PUCRS

FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

MATHEUS DUARTE VASCONCELOS

UMA ARQUITETURA DE SEGURANÇA PARA SISTEMAS EMBARCADOS VIRTUALIZADOS

Porto Alegre
2017

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ARQUITETURA DE SEGURANÇA PARA SISTEMAS EMBARCADOS VIRTUALIZADOS

MATHEUS DUARTE VASCONCELOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fabiano Passuelo Hessel

Porto Alegre
2017

Ficha Catalográfica

V331 Vasconcelos, Matheus Duarte

Uma Arquitetura de Segurança para Sistemas Embarcados Virtualizados / Matheus Duarte Vasconcelos . – 2017.
79 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fabiano Passuelo Hessel.

1. Segurança. 2. Sistemas Embarcados. 3. Virtualização. 4. IoT. I. Hessel, Fabiano Passuelo. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Matheus Duarte Vasconcelos

Uma Arquitetura de Segurança para Sistemas Embarcados Virtualizados

Tese/Dissertação apresentada como requisito parcial para obtenção do grau de Doutor/Mestre em Ciência da Computação do Programa de Pós-Graduação em Ciencia da Computação, Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 31 de Agosto de 2017.

BANCA EXAMINADORA:

Prof. Dr. Jorge Barbosa (Unisinos)

Prof. Dr. Alexandre de Moraes Amory (PPGCC/PUCRS)

Prof. Dr. Fabiano Passuelo Hessel (PPGCC/PUCRS - Orientador)

“O sucesso é ir de fracasso em fracasso sem perder entusiasmo.”
(Winston Churchill)

AGRADECIMENTOS

Gostaria de expressar minha mais sincera gratidão a todos que me ajudaram, direta ou indiretamente, durante o período de mestrado. Primeiramente, gostaria de agradecer ao Prof. Fabiano Passuelo Hessel, antes de tudo pela confiança em mim depositada, pelos ensinamentos e conselhos dados. Também agradeço ao Prof. Avelino Francisco Zorzo pelos ensinamentos passados a mim. À instituição por proporcionar um ambiente criativo e amigável. Agradeço também aos colegas do GSE/PUCRS, em especial ao Carlos Moratelli pelas orientações e apoio durante esta pesquisa.

Agradeço também a minha esposa, Aline Vasconcelos, pessoa com quem escolhi passar o resto de minha vida, pelo amor, cumplicidade, paciência e apoio incondicional durante este período. Agradeço também aos meus pais por todo amor e carinho durante toda minha vida.

UMA ARQUITETURA DE SEGURANÇA PARA SISTEMAS EMBARCADOS VIRTUALIZADOS

RESUMO

Historicamente os sistemas embarcados (SE) eram desenvolvidos para realizar uma única tarefa em toda a sua vida. Entretanto, esta visão mudou com o novo paradigma da computação chamado Internet das Coisas ou IoT. Um ambiente onde a IoT pode ser aplicada são as cidades inteligentes por meio da criação de produtos como, por exemplo, os postes inteligentes. Assim, os postes inteligentes podem ser responsáveis não só pela iluminação da cidade, mas encarregados também pelo controle de câmeras de segurança, além de sensores de temperatura e ruído. Neste cenário, a técnica de virtualização em SE surge para contribuir no desenvolvimento de dispositivos IoT, pois permite uma melhor utilização dos recursos disponíveis nos SE além de auxiliar para o aumento da segurança. A segurança dos SE tem sido negligenciada e os SE voltados para IoT têm atraído ataques maliciosos, visto que, desempenham um papel central no funcionamento de serviços essenciais para as pessoas e empresas. O objetivo deste trabalho é identificar um conjunto de mecanismos de segurança que utilizam técnicas de criptografia que, combinados com a técnica de virtualização, possam estabelecer uma arquitetura de segurança para os SE virtualizados (SEV) voltados para IoT. Assim, estabelecendo um nível de confiança mínimo entre os usuários e os SEV. Além disso, foram implementados dois mecanismos de segurança no prplHypervisor: a verificação de integridade e a introspecção das hypercalls do sistema convidado. Os resultados mostram que para um sistema convidado com tamanho de 256kB o mecanismo de verificação de integridade impôs um tempo de atraso na inicialização de 150,33ms enquanto o mecanismo de introspecção impôs 10,57ms de atraso na inicialização. Foram adicionados 2.029 linhas de código ao prplHypervisor para realizar a verificação de integridade e 120 linhas de código para implementar o mecanismo de introspecção. O tamanho final do prplHypervisor possui 32kB o que representa um aumento de 53% em

relação ao código original. Todavia, o crescimento não inviabiliza o uso dos mecanismos de segurança, dado que, a capacidade de armazenamento disponível na plataforma utilizada é de 2MB.

Palavras-Chave: Segurança, Sistemas Embarcados, Virtualização, IoT.

AN SECURITY ARCHITECTURE FOR VIRTUALIZED EMBEDDED SYSTEMS

ABSTRACT

Historically embedded systems (ES) were designed to perform a single task throughout their lifetime. However, this view has changed with the new paradigm of computing called the Internet of Things or IoT. An example of environment where IoT can be applied are smart cities by creating products such as smart poles. Thus, smart poles can be responsible not only for city lighting, but also for the control of security cameras, in addition to temperature and noise sensors. In this scenario, the virtualization technique in ES appears to contribute to the development of IoT devices since it allows a better use of the available resources in the ES besides contributing to the increase of the security. ES security has been neglected and IoT oriented ES have attracted malicious attacks as they play a central role in the operation of essential services for individuals and enterprises. Therefore, the objective of this work is to identify a set of security mechanisms that use cryptography techniques that, combined with the virtualization technique, can establish a security architecture for IoT oriented virtualized ES (VES). Thus, establishing a minimum level of confidence between the users and the SEV. Two security mechanisms have been implemented in prplHypervisor: integrity checking and introspection of guest system hypercalls. The results show that for a guest system with a size of 256kB the integrity check mechanism imposed a 150.33ms initialization delay time while the introspection engine imposed 10.57ms of initialization delay. 2,029 lines of code have been added to the prplHypervisor to perform the integrity check and 120 lines of code to implement the introspection engine. The final size of the prplHypervisor has 32kB which represents a 53% increase over the original code. However, growth does not prevent the use of security mechanisms since the storage capacity available on the platform is 2MB.

Keywords: Secutiry, Embedded System, Virtualization, IoT.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

VMM – Virtual Machine Monitor

IOT – Internet Of Things

SE – Sistemas Embarcados

GSE – Grupo de Sistemas Embarcados

PUCRS – Pontifícia Universidade Católica do Rio Grande do SUL

VM – Virtual Machine

ABI – Application Binary Interface

RFID – Radio-Frequency IDentification

EC – Elliptic Curves

ECC – Elliptic Curve Cryptography

ECDLP – Elliptic Curve Discrete Logarithm Problem

DLP – Discrete Logarithm Problem

MAC – Message Authentication Code

ROT – Root of Trust

DES – Data Encryption Standard

AES – Advanced Encryption Standard

NIST – National Institute of Standards and Technology

DSS – Digital Signature Standard

DSA – Digital Signature Algorithm

FV – Full Virtualization

PV – Para Virtualization

HAV – Hardware-Assisted Virtualization

TPM – Trusted Platform Module

DMA – Direct Memory Access

I/O – Input/Output

TCG – Trusted Computing Group

DPA – Differential Power Analysis

CPA – Correlational Power Analysis

DOS – Denial of Service

IDS – Intrusion-Detection System

TVMM – Trusted Virtual Machine Monitor

VTPM – Virtualizing the Trusted Platform Module

SECG – Standards for Efficient Cryptography Group

ROM – Read Only Memory

FPU – Floating Point Unit

HAL – Hardware Abstraction Layer

TK – Tool Kit

VCPU – Virtual Central Processing Unit

CPU – Central Processing Unit

USB – Universal Serial Bus

CAN – Controller Area Network

ECDH – Elliptic Curve Diffie–Hellman

ECDSA – Elliptic Curve Digital Signature Algorithm

LOC – Lines of Code

SUMÁRIO

1. INTRODUÇÃO

A Internet das Coisas (do inglês, *Internet of Things* ou IoT) é um paradigma recente para computação. O conceito de IoT para um futuro próximo é de que, por intermédio dos sistemas embarcados (SE), cada objeto presente no dia a dia das pessoas possa comunicar-se com outros sistemas computacionais formando uma grande rede de objetos conectados. Os SE que formam esta rede de objetos conectados possuem diferentes capacidades computacionais de acordo com sua aplicação, provendo informações úteis aos usuários criando serviços e produtos, melhorando a qualidade de vida das pessoas [16].

Segundo a Gartner [11], o mercado de IoT apresenta uma expectativa de crescimento para 20 bilhões de dispositivos em uso no mercado até o ano de 2020. Um dos ambientes onde a IoT possui múltiplas possibilidades de aplicações e criação de novos negócios são as cidades inteligentes (do inglês, *Smart Cities*), como, por exemplo [16]: monitoramento do tráfego, da poluição do ar e do ruído; controle da qualidade; uso e distribuição da água além do controle dos postes responsáveis pela iluminação das ruas.

Aplicar o conceito de IoT para tornar as cidades inteligentes é particularmente interessante para os administradores da cidade, pois é possível melhorar os serviços de manutenção da cidade reduzindo os custos e diminuindo o tempo de resposta para resolução de problemas como acidentes ou eventos climáticos. Por exemplo, um poste responsável pela iluminação das ruas pode enviar informações sobre uma possível falha na sua lâmpada ou ser controlado remotamente para ajustar a intensidade de luz emitida. Consequentemente, evitando a ida de pessoas para verificar periodicamente o funcionamento da iluminação. Os postes também podem ser equipados com câmeras, responsáveis por monitorar as ruas e informar os serviços de emergência em caso de acidentes; além de diferentes sensores responsáveis por medir temperatura, ruído, poluição e umidade, informando os cidadãos sobre as condições climáticas e do ambiente [53], a Figura 1.1 refere-se à este cenário.

Portanto, com esta vasta gama de aplicações com potencial de impacto direto na vida das pessoas, a preocupação com o aspecto da segurança dos SE está cada vez maior, passando a ser uma questão central e prioritária no desenvolvimento de soluções para IoT. Isto se deve ao fato de que estes dispositivos evoluíram e deixaram de ser responsáveis apenas por uma única tarefa isolada e especifica passando a exercer funções mais complexas, assumindo um protagonismo maior. Agora os SE estão encarregados de processar, armazenar e enviar informações confidenciais, críticas e valiosas para as pessoas [46].

Figura 1.1 – Ilustração de um poste inteligente.
Fonte: Elaborado pelo autor.

1.1 Motivação

No cenário descrito na Figura 1.1, a técnica de virtualização em SE é uma peça chave para criar a base de desenvolvimento dos SE voltados para IoT. A virtualização oferece novas possibilidades para os SE, como, por exemplo: executar diferentes sistemas operacionais em um único processador; permitir a utilização mais eficiente dos recursos disponíveis no hardware (processador, memória e periféricos). Além disso, a virtualização também contribui para aumentar a segurança dos SE por meio da separação espacial e temporal existente entre os diferentes sistemas operacionais executando na plataforma [1, 18, 19].

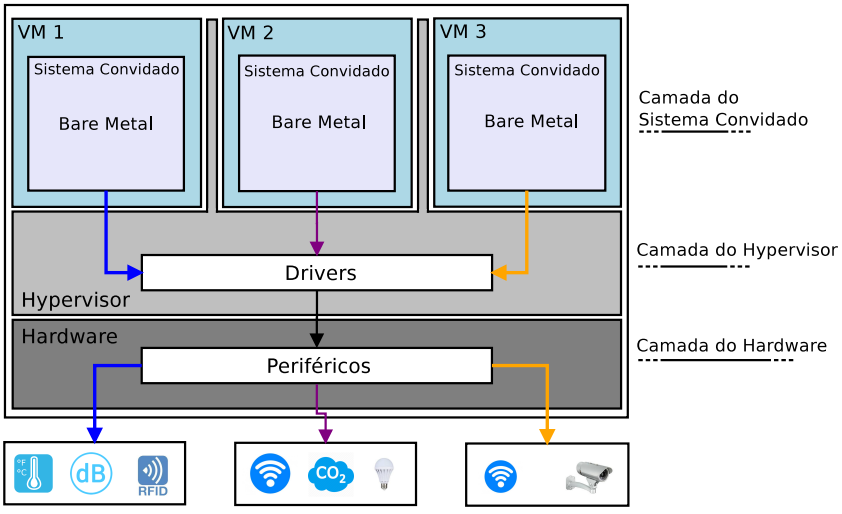

Figura 1.2 – Exemplo de um poste inteligente implementado com um SEV.
Fonte: Elaborado pelo autor.

A Figura 1.2 refere-se à um exemplo de aplicação de SE Virtualizados (SEV) no ambiente de cidades inteligentes para construir um poste encarregado pela iluminação das ruas da cidade, monitoramento da temperatura, poluição, ruído e umidade além de controle de câmeras de segurança. A arquitetura é mostrada em camadas, na parte inferior está a Camada do Hardware e os diferentes dispositivos que compõem o poste inteligente.

Em seguida, na camada central da Figura 1.2, está a Camada do *Hypervisor* ou Monitor de Máquina Virtual (do inglês, *Virtual Machine Monitor* ou VMM) que é responsável por gerenciar o tempo de execução para os sistemas convidados de acordo com as políticas de escalonamento, mantê-los em isolamento espacial e temporal e dar acesso aos recursos do hardware, como, por exemplo, processador, memória e periféricos. Os sistemas convidados, neste exemplo, são do tipo *bare metal*, porque não possuem nenhum sistema operacional, contudo, eles também podem ser um sistema operacional Linux ou também um sistema operacional de tempo real (do inglês, *Real Time Operating System* ou RTOS) de acordo com as necessidades da aplicação. Neste trabalho é utilizado o *hypervisor* desenvolvido pelo Grupo de Sistemas Embarcados (GSE) da Pontifícia Universidade Católica do Rio Grande do SUL (PUCRS), denominado prplHypervisor [34]. Este é um *hypervisor* desenvolvido para atender as necessidades dos SE. O prplHypervisor possui dois escalonadores, um de melhor esforço (do inglês, *Best-Effort Scheduler*) e outro de tempo real (do inglês, *Real-Time Scheduler*) para atender as necessidades dos SE.

A camada acima do *hypervisor* é chamada de Camada do Sistema Convidado. Os sistemas convidados ficam encapsulados em ambientes chamados Máquinas Virtuais (do inglês, *Virtual Machine ou VM*). No exemplo ilustrado na Figura 1.2 a VM 1 é responsável por implementar o controle dos sensores específicos de temperatura, ruído e Identificação por Radio Frequência (do inglês, *Radio-Frequency Identification* ou RFID). As setas em azul na Figura 1.2 ilustram a comunicação da VM 1 com o hardware. Desta forma, este sistema convidado não tem acesso aos outros periféricos disponíveis no hardware, como a lâmpada e o módulo *WiFi*, acessíveis apenas para o sistema convidado executando na VM 2 (setas roxas) ou a câmera e módulo *WiFi*, disponíveis apenas para o sistema convidado na VM 3 (setas amarelas). Outra característica importante alcançada por meio da virtualização é o compartilhamento de módulos, neste exemplo, o *hypervisor* pode prover acesso a um mesmo módulo para diferentes sistemas convidados como é o caso do módulo *WiFi*.

Logo, o poste inteligente é bastante atraente para invasores, pois uma vez comprometida alguma camada do SEV, é possível causar grandes perdas para os administradores da cidade. Assim, caso um invasor consiga executar um código malicioso no sistema convidado, diversos ataques podem ser feitos, como, por exemplo: apagar as luzes das ruas da cidade à noite, deixando a população insegura, além de manter as lâmpadas ligadas durante o dia, causando perdas financeiras com o consumo desnecessário de energia elétrica. A contribuição deste trabalho é mostrar como o uso de diferentes técnicas, aplicadas aos SE, podem contribuir para o aumento da segurança.

1.2 Contexto do Trabalho

A segurança para dispositivos IoT é considerada muito importante para o avanço desta tecnologia. Segundo pesquisa realizada pela Gartner [12] 25% dos ataques maliciosos contra empresas envolverão dispositivos IoT até 2020 e os investimentos em segurança neste mercado podem atingir $547 milhões de dólares em 2018. Este trabalho foca nas vulnerabilidades dos SEV voltados para IoT, quais os ataques conhecidos na literatura que podem ser realizados contra os SEV e como os ataques podem ser mitigados.

Na área de SE ou virtualização não existe uma padronização ou regulamentação que defina os componentes de segurança mínimos necessários para estabelecer níveis de segurança. No entanto, existem iniciativas isoladas que buscam a solução para casos específicos. Assim, chegam ao mercado SE desenvolvidos sem os devidos cuidados com a segurança, expondo dados de usuários ou potencialmente colocando suas vidas em risco.

Portanto, este trabalho busca identificar e definir um conjunto mínimo de mecanismos de segurança para SEV formando uma arquitetura de segurança padrão. Esta arquitetura pode proteger as três camadas expostas na Figura 1.2 combinando técnicas de criptografia e virtualização. Estes mecanismos de segurança são definidos com base nos ataques contra sistemas virtualizados conhecidos na literatura. Naturalmente, este conjunto de mecanismo deve evoluir com o tempo assim como os ataques evoluem. Por esta razão, estes mecanismos de segurança não são definitivos, mas sim uma base sólida para estabelecer níveis mínimos de segurança que ajudem no desenvolvimento da IoT contribuindo para a consolidação de uma padronização de segurança para este mercado.

1.3 Objetivos

O objetivo principal deste trabalho é investigar a hipótese do uso de virtualização com técnicas de criptografia adequadas para SE de maneira a aumentar a sua segurança. Para alcançar este objetivo, propõe-se uma arquitetura de segurança padrão capaz de proteger os SEV por meio da combinação de técnicas de criptografia e virtualização. Além disso, os objetivos específicos almejados por esta pesquisa são descritos a seguir:

- Identificar um conjunto mínimo de mecanismos de segurança que juntos são capazes de proteger todas as camadas da arquitetura dos SEV contra ataques maliciosos;
- Definir e implementar dois mecanismos de segurança no *hypervisor* voltados para proteção dos sistemas convidados. O primeiro mecanismo é responsável pela verificação de integridade do sistema convidado no momento da sua inicialização. Já o segundo mecanismo se encarrega de monitorar periodicamente o sistema convidado.

A escolha de implementar primeiro os mecanismos para proteção do sistema convidado se justifica pelo fato desta ser a camada da arquitetura mais exposta, atrativa e vulnerável aos ataques contra os SEV, pois é responsável por comunicar-se com outros sistemas computacionais (característica dos dispositivos IoT), o que a torna suscetível a ataques remotos. Além disto, o sistema convidado é encarregado de processar informações sensíveis pertencentes aos usuários, o que também atrai a atenção de atacantes maliciosos.

Os requisitos de segurança para avaliar a implementação dos mecanismos de segurança são: a garantia de detecção de código malicioso, o crescimento do código do *hypervisor* em comparação a versão original, uma vez que o SE possui uma quantidade de memória limitada e o tempo necessário para a detecção do código malicioso, pois, o tempo de atraso imposto pelos mecanismos de segurança não deve comprometer o desempenho.

É importante ressaltar que a segurança não deve impor atrasos significativos no desempenho do sistema onde será implementado, neste caso no *hypervisor*. Logo, criar mecanismos de segurança para SE é uma tarefa desafiadora, visto que se busca causar o menor impacto possível no tamanho do *hypervisor*, na usabilidade, no custo e tempo de desenvolvimento, ao mesmo tempo em que se tenta torná-lo mais confiável. Assim, para orientar esta pesquisa, são definidas abaixo questões relacionadas com os objetivos.

- O que necessita ser protegido em uma arquitetura de SEV?
- Quais os principais ataques contra sistemas virtualizados? Estes ataques podem ser realizados contra SEV? Como os ataques podem ser mitigados?
- Quais os mecanismos mínimos para criar uma arquitetura de segurança para SEV?
- Qual o estado da arte dos mecanismos para verificação de integridade de sistemas convidados em sistemas virtualizados? Estes podem ser adaptados aos SEV?
- Qual o estado da arte dos mecanismos de introspeção de sistemas convidados implementados em sistemas virtualizados? Estes podem ser adaptados aos SEV?

1.4 Organização do Trabalho

Este trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta os conceitos básicos sobre os SE, virtualização e criptografia. O Capítulo 3 apresenta uma pesquisa com os ataques maliciosos conhecidos na literatura contra sistemas virtualizados. O Capítulo 4 mostra o estado da arte das diferentes abordagens para aumentar a segurança dos sistemas virtualizados. O Capítulo 5 detalha a arquitetura de segurança proposta, bem como a implementação dos mecanismos de segurança. O Capítulo 6 descreve os resultados alcançados. O Capítulo 7 apresenta a conclusão, as contribuições oferecidas e apontadas as direções para trabalhos futuros.

2. CONCEITOS BÁSICOS

Este capítulo apresenta os principais temas abordados por este trabalho. A Seção 2.1 introduz o conceito de SE. A Seção 2.2 apresenta os conceitos sobre virtualização e as técnicas necessárias para que possa ser implementada. A Seção 2.3 mostra os requisitos de segurança desejados, além disso, apresenta uma introdução sobre criptografia simétrica e assimétrica, apontando suas diferentes aplicações e como podem contribuir para aumentar a segurança. A Seção 2.4 descreve as funções *hash* e a Seção 2.5 oferece a definição de assinaturas digitais, seu funcionamento e importância. A Seção 2.6 apresenta as considerações finais.

2.1 Sistemas Embarcados

Segundo Noergaard [35] os SE se caracterizam por serem mais limitados, tanto em software quanto em hardware, em relação a um sistema computacional de propósito geral como computadores pessoais e *smartphones*. Do ponto de vista do hardware, significa dizer que possuem menor capacidade de processamento, consumo de energia e memória, pois são projetados com um propósito específico. Quanto ao software, refere-se à ter um número de aplicações reduzidas, um sistema operacional reduzido ou nem possuir um.

Os exemplos de aplicações dos SE comumente remetem a sistemas computacionais específicos, como, por exemplo: uma calculadora, um roteador para internet ou um monitor cardíaco. Contudo, os SE atravessam um momento de profunda mudança. O novo paradigma da computação chamado de Internet das Coisas, que consiste em conectar à internet grande parte dos objetos ao nosso redor, têm mudado a forma como são desenvolvidos os SE. Os SE deixam de cumprir uma tarefa específica e passam a ser responsáveis por processar, armazenar, enviar e receber informações sensíveis tanto de empresas quanto de usuários.

Com o empoderamento dos SE, estes passam a atrair a atenção para ataques maliciosos. Logo, a segurança, antes negligenciada ou deixada em segundo plano, passa a ser uma prioridade dos novos produtos voltados para à IoT. Neste cenário, a virtualização emerge como uma técnica capaz de oferecer novas possibilidades para a criação de mecanismos de segurança sofisticados com características antes indisponíveis para os SE.

2.2 Virtualização

A virtualização é uma técnica que permite a criação de ambientes virtuais separados espacialmente, comumente chamados de máquinas virtuais. Com o uso da virtualização é

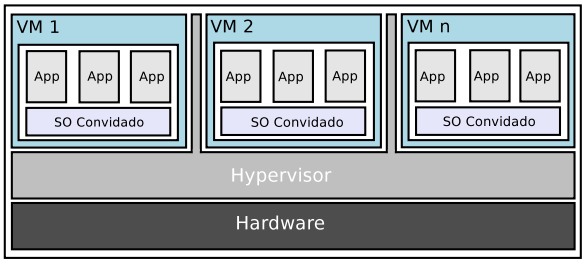

Figura 2.1 – *Hypervisor* do tipo 1. Adaptado de [44].

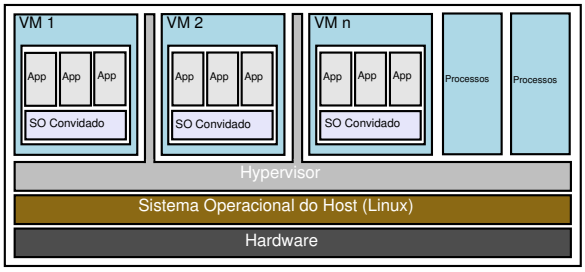

Figura 2.2 – *Hypervisor* do tipo 2. Adaptado de [44].

possível ocultar os recursos de hardware, bem como as suas características, de tal maneira que permita aos sistemas virtualizados e aplicações, de forma compartilhada, interagirem com o hardware. Para implementar esta técnica, é necessário adicionar uma nova camada de software, posicionada acima do hardware, capaz de gerenciar o acesso ao hardware feito pelos sistemas convidados. Este novo software recebe o nome de *hypervisor*. A virtualização tem sido amplamente explorada porque pode trazer benefícios aos SE, como, por exemplo [44]: a possibilidade de executar diferentes sistemas operacionais na mesma plataforma; permitir um melhor aproveitamento dos recursos do hardware, diminuindo os custos; maior flexibilidade e agilidade no desenvolvimento de novos produtos; contribuir para o aumento da segurança, mantendo os sistemas convidados em isolamento espacial.

2.2.1 Tipos de Hypervisors

Para compreender a técnica de virtualização se faz necessário entender como funcionam os *hypervisors*. Assim, os *hypervisors* são classificados em dois tipos, do tipo 1 e tipo 2, que diferem principalmente quanto a sua localização na arquitetura do sistema virtualizado, o que impacta diretamente nas capacidades oferecidas por cada abordagem.

No modelo apresentado pelo tipo 1 a camada do *hypervisor* é posta entre o hardware e o sistema operacional convidado, deste modo é possível emular todo o sistema convidado, uma vez que a camada do *hypervisor* emula a Arquitetura do Conjunto de Instruções (do inglês, *Instruction Set Architecture* ou ISA). Este *hypervisor* também é chamado de *bare metal* ou nativo, pois não necessita de um sistema operacional para executar. Por este

motivo é considerado mais eficiente que o *hypervisor* do tipo 2, dado que não existe atraso causado pelo sistema operacional. A Figura 2.1 refere-se à um *hypervisor* do tipo 1.

Já no tipo 2 a camada do *hypervisor* é inserida na Interface Binária de Aplicação (do inglês, *Application Binary Interface* ou ABI), portanto, ficando posicionada acima do sistema operacional. Desta maneira, podendo emular instruções de nível de usuário e chamadas de sistema. Este *hypervisor* executa como um processo do sistema operacional, respeitando as políticas de escalonamento. A Figura 2.2 refere-se à um *hypervisor* do tipo 2.

2.2.2 Técnicas para Virtualização

Existem diferentes formas de se implementar virtualização em sistemas computacionais, cada uma apresenta vantagens e desvantagens. Na prática, os sistemas virtualizados são implementados com uma combinação das técnicas explicadas nesta subseção.

A Para-Virtualização (do inglês, *Para-Virtualization* ou PF) requer recompilação do sistema operacional convidado antes de ser encapsulado na VM onde será executado. Estas alterações envolvem substituir instruções privilegiadas do sistema operacional por chamadas ao *hypervisor* denominadas *Hypercalls*. As *Hypercalls* são chamadas feitas pelo sistema convidado para o *hypervisor* sempre que é necessário executar instruções privilegiadas exclusivas do *hypervisor*. Ao comparar o *hypervisor* a um sistema operacional, as *hypercalls* são o equivalente às chamadas de sistema feitas pelas aplicações ao sistema operacional.

A Virtualização-Completa (do inglês, *Full Virtualization* ou FV) suporta os sistemas convidados sem a necessidade de recompilação. Logo, as instruções privilegiadas devem ser emuladas pelo *hypervisor*. Assim, a FV permite uma redução no custo de desenvolvimento além de permitir maior portabilidade do sistema convidado entre diferentes plataformas.

Na Virtualização Assistida por Hardware (do inglês, *Hardware-assisted Virtualization* ou HAV) é necessário implementações no processador, assim, aumentando sua complexidade. Este suporte em hardware torna a implementação do *hypervisor* mais simples, possibilitando um ganho de desempenho. Contudo, o custo do processador pode aumentar, bem como o consumo de energia, pontos relevantes para os SE. Dentre as características que podem ser encontradas em um processador com suporte a virtualização estão à adição de níveis de privilégio. Esta característica permite a execução do sistema convidado em modos de usuário e kernel, estando o *hypervisor* em um nível de maior privilégio.

2.3 Criptografia

Nesta seção são descritos os requisitos de segurança desejados em sistemas computacionais, assim como criptografia simétrica, criptografia assimétrica, funções *hash* além das assinaturas digitais. Compreender estes conceito é importante uma vez que os mecanismos de segurança são criados utilizando uma combinação destas técnicas criptográficas para tornar os sistemas computacionais mais confiáveis aos usuários.

2.3.1 Requisitos de Segurança

A definição dos requisitos de segurança é importante, pois auxilia na tomada de decisões no momento da criação dos mecanismos de segurança. Além disso, os requisitos de segurança oferecem uma forma objetiva para avaliar o nível de segurança alcançado.

O requisito de Integridade tem como objetivo assegurar que os dados recebidos permaneçam como foram enviados, sem alteração, inserção ou substituição dos dados [45].

A Disponibilidade visa garantir que um determinado serviço esteja disponível à um usuário autorizado e de acordo com as especificações de desempenho desejadas [45].

Já o requisito de Confidencialidade oferece a proteção da informação, tornando-a inelegível para um usuário não autorizado. Pode ser aplicada em diferentes níveis, podendo proteger todos os dados trocados entre usuários por um período pré-determinado de tempo, apenas alguns pacotes de uma mensagem ou toda a informação armazenada [45].

O Não Repúdio garante que uma entidade não poderá contestar o envio de uma determinada mensagem ou informação a outro usuário. Este requisito é importante para análise de falhas, caso ocorra algum incidente de segurança ou mau funcionamento [45].

A Autenticidade visa garantir que um usuário que se comunica com o sistema computacional é de fato quem afirma ser. A informação necessita de uma identificação única, criada por quem a envia, de modo que possa ser verificada por quem a recebe [45].

2.3.2 Criptografia Simétrica

A criptografia simétrica (do inglês, *Symmetric Cryptographic*) funciona da seguinte forma: a usuária Alice deseja enviar uma mensagem criptografada para um segundo usuário chamado Bob por meio de um canal de comunicação inseguro. O termo canal inseguro é utilizado para demonstrar que os dois usuários estão se comunicando através de uma rede passível de ataques maliciosos, como, por exemplo, a Internet. Neste exemplo, um

atacante chamado Oscar, tenta capturar os dados transmitidos neste canal de comunicação com objetivo de extrair informações valiosas enviadas pela usuária Alice para Bob [36].

É justamente no cenário descrito que o sistema de criptografia simétrica pode ser aplicado. Desta forma, Alice pode criptografar a mensagem com uma chave secreta e envia-la para Bob (a chave secreta deve ser previamente compartilhada com Bob). O usuário Bob então recebe a informação criptografada de Alice e usa a chave recebida para descriptografar a mensagem. Assim, caso Oscar (o atacante) intercepte os dados da mensagem enviada por Alice ele não conseguirá distingui-la de uma mensagem aleatória. A criptografia simétrica utiliza um canal auxiliar para funcionar, chamado de canal seguro. Ele é utilizado para que a troca de chave secreta possa ser feita sem passar pelo canal inseguro. A seguir são descritos brevemente os principais algoritmos de criptografia simétrica [36].

O DES (do inglês, *Data Encryption Standard*) foi apresentado em meados de 1972 possui seu funcionamento baseado em redes de Feistel com 16 rodadas, executando em blocos de 64 bits. As redes de Feistel utilizam praticamente as mesmas operações tanto para criptografar quanto para descriptografar, permitindo implementações eficientes.

O AES (do inglês, *Advanced Encryption Standard*), criado em 1999, é amplamente utilizado, pois se tornou um padrão recomendado pela indústria. O algoritmo AES é uma cifra de bloco com chaves que variam entre 128 bits, 192 bits e 256 bits com um tamanho de bloco de 128 bits. O número de rodadas para cada bloco varia de 10, 12 e 14 rodadas.

2.3.3 Criptografia Assimétrica

A criptografia assimétrica (do inglês, *Asymmetric Cryptography*) foi proposta por Whitfield Diffie, Martin Hellman [7]. Esta técnica surge como uma proposta para corrigir as imperfeições apresentadas pela criptografia simétrica que são difíceis de serem tratadas computacionalmente, como, por exemplo, a distribuição de chaves [36].

Portanto, na criptografia assimétrica um par de chaves (pública/privada) é selecionado de maneira que derivar a chave privada a partir da chave pública é considerado um problema computacional intratável. Assim, um usuário A que possua a chave pública de um usuário B não poderá, em tempo útil, descobrir a chave privada do usuário B [17, 36]. A seguir são apresentados os principais algoritmos de criptografia assimétrica [17, 36].

O RSA (abreviatura do nome dos autores Ronald Rivest, Adi Shamir e Leonard Adleman) é um algoritmo amplamente utilizado para troca de chaves simétrica e assinaturas digitais. A sua segurança está baseada na dificuldade de fatoração de um número inteiro grande, normalmente utiliza-ze um valor maior que 1024 bits para garantir a segurança.

Tabela 2.1 – Comparação do tamanho do módulo do RSA x ECC. Adaptado de [45].

RSA (tamanho do módulo em bits)	ECC (tamanho do módulo em bits)
1024	160-223
2048	224-255
3072	256-383

O algoritmo Diffie-Helman foi proposto por Whitfield Diffie e Martin Hellman em 1976. O sistema é utilizado para troca de chaves. A segurança é baseada na dificuldade de solução do problema do logaritmo discreto (do inglês, *Discrete Logarithm Problem* ou DLP).

O algoritmo ElGamal foi proposto em 1985 por Taher Elgamal, assim como o sistema proposto por Diffie-Helman, sua segurança também reside na dificuldade de solução do problema do logaritmo discreto. ElGamal apresenta também o conceito de chaves efêmeras.

A aplicação das Curvas Elípticas (do inglês, *Elliptic Curves* ou EC) no campo da criptografia (do inglês, *Elliptic Curve Cryptography* ou ECC) foi proposto apenas em 1985, de forma independente, por Koblitz e Victor Miller [27, 32]. A segurança reside na intratabilidade do problema do logaritmo discreto para curvas elípticas (do inglês, *Elliptic Curve Discrete Logarithm Problem* ou ECDLP). Com as curvas elípticas é possível atingir os mesmos níveis de segurança do RSA trabalhando com números menores, proporcionando uma melhor performance [36, 45]. A Tabela 2.1 corresponde à uma comparação entre ECC e RSA, onde é possível perceber uma vantagem para o algoritmo de ECC [45].

2.4 Funções Hash

As funções *Hash* ou Algoritmo de Dispersão Seguro (do inglês, *Secure Hash Algorithm* ou SHA) nos permitem criar uma representação única para uma determinada mensagem. Estas funções são amplamente utilizadas em conjunto com outros mecanismos de segurança, tais como: assinaturas digitais (do inglês, *Digital Signature*) e códigos de autenticação de mensagem (do inglês, *Message Authentication Code* ou MAC). As funções *hash* aceitam em sua entrada uma mensagem M, de tamanho variável, e produzem na saída um valor de tamanho fixo. As propriedades de cinco funções *hash* são apresentadas na Tabela 2.2; os algoritmos diferem entre si pelo nível de segurança que oferecem, além do tamanho dos blocos. As duas principais características que tornam as funções *hash* seguras são [36, 45]: ser computacionalmente inviável (nenhum ataque é significativamente melhor que a força bruta) encontrar a mensagem que corresponde há um determinado *hash*; não ser possível produzir o mesmo *hash* a partir de duas mensagens diferentes.

Tabela 2.2 – Comparação entre as funções *Hash*. Adaptado de [36].

Algoritmo	Saída em bits	Entrada em bits	Número de rodadas	Colisões encontradas
SHA-1	160	512	80	sim
SHA-2	224	512	64	não
SHA-2	256	512	64	não
SHA-2	384	1024	80	não
SHA-2	512	1024	80	não

2.5 Assinaturas Digitais

O NIST (do inglês, *National Institute of Standards and Technology*) propôs, em agosto de 1991, o Algoritmo de Assinatura Digital (do inglês, *Digital Signature Algorithm* ou DSA), especificado no FIPS 186 como Padrão para Assinatura Digital (do inglês, *Digital Signature Standard* ou DSS); visto como uma variação do sistema de assinatura digital proposto por ElGamal [8]. O algoritmo DSS foi criado com a intenção de ser aplicado em áreas, como, por exemplo: e-mail, transferências bancárias e distribuição de software.

As assinaturas digitais representam uma versão eletrônica análoga às assinaturas escritas. Para que sejam seguras, são desejadas as seguintes propriedades [25, 36]: a assinatura digital deve ter um padrão de bits que depende da mensagem; é desejável um baixo custo computacional para produzi-la e reconhecê-la; ser computacionalmente inviável forjá-la em um tempo útil; em termos de armazenamento, deve ser pouco custoso mantê-la.

2.6 Considerações Finais

Neste capítulo foram apresentados os conceitos fundamentais sobre os tópicos abordados por esta pesquisa. O capítulo inicia pela definição e uma visão geral dos SE. Posteriormente foram mostrados os conceitos de virtualização, assim como as técnicas utilizadas para a sua implementação, suas características e como podem ajudar a criar e manter os SE mais seguros. Além disso, foram abordados os principais conceitos no campo da criptografia, os algoritmos utilizados, os requisitos de segurança desejados e as técnicas utilizadas para criação de mecanismos de segurança. Nota-se que existem diferentes algoritmos criptográficos capazes de aumentar a segurança dos sistemas computacionais, logo, é necessário escolher quais destes algoritmos melhor se adaptam melhor aos SE considerando o que eles são capazes de proteger, bem como o desempenho que alcançam.

No capítulo 3 será apresentada uma pesquisa sobre os principais ataques aos quais os sistemas virtualizados estão expostos, também é discutido como eles impactam nos requisitos de segurança desejados, bem como os ataques podem ser usados contra os SE.

3. ATAQUES A SISTEMAS VIRTUALIZADOS

A primeira etapa para criação de mecanismos de segurança é a identificação das ameaças às quais o sistema computacional está exposto. Além disso, esta etapa é fundamental para a definição de quais camadas da arquitetura estão mais vulneráveis; isto nos permite priorizar quais mecanismos de segurança devem ser projetados primeiro e quais os seus objetivos. Para tanto, este capítulo apresenta os ataques conhecidos contra os sistemas computacionais que utilizam a técnica de virtualização. Assim, aborda-se ataques contra *hypervisors* amplamente utilizados no mercado, tanto do tipo 1 e tipo 2.

Este capítulo está dividido da seguinte forma: a Seção 3.1 classifica os ataques conforme a camada alvo na arquitetura e são mostrados exemplos reais bem sucedidos destes ataques contra sistemas computacionais virtualizados. A Seção 3.2 apresenta as considerações sobre as vulnerabilidades encontradas.

3.1 Ataques Frente à Camada Alvo

Esta seção apresenta os ataques contra as camadas do sistema convidado, do *hypervisor* e do hardware. Além disso, são apontados exemplos de ataques bem sucedidos, ou seja, que de alguma maneira foram capazes de extrair alguma informação útil do sistema computacional ao qual se propunham atacar. A Figura 3.1 refere-se a posição dos ataques em cada camada da arquitetura de um SEV.

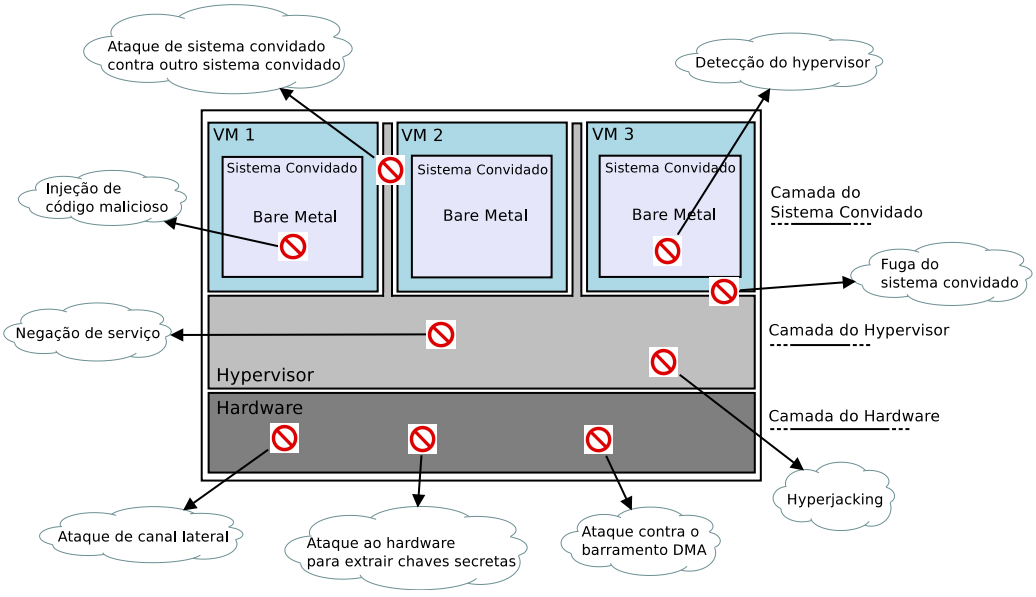

Figura 3.1 – Posição dos ataques na arquitetura de um SEV do tipo 1.
Fonte: Elaborado pelo autor.

3.1.1 Ataques ao Sistema Convidado

- Mobilidade de um Sistema Convidado: Consiste na possibilidade que alguns *hypervisors* têm de gerar imagens instantâneas do sistema convidado (do inglês, *Snapshots*). Estas imagens contêm dados salvos na memória, como, por exemplo, nome de usuário e senha. Os snapshots são amplamente utilizados para criar cópias de segurança ou backups de sistemas convidados e que podem ser transferidas para outro hardware [40]. Assim, um atacante que tenha acesso a esta imagem pode acessar informações que ficaram salvas na memória. Entretanto, os *hypervisors* desenvolvidos para SE usualmente não possibilitam a criação de imagens do sistema convidado devido aos recursos de limitados de memória RAM e capacidade de armazenamento. Assim, este ataque não representa uma ameaça, ao menos por enquanto, aos SE;

- Modificação Externa de um Sistema Convidado: Este ataque consiste em um atacante injetar código malicioso no sistema convidado. O atacante pode tentar roubar informações do sistema convidado, atacar a camada do *hypervisor* ou mesmo outros sistemas convidados. Uma maneira de evitar este ataque é por meio da criação de um mecanismo capaz de verificar a integridade do sistema convidado na inicialização ou em tempo de execução [39].

- Fuga do Sistema Convidado: Neste ataque, o termo fuga é empregado para identificar que o sistema convidado foi capaz de burlar o isolamento espacial entre o sistema convidado e o *hypervisor*, podendo assim assumir o controle do *hypervisor*. Logo, este ataque permite que o sistema convidado execute funções do *hypervisor*, como criar ou desligar sistemas convidados ou monitorar o tráfego de dados [39]. Geffner [13] demonstrou uma vulnerabilidade no acesso ao disco, utilizado em diversos *hypervisors* (KVM, Xen e Virtual Box), que permite ao sistema convidado escapar de seu isolamento espacial, conseguindo acesso a outras máquinas virtuais executando na mesma plataforma.

- Ataque de um Sistema Convidado contra Outro: Tratada como uma falha no isolamento espacial entre as VMs, este ataque caracteriza-se por permitir que um sistema convidado malicioso possa atingir outro sem a necessidade de burlar o *hypervisor*, o que difere do ataque descrito como Fuga do Sistema Convidado. Neste caso, o atacante pode ter acesso às informações sensíveis de outro sistema convidado, como, por exemplo, acessando sua região de memória, monitorando os pacotes na comunicação de rede e até mesmo desviando-os [39]. Zhang et al. [55] demonstrou um ataque entre sistemas convidados capazes de extrair uma chave privada do usuário explorando a memória cache L1. Outro exemplo é apresentado por Irazoqui et al. [23],

demonstrando que é possível recuperar a chave em uma implementação AES com a biblioteca Openssl [52].

3.1.2 Ataques ao Hypervisor

- Detecção do *Hypervisor*: Em um primeiro momento, é importante para um atacante detectar que um sistema operacional está executando em uma plataforma virtualizada para então planejar um ataque contra o *hypervisor* ou outro sistema convidado. Este ataque pode ocorrer quando o atacante não conhece ou não possui acesso ao SE e tenta infecta-lo remotamente. Uma forma de detectar a presença do *hypervisor* é através comparação do tempo de execução de instruções, dado que a camada do *hypervisor* insere um atraso no tempo de execução;

- *Hyperjacking*: Este ataque consiste em injetar um código malicioso capaz de tomar o controle do sistema convidado, se passando por um *hypervisor* [38]. Assim, esta técnica permite ao intruso executar aplicações e monitorar a comunicação entre os sistemas convidados.

- Negação de Serviço: Um ataque de Negação de Serviço (do inglês, *Denial of Service* ou DoS), caracteriza-se quando um atacante tenta consumir todos os recursos disponíveis [5]. Como os sistemas convidados compartilham os recursos, como processador, memória, dispositivos de rede e periféricos, é possível fazer um ataque de negação de serviço contra estes recursos, tornando-os indisponíveis para outros sistemas convidados.

3.1.3 Ataques ao Hardware

- Ataques ao DMA: É o recurso que permite o acesso diretamente a memória de maneira independente da CPU, denominado Acesso Direto a Memória (do inglês, *Direct Memory Access* ou DMA). O DMA permite que a CPU principal inicie uma transferência de dados da memória enquanto realiza outras operações. Stewin e Bystrov [47] demostraram uma aplicação capaz de monitorar dados do teclado de um usuário sendo transferidos pelo DMA;

- Ataques ao TPM: Um Módulo de Hardware Confiável (do inglês, *Trusted Platform Module* ou TPM) [15] é um dispositivo dedicado a operações criptográficas especificadas pelo Grupo de Computação Confiável (do inglês, *Trusted Computing Group* ou TCG). O objetivo é executar operações como armazenamento seguro de chaves secretas,

código e geração de números aleatórios. O objetivo principal deste ataque é extrair os dados armazenados no hardware, como, por exemplo, chaves privadas;

- Ataque de Canal Lateral: Para realizar este tipo de ataque é necessário que o atacante possua acesso ao hardware, pois o objetivo é explorar a implementação física do sistema criptográfico para encontrar a chave secreta utilizada ao invés de explorar as possíveis fraquezas teóricas dos algoritmos de criptografia. As técnicas mais comumente utilizadas são: Análise de Energia Diferencial (do inglês, *Differential Power Analysis* ou DPA) e Análise de Energia Correlacional (do inglês, *Correlational Power Analysis* ou CPA). A DPA [28] utiliza a análise estática da energia consumida, pois, existe uma relação entre as operações realizadas pelo hardware e o consumo de energia. A CPA é baseada na análise da dependência entre o consumo de energia do circuito eletrônico e a distância do Hamming, ou o número de bits em nível lógico em 1 dos dados processados [28]. Belgarric et al. [3, 14] demostraram um ataque bem sucedido contra smartphones.

3.2 Considerações Finais

Neste capítulo apresentou-se um conjunto de ataques cujo principal objetivo é comprometer os sistemas computacionais virtualizados. Além disso, os ataques maliciosos foram categorizados conforme a camada alvo (Sistema Convidado, *Hypervisor* e Hardware).

Conforme visto até aqui, as camadas do Hardware e do Sistema convidado estão mais suscetíveis a ataques. O Hardware por estar mais acessível nos SE se comparado com sistemas computacionais de propósito geral, como, por exemplo, os postes nas ruas da cidade. Já a camada do Sistema Convidado torna-se atraente, dado que a lógica de controle do hardware, como a lâmpada ou a câmera em um poste inteligente, é feita pelo Sistema Convidado. Além disso, os sistemas convidados se comunicam com outros sistemas computacionais, assim, ficando expostos a ataques remotos através das diferentes redes de comunicação. A camada do *Hypervisor* é a mais difícil de ser atacada, uma vez que exige um conhecimento aprofundado do *hypervisor* para se descobrir suas vulnerabilidades. Contudo, se comprometida, esta camada coloca em risco todos os sistemas convidados.

Os sistemas computacionais virtualizados têm atraído à atenção de atacantes, pois a cada dia tornam-se responsáveis pela disponibilidade de serviços e pelo processamento de informações críticas de usuários e empresas. Logo, este mesmo fenômeno ocorrerá com os SE à medida que o mercado de IoT expandir-se e a virtualização tornar-se cada vez mais utilizada na construção de dispositivos IoT. Portanto, é fundamental criar maneiras de proteger os SEV contra os ataques já conhecidos. Apresenta-se na Tabela 3.1 como os ataques impactam nos requisitos de segurança, a Tabela 3.2 corresponde à um resumo da relação dos ataques conhecidos com cada camada alvo

Tabela 3.1 – Relação dos ataques com os requisitos de segurança.
Fonte: Elaborado pelo autor.

Ataques	Requisitos de segurança				
	I [1]	A[2]	C[3]	NP[4]	D [5]
Modificação externa do sistema convidado	✓	✓	✓	✓	
Mobilidade do sistema convidado	✓		✓		
Fuga do sistema convidado	✓		✓		✓
Ataque de um sistema convidado contra outro	✓		✓		
Hyperjacking	✓	✓			
Detecção do hypervisor			✓		
Negação de serviço					✓
Ataque ao DMA			✓		
Ataque ao TPM			✓		
Ataque de canal lateral			✓		

[1] Integridade;
[2] Autenticidade;
[3] Confidencialidade;
[4] Não Repúdio;
[5] Disponibilidade.

Tabela 3.2 – Resumo dos ataques frente à camada alvo na arquitetura.
Fonte: Elaborado pelo autor.

Ataques	Camada alvo		
	Sistema Convidado	Hypervisor	Hardware
Modificação externa do sistema convidado	✓	✓	
Mobilidade do sistema convidado	✓		
Fuga do sistema convidado	✓	✓	
Ataque de um sistema convidado contra outro	✓	✓	
Hyperjacking		✓	
Detecção do hypervisor		✓	
Negação de serviço	✓	✓	
Ataque ao DMA			✓
Ataque ao TPM			✓
Ataque de canal lateral			✓

No capítulo 4 será apresentado em detalhes um conjunto de mecanismo de segurança existentes voltados para a proteção dos sistemas virtualizados, os requisitos de segurança alcançados e as camadas protegidas por estes mecanismos.

4. ESTADO DA ARTE DOS MECANISMOS DE SEGURANÇA PARA SISTEMAS VIRTUALIZADOS

Este capítulo apresenta o estado da arte dos mecanismos voltados para aumentar a segurança dos sistemas virtualizados. São abordados mecanismos implementados em *hypervisors*, do tipo 1 e do tipo 2, conhecidos na literatura da área. O conjunto de mecanismos tratados neste capítulo formará a base da arquitetura de segurança para SEV.

O capítulo está dividido da seguinte maneira: a Seção 4.1 apresenta as propostas que implementaram um mecanismo de introspecção para detectar atividades maliciosas que ocorram no sistema convidado ou nas aplicações em tempo de execução; a Seção 4.2 mostra as soluções desenvolvidas para criar uma área de memória isolada com objetivo de executar aplicações e sistemas convidados separados espacialmente. A Seção 4.3 apresenta os mecanismos que utilizam um hardware dedicado há criptografia, sendo virtualizado e disponibilizado aos sistemas convidados, permitindo o uso de técnicas de criptografia nas camadas de virtualização e sistema convidado na arquitetura. A Seção 4.4 apresenta uma proposta de verificação do sistema convidado antes de sua execução, no momento do boot. Por fim, a Seção 4.5 oferece as considerações finais.

4.1 Sistemas de Detecção de Intruso

Uma maneira de aumentar a segurança dos sistemas virtualizados é com a utilização de um Sistema de Detecção de Intrusos (do inglês, *Intrusion-Detection Systems* ou IDS). Estes mecanismos funcionam por meio da verificação continua das atividades do sistema convidado em busca de alterações maliciosas de código [44]. Quando utilizado em um sistema virtualizado, o mecanismo de IDS pode ser implementado no *hypervisor*. A vantagem desta abordagem é manter a separação, espacial e temporal, entre o agente de monitoração (IDS) e o sistema convidado monitorado. Outro ponto de destaque desta técnica é a possibilidade de verificar o sistema convidado em tempo de execução, assim, caso um sistema convidado seja infectado após sua execução, é possível identificar a atividade maliciosa. Contudo, esta técnica requer conhecimento do ataque que se deseja monitorar, por exemplo, é preciso saber onde o código malicioso será inserido (no nível do kernel ou na aplicação) e qual a diferença de comportamento em relação há um código não infectado.

Azab et al. [2] propõe avaliar a integridade dos sistemas convidados, incluindo o kernel e as aplicações executando acima do *hypervisor* com o mecanismo chamado HIMA. O HIMA é implementado como um componente do *hypervisor* XEN. O XEN é um *hypervisor* do tipo 1 amplamente utilizado no mercado de virtualização em nuvem e já foi portado para SE.

O monitoramento ativo implementado no HIMA inclui a análise das interrupções, chamadas do sistema e exceções; além do tipo de evento e registradores.

Shimada e Nakajima [43] propõe um IDS para analisar as estruturas do Kernel do sistema convidado. O protótipo foi implementado com o *hypervisor* SPUMONE (acrônimo de *Software Processing Unit, Multiplexing ONE*) e tem o Linux como sistema convidado. SPUMONE é um *hypervisor* do tipo 2 voltado para SE proposto por Kendra et al. [24].

Lee et al. [29] apresentam um IDS implementado no *hypervisor*, do tipo 2, chamado KVM [26]. O IDS utiliza técnicas de inteligência artificial para analisar se um sistema convidado está executando código malicioso. O mecanismo coleta os dados de diversos eventos e encaminha ao analisador, que será treinado para identificar e diferenciar o comportamento normal de uma atividade suspeita em um sistema convidado infectado.

Fattori et al. [9] apresentam o *hypervisor* AccessMiner capaz de capturar o comportamento das aplicações ao utilizarem os recursos do sistema convidado e diferenciar quais aplicações podem ter sido infectadas com uma acurácia de 90%. A proposta funciona criando um modelo de atividades das aplicações analisando o acesso aos diretórios no sistema de arquivos, bem como das chaves de registro. Para analisar estas atividades o *hypervisor* coleta anonimamente os *logs* das chamadas de sistema e os envia para um repositório central capaz de analisar os dados. O IDS apresentou resultados satisfatórios, entretanto, funciona apenas para o sistema operacional Microsoft Windows.

4.2 Segurança por Separação Espacial de Aplicações

Como a virtualização permite a criação de áreas de memória completamente isoladas para cada sistema convidado, esta característica possibilita o uso da técnica conhecida como Caixa de Areia (do inglês, *SandBox*). Esta técnica consiste na criação de áreas dedicadas para a execução de binários não confiáveis. Em virtualização, isto pode ser feito para executar, por exemplo, máquinas virtuais de sistemas convidados legados, assim como apenas algumas aplicações dentro de um sistema convidado não confiável [44].

McCune et al. [31] apresentam o *hypervisor* chamado TrustVisor. Considerado pequeno, com 6000 linhas de código, o Trustvisor visa atingir dois objetivos: proteger blocos de código com potencial malicioso e obter alta performance para sistemas legados. As aplicações precisam executar no ambiente protegido, para isto, devem implementar as *hypercalls* necessárias para a comunicação direta com o TrustVisor, logo, nenhuma modificação no sistema legado precisa ser feita. A principal contribuição do Trustvisor é permitir a execução de código em uma área separada espacialmente do sistema convidado chamada Pedaços de Lógica da Aplicação(do inglês, *Pieces of Application Logic* ou PALs).

Hofmann et al. [20] propõe o *hypervisor* Inktag, projetado para permitir execução de aplicações em sistemas convidados não confiáveis. O objetivo é proteger aplicações confiáveis de sistemas não confiáveis por meio da separação de espaços de memória. Desta forma, InkTag introduz o conceito de Paraverificação (do inglês, *Paraverification*) que permite averiguar o comportamento do sistema convidado. Em vez de verificar alguma modificação no sistema convidado, o InkTag requer que o sistema convidado forneça algumas informações para o *hypervisor* e aplicações, como atualizações e estado dos processos executando. Assim, é possível que as aplicações estabeleçam políticas de acesso aos seus dados.

Weiss at al. [49] apresentam um mecanismo para verificação de integridade de binários. A proposta foi implementada no *hypervisor* L4/Fiasco.OC [51] e tem como objetivo estabelecer a integridade e autenticidade de um binário recebido remotamente. O mecanismo deve gerar uma prova confiável que posteriormente será enviada para o servidor remoto. O mecanismo utiliza um sistema de criptografia híbrido, fazendo uso de criptografia simétrica com AES e da criptografia assimétrica com RSA para verificar a assinatura digital.

4.3 Virtualização de um Hardware Intrinsecamente Confiável

Outra maneira de trazer segurança aos sistemas virtualizados é utilizando de um Hardware Confiável (do inglês, *Root of Trust* ou RoT). A virtualização pode oferecer o acesso a este dispositivo para todos os sistemas convidados. Esta abordagem possui a vantagem de delegar o processamento crítico ao hardware além do dispositivo ser resistente à violação, caso alguém tente ter acesso às chaves armazenadas. A principal desvantagem desta técnica é que, na medida em que os algoritmos implementados no hardware tornam-se obsoletos, é necessário um investimento significativo de capital na atualização da plataforma.

Garfinkel et al. [10] introduzem o *hypervisor* Terra. A proposta se destaca por ser baseada em hardware, chamado Monitor de Máquina Virtual Confiável (do inglês, *Trusted Virtual Machine Monitor* ou TVMM). É por meio da verificação de integridade que é possível a proteção das VMs e das aplicações contra ataques que tentem realizar modificações maliciosas, quando executadas em áreas de memória separadas espacialmente.

Perez et al. [37] apresentam o *hypervisor* vTPM (do inglês, *Virtualizing the Trusted Platform Module*) como uma proposta capaz de virtualizar um hardware resistente à violação. Cada um dos sistemas convidados que utilizam o hardware TPM devem criar sua própria instância. Assim, cada solicitação realizada para operações criptográficas passe por um gerenciador de solicitações, que destina para a instância vTPM correta [37]. Shi et al. [42] apresentam uma proposta semelhante implementada no simulador QEMU com o *hypervisor* KVM. Contudo, os autores não citam o algoritmo de criptografia utilizado.

4.4 Verificação do Sistema Convidado no Processo de Boot

Outra forma de aumentar a segurança dos sistemas virtualizados é por meio da verificação de integridade dos sistemas convidados no momento da sua inicialização.

Seshadri et al. [41] propõe o *hypervisor* chamado SecVisor, que visa assegurar a verificação de integridade do kernel do sistema convidado, assim, protegendo contra os ataques de injeção de código malicioso. O processo de verificação de integridade foi implementado alterando a sequência de boot do Linux. Logo, após o kernel ser descomprimido, o SecVisor é executado e recebe como parâmetro o endereço final e inicial do kernel. Para que o código seja considerado seguro, é computado o *hash* SHA-1 do kernel e verificado se ele consta em uma lista previamente salva, chamada de Lista Branca (do inglês, *Whitelist*).

Zhang et al. [54] propõe uma plataforma chamada Hypercheck, baseada em um processador exclusivo para operações criptográficas cujo objetivo é assegurar a integridade do *hypervisor*, bem como dos sistemas convidados. Assim, sendo capaz de detectar a injeção de código malicioso. O mecanismo foi implementado no *hypervisor* XEN sendo capaz de criar uma Imagem Instantânea (do inglês, *Snapshot*) do sistema convidado inteiro; esta imagem contempla o estado atual da CPU além da memória da VM. A imagem é envida para um servidor remoto responsável por comparar esta informação com uma imagem previamente salva, referente ao momento em que o sistema convidado foi inicializado.

4.5 Considerações Finais

Neste capítulo foram discutidas diferentes abordagens criadas para tornar os sistemas virtualizados mais seguros. Até o momento as soluções foram desenvolvidas para atender aos sistemas de virtualização para servidores ou computação na nuvem, nestes cenários não são impostas fortes restrições de desempenho ou tamanho do *hypervisor*. As propostas apresentadas para estabelecer um nível maior de segurança utilizam a função *hash* SHA-1, já considerada insegura por apresentar colisões. Além disso, para as propostas que utilizam mecanismos de IDS, as técnicas exigem a utilização de muitos recursos computacionais para atingirem os objetivos, por esta razão, são inviáveis para aplicações voltadas para IoT. Outro ponto importante é que as soluções visam atender um sistema convidado de propósito geral, normalmente o Linux, entretanto, em um ambiente de SE pode-se ter sistemas com propósitos específicos, como sistemas de tempo real ou *bare metal*. Portanto, é necessário criar um modelo de mecanismo versátil capaz de atender a diversidade dos SE.

Apresenta-se na Tabela 4.1 um resumo das propostas desenvolvidas para trazer maior segurança aos sistemas virtualizados. São descritos, na primeira coluna, os *hypervisors* utilizados. A segunda coluna mostra o momento quando é feita a verificação (boot ou

durante a execução do sistema convidado). A terceira coluna apresenta a camada protegida pelo mecanismo de segurança (aplicações, sistema convidado ou o *hypervisor*). A quarta e a quinta coluna referem-se, respectivamente, ao algoritmo de criptografia e a função *hash* utilizada. Na última coluna identifica-se o sistema de detecção de intruso utilizado.

Tabela 4.1 – Resumo do estado da arte.
Fonte: Elaborado pelo autor.

Hypervisor	Momento da Verificação	Camada Protegida	Verificação de Integridade	Função Hash	IDS
Spumone	Execução	Sistema Convidado	-	-	Estruturas do kernel
Hima	Execução	Aplicação	-	SHA-1	Interrupções, Chamadas de Sistema
Kvm	Execução	Sistema Convidado	-	-	Inteligência Artificial
AccessMiner	Execução	Aplicação	-	-	Chamadas de Sistema
Fiasco	Boot	Aplicação	RSA, AES	SHA-1	-
Inktag	Execução	Aplicação	-	SHA-1	-
Trustvisor	Execução	Aplicação	RSA	SHA-1	-
Terra	Boot	Aplicação, Sistema Convidado	HMAC	SHA-1	-
vTPM	Execução	Sistema Convidado	RSA	SHA-1	-
SecVisor	Boot	Sistema Convidado	-	SHA-1	-
Hypercheck	Execução	Hypervisor, Sistema Convidado	-	-	-
prplHypervisor	Execução, Boot	Sistema Convidado	ECDSA	SHA-2	*hypercalls*

A última linha resume as características dos dois mecanismos de segurança voltados para proteção dos sistemas convidados. Escolheu-se estes dois mecanismo para implementação, pois esta é a camada mais vulnerável e exposta a ataques maliciosos na arquitetura dos SEV. O prplHypervisor é um hypervisor do tipo 1 voltado para trabalhar com SE desenvolvido pelo GSE da PUCRS. A abordagem proposta se diferencia do estado da arte por combinar técnicas criptográficas seguras (SHA-2 e ECDSA) adequadas à capacidade computacional dos SE, tirando proveito dos benefícios oferecidos pela virtualização e não comprometendo o seu uso. Os modelos destes mecanismos, bem como a arquitetura e o *hypervisor* serão descritos em detalhes no capítulo 5 e capítulo 6.

5. PROPOSTA DE ARQUITETURA DE SEGURANÇA PARA SEV

Este capítulo apresenta o modelo de arquitetura de segurança para SEV que contemple mecanismos de proteção contra os ataques conhecidos na literatura. Como discutido anteriormente, a virtualização já é uma realidade para os SE voltados para o mercado da IoT; também é notório que ataques maliciosos contra os SE e sistemas virtualizados estão cada vez mais sofisticados e com potencial de dano cada vez maior aos usuários e empresas. Assim, apresenta-se uma abordagem para estabelecer um conjunto de mecanismo de segurança capazes de criar uma ambiente seguro para aos SEV, combinando a técnica de virtualização com técnicas de criptografia.

Este capítulo está organizado da seguinte maneira: a Seção 5.1 descreve os mecanismos desejados para a formação da arquitetura de segurança, apresentando sua importância, o benefício de sua adoção para a segurança, o seu impacto nos ataques conhecidos na literatura. A Seção 5.2 apresenta em detalhes os modelos dos mecanismos de segurança implementados. Na Seção 5.3 discuti-se as limitações encontradas no desenvolvimento da proposta de trabalho. A Seção 5.4 apresenta as considerações finais.

5.1 O Modelo de Arquitetura Proposto

Esta seção apresenta o conjunto de mecanismos de segurança propostos. Discuti-se também como cada mecanismo contribui para aumentar segurança dos SEV.

Baseado na análise dos trabalhos relacionados, verificou-se que não existe uma definição de arquitetura de segurança padronizada consolidada para a segurança dos SEV. Desta maneira, este trabalho sugere a adoção de um conjunto de mecanismos essenciais para estabelecer um ambiente seguro. Os mecanismos de segurança sugeridos podem ser implementados em diferentes SEV e de diversas maneiras, ou seja, é possível definir diferentes modelos de implementação de cada um dos mecanismos, desde que se alcancem os requisitos de segurança propostos. Esta característica é particularmente importante, pois permite a comparação dos níveis de segurança alcançados por cada SEV voltado para à IoT.

Assim, ilustra-se na Figura 5.1 a arquitetura de segurança proposta, bem como os seus componentes. A arquitetura é apresentada em camadas (Hardware, *Hypervisor* e os Sistemas Convidados). Mostra-se também o *hypervisor* prplHypervisor que é utilizado nesta pesquisa; as suas características são apresentadas em detalhes no capítulo 6. A seguir são descritos cada um dos mecanismos, iniciando pelos mecanismos presentes na camada do hardware, em seguida o *hypervisor* e por último a Cadeia de Confiança, bem como suas contribuições para a segurança dos SEV.

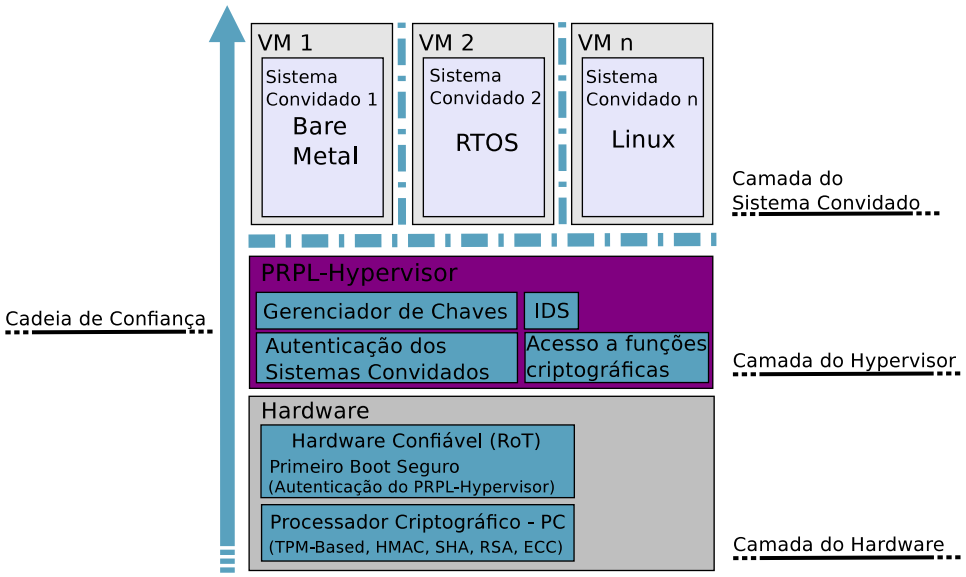

Figura 5.1 – Arquitetura de segurança proposta para os SEV.
Fonte: Elaborado pelo autor.

Os algoritmos criptográficos são essenciais para os SE voltados para IoT, uma vez que a característica principal destes dispositivos é a capacidade de comunicação com outros sistemas computacionais. Entretanto, algoritmos como AES, DES e RSA, utilizados para estabelecer uma comunicação segura, apresentam uma performance insatisfatória quando implementados em software, pois demandam muitos recursos computacionais. Portanto, é desejável que a implementação destes algoritmos esteja disponível no hardware, de maneira que o *hypervisor* seja capaz de prover acesso a este recurso para os sistemas convidados. O Processador Criptográfico está localizado na camada do hardware e o Acesso ás funções criptográficas na camada do *hypervisor* na Figura 5.1.

A necessidade de uma comunicação segura indica que os SE necessariamente trocarão chaves secretas com outros sistemas computacionais. As chaves criptográficas utilizadas pelos protocolos de comunicação devem ser armazenadas de maneira segura, preferencialmente no hardware. Contudo, um risco inserido pelo uso da técnica de virtualização é que este hardware é compartilhado com todos os sistemas convidados por intermédio do *hypervisor*. Assim, o *hypervisor* deve oferecer um Gerenciador de Chaves criptográficas salvas no hardware e então distribuir corretamente para os sistemas convidados. Desta forma é possível assegurar que as chaves serão lidas pelo sistema convidado autorizado. O Gerenciador de Chaves criptográficas está localizado na camada do *hypervisor* na Figura 5.1.

Outro importante mecanismo de segurança é a Separação Espacial. É fundamental que o *hypervisor* promova uma separação espacial entre os sistemas convidados e também entre os sistemas convidados e o *hypervisor*. O benefício de adotar um *hypervisor* com esta característica é evitar ataques que ocorram a partir de um sistema convidado infectado que tente acessar a área de memória dos outros sistemas convidados ou do *hypervisor*.

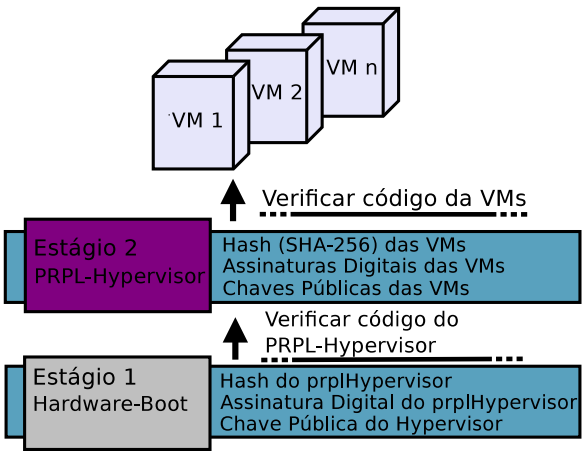

Figura 5.2 – Arquitetura da cadeia de confiança para SEV.
Fonte: Elaborado pelo autor.

Comprometer o *hypervisor* é particularmente importante uma vez que ele pode controlar todos os outros sistemas convidados.

Também na camada do *hypervisor* na Figura 5.1 está presente um Sistema de Detecção de Intruso (IDS), pois é importante que o *hypervisor* disponha de um mecanismo de segurança capaz de monitorar as atividades dos sistemas convidados periodicamente. O objetivo é identificar modificações maliciosas no sistema convidado em tempo de execução. Este mecanismo de segurança é particularmente importante para os SE, uma vez que podem ser encarregados por executar tarefas que demandam o funcionamento ininterrupto por longos períodos de tempo. Além disso, este mecanismo, se implementado no *hypervisor*, pode ser beneficiado pela separação espacial. Assim, evitando ataques maliciosos que possam ocorrer a partir do sistema convidado contra o próprio mecanismo de segurança.

Na Figura 5.2 ilustra-se em detalhes o mecanismo de segurança denominado de Cadeia de Confiança. Este mecanismo é composto por dois mecanismos de segurança, que são: um hardware confiável (RoT) capaz armazenar um software de boot de maneira segura (resistente a violações) que por sua vez deve autenticar o *hypervisor* na inicialização; e um mecanismo de segurança encarregado por autenticar os sistemas convidados na inicialização (implementado na camada do *hypervisor*). É importante salientar que as assinaturas digitais e as chaves públicas, tanto do *hypervisor* quanto das VMs devem ser armazenadas de forma segura em uma memória resistente a violações, de maneira que após serem gravadas não possam ser modificadas.

Ao ser energizado, o SEV deve executar um software de boot, a partir de uma do hardware confiável com objetivo de garantir que o código executado está integro, ou seja, não foi modificado de forma maliciosa. Logo, este hardware onde está o boot é considerado intrinsecamente seguro, pois um atacante malicioso não poderá alterar os dados já armazenados. Assim, o boot é o primeiro elo na cadeia de confiança (estágio 1 na

Figura 5.2). Desta maneira, o boot passa a ser o responsável por verificar a autenticidade e integridade do *hypervisor* no estágio 2 (ilustrado na Figura 5.2).

Após a verificação do código do *hypervisor*, realizada pelo boot seguro, o *hypervisor* deve verificar a autenticidade e a integridade de cada sistema convidado (estágio 2 na Figura 5.2). Isto é feito por meio do mecanismo de segurança de Autenticação dos Sistemas Convidados. Após todas as camadas de software serem verificadas na inicialização, a Cadeia de Confiança está completa, assegurando que nenhum software modificado maliciosamente execute. Este nível de segurança pode ser alcançado com o uso de criptografia assimétrica combinada com funções de *hash* em todos os estágios da Cadeia de Confiança.

A Tabela 5.1 refere-se à relação os mecanismos de segurança propostos com os ataques aos quais os SEV estão expostos. Identificando como cada ataque pode ser mitigado com a adoção de um ou da combinação de diferentes mecanismos de segurança.

Tabela 5.1 – Ataques abordados pelos mecanismos de segurança.
Fonte: Elaborado pelo autor.

Ataques	Mecanismos de Segurança					
	CoT [1]	RoT [2]	Separação Espacial	IDS [3]	Gerenciador de Chaves	Processador Criptográfico
Modificação Externa do Sistema Convidado	✓	✓		✓		✓
Mobilidade do Sistema Convidado	✓					
Fuga do Sistema Convidado	✓		✓	✓	✓	✓
Ataques entre Sistemas Convidados	✓		✓	✓	✓	✓
Hyperjacking	✓	✓	✓			
Detecção do Sistema Convidado	✓			✓		
DoS	✓			✓		
Ataques ao DMA		✓				
Ataque de Canal Lateral		✓				

[1] Chain of Trust;
[2] Root of Trust;
[3] Intrusion detection system.

5.2 Especificação dos Modelos Implementados

A seguir são descritos o funcionamento e as técnicas de criptografia e virtualização utilizadas nos dois mecanismos de segurança implementados no decorrer deste trabalho. Os mecanismos têm como objetivo a proteção do sistema convidado, tanto em tempo de boot

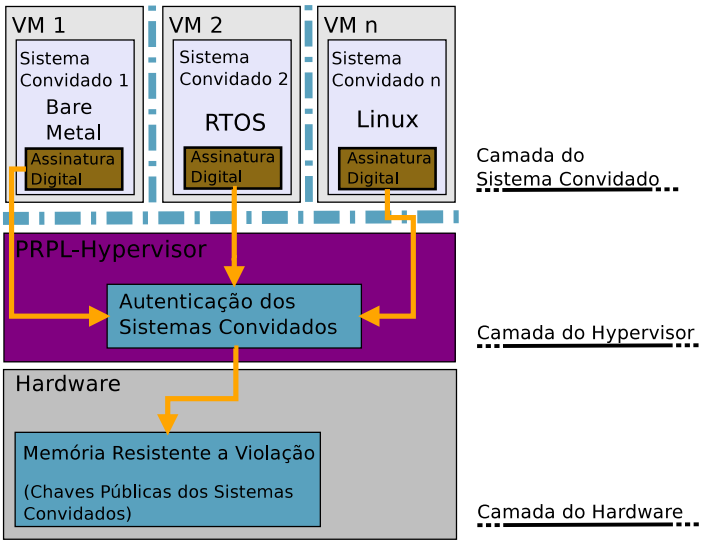

Figura 5.3 – Arquitetura da proposta de verificação de integridade para SEV.
Fonte: Elaborado pelo autor.

como em tempo de execução, pois como discutido anteriormente, existe uma quantidade significativa de ataques maliciosos que visam modificar o sistema convidado.

5.2.1 Mecanismo de Autenticação do Sistema Convidado

A Figura 5.3 ilustra o modelo de implementação para um mecanismo de verificação de integridade do sistema convidado. Cada um dos sistemas convidados possui uma assinatura digital que será verificada na inicialização. Esta assinatura digital é criada no momento da geração do arquivo binário do sistema convidado pelo desenvolvedor, bem como o *hash* calculado e o par de chaves criptográficas. Portanto, as assinaturas digitais devem ser adicionadas em cada um dos sistemas convidados. Uma vantagem desta abordagem é que as assinaturas digitais não dependem do tamanho do sistema convidado, uma vez que a assinatura digital é realizada sobre o valor do *hash*, que por sua vez possui tamanho fixo dependente da função escolhida (SHA-1, SHA-2).

É por meio do uso das assinaturas digitais que a abordagem é capaz de alcançar os seguintes requisitos de segurança: integridade, autenticidade, assim como o não repúdio. Além disto, é possível verificar que o sistema convidado não sofreu modificações maliciosas após sua criação (integridade). Que o sistema convidado foi fornecido por um desenvolvedor confiável (autenticidade). Por fim, se um sistema convidado apresentar alguma falha que comprometa a segurança, o responsável não poderá negar sua autoria (não repúdio).

No mecanismo proposto a assinatura digital para cada um dos sistemas convidados é gerada com o uso de algoritmos de curvas elípticas. Os algoritmos foram implementados por meio da biblioteca *Micro Elliptic Curve Cryptography* [30], desenvolvida utilizando os

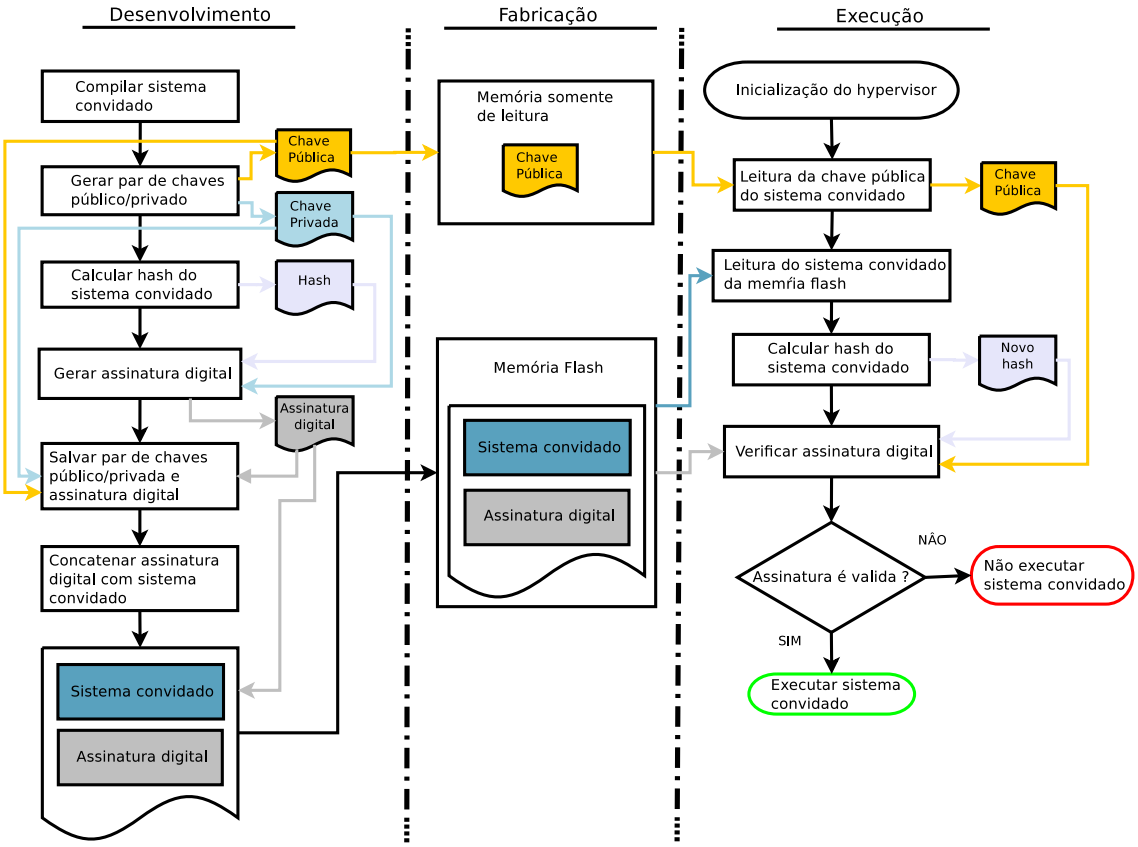

Figura 5.4 – Fluxo de funcionamento do mecanismo de verificação de integridade do sistema convidado.

Fonte: Elaborado pelo autor.

padrões recomentados pelo Grupo de Padronização para Eficiência em Criptografia (do inglês, *Standards for Efficient Cryptography Group* ou SECG) [4]. A Micro ECC implementa os algoritmos de troca de chaves Diffie–Hellman com curvas elípticas (do inglês, *Elliptic Curve Diffie–Hellman* ou ECDH) e de Assinatura Digital de Curvas Elípticas (do inglês, *Elliptic Curve Digital Signature Algorithm* ou ECDSA) disponíveis na licença BSD 2 [22]. As características que levaram a escolha desta biblioteca foram: ser escrita estritamente em linguagem C, suporte para arquiteturas de 32 *bits*, não utiliza alocação dinâmica de memória e suporta, até o momento, cinco curvas elípticas (secp160r1, secp192r1, secp224r1, secp256r1 e secp256k1). A curva elíptica escolhida para este trabalho foi a Koblitz Secp256K1 que também é utilizada na implementação do protocolo Bitcoin. Está curva é considerada segura, com força de 128 *bits* e tamanho de 256 *bits*.

Para as operações de *hash* foi utilizada a biblioteca escrita por Brad Conte denominada *Crypto Algorithms* [6] com o algoritmo SHA-256 (mesmo tamanho da curva elíptica) disponível em domínio público. Para a implementação adicionou-se os arquivos sha256.c e sha256.h ao projeto do prplHypervisor.

A Figura 5.4 ilustra o processo para realizar a verificação de integridade do sistema convidado. Assim, em tempo de desenvolvimento, ao compilar o sistema convidado uma

aplicação auxiliar é invocada para criar o par de chaves pública/privada, calcular o *hash* do sistema convidado, além de, com a chave privada e o *hash* gerar a assinatura digital e concatenar-la ao sistema convidado. A chave privada permanece com o desenvolvedor responsável por criar o sistema convidado, porque pode ser reutilizada para uma atualização do sistema convidado sem alterar a chave pública. A chave pública é gravada na ROM.

Após o desenvolvimento, o sistema convidado deve ser salvo na memória *flash* do SEV. Já em tempo de execução, quando o *hypervisor* for inicializado, ele deve ler a chave pública da memória ROM, em seguida realizar a leitura do sistema convidado para calcular o valor do *hash* e por fim ler a assinatura digital concatenada ao sistema convidado. Logo após, o *hypervisor* deve executar a verificação da assinatura digital e, se estiver correta, o sistema convidado é autorizado a executar, do contrário, o *hypervisor* impedirá a execução do sistema convidado. Esta verificação ocorre sempre que um sistema convidado é inicializado.

A proposta visa avançar no estado da arte apresentando um mecanismo de verificação de integridade que pode ser executado em SEV, capaz de detectar a injeção de código malicioso nos sistemas convidados, em tempo de boot, com as seguintes características:

- O mecanismo não requer modificação no sistema convidado. Assim, a solução pode ser estendida para um sistema de propósito geral, um sistema RTOS, assim como uma aplicação *bare metal*. Esta característica é relevante para os SEV devido à variedade de sistemas operacionais disponíveis nesta área, além dos sistemas proprietários;
- A garantia de não repúdio, não permite que um fabricante negue sua responsabilidade pela criação de um sistema convidado. Assim, caso um sistema convidado apresente mau funcionamento, o fabricante que forneceu o sistema convidado com problemas pode ser identificado e informado do ocorrido. Não podendo negar sua autoria.

5.2.2 Mecanismo de Introspecção do Sistema Convidado

A Figura 5.5 refere-se à arquitetura do mecanismo de introspecção do sistema convidado. O mecanismo de segurança funciona em duas etapas, no momento do boot, é necessário analisar o sistema convidado com objetivo de identificar quais serviços poderão ser utilizados por intermédio das chamadas de *hypercalls*. Isto é possível, pois cada *hypercall* utilizada no sistema convidado possui uma instrução de máquina já conhecida, assim, o mecanismo analisa as instruções de máquina presentes no binário do sistema convidado salvo na memória do SEV. Após a identificação, o mecanismo de introspecção deve gerar uma lista de *hypercalls* autorizadas. A seguir, já em tempo de execução, o mecanismo é responsável por interceptar todas as chamadas de *hypercalls* feitas pelo sistema convidado e verificar se a *hypercall* pertence à lista gerada na etapa anterior. Desta forma, se algum serviço não autorizado for acessado, esta chamada é interceptada e

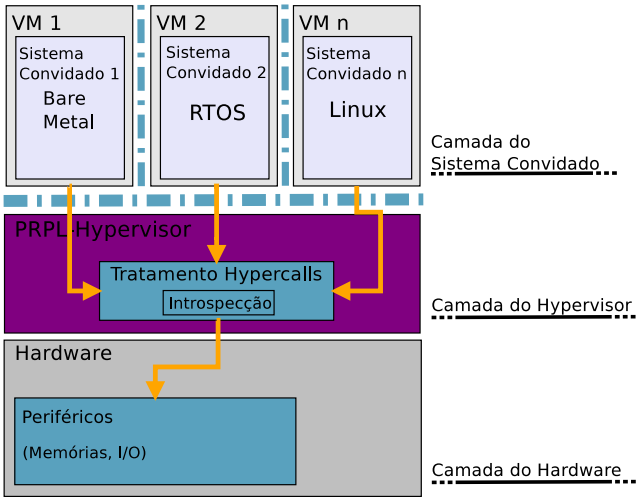

Figura 5.5 – Arquitetura do mecanismo de introspecção para SEV.
Fonte: Elaborado pelo autor.

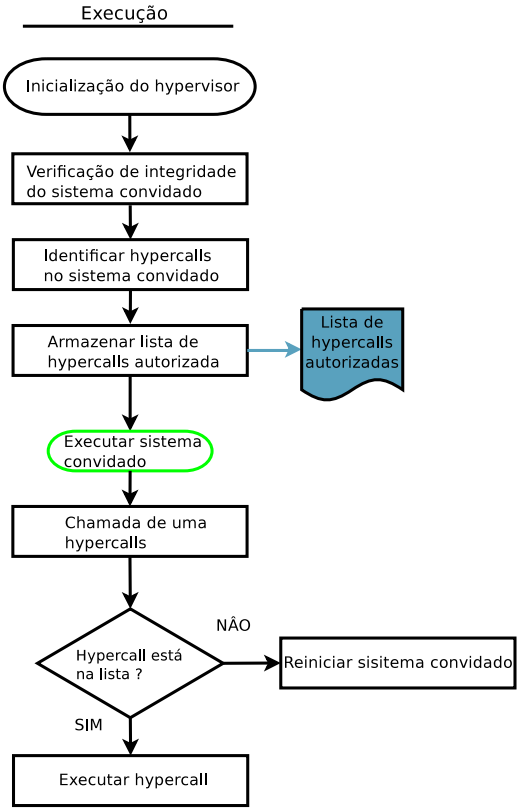

Figura 5.6 – Fluxo de funcionamento do mecanismo de introspecção do sistema convidado.
Fonte: Elaborado pelo autor.

bloqueada pelo *hypervisor*, assim, o sistema convidado é reiniciado e outra verificação de integridade será feita no momento do boot para identificar possíveis modificações maliciosas que possam ter ocorrido. A Figura 5.6 corresponde ao fluxo de funcionamento do mecanismo de introspecção de *hypercalls* do sistema convidado.

Para identificar as chamadas das *hypercalls* utilizou-se a ferramenta *objdump* para analisar o arquivo binário do sistema convidado. Esta ferramenta mostra informações

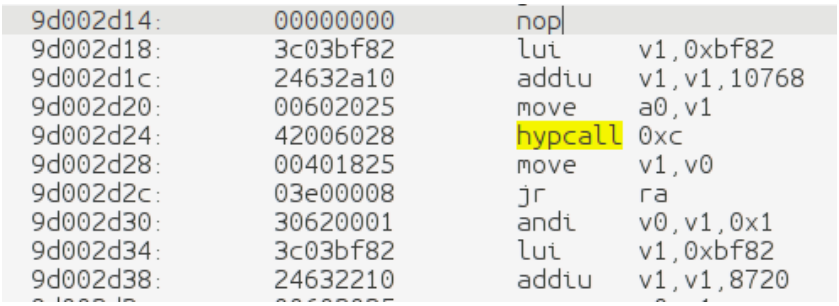

Figura 5.7 – Identificação da instrução das *hypercalls*.
Fonte: Elaborado pelo autor.

sobre um arquivo binário, como, por exemplo, apresentar as instruções de máquina de um arquivo. A Figura 5.7 ilustra um exemplo do resultado da execução do comando *"mips-mti-linux-gnu-objdump -d sistema-convidado.elf > objdump.txt"*. Destacado em amarelo está a *hypercall* seguida pelo código 0xc, referente à função readio. O processo de analisar o sistema convidado é realizado sempre que um sistema convidado é inicializado, embora esta abordagem imponha um tempo de atraso, ela não requer a atualização do mecanismo sempre que uma nova *hypercall* for criada, além disso, não é necessário que o prplHypervisor receba uma lista com as *hypercalls* autorizadas, o que acarretaria em uma forma segura para o envio e armazenamento de uma lista prévia. A lista de *hypercalls* é descrita a seguir:

1. guestid: solicitar número de identificação do sistema convidado;
2. ipc recv: receber mensagem entre sistemas convidados;
3. ipc send: enviar mensagem entre sistemas convidados;
4. guestup: verificar se o sistema convidado está executando;
5. eth watch: verificar o *link ethernet*;
6. eth send frame: enviar um dado para porta *ethernet*;
7. eth recv frame: receber um dado da porta *ethernet*;
8. eth mac: realizar leitura do endereço físico associado à porta *ethernet*;
9. usb polling: atualizar estado da porta do Barramento Serial Universal (do inglês, *Universal Serial Bus* ou USB);
10. usb device descriptor: ler o descritor da porta USB;
11. usb send: enviar dado para porta USB;
12. writeio: escrever endereços de memória virtual de I/O;
13. readio: ler endereços de memória virtual de I/O;

14. reenable interrupt: habilitar novamente uma interrupção;

15. get guestid: permitir ao sistema convidado ler o seu próprio número de identificação.

O mecanismo de introspecção do sistema convidado apresentado procura avançar o estado da arte apresentando uma abordagem que não necessita de recebimento de uma lista de serviços autorizados a serem utilizados pelo sistema convidado, assim, evitando a criação de mecanismos de envio e validação da lista, diminuindo a superfície exposta a ataques maliciosos. Além disso, a proposta pode ser implementada em SE voltados para IoT, sem que ocorra um degradação do desempenho para executar as chamadas de *hypercalls* e que o mecanismo não cause um impacto significativo no tamanho final do *hypervisor*. As principais características do mecanismo de segurança são:

- não é necessário que o *hypervisor* receba uma lista com as *hypercalls*, o mecanismo é capaz de extrair do sistema convidado à informação das *hypercalls* autorizadas.

- o mecanismo pode ser estendido para funcionar com outros sistemas convidados sem que seja necessário realizar modificações no seu código. Este aspecto é particularmente importante para os SE, pois pode-se trabalhar com sistemas convidados que executam Linux ou um sistema de tempo real, dependendo da aplicação final do SE.

5.3 Limitações da Implementação

A limitação do mecanismo de IDS implementado é a impossibilidade de detectar alterações de código maliciosas realizadas em tempo de execução do sistema convidado que não utilizam as *hypercalls*, isto inclui alguns tipos de ataques que podem tentar capturar informações dos usuários. Contudo, como estes ataques modificam o código do sistema convidado, podem ser detectados se houver uma reinicialização do sistema convidado.

Quanto ao mecanismo de verificação de integridade do sistema convidado feito pelo *hypervisor*, a principal limitação encontrada foi à indisponibilidade de uma memória resistente a violação disponível na placa de desenvolvimento. Assim, as chaves públicas foram armazenadas diretamente no código do *hypervisor*. Contudo, esta limitação não compromete a validade do experimento, além disso, outras placas de desenvolvimento do com o mesmo processador que utilizamos possuem uma memória resistente à violação, podendo este recurso ser implementado na medida em que se tenha acesso às placas.

5.4 Considerações Finais

Este capítulo apresentou um conjunto de mecanismos de segurança que podem ser utilizados para estabelecer níveis mínimos de confiança para os SEV. A arquitetura de segurança proposta é fundamentada com os mecanismos de segurança que hoje são implementados de maneira isolada em sistemas virtualizados com o objetivo de proteger todas as camadas da arquitetura. Além disso, detalhou-se dois modelos de mecanismos de segurança com objetivo é identificar modificações maliciosas no sistema convidado.

O modelo de verificação de integridade dos sistemas convidados durante a inicialização utiliza algoritmos de assinaturas digitais com curvas elípticas, bem como a função *hash* SHA-256, com objetivo de assegurar os seguintes requisitos de segurança: autenticidade, integridade e não repúdio. O uso de curvas elípticas em SE têm sido estudado por oferecer boa performance em sua computação se comparado ao RSA. Neste trabalho foi possível implementar a verificação de integridade aliado a um desempenho adequado mesmo com as restrições dos SEV. O mecanismo está localizado na camada do *hypervisor*, que está separado espacialmente dos sistemas convidados.

Na proposta para criação do mecanismo de introspecção do sistema convidado, analisou-se os eventos de chamadas de *hypercalls*, ou seja, analisou-se se o sistema convidado está tentando acessar algum recurso não autorizado a ele. Uma limitação da abordagem ocorre se um sistema convidado infectado tentar atacar algum recurso que lhe foi previamente autorizado, desta maneira não seria possível identificar o ataque. Contudo, no futuro o mecanismo pode ser estendido para analisar outros eventos.

No capítulo 6 serão apresentados os resultados práticos obtidos em decorrência da implementações dos mecanismo de segurança de verificação de integridade e introspecção dos sistemas convidado, cujo modelos foram detalhados neste capítulo.

6. RESULTADOS

Neste capítulo são apresentados os resultados alcançados pela implementação dos dois mecanismos de segurança no prplHypervisor. O capítulo está dividido da seguinte forma: na Seção 6.1 é descrito o hardware e o *hypervisor* onde os mecanismos de segurança foram implementados; na Seção 6.2 são apresentados os critérios utilizados para a avaliação. A Seção 6.3 mostra os resultados da implementação do mecanismo de verificação de integridade. A Seção 6.4 apresenta os resultados da implementação do mecanismo de introspecção. Já a Seção 6.5 oferece as conclusões sobre os resultados obtidos.

6.1 Ambiente de Implementação

Esta seção apresenta o ambiente onde foram realizadas as implementações dos mecanismos de segurança para verificação de integridade na inicialização e de introspecção do sistema convidado. Por fim, são descritos o hardware e o *hypervisor* utilizados.

6.1.1 Placa de Desenvolvimento Microchip PIC32-MZ

Para conduzir os testes foram utilizados a placa de desenvolvimento fornecida pela Microchip chamada PIC32MZ, que oferece uma forma rápida e barata para conduzir experimentos com SE voltados para a IoT. A placa possui o processador MIPS M5150 capaz de executar a 200 MHz. Além disso, oferece 2 MB de memória *flash* e 512kB de memória RAM. Também estão disponíveis os seguintes periféricos: uma porta USB, barramento de Controle de Rede(do inglês, *Controller Area Network* ou CAN), duas portas seriais e uma porta *ethernet*.

6.1.2 O Processador MIPS M5150

O processador criado peal empresa Imagination modelo MIPS M5150 foi criado especificamente para os SE e disponibilizado para o mercado em 2013. As principais características do processador são: um núcleo, suporte a virtualização em hardware com o módulo MIPS VZ, 32-bits de endereçamento, Unidade de Ponto Flutuante (do inglês, *Floating Point Unit* ou FPU), até 16 registradores de propósito geral e *pipeline* de 5 estágios [21].

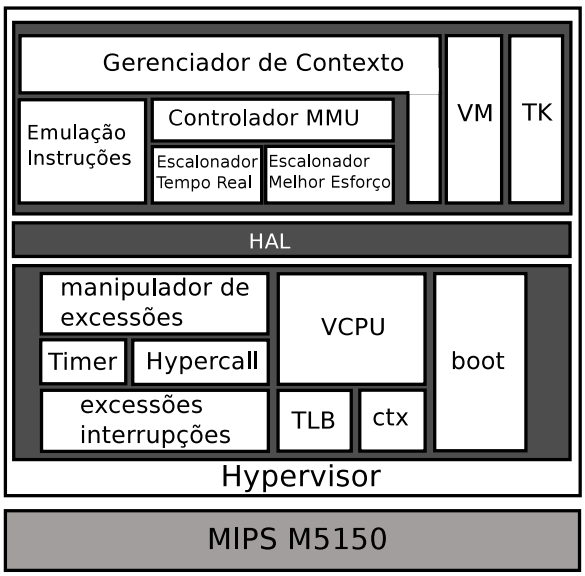

Figura 6.1 – Arquitetura do prplHypervisor onde os mecanismos de segurança foram implementados. Adaptado de [34].

6.1.3 O Hypervisor prplHypervisor

O prplHypervisor foi desenvolvido para trabalhar com SE com uma camada de virtualização bastante reduzida, o que permite a execução em SE com restrição de capacidade de memória e processamento. O prplHypervisor utiliza uma abordagem de virtualização híbrida. A virtualização completa é utilizada para virtualizar os sistemas convidados. Já a para-virtualização é utilizada para oferecer suporte as *hypercalls* criando serviços como, por exemplo, comunicação entre sistemas convidados. Além disso, é implementada a separação espacial e temporal entre os sistemas convidados e também entre o *hypervisor* e os sistemas convidados, contribuindo para a segurança do SEV [33]. A Figura 6.1 refere-se à arquitetura do prplHypervisor mostrando os diferentes componentes que formam esta camada de software [34]. Cada um dos componentes que formam a arquitetura do prplHypervisor é descrito a seguir.

A Camada de Abstração do Hardware(do inglês, *Hardware Abstraction Layer* ou HAL) é a responsável pela implementação das rotinas de baixo nível, mantendo os detalhes do hardware ocultos das camadas de mais alto nível. O boot é o primeiro código a executar após a reinicialização do processador, ficando encarregado de configurar o processador e copiar o código do *hypervisor* para a área de memória onde será executado. Esta camada é utilizada quando o desenvolvimento ocorre no simulador IASim, caso contrário, o boot utilizado é nativo da plataforma Microchip PIC32MZ utilizada na implementação.

O prplHypervisor possui dois escalonadores. O de melhor esforço é responsável por escalonar as CPUs de propósito geral. Já o de tempo real se encarrega de escalonar as CPUs que precisam atender rotinas dentro de uma janela de tempo restrita. O gerenciador

de contexto é responsável por encaminhar a VCPU para a CPU física após o escalonador definir qual processo será executado, utilizando as rotinas implementadas na HAL.

O emulador de instruções é o módulo responsável por executar as instruções privilegiadas que os sistemas convidados não podem realizar, como por exemplo, alterar o modo de gerenciamento de energia. Já o módulo de instância da VM é utilizado para inicialização e configuração das máquinas virtuais. O *hypervisor* ainda possui o módulo de ferramentas (do inglês, *Toolkit* ou TK), que implementa algoritmos e funções de propósito geral, como, por exemplo, listas encadeadas e funções de manipulação de *strings* [34].

6.2 Critérios de Avaliação

Para analisar a proposta a validação foi dividida em dois requisitos aplicados aos dois mecanismos implementados: funcionalidade e desempenho. Quanto à funcionalidade, foi avaliado se os mecanismos de segurança detectam os ataques maliciosos aos quais se propõe mitigar, além disso, faz parte do requisito de funcionalidade analisar o impacto do mecanismo de segurança no crescimento do código do prplHypervisor, mostrando a viabilidade da adoção do mecanismo de segurança para os SEV. No que diz respeito ao desempenho, foi analisado o impacto da segurança no tempo de atraso imposto na inicialização do SEV, bem como no tempo para gerar a lista de *hypercall* no mecanismo de introspecção.

6.3 Resultados Obtidos com o Mecanismo de Verificação da Integridade do Sistema Convidado

6.3.1 Detecção de Ataques

Para assegurar que o mecanismo de verificação de integridade pode detectar modificações maliciosas no sistema convidado ou na assinatura digital foram realizadas alterações com objetivo de simular ataques maliciosos. Assim, provando o funcionamento do mecanismo de segurança. Os testes foram executados 200 vezes para cada ataque.

O primeiro teste consiste em modificar (apenas um byte) o código do sistema convidado para simular o ataque malicioso após a geração da assinatura digital. Assim, o sistema convidado salvo na memória do SE contém uma assinatura digital que não corresponde ao sistema convidado modificado. Desta maneira, demonstra-se que a alteração no sistema convidado foi detectada e é suficiente para impedir a sua execução. Isto ocorre, pois o resultado do cálculo da função *hash* SHA-256 é diferente do sistema convidado não

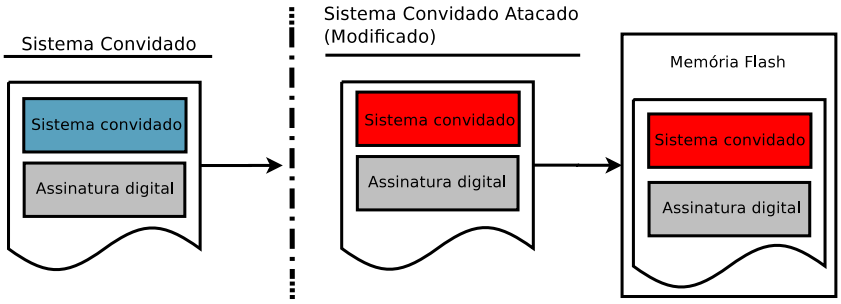

Figura 6.2 – Teste com modificação do sistema convidado.
Fonte: Elaborado pelo autor.

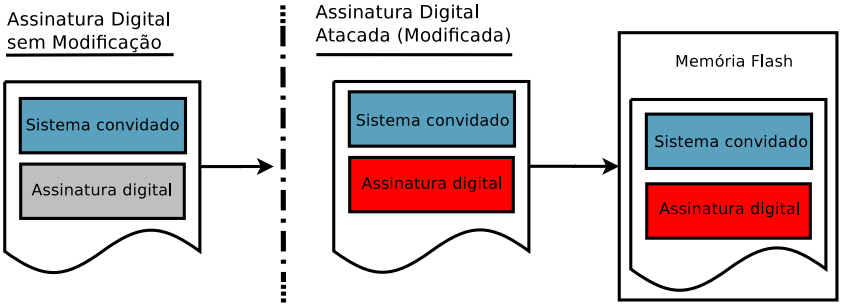

Figura 6.3 – Teste com modificação da assinatura digital.
Fonte: Elaborado pelo autor.

modificado, logo, no momento da verificação da assinatura digital com a chave pública e o novo *hash*, o ataque pode ser identificado. Na Figura 6.2 ilustra-se este tipo de ataque malicioso, é apresentado o sistema convidado sem modificações (em azul) que é alterado pelo atacante (em vermelho) e em seguida é armazenado na memória *flash* do SEV.

O segundo teste conduzido foi a alteração de um byte da assinatura digital do sistema convidado. Este tipo de ataque pode ocorrer caso um atacante tente gerar uma assinatura digital com uma chave privada que não corresponde à chave pública correta do sistema convidado. Os testes mostraram que o mecanismo de segurança é capaz de identificar que a assinatura digital não foi gerada com a chave privada correspondente ao sistema convidado. Portanto, este tipo de ataque também foi identificado pelo mecanismo de segurança. Na Figura 6.3 mostra-se como é feito este ataque. Em cinza é representada a assinatura digital correspondente ao sistema convidado, em seguida ocorre o ataque contra a assinatura digital (em vermelho) que posteriormente é salva na memória *flash* do SE.

6.3.2 Crescimento do Código do prplHypervisor

A Tabela 6.1 corresponde ao tamanho final do prplHypervisor, assim como o crescimento do código do prplHypervisor em relação à versão original após a implementação do mecanismo de verificação de integridade do sistema convidado. Observa-se um crescimento

Tabela 6.1 – Tamanho final do prplHypervisor após implementação do mecanismo de verificação de integridade.

Fonte: Elaborado pelo autor.

Segmento	Inicial (bytes)	Final (bytes)	Diferença (bytes)	Crescimento (%)
Código (text)	20.996	32.192	11.196	53,32

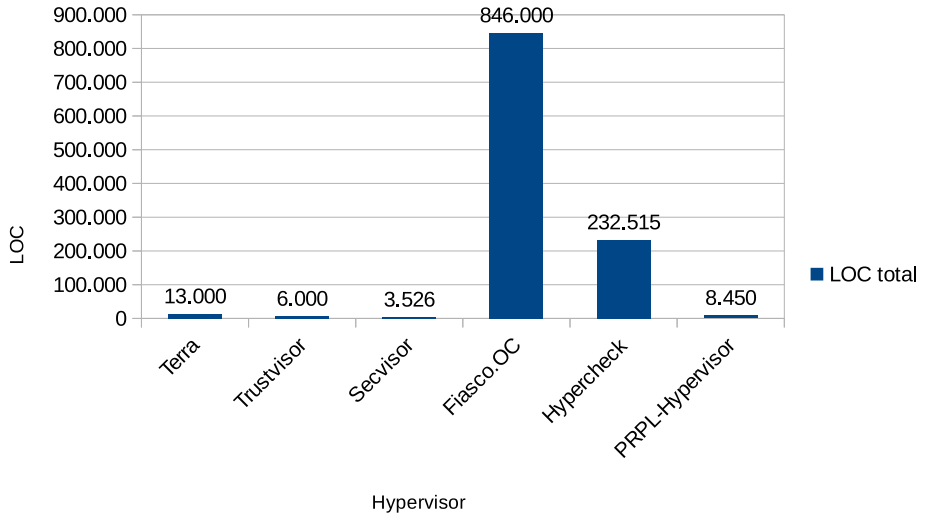

Figura 6.4 – Comparação do número de linhas do prplHypervisor com o estado da arte.
Fonte: Elaborado pelo autor.

no código original do prplHypervisor. Contudo, o tamanho total da memória *flash* disponível para o prplHypervisor e os sistemas convidado é de 2MB (ou 2.097.152 bytes). Portanto, após a implementação do mecanismo de segurança, o prplHypervisor passou a ocupar 32.192 bytes da memória *flash*, o que representa o uso de 1,1% do espaço disponível na memória. Assim, fica disponível para os sistemas convidados 2.064.960 bytes de memória.

Outra maneira de medir o crescimento do software é por meio do número de Linhas de Código (do inglês, *Lines of Code* ou LOC). Esta medida é importante, pois também permite a comparação com outras implementações. Foram necessárias 2.149 LOC em linguagem C para implementar o mecanismo de segurança, incluindo as funções de *hash* e criptografia assimétrica. O prplHypervisor sem este mecanismo de segurança possui 6.301 LOC. Na Figura 6.4 apresenta-se uma comparação com os trabalhos relacionados, os valores são apresentados em LOC e mostram o tamanho total da implementação, ou seja, não está estratificado o quanto cada proposta utilizou de LOC para a implementação dos mecanismos de segurança. Assim, é possível notar que os *hypervisors* Hypercheck e Fiasco apresentam um número de LOCs elevado, pois são desenvolvidos para sistemas computacionais sem restrições de recursos. A medição de LOC foi realizada com a ferramenta desenvolvida por Wheeler chamada SLOCCount [50].

Em comparação com o estado da arte, a proposta apresentada neste trabalho se equipara, em LOCs, com os mecanismos implementados nos *hypervisors* Terra, TrustVisor

e SecVisor. Entretanto, os trabalhos citados não implementam os mesmos recursos que o prplHypervisor utilizado neste trabalho, como, por exemplo, dois escalonadores ou *hypercalls*. Além disso, não utilizam funções criptográficas como, por exemplo, curvas elípticas implementadas em software, o que impacta no total de LOCs.

Estes resultados indicam que a adoção do mecanismo de segurança no prplHypervisor pode ser realizada sem comprometer o seu desempenho. O crescimento apresentado no código não inviabiliza sua utilização, uma vez que o espaço de memória disponível na plataforma onde foi implementado não foi comprometido. O código com as implementações está disponível no Github em [48].

6.3.3 Impacto no Tempo de Execução

A medição do tempo de execução de cada tarefa foi feita utilizando um contador atualizado a uma frequência de 100Mhz. Os dados são apresentados em milissegundos, que representa o tempo decorrido para realizar a tarefa. As tarefas analisadas foram:

- O tempo desprendido para calcular o *hash* SHA-256 do sistema convidado inteiro;
- O tempo necessário para verificar se a chave recebida possui o tamanho esperado para a curva selecionada, bem como se é um ponto válido para uma curva elíptica;
- O tempo total desprendido para executar a verificação da assinatura digital.

Os resultados são calculados como uma média simples de 200 iterações para sistemas convidados de diferentes tamanhos do tipo *bare metal*. Os valores foram medidos para sistemas convidados não modificados, portanto, caso não ocorra um ataque, este é o tempo médio de atraso imposto pelo mecanismo de segurança a cada inicialização do sistema convidado. O código da implementação está disponível no Github em [48].

Apresenta-se na Tabela 6.2 um resumo dos tempos necessários para cada operação. Como esperado, o tempo de verificação da assinatura digital e da validação da chave pública independe do tamanho do sistema convidado. Para sistemas convidados de até 128kB, a verificação da assinatura digital consome a maior fatia de tempo. Contudo, a função *hash* cresce conforme o aumento do tamanho do sistema convidado, que a partir de 256kB já consome 60% do tempo total. Para o tamanho máximo de um sistema convidado (1.024kB) o tempo de cálculo do *hash* representa, em média, 76% do tempo total de toda a verificação.

A Figura 6.5 refere-se à um gráfico para melhor visualização do tempo total para verificação de integridade dos sistemas convidados com tamanho entre 32kB e 1.024kB.

O tempo total desprendido para inicializar os sistemas convidados é de aproximadamente 68,38ms sem o mecanismo de verificação de integridade (este tempo não depende

Tabela 6.2 – Tempo de atraso médio (200 iterações) imposto na inicialização.
Fonte: Elaborado pelo autor.

Tamanho do Sistema Convidado (KB)	Hash (ms)	Chave Pública (ms)	Assinatura Digital (ms)	Total (ms)
32	11,5708	0,0464	55,3707	66,9879
64	23,1655	0,0476	57,0077	80,2208
128	46,3549	0,0467	55,7378	102,1395
256	92,7338	0,0466	57,5575	150,3380
512	156,2587	0,0471	56,7774	213,0833
1024	185,4916	0,0474	56,5885	242,1276

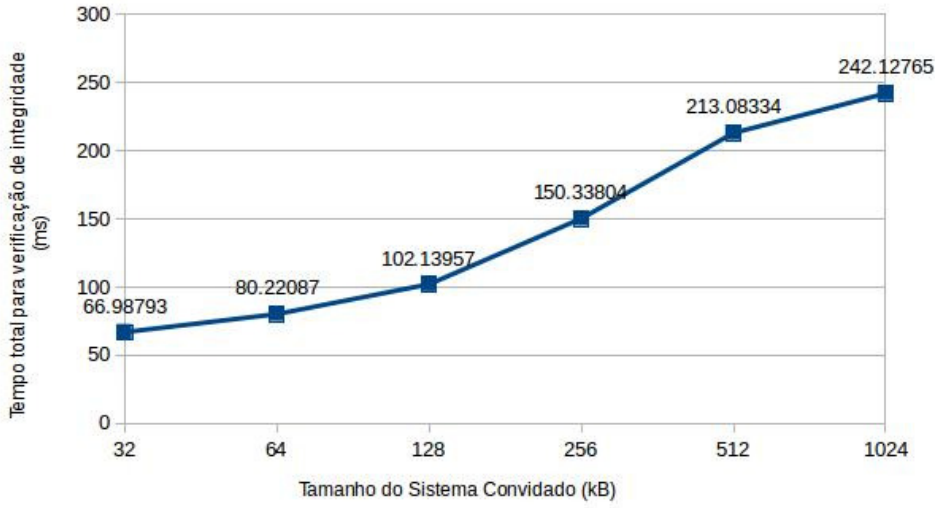

Figura 6.5 – Tempo de atraso médio para 200 iterações.
Fonte: Elaborado pelo autor.

do tamanho do sistema convidado). Logo, o tempo total de inicialização de um sistema convidado de 32KB é de 135,36ms, este tempo representa a soma do tempo de inicialização (68,38ms) somado ao tempo necessário para verificação do sistema convidado (66,98ms).

6.4 Resultados Obtidos com o Mecanismo de Introspecção do Sistema Convidado

6.4.1 Detecção de Ataques

Para verificar se o mecanismo de introspecção de *hypercalls* realmente seria capaz de detectar uma modificação no sistema convidado que realiza uma chamada de *hypercall* não autorizada, foi conduzido um teste para simular este tipo de ataque. O teste consiste basicamente em criar uma tabela de *hypercalls* inválida, ou seja, que não corresponde ao

Tabela 6.3 – Tamanho final do prplHypervisor após implementação do mecanismo de introspecção de *hypercalls*.

Fonte: Elaborado pelo autor.

Segmento	Inicial (bytes)	Final (bytes)	Diferença (bytes)	Crescimento (%)
Código (text)	20.996	21124	128	0,61

Tabela 6.4 – Tamanho final do prplHypervisor somados os dois mecanismo de segurança implementados.

Fonte: Elaborado pelo autor.

Segmento	Inicial (bytes)	Final (bytes)	Diferença (bytes)	Crescimento (%)
Código (text)	20.996	32.320	11.324	53,93

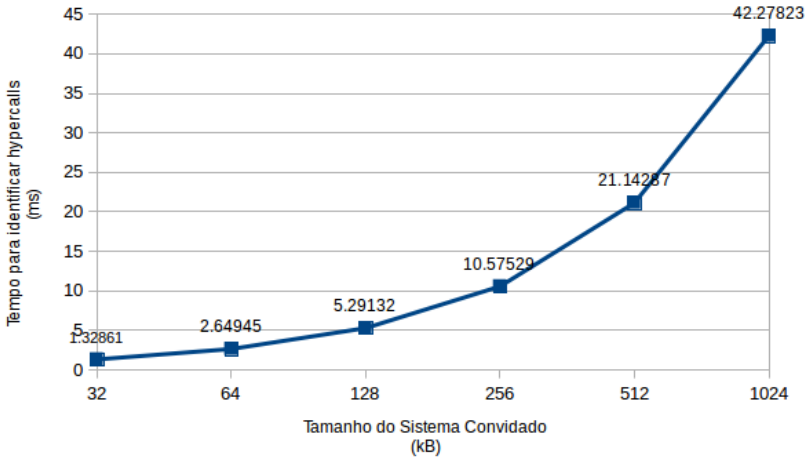

Figura 6.6 – Tempo médio necessário para identificar *hypercalls* no sistema convidado.

Fonte: Elaborado pelo autor.

sistema convidado. Já durante a execução do sistema convidado, foram feitas diferentes chamadas de *hypercalls* não autorizadas. O prplHypervisor foi capaz de detectar de forma eficiente todas as *hypercalls* não autorizadas, demonstrando que o mecanismo funciona.

6.4.2 Crescimento do Código do prplHypervisor

Para realizar a implementação do mecanismo de segurança foi necessário adicionar 120 LOC ao código original do prplHypervisor. A Tabela 6.3 refere-se ao crescimento do prplHypervisor para o mecanismo de introspecção. Mostra-se na Tabela 6.4 o tamanho final do prplHypervisor após a implementação dos dois mecanismos de segurança implementados neste trabalho (a verificação de integridade e introspecção do sistema convidado).

6.4.3 Impacto no Tempo de Execução

Foram realizadas 200 iterações e calculada uma média simples para os diferentes tamanhos de sistemas convidados do tipo *bare metal*. O código da implementação está disponível em [48]. Para o maior sistema convidado possível (1.024kB) o tempo necessário para identificar as *hypercalls* foi de 42ms. Este é o tempo de atraso imposto no momento da inicialização. A Figura 6.6 ilustra o tempo necessário para identificar as *hypercalls* em sistemas convidados com tamanhos que variam entre 32kB até 1.024kB. Foram testados sistemas convidados com treze *hypercalls* implementadas. É importante ressaltar que o tempo de atraso é imposto sempre que o sistema convidado for inicializado, pois o prplHypervisor não armazena ou recebe previamente a lista de *hypercalls* autorizadas.

6.5 Considerações Finais

Neste capítulo apresentou-se os resultados obtidos a partir da implementação dos dois mecanismos de segurança especificados no capítulo 5. Foram estabelecidos também critérios de desempenho e crescimento do código do prplHypervisor para avaliar a proposta. Foi apresentado também o ambiente onde os dois mecanismo de segurança foram implementados, descrevendo as principais características presentes no hardware utilizado nesta pesquisa.

Os resultados encontrados são considerados satisfatórios, uma vez que, os mecanismos implementados foram eficazes na detecção de código malicioso no sistema convidado, o que era o principal objetivo. Além disso, o tempo de atraso imposto por cada um dos mecanismos de segurança é aceitável, mostrando que é possível balancear a utilização de mecanismos de segurança com técnicas de criptografia adequadas aos SE sem comprometer o desempenho dos SEV. É importante ressaltar que a implementação resultou em um crescimento significativo do código do prplHypervisor apenas para o mecanismo de verificação de integridade. Entretanto, este crescimento não inviabiliza a adoção deste mecanismo de segurança. O mecanismo de introspecção de *hypercalls* não impôs atrasos significativos além de não aumentar significativamente o tamanho do prplHypervisor.

Em comparação com o estado da arte, os resultados apresentados indicam que o número de linhas do prplHypervisor se equipara com os menores *hypervisor* estudados. Além disso, foram conduzidos testes com diferentes tamanhos de sistemas convidados, o que também não comprometeu a adoção dos dois mecanismo de segurança. No capítulo 7 são oferecidas as conclusões finais sobre esta pesquisa, as principais contribuições, assim como são apontados os trabalhos futuros derivados desta pesquisa.

7. CONCLUSÕES FINAIS

Este capítulo concluí esta dissertação revisitando os temas abordados durante a condução desta pesquisa. O capítulo está dividido da seguinte maneira: a Seção 7.1 apresenta as conclusões finais. Já a Seção 7.2 resume as principais contribuições oferecidas por este trabalho. A Seção 7.3 indica as direções para os trabalhos futuros.

7.1 Conclusões

Os SE ainda são desenvolvidos com foco apenas nas suas funcionalidades, com intuito de criar soluções inovadoras para os problemas do dia a dia das pessoas, contudo, acaba-se deixando a segurança em segundo plano. Entretanto, devido à importância dos SE voltados para a IoT a segurança passa a ser uma característica determinante para o sucesso das empresas que atuam neste novo mercado. É justamente neste cenário, onde os SE alcançam uma larga escala e estão cada vez mais presentes na vida das pessoas, como, por exemplo, nas cidades inteligentes, que a técnica de virtualização emerge como uma solução viável capaz de contribuir para melhorar a utilização dos recursos disponíveis no hardware, no tempo de desenvolvimento dos SE, bem como no aumento da segurança.

Todavia, no que diz respeito unicamente à segurança, a técnica da virtualização traz consigo novas ameaças antes não encontradas nos SE, uma vez que é adicionada uma camada de software nesta nova arquitetura. Assim, somente a virtualização não é capaz de proteger os SE contra os ataques maliciosos conhecidos aos quais os dispositivos IoT estão expostos. Logo, é necessário o uso de técnicas de criptografia adequadas aos SE combinadas com a virtualização de maneira que permita a criação de ambientes seguros. É importante salientar que os SE possuem um poder computacional limitado se comparados aos sistemas computacionais de propósito geral, portanto, é importante escolher cuidadosamente as técnicas de criptografia adequadas para os mecanismo de segurança.

Como agravante deste cenário, não existe uma padronização e tão pouco uma legislação que estabeleça ou recomende um conjunto de mecanismos de segurança que devem ser adotados na criação de novos produtos com objetivo de resguardar a privacidade, assim como os dados de usuários e empresas. A segurança precisa estar presente desde a concepção dos produtos voltados para IoT, passando a ser tão importante quanto a sua funcionalidade. Assim, a segurança não pode ser vista apenas como um acessório ou característica adicionada aos produtos após falhas e incidentes já terem ocorrido.

Por conseguinte, o objetivo deste trabalho foi identificar os ataques maliciosos conhecidos na literatura capazes de comprometer a segurança dos SEV, categoriza-los frente à camada alvo e identificar como os ataques podem ser aplicados contra os SEV

voltados para IoT. Assim, com base nos ataques definiu-se um conjunto de mecanismos de segurança possíveis de serem implementados nos SEV, criando assim uma arquitetura de segurança padronizada, independente de tecnologias proprietárias, que visa estabelecer um nível mínimo de confiança entre usuários e os SEV. Os mecanismos de segurança sugeridos podem ser implementados seguindo diferentes modelos, esta característica é importante pois os mecanismo precisam evoluir com o tempo na medida em que novas técnicas de criptografia sejam propostas. Além disso, existe uma vasta gama de desenvolvedores e fabricantes de SE, permitindo que cada um implemente os mecanismo de segurança da maneira que melhor se adapta aos seus ambientes e de acordo com seus objetivos.

Além da arquitetura de segurança, nesta pesquisa foram implementados dois dos mecanismos de segurança que compõe a arquitetura proposta, especificando seus modelos e expondo as técnicas utilizadas. Implementou-se um mecanismo responsável pela verificação de integridade dos sistemas convidados durante a inicialização e, em seguida, implementou-se um mecanismo dedicado para a introspecção dos sistemas convidados durante sua execução. Os mecanismos foram implementados na placa de desenvolvimento PIC32MZ.

Os resultados indicam que é viável a adoção dos dois mecanismos de segurança propostos em um SEV por meio de técnicas de criptografia adequadas, que asseguram que atingiu-se os requisitos de segurança mínimos desejados sem comprometer o desempenho dos SEV. Os mecanismos de segurança implementados detectam os ataques maliciosos, bem como não comprometem as restrições de tamanho de memória, assim como não impõe atrasos significativos durante a execução do *hypervisor* ou dos sistemas convidados. Desta maneira, foi estabelecido um nível de confiança mínimo entre os usuários e os SEV.

7.2 Contribuição

Abaixo são listadas a principais contribuições oferecidas por esta pesquisa.

- A definição de uma arquitetura de segurança para SEV com intuito de mitigar ataques maliciosos conhecidos contra sistemas virtualizados. Os mecanismos presentes na arquitetura são baseados em protocolos e técnicas de segurança já existentes, combinando virtualização com criptografia. Além disso, a arquitetura pode ser implementada em outras plataformas, uma vez que é independente de tecnologias proprietárias;
- Implementação de um mecanismo de segurança para verificação de integridade de sistemas convidados no prplHypervisor. O mecanismo não compromete o desempenho e é capaz de garantir os seguintes requisitos: integridade, autenticidade e não repúdio.
- Implementação de um mecanismo de introspecção de *hypercalls* no prplHypervisor. O mecanismo é capaz de identificar se um sistema convidado tentou acessar algum

serviço não autorizado sem comprometer a performance e o tamanho do *hypervisor*. Assim, é possível identificar que o mecanismo foi infectado com código malicioso;

- Uma ampla pesquisa abrangendo as principais técnicas e protocolos de segurança existentes, as vulnerabilidades e requisitos de segurança que afetam os SE e como a virtualização combinada a técnicas de criptografia podem mitigá-los, além do estado da arte dos mecanismos de verificação de integridade de sistemas convidados e introspecção, implementados em *hypervisors* amplamente utilizados no mercado.

7.3 Trabalhos Futuros

Esta seção indica as oportunidades de melhorias que podem ser implementadas no futuro, bem como caminhos a serem explorados na continuação deste trabalho.

1. Implementou-se no prplHypervisor a geração de números aleatórios utilizando-se a funcionalidade disponível no hardware do processador M5150 o que permitiu o desenvolvimento dos algoritmos ECDSA e ECDH através do uso da biblioteca [30]. Desta forma, o prplHypervisor agora é capaz de assinar mensagens e trocar chaves de forma segura. o próximo passo é comparar a performance obtida com estes dois algoritmos com o estado da arte, bem como procurar otimizar o seu desempenho;
2. Implementar outros elementos propostos na arquitetura de segurança, como, por exemplo, a cadeia de confiança com o boot seguro para verificar a integridade do prplHypervisor, além de acessar funções criptográficas disponíveis no hardware.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Aguiar, A.; Hessel, F. “Embedded systems’ virtualization: The next challenge?” In: 21st IEEE International Symposium on Rapid System Protyping, 2010, pp. 1–7.

[2] Azab, A. M.; Ning, P.; Sezer, E. C.; Zhang, X. “Hima: A hypervisor-based integrity measurement agent”. In: Annual Computer Security Applications Conference, 2009, pp. 461–470.

[3] Belgarric, P.; Fouque, P.-A.; Macario-Rat, G.; Tibouchi, M. “Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones”. Cham: Springer International Publishing, 2016, pp. 236–252.

[4] Brown, D. R. “Sec 2: Recommended elliptic curve domain parameters”, *Standars for Efficient Cryptography*, vol. 1, 2010.

[5] Center, C. C. “Understanding denial-of-service attacks”. Capturado em: https://www.us-cert.gov/ncas/tips/ST04-015, Fevereiro 2013.

[6] Conte, B. “Crypto algorithms.” Capturado em: https:/github.com/B-Con/crypto-algorithms, Dezembro 2015.

[7] Diffie, W.; Hellman, M. “New directions in cryptography”, *IEEE Transactions on Information Theory*, vol. 22–6, Nov 1976, pp. 644–654.

[8] Elgamal, T. “A public key cryptosystem and a signature scheme based on discrete logarithms”, *IEEE Transactions on Information Theory*, vol. 31–4, Jul 1985, pp. 469–472.

[9] Fattori, A.; Lanzi, A.; Balzarotti, D.; Kirda, E. “Hypervisor-based malware protection with accessminer”, *Computers and Security*, vol. 52, 2015, pp. 33 – 50.

[10] Garfinkel, T.; Pfaff, B.; Chow, J.; Rosenblum, M.; Boneh, D. “Terra: A virtual machine-based platform for trusted computing”. In: 19th ACM Symposium on Operating Systems Principles, 2003, pp. 193–206.

[11] Gartner. “Gartner says 6.4 billion connected "things"will be in use in 2016, up 30 percent from 2015”. Capturado em: http://www.gartner.com/newsroom/id/3165317, Maio 2016.

[12] Gartner. “Gartner says worldwide iot security spending to reach 348 million in 2016”. Capturado em: http://www.gartner.com/newsroom/id/3291817, Abril 2016.

[13] Geffner, J. “Venom vulnerabilidade”. Capturado em: http://venom.crowdstrike.com/, Agosto 2015.

[14] Genkin, D.; Pachmanov, L.; Pipman, I.; Tromer, E.; Yarom, Y. “Ecdsa key extraction from mobile devices via nonintrusive physical side channels”. In: ACM Conference on Computer and Communications Security, 2016, pp. 1626–1638.

[15] Group, T. C. “Trusted platform module specifications”. Capturado em: https://www.trustedcomputinggroup.org/specs/TPM, Julho 2016.

[16] Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. “Internet of things (iot): A vision, architectural elements, and future directions”, *Future Generation Computer Systems*, vol. 29–7, 2013, pp. 1645–1660.

[17] Hankerson, D.; Menezes, A. J.; Vanstone, S. “Guide to elliptic curve cryptography”. Springer Science and Business Media, 2006.

[18] Heiser, G. “The role of virtualization in embedded systems”. In: 1st Workshop on Isolation and Integration in Embedded Systems, 2008, pp. 11–16.

[19] Heiser, G. “Virtualizing embedded systems: Why bother?” In: 48th Design Automation Conference, 2011, pp. 901–905.

[20] Hofmann, O. S.; Kim, S.; Dunn, A. M.; Lee, M. Z.; Witchel, E. “Inktag: Secure applications on an untrusted operating system”, 2013, pp. 265–278.

[21] Imagination. “M-class m51xx core family”. Capturado em: https://imgtec.com/mips/warrior/m-class-m51xx-core-family/, Julho 2016.

[22] Initiative, O. S. “The 2-clause bsd license”. Capturado em: https://opensource.org/licenses/BSD-2-Clause, Janeiro 2017.

[23] Irazoqui, G.; Eisenbarth, T.; Sunar, B. “A: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes”. In: IEEE Symposium on Security and Privacy, 2015, pp. 591–604.

[24] Kanda, W.; Yumura, Y.; Kinebuchi, Y.; Makijima, K.; Nakajima, T. “Spumone: Lightweight cpu virtualization layer for embedded systems”. In: IEEE International Conference on Embedded and Ubiquitous Computing, 2008, pp. 144–151.

[25] Kerry, C. F.; Director, C. R. “186-4: Federal information processing standards publication. digital signature standard (dss)”, *Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD*, 2013, pp. 20899–8900.

[26] Kivity, A.; Kamay, Y.; Laor, D.; Lublin, U.; Liguori, A. “kvm: the linux virtual machine monitor”. In: Linux symposium, 2007, pp. 225–230.

[27] Koblitz, N. “Elliptic curve cryptosystems”, *Mathematics of computation*, vol. 48–177, 1987, pp. 203–209.

[28] Kocher, P.; Jaffe, J.; Jun, B.; Rohatgi, P. “Introduction to differential power analysis”, *Journal of Cryptographic Engineering*, vol. 1–1, Abril 2011, pp. 5–27.

[29] Lee, S. W.; Yu, F. “Securing kvm-based cloud systems via virtualization introspection”. In: 47th Hawaii International Conference on System Sciences, 2014, pp. 5028–5037.

[30] MacKay, K. “Biblioteca micro ecc.” Capturado em: https://github.com/kmackay/micro-ecc, Julho 2015.

[31] McCune, J. M.; Li, Y.; Qu, N.; Zhou, Z.; Datta, A.; Gligor, V.; Perrig, A. “Trustvisor: Efficient tcb reduction and attestation”. In: IEEE Symposium on Security and Privacy, 2010, pp. 143–158.

[32] Miller, V. S. “Use of elliptic curves in cryptography”. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 417–426.

[33] Moratelli, C.; Johann, S.; Neves, M.; Hessel, F. “Embedded virtualization for the design of secure iot applications”. In: 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, 2016, pp. 2–6.

[34] Moratelli, C. R. “A lightweight virtualization layer with ardware-assistance for embedded systems”, Tese de Doutorado, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, RS, BR, 2016, 135p.

[35] Noergaard, T. “Embedded systems architecture: a comprehensive guide for engineers and programmers”. Langford Lane, Kidlington, Oxford: Newnes, 2012, 672p.

[36] Paar, C.; Pelzl, J. “Understanding cryptography: a textbook for students and practitioners”. Springer Science and Business Media, 2009.

[37] Perez, R.; Sailer, R.; van Doorn, L.; et al.. “vtpm: virtualizing the trusted platform module”. In: 15th Conference on USENIX Security Symposium, 2006, pp. 305–320.

[38] Ray, E.; Schultz, E. “Virtualization security”. In: 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, 2009, pp. 42:1–42:5.

[39] Sahoo, J.; Mohapatra, S.; Lath, R. “Virtualization: A survey on concepts, taxonomy and associated security issues”. In: 2nd International Conference on Computer and Network Technology, 2010, pp. 222–226.

[40] Scarfone, K. A.; Souppaya, M. P.; Hoffman, P. “Sp 800-125. guide to security for full virtualization technologies”, *National Institute of Standards and Technology*, 2011.

[41] Seshadri, A.; Luk, M.; Qu, N.; Perrig, A. “Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses”, *ACM Special Interest Group in Operating Systems*, vol. 41–6, Out 2007, pp. 335–350.

[42] Shi, Y.; Zhao, B.; Yu, Z.; Zhang, H. “A security-improved scheme for virtual tpm based on kvm”, *Wuhan University Journal of Natural Sciences*, vol. 20–6, 2015, pp. 505–511.

[43] Shimada, H.; Nakajima, T. “External integrity checking with invariants”. In: 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2011, pp. 122–125.

[44] Smith, J.; Nair, R. “Virtual machines: versatile platforms for systems and processes”. San Francisco, CA, USA: Elsevier, 2005, 656p.

[45] Stallings, W. “Cryptography and Network Security: Principles and Practice”. Pearson Higher Ed, 2013.

[46] Stankovic, J. A. “Research directions for the internet of things”, *IEEE Internet of Things Journal*, vol. 1–1, Feb 2014, pp. 3–9.

[47] Stewin, P.; Bystrov, I. “Understanding dma malware”. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2012, pp. 21–41.

[48] Vasconcelos, M. D. “Implementação dos mecanismos de verificação de integridade e introspecção de sistemas convidados no prplhypervisor.” Capturado em: https://github.com/mathDv/prpl-hypervisor, Maio 2017.

[49] Weiß, M.; Wagner, S.; Hellman, R.; Wessel, S. “Integrity verification and secure loading of remote binaries for microkernel-based runtime environments”. In: 13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2014, pp. 544–551.

[50] Wheeler, D. A. “Sloccount tool.” Capturado em: https://www.dwheeler.com/sloccount/, Janeiro 2017.

[51] Yang, J.; Kim, H.; Park, S.; Hong, C.; Shin, I. “Implementation of compositional scheduling framework on virtualization”, *Special Interest Group on Embedded Systems*, vol. 8–1, Mar 2011, pp. 30–37.

[52] Young, E. A.; Hudson, T. J. “The openssl project”. Capturado em: https://www.openssl.org/, Maio 2016.

[53] Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. “Internet of things for smart cities”, *IEEE Internet of Things Journal*, vol. 1–1, Feb 2014, pp. 22–32.

[54] Zhang, F.; Wang, J.; Sun, K.; Stavrou, A. “Hypercheck: A hardware-assistedintegrity monitor”, *IEEE Transactions on Dependable and Secure Computing*, vol. 11–4, July 2014, pp. 332–344.

[55] Zhang, Y.; Juels, A.; Reiter, M. K.; Ristenpart, T. “Cross-vm side channels and their use to extract private keys”. In: ACM Conference on Computer and Communications Security, 2012, pp. 305–316.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br