Um Mecanismo de aprovisionamento de Identidades para Microsserviços Baseado na Integridade do Ambiente de Execução

Marcela Tassyany[1], Ramon Sarmento[1], Eduardo Falcão[2], Reinaldo Gomes[1], Andrey Brito[1]

[1]Universidade Federal de Campina Grande
Departmento de Sistemas e Computação, Campina Grande, Brasil

[2]Universidade Federal do Rio Grande do Norte
Departamento de Engenharia de Computação e Automação, Natal, Brasil

{marcela,ramon.sarmento}@lsd.ufcg.edu.br, eduardo@dca.ufrn.br

{reinaldo,andrey}@computacao.ufcg.edu.br

Abstract. *This work proposes a new mechanism for microservices identity provisioning based on integrity evidences. The proposed solution relies on the verification of containers' integrity, building a chain of trust based on a TPM chip. We implemented the mechanism as an attestation plugin, following a standard framework for issuing identities. Evaluations were carried out to analyze the overhead imposed on the system. The results show the efficient performance of the solution, with an average time for provisioning an identity in* $252\ ms$*, considering 150 microservices being attested in parallel in the same node.*

Resumo. *Este trabalho propõe um novo mecanismo para aprovisionamento de identidade para microsserviços baseado em evidências de integridade. A solução proposta se baseia em um mecanismo de verificação de integridade para contêineres, cuja cadeia de confiança é construída a partir de um chip TPM. Implementamos o modelo como um plugin em um framework de atestação, seguindo um padrão para emissão de identidades. Avaliações foram realizadas a fim de analisar a sobrecarga imposta ao sistema. Os resultados evidenciam o desempenho eficiente da solução, com o tempo médio de aprovisionamento de uma identidade em* $252\ ms$*, considerando 150 microsserviços sendo atestados paralelamente em um mesmo nó.*

1. Introdução

Ao longo dos anos as tradicionais arquiteturas monolíticas têm sido migradas rapidamente para arquiteturas cada vez mais distribuídas, heterogêneas e modularizadas em microsserviços [Taibi et al. 2020]. Essa transição tem demandado o aprovisionamento de mecanismos de segurança que atendam as particularidades deste cenário emergente. Nesse sentido, o mercado e a comunidade científica têm despendido esforços a fim de propor novas possibilidades para soluções em segurança, entre elas para identificação segura de microsserviços [Yarygina and Bagge 2018].

Sabe-se que a autenticidade é um dos principais pilares em segurança da Tecnologia da Informação (TI). Mecanismos de aprovisionamento de identidade formam uma

linha de frente na proteção dos diferentes elementos da arquitetura de microsserviços. De acordo com Hannousse (2020), as principais soluções usadas para identificar serviços estão relacionadas ao uso de certificados, gerenciamento de controle de acesso centralizado (do inglês, *Centralized Access Control Manager*), login único (SSO, do inglês *Single Sign On*), *Open ID*, entre outros.

Em geral, microsserviços são disponibilizados em contêineres. A virtualização de contêineres tem como particularidade empregar o compartilhamento de *kernel*. Isso torna a alocação de recursos mais ágil, mas gera vulnerabilidades de segurança [Sultan et al. 2019]. No estudo realizado por Lin et al. (2018), foram identificados 50 *exploits* que conseguem explorar as vulnerabilidades de contêineres Docker em sua configuração padrão. Logo, essas fragilidades se tornam impasses frente à adoção de contêineres, e por consequência de microsserviços.

Diferentes tecnologias, como chips TPM (*Trusted Platform Module*) e Ambientes de Execução Confiável (TEEs, do inglês *Trusted Execution Environment*) [Sabt et al. 2015], possibilitam atestar a integridade de um ambiente de execução de forma segura. Para mostrar a viabilidade da nossa proposta usamos o TPM, um coprocessador criptográfico, barato, seguro e que vem acoplado na maioria dos servidores modernos [Arthur and Challener 2015]. Trabalhos recentes detalham como verificar a integridade de contêineres, criando para isto uma cadeia de confiança que tem como raiz o TPM [Luo et al. 2019, De Benedictis and Lioy 2019]. Essas soluções permitem coletar evidências confiáveis com garantia de integridade sobre um container específico.

Neste trabalho é proposta uma nova abordagem para emissão de identidades para microsserviços baseada na integridade do ambiente de execução que hospeda o microsserviço e na integridade da própria aplicação do microsserviço. O modelo que propomos neste trabalho considera que um serviço apenas receba uma identidade se a integridade de sua aplicação e do ambiente de execução que o hospeda for comprovada.

Para tanto, foi utilizado uma proposta de padrão para emissão de identidades chamado SPIFFE (*Secure Production Identity Framework for Everyone*)[1], sua implementação oficial, SPIRE (*SPIFFE Runtime Environment*)[2], e a abordagem proposta por Luo et al. (2019) para atestação de contêineres usando TPM. Apesar da ideia de verificar a integridade de ambientes de execução não ser nova, no melhor do nosso conhecimento, sua utilização com o propósito de emissão de identidades ainda não foi explorada. A principal vantagem dessa abordagem, em comparação com a entrega de identidades baseada em alguns poucos atributos, é a verificação completa da integridade do ambiente de execução que hospeda a aplicação. Além disso, os resultados experimentais evidenciam que o processo de aprovisionamento de identidade leva décimos de segundos mesmo em cenários com uma alta quantidade de microsserviços sendo atestadas paralelamente, chegando a uma média de $252\ ms$, com 150 microsserviços sendo atestados paralelamente.

O restante deste trabalho está organizado da seguinte forma: nas Seções 2 e 3 são apresentados os conceitos básicos, úteis para a compreensão da proposta. Na Seção 4 são discutidos alguns trabalhos relacionados. Na Seção 5 é descrito o mecanismo proposto. Na Seção 6 são apresentados os principais resultados obtidos. Na Seção 7 o trabalho é

[1] `https://spiffe.io/`

[2] `https://spiffe.io/docs/latest/spire-about/`

finalizado com um resumo das principais conclusões e propostas de trabalhos futuros.

2. Emissão e Verificação de Identidades

Para tornar transparente a comunicação e autenticação entre serviços em um ecossistema dinâmico e heterogêneo, a comunidade de especialistas em segurança sentiu a necessidade de padronizar a emissão e verificação de identidades. Desta demanda surgiu o SPIFFE, uma especificação sobre como emitir identidades seguras para serviços e como verificar a autenticidade dessas identidades.

O pilar do SPIFFE é a especificação de um documento de identidade criptográfica verificável, o SVID (*SPIFFE Verifiable Identity Document*). A ideia é entregar SVIDs a aplicações que tiverem sua identidade verificada após serem submetidas a um processo de atestação. Atestação no contexto do SPIFFE diz respeito ao processo de comprovação de identidade. Durante a atestação a aplicação provê informações sobre si mesma, e estas informações são usadas no processo de emissão do SVID. Cada SVID possui um SPIFFE ID, uma cadeia de caracteres que identifica de forma única uma aplicação.

A especificação SPIFFE possui uma série de implementações para emissores de identidade, como o SPIRE, Istio, HashiCorp Consul, e Kuma, além de implementações de consumidores (ou verificadores) de identidade, como os *proxies* Ghostunnel e Envoy[3]. Por ser considerada a implementação de referência do SPIFFE, neste trabalho usamos o SPIRE para implementar o mecanismo de emissão de identidades baseado na integridade do ambiente de execução do microsserviço.

2.1. SPIRE

O SPIRE verifica a identidade de ambientes de execução de aplicação e a identidade das aplicações em execução para fornecer SVIDs com segurança. No vocabulário SPIFFE, ambientes de execução são comumente referidos por "nós", e o termo aplicação pode ser referido por "carga de trabalho".

Os principais componentes do SPIRE[4] são o *servidor* e o *agente*. O servidor cria, gerencia e fornece identidades. Ele armazena entradas de registro que contêm as informações para definir quais condições devem ser atendidas para que uma identidade seja emitida. Os agentes são executados no mesmo nó que as cargas de trabalho, e são responsáveis por expor uma API que podem ser usadas por elas para solicitar suas respectivas identidades.

Para emitir identidades, é necessário criar uma cadeia de confiança que tem como raiz o servidor e se estende ao agente. O servidor é naturalmente considerado confiável, pois é implantado pelo administrador em uma máquina de sua confiança. O servidor é configurado com um certificado confiável, e este certificado é a raiz da cadeia de confiança. Para que essa cadeia de confiança seja estendida a um agente, é necessário verificar a identidade do nó que hospeda o agente. Essa verificação de identidade é considerada importante pois ao contrário do servidor, os agentes podem ser implantados em nós providos por terceiros, portanto, considerados não confiáveis. O processo de verificação de identidade, comumente referido por atestação, é descrito a seguir.

[3] `https://spiffe.io/docs/latest/spiffe-about/overview/`

[4] `https://spiffe.io/docs/latest/spire-about/spire-concepts/`

2.1.1. Atestação de Nó

A atestação de um nó consiste em verificar a identidade de um nó e emitir um SVID para o agente caso essa identidade seja comprovada. Boa parte desse processo é realizada por plugins atestadores de nó que executam tanto no servidor como no agente. Embora cada implementação de plugin possua comportamento próprio, é possível descrever a atestação de nó genericamente com os seguintes passos.

1. O agente coleta informações exclusivas do nó que sejam suficientes para comprovar sua identidade e as envia para o servidor.
2. O servidor valida as informações recebidas, cria e envia o SVID para o agente, juntamente com os registros que correspondem às cargas de trabalho que estão sob responsabilidade daquele agente.
3. Concluído o processo de atestação do nó, o agente envia para o servidor os CSRs (*Certificate Signing Request*) de cada carga de trabalho.
4. O servidor envia para o agente os SVIDs das cargas de trabalho.
5. O agente armazena os SVIDs em cache.

2.1.2. Atestação de Carga de Trabalho

Ao contrário da atestação de nó que ocorre entre servidor e agente, a atestação de carga de trabalho ocorre entre agente e carga de trabalho. O agente possui plugins atestadores de carga de trabalho que buscam informações sobre a mesma para comprovar sua identidade. Em geral, o processo de atestação de cargas de trabalho percorre os seguintes passos:

1. A carga de trabalho solicita sua identidade ao agente.
2. O agente solicita que autoridades locais coletem informações sobre a carga de trabalho. Essas autoridades dependem da plataforma subjacente e contexto no qual a carga de trabalho está executando (e.g., *kernel* do sistema operacional, ou o Kubelet, no caso do Kubernetes).
3. O agente verifica se as informações coletadas correspondem a algum registro recebido do servidor, e em caso positivo o respectivo SVID é fornecido para a carga de trabalho.

Neste trabalho empregamos o chip TPM como autoridade local para prover informações sobre a carga de trabalho. A próxima seção detalha o funcionamento do TPM e explica como a integridade das informações providas por ele são asseguradas.

3. Trusted Platform Module

Uma das funcionalidades do TPM é estabelecer confiança em um ambiente de execução remoto através da atestação remota. Esse procedimento pode ser dividido em duas etapas: i) verificar a existência de um TPM real e íntegro e ii) verificar a integridade do ambiente através da análise das medições de executáveis e arquivos de configuração reportadas[Arthur and Challener 2015].

3.1. Verificando a Autenticidade de um TPM

A existência de um TPM é verificada através de um desafio proposto por um verificador para o cliente com TPM. Em geral, o verificador criptografa um valor aleatório utilizando

a parte pública da chave de endosso (ou EK, do inglês *Endorsement Key*). Para entender a segurança do desafio, é importante compreender algumas propriedades do TPM e da EK.

Durante a manufatura, cada TPM recebe uma EK única assinada por uma Autoridade Certificadora (AC) confiável. A EK é uma chave assimétrica, e não é possível extrair sua parte privada do TPM, enquanto a parte pública pode ser obtida a partir do certificado X509 injetado na RAM não-volátil. Então, para solicitar um desafio, basta que um cliente envie o certificado X509 referente à EK.

Após extrair a EK pública do certificado x509, o verificador analisa sua autenticidade checando se a mesma foi assinada por alguma AC confiável. A assinatura é verificada usando certificados das ACs, certificados estes que são tipicamente disponibilizados publicamente pelos produtores de TPM. Se essa verificação for bem sucedida, então o verificador sabe que esta é a parte pública de uma EK cuja parte privada nunca pode ser extraída de um TPM. Portanto, o verificador criptografa algum segredo aleatório, e apenas o cliente com o TPM que possui a parte privada daquela EK será capaz de descriptografar o segredo. Finalmente, o desafio é concluído quando o cliente apresenta o segredo em texto plano ao verificador.

3.2. Verificando a Integridade de um Ambiente de Execução com TPM

Para compreender a atestação remota por TPM é necessário entender como a cadeia de confiança é ancorada ao TPM e como criar relatórios confiáveis do ambiente de execução.

O TPM contém 24 registradores, chamados de registradores de configuração de plataforma (ou PCRs, do inglês *Platform Configuration Registers*), empregados para armazenar medições. As medições são obtidas através da aplicação de algoritmos de *hashing* sobre algum conteúdo, que pode ser o binário de algum executável ou até mesmo arquivos de configuração. No entanto, cada medição não é atribuída a um PCR diretamente. Para incluir uma medição, o TPM realiza uma operação de *hash* estendido sobre o último valor do PCR. Por exemplo, se o PCR 10 for estendido com alguma medição M, então o novo valor do PCR 10 é computado da seguinte maneira: $PCR\_10 = hash(PCR\_10 + M)$.

Quando a atestação se baseia na Arquitetura de Medição de Integridade (ou IMA, do inglês Integrity Measurement Architecture), todas as medições são sumarizadas no PCR 10. No processo de inicialização, o *firmware* e toda configuração usada na inicialização são estendidos nos PCRs 0-7. Em seguida, o resultado agregado dos PCRs 0-7 é usado como base para medições do sistema operacional (SO). Portanto, a primeira medição da inicialização do SO é chamada de agregação de inicialização. Deste momento em diante, todas as aplicações do SO, incluindo bibliotecas, dependências, e arquivos de configurações importantes são medidos e estendidos no PCR 10. Logo, o PCR 10 sumariza todo o histórico sobre a integridade das aplicações carregadas em um ambiente de execução desde sua inicialização de *firmware* até a a inicialização e operação do SO.

Depois da inicialização, o primeiro passo da atestação remota é a criação de um relatório que inclui os valores dos PCRs. Antes da criação deste relatório de PCRs, deve-se estabelecer confiança entre verificador e cliente através da verificação de autenticidade de um TPM, como descrito na subseção anterior. No entanto, quando se trata de verificar a integridade de ambientes de execução, além de verificar a EK, o verificador precisa estabelecer confiança em uma chave de atestação (ou AK, do inglês *Attestation Key*). Em face

de uma AK proveniente de um TPM genuíno, o cliente poderá assinar o relatório de PCRs com essa chave de modo que a integridade do relatório de PCRs possa ser verificada.

O relatório de PCRs assegura para um verificador remoto que os valores reportados para os PCRs são íntegros. Contudo, os valores dos PCRs não são suficientes para que o verificador analise a integridade do ambiente de execução. Além do relatório de PCRs, o cliente envia um relatório de medições criado pelo IMA. Com o relatório de PCRs e o relatório de medições do IMA, o verificador consegue analisar se ambos são íntegros, e então poderá utilizar as informações do relatório de medições do IMA para verificar se alguma aplicação não autorizada executou naquele ambiente.

4. Trabalhos Relacionados

De acordo com [Hannousse and Yahiouche 2020], a maioria dos trabalhos que envolvem autenticação para microsserviços abordam a utilização de técnicas e padrões de identificação já existentes, como JWT (*JSON Web Token*) [George and Mahmoud 2017, Salibindla 2018], Certificados [Nkomo and Coetzee 2019, Yarygina and Otterstad 2018], OpenID [Nehme et al. 2019] . Diferentemente dos estudos supracitados, que se baseiam nas práticas padrões para o fornecimento de identidade, a abordagem proposta neste trabalho considera que a integridade do microsserviço é um importante atributo a ser considerado no aprovisionamento de identidade. Isso significa que, se as aplicações e o ambiente no qual são executados forem íntegros, é seguro confiar que esse sistema é quem diz ser e que não foi modificado ou utilizado indevidamente por terceiros.

A atestação remota tem sido amplamente utilizada para examinar a integridade de *hardware* e *software*. Sua implementação padrão funciona bem para plataformas físicas. No entanto, uma série de desafios são observados quando considerados cenários com nós virtuais, como VMs e contêineres [De Benedictis and Lioy 2019]. Comumentemente, aplicações que utilizam arquiteturas de microsserviços fazem uso de contêineres como plataforma de execução. Esse cenário despertou o interesse da comunidade científica em propor soluções para construção de confiança para contêineres [Berger 2016, Hosseinzadeh et al. 2016, De Benedictis and Lioy 2019, Luo et al. 2019]. Uma parte significativa desses estudos fazem uso da Computação Confiável (do inglês, *Trusted Computing*). Entre eles, uma das soluções propostas é a utilização do vTPM. O vTPM é um TPM virtual, inicialmente proposto por Perez et al. (2006), para o contexto específico de máquinas virtuais, e utilizado por Berger (2016) e Hosseinzadeh et al. (2016) para o contexto de contêineres. No entanto, essa abordagem requer a adição de mais uma camada de virtualização, bem como a modificação das imagens dos contêineres.

Nesse sentido, visando propor soluções que se sobressaiam em relação a esses impedimentos, Benedictis e Lioy (2019) propuseram o DIVE, uma abordagem para verificação de integridade de contêineres Docker. Essencialmente, os autores estenderam o IMA para que o log de medições indique a qual contêiner cada medição pertence. Para isto também foi explorado o *driver* de armazenamento de mapeamento de dispositivos, do Docker, para atribuir diferentes identificadores para processos que executam em diferentes contêineres. No entanto, o IMA continua armazenando as medições de diferentes contêineres em um único arquivo, violando a privacidade dos contêineres que não estão sendo atestados, pois no processo de atestação remota o verificador tem acesso a informações de todos os contêineres.

Visando preencher essa lacuna, Luo et al. (2019) apresentaram o Container-IMA, uma abordagem que também busca fornecer evidências de integridade para contêineres. Entretanto, diferente dos trabalhos anteriores, os autores modificaram o IMA para realizar uma divisão do arquivo de medição por *namespace*. Conjuntamente, foi proposto o cPCR, um PCR baseado em contêiner que permite garantir a integridade de cada arquivo de medição. Desta forma, a integridade do ambiente de execução de cada contêiner é assegurada por meio do cPCR, e, durante o processo de atestação remota, é possível obter informações que dizem respeito apenas ao contêiner designado.

Embora forneçam soluções para atestação de integridade de contêineres, nenhum dos trabalhos supramencionados buscam realizar o processo de emissão de identidades a partir dessas evidências de integridade. No melhor do nosso conhecimento, não foram encontrados trabalhos que realizem investigações nesse sentido.

5. Mecanismo para Aprovisionamento de Identidade

O mecanismo para aprovisionamento de identidade para microsserviços proposto neste artigo, analisa as evidências de integridade do ambiente de execução do microsserviço, i.e., o contêiner, o que envolve as aplicações que já executaram e que estão em execução no contêiner. Caso o contêiner esteja íntegro, considera-se que o sistema não foi comprometido por terceiros e, portanto, uma identidade é fornecida. Essa abordagem está considerando que o processo de atestação remota do nó que hospeda os contêineres já foi realizado, de forma que a autenticidade do TPM já foi validada e, portanto, o nó que hospeda os contêineres, por ser avaliado como íntegro, já recebeu sua identidade.

O mecanismo proposto utiliza o Container-IMA e o estende para diminuir a ocorrência de falsos negativos, i.e., atestar como corrupto um contêiner e microsserviço íntegros. O Container-IMA é um processo de verificação de integridade proposto por Luo et al. (2019). Essencialmente, Luo. et al. realizaram modificações no *kernel* do Linux para que as medições do IMA também sejam identificadas pelo namespace. Portanto, assim como os nós, cada contêiner também tem um relatório de medições de suas dependências e aplicações que foram executadas. Para proteger a integridade dos relatórios de medições de cada contêiner, os autores também implementaram um PCR virtual (armazenado em memória) para cada contêiner (cPCR). Os valores do cPCR de cada contêiner são adicionados ao PCR 12 fazendo uso da operação de *hash* estendido.

Para que seja possível recomputar o valor corrente do PCR 12, o kernel também armazena o valor anterior do PCR 12, além dos valores dos cPCRs dos contêineres iniciados no nó. Durante o processo de verificação de integridade, o valor anterior do PCR 12 é estendido com o valor de todos os cPCRs e comparado ao valor atual do PCR 12 que foi reportado no relatório de PCRs. Caso os valores correspondam, significa que é possível confiar nos valores dos cPCRs. Além disso, a fim de ter uma evidência de que a imagem do contêiner não foi modificada indevidamente, Luo. et al. também programaram o motor de execução de contêineres para capturar as medições das imagens de cada contêiner para armazená-las localmente e estendê-las no PCR 11.

No gerenciamento de *namespaces*, caso um contêiner seja parado, o *namespace* atribuído a ele pode ser reutilizado por outro contêiner. Nesse sentido, durante os testes realizados com o Container-IMA foi observado que, como o arquivo de medições estava sendo dividido pelo *namespace*, um mesmo arquivo poderia conter medições de diferentes

contêineres. Para o nosso mecanismo de aprovisionamento de identidade esse cenário poderia resultar em falsos negativos. Isto acontece pois um contêiner poderia estar íntegro, mas não receberia uma identidade pois o estado de suas medições não correspondem às expectativas devido a presença das medições de outros contêineres. Nesse sentido, a implementação original foi modificada para que o arquivo de medições fosse dividido pelo identificador do contêiner. Por ser um identificador exclusivo, cada relatório de medições conterá apenas medições de seus respectivos contêineres.

O mecanismo de aprovisionamento de identidade é iniciado apenas se o nó que hospeda os contêineres já tiver sido atestado remotamente via TPM. Nesse sentido, o aprovisionamento de identidade de cargas de trabalho não tem a participação de um verificador remoto. Conforme ilustrado no diagrama de sequência apresentado na Figura 1, o principal componente do mecanismo é o agente, que ao receber uma solicitação, coleta as informações necessárias junto ao nó que hospeda os contêineres, realiza as verificações de integridade necessárias e toma as decisões quanto ao fornecimento da identidade.

O processo inicia com a carga de trabalho solicitando sua identidade para o agente. Em face dessa requisição, o agente coleta no nó as seguintes informações: 1) valores correntes do PCR 11 e PCR 12; 2) último valor do PCR 12; 3) lista com os valores dos cPCRs; 4) arquivo com as medições do contêiner solicitante; e 5) arquivo com as medições das imagens dos contêineres.

De posse dessas informações, o agente executa operações de *hash* estendido sobre as medições das imagens dos contêineres, obtendo uma medição que sumariza todas as imagens de contêineres (MIC). A MIC obtida é comparada com o valor do PCR 11. Caso os valores sejam correspondentes, significa que é possível confiar na integridade das medições das imagens dos contêineres.

Uma vez que já confirmou a confiabilidade da imagem do contêiner, o agente pode verificar a integridade dos conêineres. Para tanto, são realizadas operações de *hash* estendido sobre o último valor do PCR 12 e cPCRs, obtendo uma medição que sumariza o estado corrente dos contêineres (MEC). A MEC é comparada com o valor do PCR 12 e, caso correspondam, conclui-se que o valor do cPCR do contêiner solicitante é íntegro e, portanto, pode ser utilizado para validar a integridade das medições do contêiner que solicitou a atestação.

O próximo passo do agente é executar operações de *hash* estendido sobre as medições do contêiner solicitante para computar uma medição que sumariza o estado corrente do contêiner em atestação (MECA). Para decidir se deve confiar nas medições reportadas pelo contêiner, o agente compara o MECA com o cPCR do contêiner em atestação.

Uma peculiaridade do MECA é que ele inclui atributos específicos do ciclo de vida de um contêiner, como por exemplo seu identificador, e por isso o MECA não é genérico e representativo para atestar contêineres arbitrários criados com a mesma imagem. Portanto, o agente computa um MECA genérico (MECA-g), e compara o MECA-g com um valor de referência que representa o estado esperado do contêiner, que chamamos de GID (do inglês *Golden Image Digest*). Caso os valores correspondam, significa que o contêiner e as aplicações em execução estão em um estado confiável e, portanto, uma identidade é fornecida à aplicação solicitante.

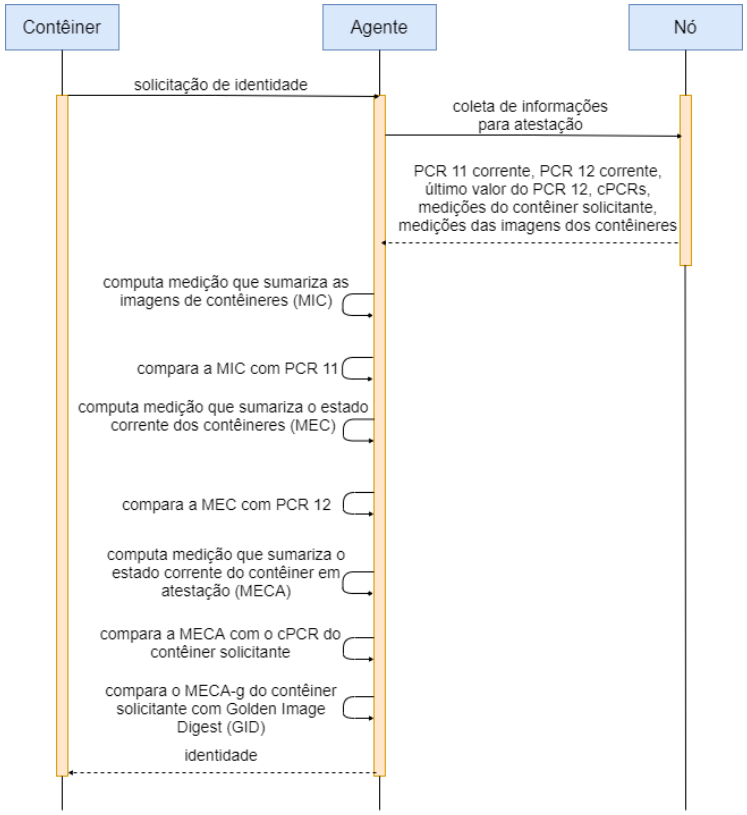

Figura 1. Processo de aprovisionamento de identidade para carga de trabalho.

5.1. Implementação

A implementação do Container-IMA estava disponível para a versão 3.13 do *kernel* Linux, sendo possível utilizá-lo apenas com Ubuntu 14. A solução foi reimplementada com as alterações descritas na Seção 5 na versão 4.19 do *kernel*, para dar suporte ao Ubuntu 16[5]. Ademais, o mecanismo de aprovisionamento de identidade foi implementado como um plugin de atestação de carga de trabalho no SPIRE. Atualmente, o SPIRE dispõe de plugins para atestação de carga de trabalho para Unix[6], Kubernetes[7] e Docker[8]. Nenhum deles se baseia em verificações de integridade para o aprovisionamento de identidade.

6. Avaliação

Nesta seção serão descritos os detalhes da avaliação da solução proposta. Será apresentado o ambiente de experimentação utilizado, a descrição do experimento realizado, bem como as métricas coletadas.

Para realização do experimento, foram criadas duas VMs, ambas com Ubuntu 16.04, 1vCPU, 3GB de memória principal e 30GB de disco. Nessas VMs foi utilizado o

[5]A reimplementação para a versão LTS mais recente (20.04) exigirá mais algumas adaptações

[6]`https://github.com/spiffe/spire/blob/master/doc/plugin_agent_workloadattestor_unix.md`

[7]`https://github.com/spiffe/spire/blob/master/doc/plugin_agent_workloadattestor_k8s.md`

[8]`https://github.com/spiffe/spire/blob/master/doc/plugin_agent_workloadattestor_docker.md`

SWTPM[9] para emular um TPM, com o intuito de representarem o comportamento do nó que hospeda os contêineres. A diferença entre as VMs é que uma delas utiliza o *kernel* 4.19 sem qualquer modificação, e a outra o *kernel* 4.19 com as modificações realizadas para dar suporte a verificação de integridade para contêineres.

6.1. Análise do Impacto das Modificações no *kernel* Linux

A primeira parte do experimento tem como objetivo avaliar a sobrecarga imposta pelas alterações realizadas no *kernel* durante a inicialização dos contêineres. Apesar de uma análise semelhante já ter sido realizada por Luo et al. durante a avaliação do Container-IMA, uma nova análise se faz necessário devido à utilização de uma versão diferente do *kernel*, além das modificações realizadas, descritas na Seção 5.

Durante o experimento variamos o número de contêineres iniciados paralelamente entre 25, 50, 100 e 150 e aferimos o tempo de inicialização do contêiner tanto no cenário com *kernel* original, como no cenário com o *kernel* modificado. Para realizar as medições foi utilizado o Docker Micro Benchmark[10], um *benchmark* utilizado para avaliações de desempenho de contêineres Docker.

Tabela 1. Diferença média no tempo de inicialização dos contêineres.

nº de contêineres	*kernel* original	*kernel* modificado	diferença	diferença percentual
25	1,53	1,93	0,40	25,93%
50	1,43	1,95	0,52	36,65%
100	1,44	1,93	0,49	34,19%
150	1,54	2,19	0,65	42,58%

Observando a Tabela 1 e a Figura 2 é possível comparar o tempo de inicialização dos contêineres para os *kernels* original e modificado. Os resultados evidenciam que o *kernel* modificado implica em um maior tempo de iniciliazação, entretanto, mesmo a diferença percentual sendo significativa, o aumento absoluto observado no tempo pode ser considerado marginal, uma vez que foi de cerca de meio segundo, e só acontece uma vez, durante a inicialização do contêiner.

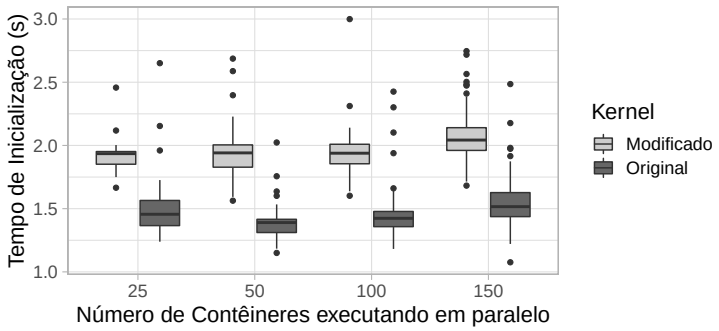

Figura 2. Tempo de inicialização dos contêineres.

[9] https://github.com/stefanberger/swtpm
[10] https://github.com/Random-Liu/docker_micro_benchmark

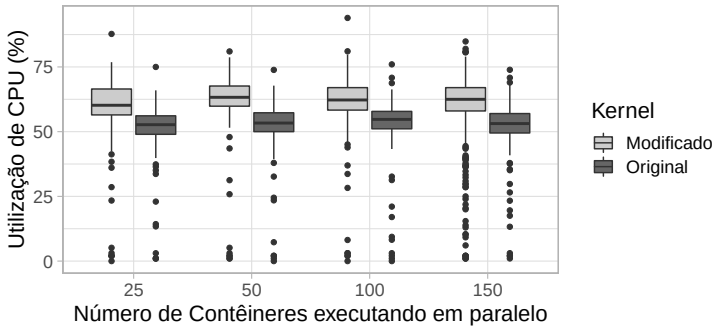

Figura 3. Percentual de utilização de CPU para os *kernels* modificado e original.

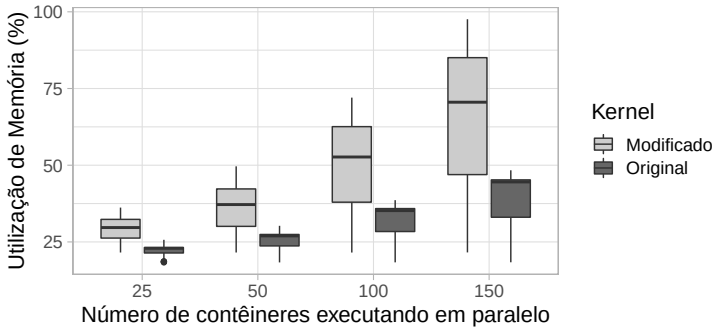

Figura 4. Percentual de uso de memória para os *kernels* modificado e original.

No que diz respeito a utilização de CPU, é possível observar na Figura 3 que o *kernel* modificado apresenta em média um leve aumento em relação ao *kernel* original. Entretanto, não é possível afirmar estatisticamente que essa diferença exista devido a variabilidade dos dados. Ademais, no que se refere a utilização de memória, expresso percentualmente na Figura 4, os resultados sugerem que o *kernel* modificado apresentou um maior consumo se comparado ao *kernel* original. Isso acontece devido a adição das operações de medição das aplicações e arquivos de configuração dos contêineres, pois para que sejam medidos eles naturalmente precisam ser carregados na memória. Todavia, vale ressaltar que, o ambiente de experimentação utilizado foi uma VM com apenas 3GB de memória principal. Espera-se que em ambientes de produção as modificações realizadas no *kernel* tenham um impacto menor devido à maior quantidade de recursos.

6.2. Análise do Impacto do Mecanismo de aprovisionamento de Identidade

A segunda parte do experimento tem como objetivo avaliar a sobrecarga imposta pelo mecanismo de aprovisionamento de identidade. Para isso, o SPIRE 0.12.1 foi configurado para executar com o nosso plugin de atestação de carga de trabalho.

O experimento consistiu em medir a latência que o mecanismo de atestação introduz na aplicação em execução, medindo o tempo decorrido entre a solicitação e recebimento da identidade pelos contêineres. A latência foi aferida considerando a execução de contêineres em paralelo. Assim como no cenário descrito na Seção 6.1, o número de

contêineres variou entre 25, 50, 100 e 150. Cada contêiner executa um cliente HTTP que, para se comunicar com o servidor HTTP, precisa antes solicitar sua identidade para que o servidor o reconheça e processe a requisição do cliente.

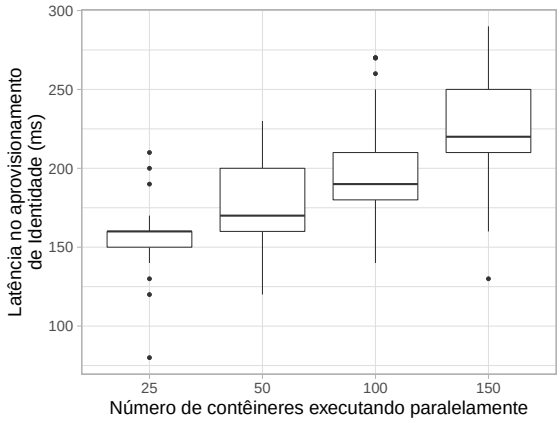

Figura 5. Latência do Mecanismo de aprovisionamento de Identidade

No geral, provisionar uma identidade implica em uma latência na ordem de décimos de segundo. Quando consideramos apenas um contêiner, esse valor corresponde em média a $143\ ms$. Como pode ser observado na Figura 5, os resultados sugerem uma tendência no aumento da latência em função do acréscimo no número de contêineres. Levando em consideração os resultados mencionados na Seção 6.1, esse comportamento provavelmente se dá devido ao aumento da utilização de recursos. Isso sugere que esse acréscimo diz respeito mais à infraesturura subjacente do que às limitações de escalabilidade do mecanismo. Além disso, o próprio uso do SPIRE pode representar um aumento na utilização de memória. Na Figura 6 observa-se que há uma tendência de maior utilização de memória quando utilizado o SPIRE, no entanto, não é possível afirmar com certeza que esse acréscimo sempre aconteça devido a variabilidade dos dados.

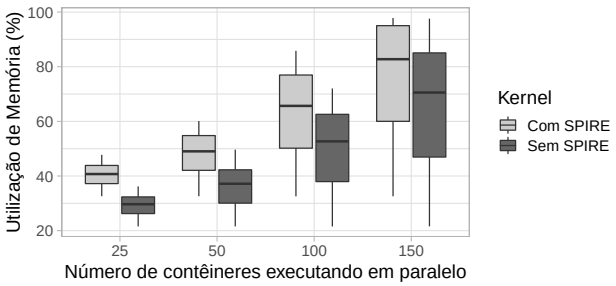

Figura 6. Percentual de Utilização de Memória para os cenários com e sem SPIRE

7. Conclusão

Este trabalho apresenta um novo mecanismo para aprovisionamento de identidade para microsserviços. O mecanismo se baseia em evidências de integridade do ambiente de execução (contêiner) e das aplicações que compõem o microsserviço. A raiz de confiança para assegurar essas evidências é construída a partir de um módulo de computação

confiável: o TPM. Nesse sentido, microsserviços só receberão identidades caso comprovem sua integridade, evidenciando que o sistema está executando como esperado e nenhuma modificação indevida foi realizada. Assim, por meio desse mecanismo, microsserviços podem atestar sua identidade para outras aplicações com as quais precisarem interagir.

A solução foi implementada como um plugin no SPIRE, sistema extensível que implementa o SPIFFE, uma especificação para padronizar a emissão de identidades, e que recentemente tem atraído o interesse de diferentes organizações. Os resultados das avaliações sugerem que as modificações realizadas no *kernel* para dar suporte à verificação de integridade de contêineres, adicionam um custo marginal de em média $0.5\ s$ ao tempo de inicialização do contêiner.

Adicionalmente, foi observado um acréscimo na utilização de memória no ambiente de experimentação utilizado, porém, esse acréscimo tende a ser menor em um ambiente de produção devido à maior quantidade de recursos. Também foi observada uma pequena sobrecarga na inicialização do contêiner, que é aceitável considerando a segurança acrescentada. Em suma, os resultados das avaliações quanto ao mecanismo de aprovisionamento de identidade demonstram que a solução proposta adiciona maior segurança com baixo custo no desempenho das aplicações.

Trabalhos futuros podem envolver a implementação desse mecanismo usando ambientes de execução confiáveis (TEEs). De modo geral, além da mesma segurança provida na atestação remota, TEEs podem prover camadas de segurança adicionais graças à execução da aplicação em partes isoladas da memória. Alguns esforços já vêm sendo feitos no sentido de portar aplicações [da Silva et al. 2019] para TEEs como o Intel SGX, e o desenvolvimento de microsserviços que executam em TEEs tem se tornado mais simples graças ao desenvolvimento de frameworks como o SCONE [Arnautov et al. 2016].

Agradecimentos

Este trabalho foi financiado através do projeto ZTPO, uma colaboração entre a Hewlett Packard Enterprise (Brasil), com recursos da Lei de Informática (Lei 8.248 de 23/10/1991), e a unidade CEEI-EMBRAPII na Universidade Federal de Campina Grande.

Referências

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M. L., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., and Fetzer, C. (2016). SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703, Savannah, GA. USENIX Association.

Arthur, W. and Challener, D. (2015). *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature.

Berger (2016). Virtual tpm proxy driver for linux containers. `https://www.kernel.org/doc/html/v4.10/security/tpm/tpm_vtpm_proxy.html`. acessado em 08/04/2021.

da Silva, M. S. L., de Oliveira Silva, F. F., and Brito, A. (2019). Squad: A secure, simple storage service for sgx-based microservices. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–9.

De Benedictis, M. and Lioy, A. (2019). Integrity verification of docker containers for a lightweight cloud environment. *Future Generation Computer Systems*, 97:236–246.

George, V. M. and Mahmoud, Q. H. (2017). Claimsware: A claims-based middleware for securing iot services. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 649–654. IEEE.

Hannousse, A. and Yahiouche, S. (2020). Securing microservices and microservice architectures: A systematic mapping study. *arXiv preprint arXiv:2003.07262*.

Hosseinzadeh, S., Laurén, S., and Leppänen, V. (2016). Security in container-based virtualization through vtpm. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 214–219.

Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., and Zhou, Q. (2018). A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 418–429.

Luo, W., Shen, Q., Xia, Y., and Wu, Z. (2019). Container-ima: a privacy-preserving integrity measurement architecture for containers. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, pages 487–500.

Nehme, A., Jesus, V., Mahbub, K., and Abdallah, A. (2019). Securing microservices. *IT Professional*, 21(1):42–49.

Nkomo, P. and Coetzee, M. (2019). Software development activities for secure microservices. In *International Conference on Computational Science and Its Applications*, pages 573–585. Springer.

Perez, R., Sailer, R., van Doorn, L., et al. (2006). vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium*, pages 305–320.

Sabt, M., Achemlal, M., and Bouabdallah, A. (2015). Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE.

Salibindla, J. (2018). Microservices api security. *International Journal of Engineering Research & Technology*, 7(1):277–281.

Sultan, S., Ahmad, I., and Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996.

Taibi, D., Lenarduzzi, V., and Pahl, C. (2020). Microservices anti-patterns: A taxonomy. In *Microservices*, pages 111–128. Springer.

Yarygina, T. and Bagge, A. H. (2018). Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20. IEEE.

Yarygina, T. and Otterstad, C. (2018). A game of microservices: Automated intrusion response. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 169–177. Springer.