not possible to apply patches or add new functionalities to the code once the database password is set without losing the data stored in the database. This can be mitigated by creating a separate service responsible only for setting and storing the database password in a safe way, and to provide this password only to authorized and attested applications or by configuring the session to trust code signed by an specific entity, such as the trusted developers or the Continuous Deployment pipeline.

3 EVALUATION

In order to find the possible trade-offs involved in adding SCONE to the application, security and performance tests were developed and ran against two versions of the application: with and without SCONE.

The evaluation process for the application is divided in two main sections: security and performance. For security, the capacity of adding malicious code and exposing sensitive data is evaluated. For performance, the average response time is evaluated.

3.1 Security

For security evaluation, a series of different possible admin-level attacks were chosen to test how the application reacts to all of them.

The first and most simple one is to run a terminal inside the container and run malicious Python code inside of it. The second was to get the database credentials from the container's environment variables and connect to the database. The third was to dump and read the container's memory.

3.1.1 No SCONE:. Running Python code inside the container

It is possible to run the Python CLI or any Python script from inside the container.

```
root@135c44e63f03:/app# python3
Python 3.7.12 (default, Sep  8 2021, 01:20:16)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Figure 2: Python CLI running inside the unprotected container

This enables the attacker to run any command, read sensitive information, or creating backdoors, taking full control of the application.

3.1.2 No SCONE:. Connecting to the database

It is possible to retrieve the database credentials from the container and connect to it.

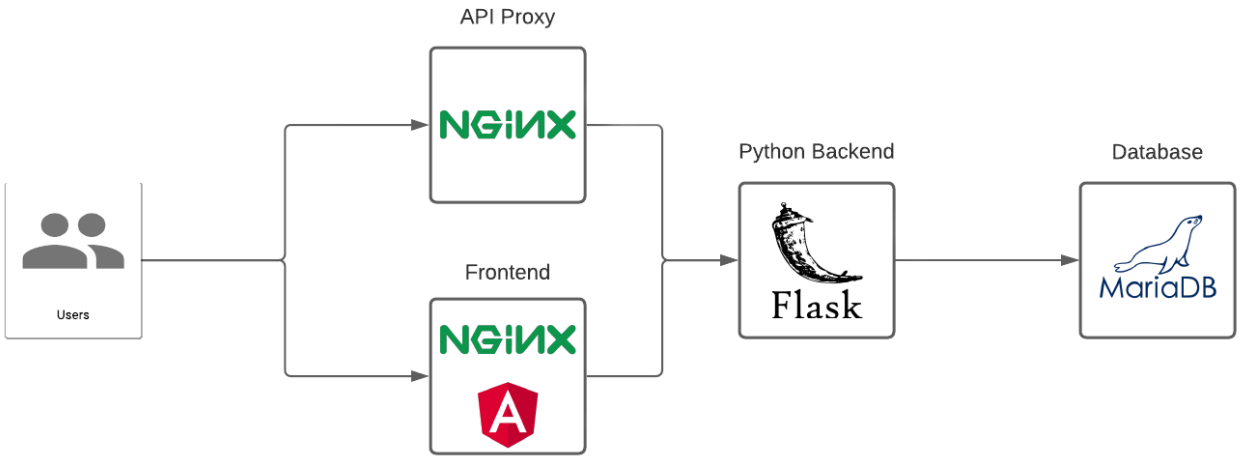

Figure 1: Architecture of the final solution

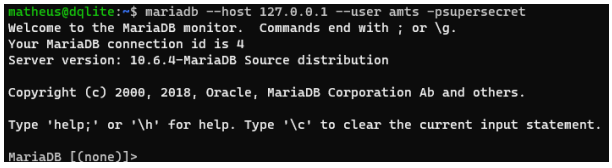

Figure 3: Successful connection to the database

This enables the attacker to read sensitive information or editing access control policies, basically taking full control of the application.

3.1.3 No SCONE:. Dumping application memory
It is possible to dump and read the application memory.

Figure 4: Readable text in the memory dump

This enables the attacker to continuously dump the memory for further inspection, potentially exposing sensitive data that is passing through the application.

3.1.4 SCONE enabled: Running Python code inside the container
When trying to run the Python CLI, an attestation error happens.

Figure 5: Attestation error when trying to run the Python CLI

The attestation error will happen for every command ran outside of the SCONE session, for which only attested code have access. This prevents malicious code to be executed inside of the container.

3.1.5 SCONE enabled: Connecting to the database

Figure 6: Attestation error while trying to read the password file

When trying to recover the new database password, an attestation error happens. This is due to SCONE's disk protection.

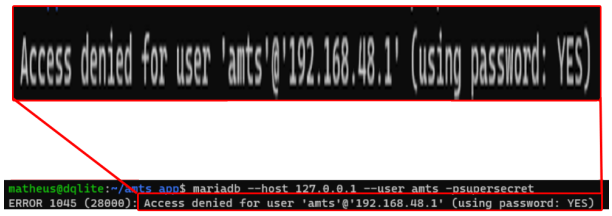

Figure 7: Access denied while trying to connect to the database

When trying to connect to the database with the default credentials, we get an access denied error. This is due to the application configuration of changing the password on the first run, securing the database access.

3.1.6 SCONE enabled: Dumping application memory

It is still possible to dump the memory, but since it is completely encrypted it is impossible to read it.

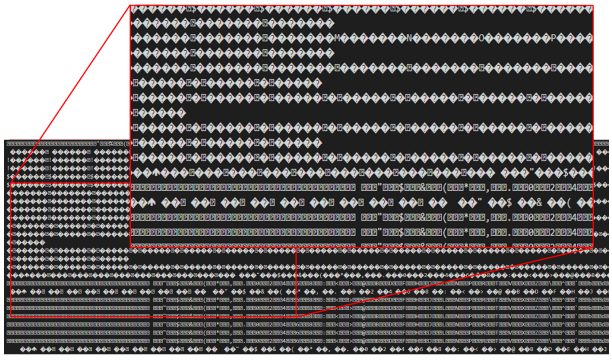

Figure 8: Unreadable data in the memory dump

SCONE encrypts the application memory, so only attested code can read it or write to it.

3.2 Performance

A Python script was developed for performance tests. This script registers users, 50 by 50, and measures response times for a request to the get all users API endpoint. It also stores CPU and RAM usage during the test.

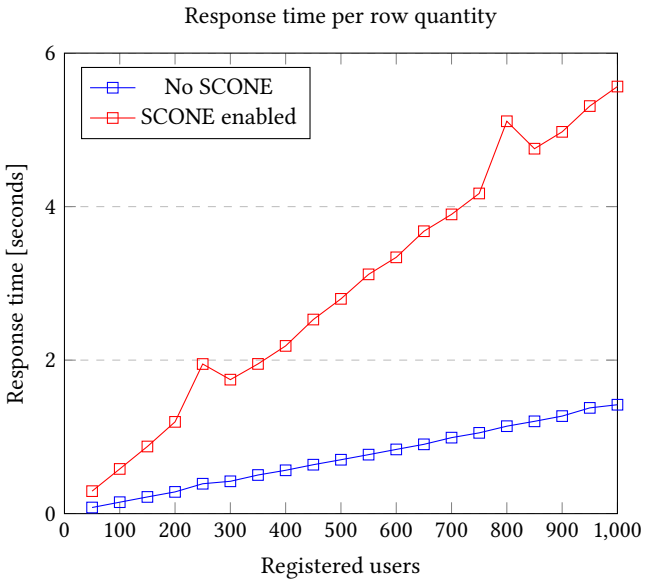

Figure 9: Response time comparison

As noticed in the response times results, the API latency increases as the amount of results to be returned rises. However, it is noticeable that the response times have a much steeper rise with SCONE than without it. This might be related to the process of memory encryption and decryption that SCONE brings to make the application more secure.

This fact can create serious scalability problems when our application must provide a substantial number of results within a single endpoint, but can be circumvented by implementing result limits and pagination.

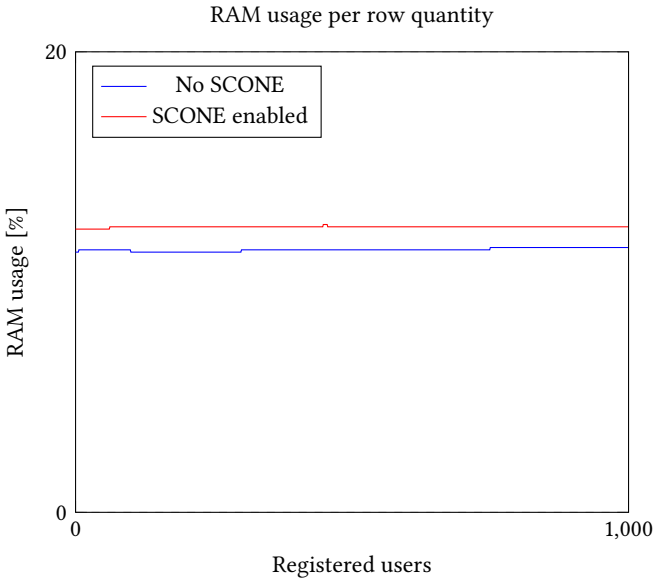

Figure 10: Memory usage comparison

During the tests, the application used around 7.3% more RAM with SCONE than without it. This can also cause scalability problems when the application faces much more parallel traffic than what is presented in the performance tests.

In the tests, the CPU results did not present any significant difference between both modes.

4 CONCLUSIONS

SCONE is a very innovative addition to the application security debate. With a previous container technology knowledge and little documentation reading and hands-on testing, one can easily make their application more secure when it comes to admin-level attacks. The level of security SCONE and Intel SGX brought to the application is noticeable. With the implementation of the two technologies and the use of well-known security techniques, it is possible to create applications which not even its creators can exploit, but it might not be the silver bullet.

The results showed that SGX and SCONE add a significant overhead to Python applications, but that might not be the case for every hardware. Further tests comparing different Intel processors generations are needed since this paper focused on bringing a comparison between two versions of the same application running on the same Intel system. Recent developments[4] showed improvements on SGX running on modern Intel processors.

SGX is suitable for a variety of use cases like internal applications, source code protection in on-premise installations, and to secure publicly available images. SCONE also has a lot of potential to become the security standard for admin-level attacks and can become much more widespread when the trade-offs are reduced or mitigated.

REFERENCES

[1] [n.d.]. Why Human Error is #1 Cyber Security Threat to Businesses in 2021. https://thehackernews.com/2021/02/why-human-error-is-1-cyber-security.html. Accessed: 2021-09-10.

[2] Sergei Arnautov Bohdan Trach Franz Grego Thomas Knauth Andre Martin Christian Priebe Joshua Lind Divya Muthukumaran Dan O'Keeff Mark L Stillwell David Goltzsch Dave Eyers Rüdiger Kapitza Peter Pietzuch Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGXg. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16).*

[3] Twaha Fuko. 2019. Intel Software Guard Extensions (SGX) Explained.

[4] Simon Johnson Raghunandan Makaram Amy Santoni Vinnie Scarlata. [n.d.]. Supporting Intel® SGX on Multi-Socket Platforms.

[5] Abhishek Verma Luis Pedrosa Madhukar R. Korupolu David Oppenheimer Eric Tune John Wilkes. 2015. Large-scale cluster management at Google with Borg. *European Conference on Computer Systems.*