Universidade de Aveiro Electrónica, Telecomunicações e Informática,
2015

Simão Reis

Ecossistema para exploração do DETIboot no âmbito de exames

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2015

Simão Reis

Ecossistema para exploração do DETIboot no âmbito de exames

"Ultimately, there are only two actions you can take in a game. Advance your own strategy, or respond to another player's strategy."

— Sora and Shiro

Simão Reis

Ecossistema para exploração do DETIboot no âmbito de exames

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica de André Ventura da Cruz Marnoto Zúquete e José Manuel Neto Vieira, professores do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president	**Professor Doutor Tomás António Mendes Oliveira e Silva** Professor Associado da Universidade de Aveiro
vogais / examiners committee	**Professor Doutor Carlos Nuno da Cruz Ribeiro** Professor Associado do Instituto Superior Técnico da Universidade de Lisboa (arguente)
	Professor Doutor André Ventura da Cruz Marnoto Zúquete Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Agradeço principalmente aos meus orientadores, Professor André Zúquete e Professor José Vieira, pela oportunidade que me deram em trabalhar num projeto que considerei original e inovador e por todo o apoio que me deram e todo o conhecimento que me transmitiram.

Agradeço também à minha família e todos os meus amigos e colegas, por todo o apoio que me deram ao longo da vida.

Palavras chave

Comunicação por Difusão Segura, Deteção de Máquinas Virtuais, Segurança em Linux

Resumo

A evolução tecnológica tem levado a que os portáteis tenham ganho cada vez mais uso, visto que estes são cada vez mais poderosos e dispõem cada vez mais de maior mobilidade, causando o desuso progressivo dos computadores fixos.

O fenómeno anterior verifica-se atualmente nas universidades, onde os alunos preferem transportar os seus computadores pessoais para realizar as tarefas das aulas, tornando de certa forma desnecessário os gastos que os institutos académicos fazem com a população das salas de aula com computadores fixos.

Numa dissertação de mestrado anterior foi desenvolvido o DETIboot, o sistema de carregamento rápido de sistemas operativos Linux em computadores remotos, oferecendo aos docentes a oportunidade de executar facilmente um sistema operativo, configurado pelos próprios, nos computadores portáteis dos alunos.

O problema coloca-se na realização de provas de exame, onde o uso dos computadores dos alunos é indesejável. O objetivo desta dissertação foi estender o sistema DETIboot, permitindo que este possa ser usado de forma segura nas provas de exame. A extensão passa pelo reforço da imagem Linux distribuída, tomando esta o controlo das permissões que os alunos têm sobre os seus computadores, passando-as assim para os docentes, e por outro lado garantir a transferência e arranque seguro do sistema operativo.

Keywords Secure Broadcast Communication, Virtual Machines Detection, Linux Security

Abstract

Technological progress led to an increased use of laptop computers, as they are becoming more powerful and granting greater mobility, causing the gradual disuse of desktop computers.

The described phenomenon is currently happening in universities, where students prefer to carry and use their laptops to work on required tasks from classes, rather than using the available desktop computers. Therefore, the budget spent by academic institutions on populating classroms with desktop computers becomes unnecessary.

In an former master's thesis was developed DETIboot, a fast on-the-fly loading system of Linux operating systems, granting teachers the opportunity to easily run an operating system, set up by them, on the student's laptops. A problem arises in exam evaluation, where the use of student computers is undesirable.

The aim of this work was to extend the DETIboot system, allowing it to be used in a secure fashion during exams. The extension englobes hardening the Linux image by passing the permission's control from the students to the teachers, and ensuring the secure transmission and booting of the operating system.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Glossário

μC micro-controlador. 65

AC Alignment Check. 47

ADC Analog to Digital Converter. 76

AFBP Authenticated File Broadcast Protocol. 30, 61, 71, 75, 76

AM Alignment Mask. 47

AMD-V AMD Virtualization. 20, 48, 52

AP Access Point. 8

API Application Programming Interface. 24

BIOS Basic Input/Output System. 45

BSSID Basic Service Set Identifier. 29, 31–34, 36, 40, 63, 71, 72, 75

CPU Central Processor Unit. 15, 20, 21, 29, 35, 43, 46–48, 50, 51, 53–55, 58

CPUID CPU IDentification. 48, 53, 55

CR0 Control Register 0. 47, 48, 50, 54

CR3 Control Register 3. 55

CR8 Control Register 8. 55

DETI Departamento de Eletrónica, Telecomunicações e Informática. 1, 61, 67, 70, 73

DHCP Dynamic Host Configuration Protocol. 69

DMI Desktop Management Interface. 45

DoS Denial of Service. 28–30, 34

DRM Digital Rights Managment. 76

E/S Entrada/Saída. 1, 3, 19, 21, 44, 45, 61, 63, 65, 69, 70, 75

EFER Extended Feature Enable Register. 55

FBP File Broadcast Protocol. 27–37, 39, 40, 62–64, 70–72, 75, 76

GDT Global Descriptor Table. 43

GDTR GDT Register. 43

GP General Protection. 22

HTTP HyperText Transfer Protocol. 64, 69

HTTPS Hyper Text Transfer Protocol Secure. 64, 69

IBSS Independent Basic Service Set. 29

IdP Identity Provider. 68, 76

IDT Interrupt Descriptor Table. 42, 43

IDTR IDT Register. 42

IP Internet Protocol. 1, 4, 24, 25, 62, 64, 67–69

JVM Java Virtual Machine. 17

LDT Local Descriptor Table. 43

LDTR LDT Register. 43

MAC Medium Access Control. 44, 45, 62, 66

MAC Message Authentication Code. 12, 14, 29

MMU Memory Management Unit. 20, 21

MSR Model-Specific Register. 46, 47, 54, 55, 58

MTRR Memory Type Range Register. 50

NFS Network File System. 61, 63, 69

NIC Network Interface Controller. 66

NOP No OPeration. 48, 56–58

NTP Network Time Protocol. 49

OUI Organizationally Unique Identifier. 44, 45

PAM Pluggable Authentication Modules. 23, 24, 69

PC Personal Computer. 1, 4, 61, 62, 64, 67, 68, 72, 75, 76

PCI Peripheral Component Interconnect). 45, 46

PTE Page Table Entries. 52, 54

RAM Random Access Memory. 3, 9, 10, 37, 63

RDMSR Read MSR. 49, 55

RDPMC Read Performance Measurement Counter. 55, 58

RDTSC Read TSC. 48–50, 55, 58

REP Repeat String. 47

RSA Rivest, Shamir e Adleman. 13, 14, 27, 30–33, 40

RSB Return Stack Buffer. 50

RSSI Received Signal Strength Indication. 36

SGDT Store GDT. 43

SHA Secure Hashing Algorithm. 27, 30–34, 39, 40

SIDT Store IDT. 42, 43

SLDT Store LDT. 43

SO Sistema Operativo. 2–4, 7–10, 14–16, 18–23, 41, 42, 47, 53, 61, 63–70, 72, 75

SSL Secure Sockets Layer. 64

STR Store Task Register. 44

SVME Secure Virtual Machine Enable. 55

TCP Transport Control Protocol. 25

TESLA Timed Efficient Stream Loss-tolerant Authentication. 40

TLB Translation Lookaside Buffer. 20, 21, 52–54

TLS Transport Layer Security. 64

TPM Trusted Platform Module. 67

TSC Time Stamp Counter. 48–50, 53, 55

TSS Task State Segment. 44

UA Universidade de Aveiro. 1, 4, 55, 61, 68, 69, 76

USB Universal Serial Bus. 4, 8, 37, 65, 70, 72

UU Utilizador Universal. 62

VM Virtual Machine. 2–5, 7, 15–17, 19–23, 41–46, 48, 50, 51, 53–55, 58, 59, 67, 69, 75

VMBC Virtual Machine Control Blocks. 20

VMCS Virtual-Machine Control data Structure. 20, 23

VMENTRY Virtual Machine Entry. 49

VMEXIT Virtual Machine Exit. 48, 50, 51, 59

VMHalt Virtual Machine Halt. 41, 47, 53–55, 58, 59, 61, 69, 75

VMM Virtual Machine Monitor. 17

VT-x Virtualization Technology. 20, 22, 23, 43, 48, 52

WBINVD Write Back and Invalidate Cache. 50

Wi-Fi Wireless Fidelity. 2, 8–10, 32, 34, 37

WRMSR Write MSR. 47

Capítulo 1

Introdução

No Departamento de Eletrónica, Telecomunicações e Informática (DETI) da Universidade de Aveiro (UA) já há muitos anos que se realizam exames práticos de programação em Personal Computer (PC) *desktop* instalados nas salas de aula. Estes possuem sistemas Linux com as ferramentas de apoio à realização das aulas práticas e exames práticos e encontram-se preparados para barrar o acesso à maior parte dos sites da Web e a dispositivos de Entrada/Saída (E/S). A ligação à Internet é feita através da infraestrutura de rede com fios montada no departamento.

Para conduzir a prova, os docentes começam por registar manualmente a presença dos alunos da sala, deslocando-se de lugar a lugar para que os alunos assinem a presença e verificam a identidade do aluno através do Cartão Universitário ou do Cartão do Cidadão. Com o objetivo de controlar a prova é criada uma conta Linux dedicada: apenas usando esta conta os alunos podem realizar a prova e entregar a mesma. Para os alunos poderem aceder à conta, os docentes fornecem as credenciais de acesso à mesma.

A entrega da prova é feita através de um canal seguro através da Internet. Este canal consiste na ligação por cabo e filtragem de endereços Internet Protocol (IP). Ou seja, um aluno que obtenha as credenciais, não pode usufruir destas fora da(s) sala(s) de avaliação.

Nos últimos anos, os computadores portáteis tornaram-se progressivamente mais poderosos, e o seu preço tem descido de forma acentuada. Isto fez com que um número maior de utilizadores abdique dos PC *desktop*. No cursos lecionados por docentes do DETI tem-se assistido a uma crescente utilização de portáteis por parte dos alunos, abdicando estes da utilização dos PC *desktop* existentes nas salas de aula usadas em aulas práticas ou laboratórios. Porém, os computadores disponibilizados são bastante dispendiosos, tanto em investimento como em manutenção, e com o aumento do seu desuso justifica-se diminuir estes custos.

O abandono dos PC *desktop* vai levar a que todas as aulas práticas, laboratórios e exames de informática usem apenas os portáteis dos alunos como ferramenta de trabalho. Um dos problemas de mais difícil resolução é o da realização de exames usando os portáteis pessoais dos alunos. No cenário anterior, em que se utilizavam apenas os PC *desktop* do DETI, os docentes podiam configurar o sistema de modo a barrar os serviços indesejados, mas com a utilização dos portáteis dos alunos tal não é aparentemente possível.

1.1 Objectivo

O primeiro passo para resolver o problema anterior passou pela criação do DETIboot, concebido e concretizado no âmbito de uma dissertação de mestrado anterior. O DETIboot é um sistema baseado numa *pen bootable* que arranca uma distribuição Live de Linux a partir de um servidor usando a rede Wireless Fidelity (Wi-Fi) e códigos Fountain. A principal vantagem do DETIboot é que permite que um número ilimitado de portáteis fiquem plenamente operacionais com o mesmo Sistema Operativo (SO) em apenas cerca de um minuto, sem recorrer a qualquer infraestrutura e sem alterar de forma persistente o conteúdo do portátil.

O DETIboot foi originalmente idealizado com o intuito de ser explorado na realização de aulas práticas e de exames usando os computadores pessoais dos alunos, mas até ao início desta dissertação apenas se tinha alcançado o objetivo de conseguir distribuir rapidamente uma mesma imagem Linux para uma população arbitrariamente grande de computadores pessoais heterogéneos.

A realização das provas de exame nos computadores portáteis dos alunos com a distribuição de uma imagem dedicada ao apoio e controlo da prova através do DETIboot traz novos problemas com que vamos ter de lidar. Nesta dissertação procurou-se atender aos múltiplos requisitos de segurança que envolve a exploração do DETIboot numa prova de exame, bem como os requisitos operacionais de produção de imagens Linux bem adaptadas a tais provas e a aulas práticas, e a sua distribuição fidedigna através do DETIboot.

1.2 Problema

Nos parágrafos seguintes são elencados os vários problemas que se identificaram relativamente à exploração do DETIboot e de imagens de SO Linux para a realização de exames usando os computadores portáteis dos alunos.

O DETIboot faz uso de um protocolo de difusão de ficheiros. Este usa códigos Fountain, que permitem que os recetores consigam reconstruir um ficheiro transmitido pelo emissor usando quaisquer *codewords* transmitidas. Uma *codeword* é uma combinação de blocos dos ficheiro original. Atualmente os recetores não validam a imagem Linux descarregada. Mais, sendo o meio de transmissão uma rede Wi-Fi *ad hoc* aberta, é possível a um atacante modificar e retransmitir tramas anteriormente difundidas ou injetar novas tramas.

Num exame um aluno pode executar a imagem distribuída sobre uma Virtual Machine (VM), tendo desse modo a possibilidade de correr no seu computador dois sistemas em paralelo: o sistema hospedeiro da VM e o sistema distribuído para realizar o exame (hospedado na VM). Caso isso fosse permitido o aluno poderia beneficiar de facilidades facultadas pelo hospedeiro que não são desejadas durante o exame (acesso à Internet ou a ficheiros do seu computador) mas que não estão sob a alçada do sistema hospedado (imagem difundida que, supostamente, deveria ter um controlo total sobre o computador do aluno).

A difusão da imagem é feita através de uma rede sem fios Wi-Fi aberta, a qual todos os recetores existentes no seu raio de ação podem ter acesso. Tal permite que um aluno localizado no exterior da sala de exame possa ter acesso a imagem difundida e realizar o exame por um colega que esteja dentro da sala de exame.

Assim, sem controlo da unicidade de sessão em prova, dois alunos podem entrar com a mesma conta e realizar a prova em conjunto, partilhando o trabalho um com o outro.

Até aqui o DETIboot tem sido usado com imagens Linux existentes para uso genérico (e.g.

SLAX), as quais podem ser suficientes para muitas aulas práticas. Porém, as imagens que se pretende usar num exame deverão seguir o princípio do privilégio mínimo: tudo o que nelas não for necessário deverá ser eliminado porque pode ser usado pelo aluno para realizar ações indesejadas. Isto significa que em cada exame é preciso criar uma imagem bem adaptada ao tipo de acessos que é suposto estarem ao alcance dos alunos, e barrar todos os demais.

Decorre do ponto anterior que o bloqueio do acesso a dispositivos de E/S terá de ser feito com o intuito do aluno perder os privilégios de leitura (e escrita) de qualquer dispositivo de memória persistente do seu computador (ou ligados ao seu computador). Por este facto, o trabalho realizado durante um exame não pode ser salvaguardado em memória persistente no computador do aluno, existe apenas em Random Access Memory (RAM), o que representa um risco de perda total em caso de falha do computador.

De forma a resolver o problema anterior, os alunos entregam o trabalho que produzem de forma automatizada através de um sistema centralizado. Os docentes podem usar esse sistema para recolher o trabalho entregue pelos alunos. O problema é que, mesmo que o docente consiga verificar que o trabalho foi entregue por um dado aluno, também tem que conseguir verificar que foi entregue através do SO difundido e apenas por alunos dentro da sala de exame.

Finalmente, existindo uma forma de recuperação do trabalho realizado por um aluno após uma falha do seu portátil, o mesmo poderia alternar entre o uso, no seu portátil, do sistema nativo do mesmo e do sistema difundido, fugindo assim ao controlo deste último. Para evitar que tal seja feito, é necessário assegurar que a falha na execução do sistema difundido seja detetada pelo docente logo após a sua ocorrência, e não apenas quando o aluno pretender recuperar trabalho salvaguardado.

1.3 Contribuição

Neste trabalho pretendeu-se melhorar o ecossistema em redor do DETIboot, contribuindo com várias funcionalidades de segurança que são vitais para a realização de exames usando os portáteis dos alunos. Como se apresentou na secção anterior, os problemas identificados têm índoles muito diversas, e por isso a solução dos mesmos implicou o estudo e concretização de soluções em grande medida independentes, que iremos seguidamente descrever de forma sumária.

A primeira contribuição é um mecanismo de autenticação de origem para a distribuição de imagens de SO através do DETIboot. De forma a não alterar o comportamento base do protocolo de difusão de ficheiros do DETIboot adicionámos ao mesmo elementos adicionais para autenticação de *codewords* [21]. Concretamente, foi introduzindo um novo tipo de mensagem (autenticador) que é transmitida periodicamente e que autentica um conjunto das próximas *codewords* ainda por transmitir. Para o realizar de forma eficiente, o autenticador transporta a síntese das *codewords* que autentica, assinadas pelo emissor. A chave pública de validação da assinatura digital é também transportada pelo autenticador. Esta estratégia de transmissão oferece vantagens ao nível de implementação, desempenho computacional e do aproveitamento dos recursos de rede.

A segunda contribuição é um sistema de verificação que, durante o arranque da imagem do SO para exame, detete se está ou não a executar sobre uma VM. Este foi implementado sobre a forma de um módulo Linux integrado na imagem distribuída. Sendo um módulo Linux, a verificação é durante o processo de arranque da imagem. O módulo corre vários testes, e caso

um deles detete a presença de uma VM, o SO termina imediatamente a sua execução.

A terceira contribuição foi a definição do que deverá ser excluído de, ou adicionado a, uma imagem Linux para uma prova de exame, tendo em conta as imagens usualmente disponíveis para uso geral.

A imagem para exame deve incluir vários mecanismos que ponham em prática um conjunto de políticas restritivas definidas pelo docente: (i) impedir o acesso a quaisquer dispositivos que possam ser usados como fonte não autorizada de informação, através do bloqueio dos módulos Linux (*device drivers*); (ii) autorizar o acesso IP a apenas a alguns endereços da Internet (a plataforma educacional moodle, por exemplo), através dos mecanismos nativos de *firewall* do Linux; (iii) desativar os binários que permitem a promoção do nível de privilégios do utilizador em Linux, como `su` e `sudo`. Esta imagem é construída a partir de uma outra base a partir de um processo orientado pelo paradigma "tudo o que não é necessário deverá ser negado". Ou seja, cabe ao docente especificar que direitos devem ser mantidos, e não os que devem ser negados.

A transferência segura de elementos relevantes para o exame (enunciado e demais ficheiros de apoio), é facilmente concretizado incluindo os elementos na imagem Linux difundida quando esta é produzida pelo docente, não sendo necessário distribuir estes através de outros canais.

A transferência segura periódica dos resultados produzidos por um aluno é concretizada durante a realização do exame através de um repositório central do docente.

O aluno vai registar-se e autentica-se no exame com as suas credencias universais da UA, o que é vantajoso para o aluno porque este não tem que memorizar novas credenciais, e é vantajoso para o sistema de armazenamento, porque diminui a sua complexidade, não sendo necessário implementar um sistema de informação para registar e armazenar os dados dos alunos para o âmbito de exames. O docente tem acesso aos dados dos alunos inscritos na unidade curricular e pode confrontar com a lista dos alunos que depositaram o seu trabalho no servidor central.

Para verificar que a origem de entrega do trabalho produzido pelo aluno é o SO distribuído pelo docente, a imagem distribuída traz embutido uma "impressão digital" que a distingue de todas as outras imagens.

Para evitar que um aluno alterne entre o uso, no seu computador, do SO nativo do mesmo e do SO difundido, é necessário assegurar que a falha na execução do sistema difundido seja detetada pelo docente logo após a sua ocorrência. Usamos um esquema de *heartbeat*, em que o SO corre um processo que envia, periodicamente, mensagens *keep alive* para o docente de forma a monitorizar os alunos.

Finalmente, a quarta contribuição foi a definição de estratégias para resolver o problema da distinção dos alunos dentro e fora da sala de exame. São propostas duas estratégias. A primeira envolve a ligação de um dispositivo aos computadores dos alunos dentro da sala de aula (e.g. via Universal Serial Bus (USB)); apenas com esse dispositivo o SO fica autorizado a participar no exame. A segunda estratégia passa por ligar os computadores dos alunos a uma infraestrutura com acesso à rede por cabo. A prova do exame apenas pode ser entregue através das interfaces de rede dentro da sala de aula (por filtragem IP). Esta solução segue o modelo atual de exames usando os PC *desktop* montados nas salas de aula.

1.4 Organização

A dissertação está organizada da seguinte forma. Depois desta introdução, no Capítulo 2 introduzimos fundamentos teóricos úteis à compreensão do texto restante. O leitor familiarizado com os tópicos de comunicação por difusão segura, máquinas virtuais ou segurança em geral em sistemas operativos Linux poderá saltar este capítulo. Contudo, recomendamos fortemente a leitura da secção 2.1, em que descrevemos o sistema DETIboot.

Nos capítulos seguintes descrevemos o trabalho realizado e os resultados alcançados. No Capítulo 3 descrevemos o novo protocolo desenvolvido para a validação pelos recetores do conteúdo que lhes é transmitido por difusão via DETIboot; o conteúdo deste capítulo é fundamentalmente uma tradução de [21].

No Capítulo 4 descrevemos o sistema de deteção de VM.

No Capítulo 5 descrevemos o ecossistema de configuração do ambiente de execução da prova de exame após o arranque da imagem correta nos computadores dos alunos.

Finalmente, no Capítulo 6 recapitulamos todo o trabalho realizado, terminando com os problemas deixados em aberto.

Capítulo 2

Contexto

Neste capítulo contextualizamos os fundamentos teóricos aplicados no desenvolvimento desta dissertação.

Começamos por descrever a arquitetura do DETIboot, o sistema de arranque rápido de portáteis remotamente, de forma a clarificar o cenário que precisa de ser protegido. Exploramos os vários componentes que constituem o DETIboot, as ferramentas usadas na sua produção, na sua instalação e na sua operacionalização. De seguida introduzimos várias tecnologias que vão permitir proteger o DETIboot durante a prova de exame.

Começamos com criptografia básica. Esta é a base da comunicação segura por difusão, necessária para a validação por parte dos alunos da imagem difundida pelo docente.

De seguida exploramos as capacidades dos protocolos de atestação de software, que envolve a prova de identidade de um software, como um SO por exemplo, a uma entidade remota. No nosso caso estamos interessados em conseguir verificar a identidade do SO que corre no computador de cada aluno.

Depois exploramos os principais paradigmas de virtualização do ambiente de execução de um SO. Estes são fundamentais à compreensão das estratégias de deteção da existência virtualização realizada por VM originais ou modificadas.

Por último resumimos alguns mecanismos de segurança que operam a vários níveis dos sistemas operativos Linux (*firewall*, autenticação e controlo de acesso). Estes são necessários à criação de uma imagem reforçada que controle os computadores dos alunos durante a prova de exame.

O leitor familiarizado com todos estes tópicos pode avançar de imediato para o próximo capítulo.

2.1 DETIboot

O DETIboot baseia-se numa arquitetura cliente-servidor [6], onde um nó central (servidor ou emissor) difunde uma imagem de um SO e os restantes nós (clientes ou recetores) descarregam a imagem e arrancam a mesma após a transferência estar concluída. A arquitetura encontra-se ilustrada na Figura 2.1.

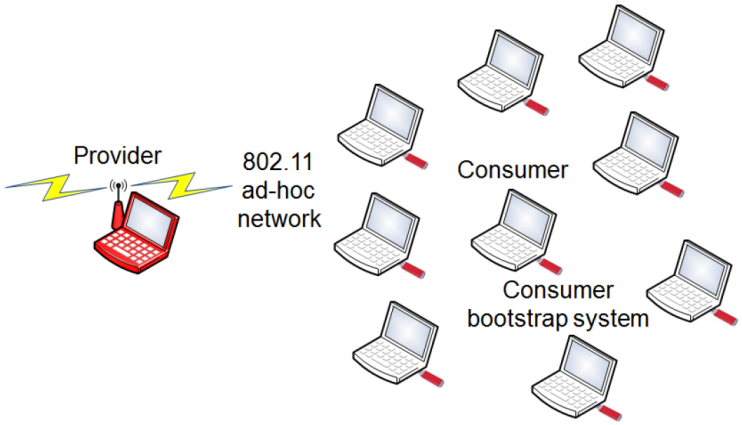

Figura 2.1: Arquitetura do DETIboot. O computador portátil vermelho (*Provider*) difunde a imagem do SO através de uma rede 802.11 em modo *ad hoc*. Os restantes computadores portáteis (*Consumers*) usam o método de arranque do DETIboot (*Consumer bootstrap system*), este armazenado em dispositivos de armazenamento de dados externos USB (pens USB a vermelho) para descarregar e arrancar o novo SO.

2.1.1 Comunicação

A comunicação é feita utilizando redes sem fios Wi-Fi. Estas permitem comunicação entre os clientes e o servidor num determinado espaço geográfico limitado de forma rápida e sem a necessidade de pré-instalação, contrariamente às redes com ligação por cabo onde é necessário equipamento fixo previamente instalado. Portanto é necessário que os sistemas computacionais, estejam munidos de dispositivos de rede sem fios Wi-Fi, de forma a poderem comunicar entre si.

As redes sem fios Wi-Fi podem ser estruturadas ou *ad hoc*. Visto que a utilização de redes estruturadas implicaria que o servidor tivesse caraterísticas de Access Point (AP), optou-se pela utilização de uma rede Wi-Fi do tipo *ad hoc*. O servidor fica apenas responsável por difundir a imagem, e os clientes não necessitam de se associar ao AP ao entrar e sair da transmissão, nem realizar qualquer tipo de *handshake* protocolar.

O DETIboot utiliza comunicação em difusão por parte do servidor de forma a que este tenha uma taxa de transmissão de dados constante e independente do número de clientes presentes na rede, tornando assim o sistema escalável. Devido à inexistência de *feedback* na comunicação por difusão, não existe retransmissão das tramas perdidos. Para suprimir as perdas são usados códigos Fountain (ver 3.1), permitindo que os clientes consigam reconstruir o ficheiro transmitido pelo servidor a partir de quaisquer *codewords* transmitidos, com o agravamento de serem necessários cerca de 5% mais blocos do que os que compõem o ficheiro original transmitido.

Como a transmissão é feita diretamente e por difusão, não existe necessidade de reencaminhamento de tráfego. Não é necessário nenhum protocolo de transporte (L4), nem nenhum protocolo de rede (L3). A comunicação é feita de forma canónica sobre a camada de ligação de dados (L2). A camada de ligação de dados é responsável por garantir a integridade das tramas e a utilização de códigos Fountain para lidar com perdas (*erasures*).

Para que os clientes possam entender a transmissão do servidor, foi desenhado um pro-

tocolo de distribuição por difusão (ver 3.1), permitindo ao servidor transmitir uma *codeword* por trama.

2.1.2 Cliente

Os clientes são máquinas que recebem e arrancam o SO transmitido pelo servidor. É necessário que estes executem o método de arranque DETIboot na sua fase de arranque.

Para que todo o processo de receção, armazenamento e arranque do SO seja feito na fase de arranque da máquina cliente, o *bootstrap system* do DETIboot contem os programas de arranque e os *drivers wireless* necessários para a utilização de dispositivos de rede sem fios Wi-Fi [6]. O novo SO é arrancado usando o núcleo Linux e respetivo *initramfs*[1] da imagem descarregada.

Os clientes têm como requisitos mínimos: (i) possuir um dispositivo de comunicação sem fios Wi-Fi. Todos os computadores portáteis hoje em dia já vêm com este dispositivos integrados; (ii) ter acesso ao *bootstrap system* do DETIboot. Este pode estar armazenado num dispositivo de armazenamento de dados, interno ou externo, desde que seja acessível na fase de arranque das máquinas em questão.

Como o DETIboot foi desenhado para a realização de aulas e exames práticos, este tem que garantir que não há estragos nas máquinas dos clientes. Portanto o SO descarregado é armazenado em memória RAM [6]. Tal acarreta algumas potenciais limitações, tal como a necessidade de garantir que os clientes possuam memória RAM suficiente para armazenar o sistema operativo (núcleo mais *initramfs*).

O arranque do SO é auxiliado pela ferramenta `kexec`[2], permitindo arrancar o sistema operativo descarregado sem a necessidade de o transferir permanentemente para um dispositivo de persistência de dados e sem a necessidade de reiniciar o hardware do sistema computacional.

O restante processo de arranque, após a execução do `kexec`, é da inteira responsabilidade do SO importado. O esquema encontra-se ilustrado na Figura 2.2.

2.1.3 Imagens de arranque

O armazenamento do SO na RAM traz outro inconveniente. Durante a mudança de núcleo (`kexec`) toda a RAM é libertada, sendo apenas mantidos os dados referentes ao novo núcleo e ao novo *initramfs* carregados [6]. Este problema é resolvido na fase de construção da imagem do SO, sendo possível optar por duas soluções: (i) utilizar distribuições cujo sistema de ficheiros seja no formato *initramfs* (ex. Slitaz); (ii) incluir o sistema de ficheiros no *initramfs* de arranque. A primeira tem a desvantagem de limitar a gama de imagens que podem ser usadas no DETIboot. Portanto geralmente opta-se pelo segundo esquema na montagem das imagens.

2.1.4 Servidor

O servidor é uma máquina responsável pela transmissão do SO para todos os clientes alcançáveis pelo mesmo e presentes na rede.

O servidor tem como requisitos mínimos: (i) possuir um dispositivo de comunicação sem fios Wi-Fi, de forma a possibilitar a comunicação com os clientes; (ii) dispor de acesso à

[1] `https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt`

[2] `https://www.kernel.org/pub/linux/kernel/people/geoff/petitboot/kexec-man-11.10.21-g90da2c37.txt`

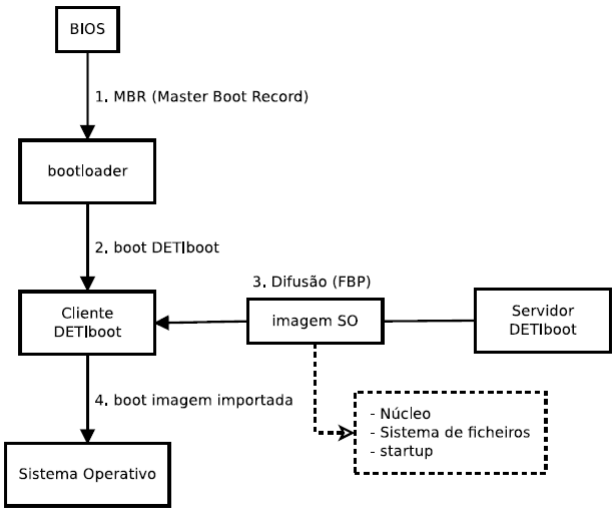

Figura 2.2: Método de arranque do DETIboot (imagem extraída de [6]). O Cliente DETIboot é arrancado através do bootloader do computador do aluno. Este espera pela difusão do Servidor DETIboot. O servidor DETIboot difunde a imagem que contém o núcleo, *initramfs* da imagem Linux e um *script* de arranque. O Cliente DETIboot ao descarregar a imagem reinicia, executa o novo SO.

---

imagem do SO a transmitir. Outros dois requisitos podem influenciar drasticamente o desempenho do servidor são: (i) a capacidade de memória RAM do mesmo, que pode limitar o tamanho da imagem a transmitir; (ii) possuir mecanismos para definir o *basic rate* e *multicast rate* da interface de rede Wi-Fi utilizada, de modo a aumentar a taxa de transmissão do sistema operativo. Por omissão, a taxa de transmissão de tramas por difusão nas redes Wi-Fi é limitada à base de 1 Mb/s, sendo 54 Mb/s o valor máximo mais adequado de forma a manter a compatibilidade com a norma mais utilizada (802.11g), e afetando apenas a norma obsoleta (na prática) 802.11b. Porém, nem sempre se consegue usar a velocidade máxima de 54 Mb/s; empriricamente verificou-se que tal depende dos *drivers* (ou do *chipset*) das interfaces Wi-Fi [6, 9].

A difusão da imagem do SO por parte do servidor como já referido é feita utilizando códigos Fountain através de uma interface de rede Wi-Fi, previamente configurada como uma rede *ad hoc*. Esta emissão consiste na geração prévia e difusão constante de *codewords* sobre tramas Ethernet (ver secção 3.1).

2.2 Criptografia

Criptografia é a ciência que permite escrever conteúdos de forma oculta. O objetivo é permitir que apenas um conjunto limitado de entidades possam trocar informação que é ininteligível para terceiros. A criptografia baseia-se no uso de cifras. Uma cifra é uma técnica concreta de criptografia, uma forma específica de ocultar informação [27].

$$\text{texto em claro} \xrightarrow{\text{cifra}} \text{criptograma} \xrightarrow{\text{decifra}} \text{texto em claro original}$$

A operação da maioria das cifras e decifras é definida através da especificação de um

algoritmo e de uma chave. O algoritmo define o modelo de transformação de dados e pode ser conhecido por qualquer entidade. A chave é um parâmetro do algoritmo que permite variar o seu comportamento de forma complexa, devendo ser conhecida só pelas entidades com que se pretende trocar informação. No caso da comunicação entre sistemas computacionais, a chave é um valor numérico com uma dimensão que torna a sua combinatória o suficientemente grande para não ser reproduzida em tempo útil.

Segundo o tipo de chave, as cifras subdividem-se em cifras simétricas e cifras assimétricas.

2.2.1 Cifras simétricas

As cifras simétricas, ou de chave partilhada, usam uma chave secreta comum. Esta chave partilhada, é usada tanto para o algoritmo de cifra como para o algoritmo de decifra, $K_{\text{cifra}} = K_{\text{decifra}}$. Só os detentores da chave secreta podem decifrar a informação cifrada. Considerando um algoritmo de cifra E e o algoritmo inverso de decifra D, um texto em claro P e o criptograma C obtido usando a chave K_{cifra}, as cifras e decifras simétricas são feitas do seguinte modo:

$$\begin{aligned} \text{cifra} &\rightarrow C = E_{K_{\text{cifra}}}(P), \\ \text{decifra} &\rightarrow P = D_{K_{\text{cifra}}}(C). \end{aligned}$$

As cifras simétricas destinam-se a garantir a confidencialidade de dados conhecidos apenas por uma entidade, detentora da chave secreta ou para garantir a confidencialidade dos dados trocados entre um grupo de duas ou mais entidades.

2.2.2 Cifras assimétricas

As cifras assimétricas, ou de par chave pública e privada, usam um par de chaves distintas mas relacionadas. É usada uma chave pública para cifrar e uma chave privada para decifrar. Não é possível dada uma chave pública, calcular a chave privada.

É comum que o algoritmo de cifra e decifra seja o mesmo num esquema assimétrico, $E = D$. O que diferencia a operação como sendo de cifra e decifra é a chave usada para o efeito. Considerando uma chave pública K_{publica} e e a chave privada respetiva K_{privada}, as cifras e decifras assimétricas são feitas do seguinte modo:

$$\begin{aligned} \text{cifra} &\rightarrow C = E_{K_{\text{publica}}}(P), \\ \text{decifra} &\rightarrow P = E_{K_{\text{privada}}}(C). \end{aligned}$$

Os pares de chaves assimétricas são personalizados, ou seja, são associados ao seu detentor (pessoa, serviço ou servidor). Apenas o detentor da chave privada poderá decifrar o texto. A chave pública pode e deve ser publicamente divulgada para poder ser usada por qualquer entidade.

2.2.3 Funções de síntese

As funções de síntese (*digest functions*), não são funções criptográficas, uma vez que não servem para cifrar ou decifrar dados. Estas são úteis para complementar outros mecanismo

de segurança. Enquanto as cifras providenciam confidencialidade das mensagens, as funções de síntese são úteis (no âmbito deste texto) para a verificação de integridade das mensagens.

Consideramos o envio de um texto M. O remetente do texto vai envia-lo, acompanhado pela sua síntese $H = h(M)$. O destinatário ao receber o texto, calcula a síntese do texto $H' = h(M)$. Se $H \equiv H'$, é porque a mensagem está intacta. Caso contrário a mensagem foi danificada ou modificada durante a transmissão. Uma mudança nos dados pode ser detetada com a alteração do texto M, ou com a alteração da síntese H.

Uma síntese não é uma simples função de dispersão, como as usadas nas tabelas de dispersão, e também não é uma função de correção como as que são usadas para deteção e correção de bits na transmissão de dados em meios ruidosos. Isto porque: (i) o valor deve ser único, o que não acontece com as funções de dispersão e (ii) não deve ser possível modificar a mensagem de forma adaptativa, produzindo uma dada síntese, o que não acontece nas funções de correção [27].

Formalmente, uma função de síntese h tem de possuir as três seguintes propriedades [19]:

1. **Resistência à descoberta de um texto original.** Dado uma síntese H, é muito difícil descobrir um texto qualquer M, tal que $H \equiv h(M)$.

2. **Resistência à descoberta de um segundo texto original.** Dado um texto M, é muito difícil descobrir um segundo texto qualquer $M' \neq M$, tal que $h(M) \equiv h(M')$.

3. **Resistência à colisão.** É muito difícil descobrir dois textos quaisquer M e M', $M' \neq M$, tais que $h(M) \equiv h(M')$.

2.2.4 Autenticadores de mensagens

Um autenticador de mensagem Message Authentication Code (MAC) é um valor produzido a partir de uma mensagem e de uma chave simétrica. Um MAC prova que, a menos que a chave partilhada tenho sido comprometida, que um dos interlocutores produziu a mensagem. No fundo, prova a autenticidade da mensagem, uma propriedade mais forte do que integridade, fornecida pelas funções de síntese.

Uma função de síntese não faz uso de qualquer tipo de segredo, o que significa que apesar de podermos verificar a integridade da mensagem, não podíamos validar a origem da mesma. Os MAC resolvem este problema, podemos verificar que a mensagem é proveniente de um dos interlocutores que possui o segredo partilhado. Ou seja, autenticidade implica integridade da mensagem, enquanto o inverso não é verdade.

Geralmente, o algoritmo de geração de um MAC, é a extensão de uma função de síntese, de forma a que esta tenho uma chave como parâmetro, $H = h_{K_{\text{cifra}}}(M)$. A geração de MAC não está limitada a este método.

2.2.5 Assinaturas digitais

No entanto, usando MAC, nenhum dos interlocutores consegue provar, perante terceiros, por qual deles a mensagem foi enviada.

Algumas cifras assimétricas permitem efetuar a cifra ao contrário, cifrar com a chave privada e decifrar com a chave pública. Pode não ter qualquer interesse para esconder a informação, mas é importante para garantir a autoria da informação. É esse o objetivo das

assinaturas digitais, ir mais longe na garantia da origem de uma mensagem e assegurar a autoria desta perante terceiros [27].

Na prática, a assinatura digital é o criptograma da síntese do texto em claro, cifrado com a chave privada da entidade emissora. Desta forma as restantes entidades ao decifrarem a síntese da mensagem com a chave pública da entidade emissora, têm a garantia que esta foi produzida por quem esta diz ser.

Considerando a assinatura digital S produzida com a chave K_{privada}, a geração e validação de assinaturas são feitas do seguinte modo:

$$\begin{aligned} \text{assinar} &\rightarrow S = E_{K_{\text{privada}}}(h(P)), \\ \text{validar} &\rightarrow E_{K_{\text{publica}}}(S) \equiv h(P). \end{aligned}$$

Se na validação não for possível verificar a equivalência da síntese do texto em claro com decifra a assinatura, então ou o texto foi alterado, ou a assinatura foi alterada. No entanto, a validação depende da chave pública usada, a qual pode não estar correta.

2.2.6 Desempenho das primitivas criptográficas

A segurança criptográfica pode ser vista como um compromisso entre combinatória e desempenho computacional. Em sistemas computacionalmente intensivos (como a difusão de conteúdo em tempo real), o peso computacional das primitivas criptográficas torna-se um fator crítico. Para dar a noção dos tempos de calculo dos vários tipos de primitivas criptográficas usadas ao longo do texto, retiramos a Tabela 2.1 de [18], em que os autores medem o tempo de execução e o número de operações das primitivas padrão de síntese, autenticação e assinatura digital.

Tabela 2.1: Desempenho das primitivas criptográficas padrão (tabela extraída de [18]). Todas as experiências foram feitas sobre um valor de entrada de 8 bytes, usando as bibliotecas do OpenSSL[3], num processador Pentium III de 800MHz, que corre uma estação Linux.

Algoritmo	tempo por operação	operações por segundo
Funções de síntese		
MD5	1.00μs	1.00×10^6
SHA-1	1.50μs	0.66×10^6
Funções de autenticação		
HMAC-MD5	2.50μs	0.40×10^6
Funções de assinatura digital		
RSA 512 assinar	1.60μs	641
RSA 512 verificar	0.16μs	6404
RSA 1024 assinar	8.70μs	115
RSA 1024 verificar	0.48μs	2094
RSA 2048 assinar	53.90μs	19
RSA 2048 verificar	1.70μs	605

O Rivest, Shamir e Adleman (RSA) é o algoritmo mais usado para cifra assimétrica e assinatura digital. O cálculo tanto da cifra como da decifra, incluindo assinaturas digitais, envolvem o cálculo de um expoente modular. A chave pública é constituída por um expoente público e um módulo. RSA 1024 significa que o comprimento do módulo comum às chaves privada e pública, bem como da assinatura digital, é de 1024 bits. Os dados da tabela

[3] `https://www.openssl.org/`

anterior foram calculados usando um expoente público igual a 65537 ($2^{16} + 1$), embora o uso do expoente mais reduzido 3 aumentasse o desempenho do calculo do RSA cerca de oito vezes [18].

2.3 Comunicação por difusão segura

O primeiro passo de qualquer cenário de exploração do DETIboot é a difusão da imagem do SO por parte do servidor e a receção desta por parte dos clientes. Atualmente o DETIboot não permite a validação da imagem por parte dos clientes. Portanto, tentámos compreender os problemas que a comunicação por difusão acarreta, e justificar os pressupostos que levaram à solução descrita no Capítulo 3.

A comunicação em difusão é um mecanismo para distribuição escalável de informação [18]. A comunicação ponto-a-ponto dominou a Internet desde que esta surgiu. Contudo, distribuição do mesmo conteúdo para uma audiência grande não é escalável. O envio de um mesmo conteúdo para N recetores por comunicação ponto-a-ponto envolve o envio de N mensagens, enquanto que com comunicação por difusão, é somente preciso uma.

Devido à grande vantagem que a comunicação por difusão demonstra ter, as suas aplicações têm vindo a crescer ao longo dos últimos anos: televisão digital, rádio digital, conferências remotas, jogos multi-jogador, e na distribuição escalável do mesmo SO para os computadores portáteis do alunos (DETIboot).

Segundo [18], difusão segura pode ser enquadrada segundo duas perspetivas:

1. Os recetores querem certificar-se que o material que recebem vem do emissor pretendido;

2. Os emissores querem limitar a utilização do conteúdo transmitido a um conjunto de recetores delimitado.

Claramente aqui estamos interessados no primeiro ponto, a validação do SO recebido pelos alunos.

2.3.1 Autenticação em difusão

A difusão apresenta um grande poder, um pacote pode alcançar milhões de recetores. Infelizmente esta propriedade traz um enorme risco: um atacante que envie um pacote malicioso pode alcançar milhões e com apenas essa mensagem consegue afetar todos eles [18]. Isto ilustra a importância da autenticação.

O problema da autenticação vem com a eficiência. Numa comunicação entre um par (ponto a ponto), a autenticação pode ser facilmente adquirida através de um mecanismo simétrico. O emissor e recetor partilham uma chave secreta para calcular os MAC das mensagens. Quando uma mensagem com um MAC válido chega, o recetor pode estar assegurado que a mensagem foi enviada pelo emissor pretendido.

Num ambiente de difusão, a autenticação simétrica não é segura caso os recetores não sejam mutuamente confiáveis. Se todos os recetores partilham uma chave simétrica comum, qualquer um pode personificar o emissor [18].

2.3.2 Assinaturas digitais em difusão

Portanto o que procuramos não é uma solução simétrica mas sim assimétrica, como assinaturas digitais. Estas têm a propriedade desejada: o emissor gera as assinaturas com a chave privada, e todos os recetores podem verificar a assinatura com a chave pública do emissor. Como bónus adicional, as assinaturas digitais possuem uma propriedade mais forte que a autenticação, o não-repúdio.

Infelizmente como vimos na secção 2.2.6, as primitivas criptográficas assimétricas têm um custo computacional uma ordem de grandeza acima das primitivas criptográficas simétricas. Numa solução ponto a ponto, o problema do peso computacional é geralmente resolvido usando criptografia assimétrica apenas para cifrar a negociação de uma chave simétrica, e proteger a comunicação que se segue com um mecanismo simétrico. Infelizmente esta técnica não é aplicável em difusão, porque todos os recetores recebem o mesmo conteúdo.

A nossa solução, que será apresentada no Capitulo 3, amortiza o peso computacional protegendo um conjunto grande de mensagens com uma só assinatura. A nossa solução também fornece outra propriedade desejada: descarte imediato dos pacotes inválidos assim que recebidos. Por último, a nossa solução também resolve o problema da divulgação da chave pública para o caso DETIboot.

2.3.3 Integridade de dados em difusão

Integridade de informação em difusão é adquirida através de autenticação tal como nas comunicações ponto a ponto.

2.4 Máquinas virtuais

Um dos objetivos principais desta dissertação é o desenvolvimento de uma solução que permita a deteção de uma VM durante o arranque do SO destinado a suportar a realização do exame, de modo a não permitir que o SO execute nessas condições. Em princípio, não deveria haver diferenças na execução de um SO sobre o hardware nativo ou uma VM. Contudo tal não é verdade na prática; essas diferenças são o que nos permite distinguir uma VM do hardware real. Porém, há muitas formas de suportar a execução de um SO num hardware virtualizado, pelo que para compreender as várias estratégias de deteção de uma VM estudadas no Capítulo 4 é necessário primeiro compreender como as VM funcionam. Assim, nesta secção apresentamos os principais paradigmas de virtualização e as principais tecnologias atuais de máquinas virtuais e emuladores.

Uma VM é um simulador de um sistema computacional, vindo o seu uso a aumentar significativamente ao longo dos últimos anos por várias ordens de razão.

Quando uma amostra de um novo *malware* é obtido por uma organização de segurança, como uma companhia de anti-vírus, a amostra tem que ser analisada de forma aprofundada, para assim se conseguir perceber o seu comportamento, objetivos e mecanismos defensivos. De forma a analisar o *malware* num ambiente controlado e seguro, para que não prejudique os sistemas computacionais dos analistas, é comum a análise ser feita sobre uma VM (atuando como uma *sandbox*). Não só esta permite o isolamento da execução do *malware*, como permite retirar *snapshots* à memória, ou mesmo parar a sua execução, facilitando a análise do estado da memória e Central Processor Unit (CPU) [22].

As VM trazem inúmeras vantagens a negócios na nuvem, facilitando o alojamento e gestão de serviços remotos. Um *data center* pode ter um conjunto heterogéneo de máquinas físicas, mas com a exploração de VM pode tornar o seu ambiente homogéneo. É também possível ter simultaneamente várias VM sobre a mesma máquinas física. Desta forma uma, máquina física pode fornecer vários servidores virtuais, num negócio de alojamento. O isolamento que as VM providenciam permite que, caso uma seja comprometida, tal não afete as demais. Com a evolução das tecnologias de virtualização, o desempenho destas já é muito próximo do do hardware nativo, permitindo a sua aplicação numa perspetiva empresarial.

As VM tornaram-se também populares no desenvolvimento de software, permitindo desenvolver e testar software numa arquitetura ou SO distintos do sistema usado para o desenvolvimento. Isto aplica-se cada vez mais ao trabalho académico, em que muitas unidades curriculares pretendem desenvolver trabalho em sistemas operativos Linux, mas como grande parte dos alunos tem um sistema nativo Windows, estes conseguem facilmente trabalhar simultaneamente na plataforma hospedeira e hospedada.

2.4.1 Definição formal de máquina virtual

Formalmente uma VM pode ser definida como um homomorfismo entre um conjunto de estados reais $\mathcal{R}$ e um conjunto de estados virtuais $\mathcal{V}$, ou seja, uma sequência de instruções (programa) p que mapeie o estado real s_r para o estado real s'_r, e que o mesmo programa também mapeie o estado virtual s_v para o estado virtual s'_v, e que existe uma função injetiva $f : \mathcal{R} \rightarrow \mathcal{V}$ que mapeia cada estado s_r* a um estado virtual equivalente s_v* [12].

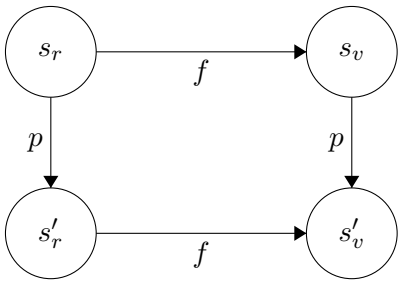

Figura 2.3: VM como um homomorfismo de estados (diagrama extraído de [12]).

2.4.2 Classes de máquinas virtuais

As VM operam segundo uma arquitetura real ou hipotética (arquitetura alvo). Uma arquitetura hipotética é uma simulação de hardware em software. Estas têm que ser simuladas sobre um sistema nativo ou sobre outra máquina virtual (arquitetura base).

As VM podem ser classificadas de acordo com o uso e correspondência a uma máquina real:

1. **Máquinas virtuais de sistemas**. Providenciam a execução de um SO completo. Normalmente simulam uma arquitetura existente. Estas são construidas para correr programas nalguma arquitetura obsoleta, cujo hardware não está disponível, ou para maximizar o proveito dos recursos computacionais, correndo vários sistemas operativos na mesma máquina física;

2. **Máquinas virtuais de processos**. São desenhadas para correr um software, permitindo portabilidade entre plataformas, sendo o exemplo mais famoso a Java Virtual Machine (JVM). Normalmente simulam uma arquitetura virtual.

Neste texto vamos explorar apenas o primeiro tipo.

2.4.3 Técnicas de virtualização

Existem dois paradigmas de simulação de hardware concretizados na prática. Um deles, usado pelas VM mais conhecidas como emuladores, traduz todas as instruções de uma arquitetura fonte para uma arquitetura alvo. Pelo contrário, as VM propriamente ditas apenas traduzem um conjunto mínimo de instruções, deixando executar a maior parte das instruções diretamente no hardware. A arquitetura fonte e alvo são as mesmas.

O primeiro paradigma apresenta a vantagem de poder simular uma arquitetura distinta à arquitetura base de onde está a ser executado. O segundo, apesar de apenas suportar a arquitetura base de onde é usado, tem a vantagem de ter um desempenho muito próximo do do hardware nativo [22].

Emuladores

Os emuladores traduzem todas as instruções da plataforma fonte para a plataforma alvo (plataforma nativa sobre a qual o emulador corre). Existem duas técnicas para implementar um emulador de um sistema computacional:

1. **Interpretação**;
2. Vários esquemas de **tradução dinâmica**.

Um interpretador lê uma instrução, traduz essa instrução e finalmente corre o resultado da tradução no hardware nativo, repetindo o processo para cada instrução. Um tradutor dinâmico, após traduzir uma instrução, armazena a tradução numa memória cache. Assim, o tradutor pode reaproveitar traduções anteriores mais adiante, aumentando o desempenho do emulador [22].

A tradução de todas as instruções traz um enorme peso computacional, que leva uma enorme perda no desempenho. A velocidade de execução não se consegue aproximar à do hardware nativo, nem de uma VM. Contudo consegue simular arquiteturas distintas da da plataforma base.

Máquinas virtuais

Uma VM tem como função duplicar de forma eficiente e isolada a máquina original. Qualquer programa que corra na VM deve exibir um comportamento idêntico como se corresse diretamente numa máquina física, com a exceção de exibir uma velocidade de execução menor. A únicas diferenças que podem ser exibidas numa VM, em relação ao hardware real, são: (i) a disponibilidade de recursos e (ii) diferenças devido a dependências de cronometragem. A execução de uma VM é controlada pelo Virtual Machine Monitor (VMM) ou Hypervisor. Um Hypervisor é um software ou hardware que cria e corre máquinas virtuais. Este deve garantir que estatisticamente a maior parte das instruções são executadas diretamente no hardware. Por último, o Hypervisor deve garantir controlo total sobre os recursos do sistema, ou seja,

um programa não deve poder aceder a zonas de memória que não lhe foram reservados. Os Hypervisors podem ser classificados em dois tipos (ver Figura 2.4):

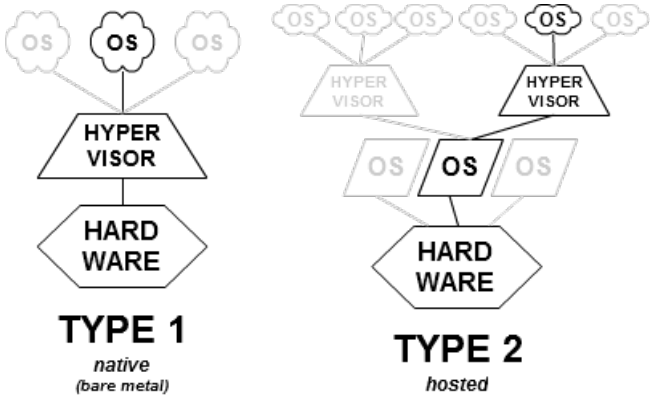

Figura 2.4: Tipos de Hypervisors[4]: nativo, corre diretamente sobre o hardware (esquerda); hospedado, corre sobre o SO hospedeiro (direita).

1. **Hypervisor nativo.** Também conhecidos como *native* ou *bare metal* Hypervisor, este tipo corre diretamente no hardware, com o objetivo de controlar o hardware e gerir os SO hospedados.

2. **Hypervisor hospedado.** Este tipo de Hypervisor corre sobre um SO convencional, tal como qualquer outro programa. Estes abstraem o SO hospedado do SO hospedeiro. Exemplos bem conhecidos são o VMware Workstation, VMware Player, Virtual Box e Virtual PC.

O primeiro tipo é usado principalmente em ambientes empresariais na nuvem, em que as máquinas físicas correm vários SO simultaneamente. O segundo tipo é normalmente comercializada e usado por utilizadores finais, em sistemas de uso geral.

2.4.4 Virtualização da arquitetura x86_64

Virtualização do CPU

Os SO concebidos para arquiteturas baseadas em processadores Intel de 32 ou 64 bits (x86 ou x86_64) foram desenhados para explorar diretamente o hardware onde é suposto executarem. Assim, é natural que assumam que possuem o controlo total do hardware. A arquitetura x86_64 oferece quatro níveis de privilégio, conhecidos como Ring 0, 1, 2 e 3. As aplicações em modo utilizador (*user space*) executam no Ring 3, enquanto o núcleo do SO, que precisa de ter acesso direto à memória e ao restante hardware, executa no Ring 0, como ilustrado na Figura 2.5.

Virtualizar a arquitetura x86_64 requereu colocar a camada de virtualização de baixo do SO (cujo núcleo está desenhado para trabalhar no nível mais privilegiado, Ring 0), de forma a que esta tivesse controlo direto sobre o hardware. Por outras palavras, os núcleos dos SO hospedados têm de executar de forma a que o Hypervisor possa sempre intervir na sua

[4] `https://upload.wikimedia.org/wikipedia/commons/e/e1/Hyperviseur.png`

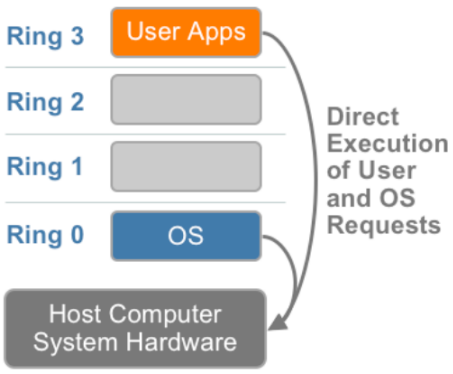

Figura 2.5: Níveis de privilégio da arquitetura x86_64 (imagem extraída de [23]). As aplicações em modo utilizador executam com nível de privilégio Ring 3, enquanto o código no núcleo do SO executa com nível de privilégio Ring 0.

operação. A colocação do nível de privilégio do núcleo de um SO hospedado no Ring 1, para ter o Hypervisor no Ring 0, é uma possibilidade, mas há outras, como veremos mais adiante.

Para complicar mais a situação, algumas instruções sensíveis não podem ser facilmente virtualizadas, porque têm diferentes semânticas quando executadas em níveis superiores ao Ring 0. Não confundir uma instrução sensível com uma privilegiada, que é uma instrução que apenas pode ser usada no Ring 0. Uma instrução sensível é uma instrução que tem de ser executada num Ring igual ou abaixo do limiar definido para a proteção de E/S (*I/O Protection Level*). Existem instruções privilegiadas que não são sensíveis e vice-versa. A dificuldade de intercetar e traduzir estas instruções sensíveis e privilegiadas em tempo de execução foi o desafio que fez com que a virtualização da arquitetura x86 parecesse impossível durante algum tempo.

Existem atualmente três técnicas para lidar com instruções sensíveis e privilegiadas [23]:

1. Virtualização completa (*full virtualization*) usando tradução binária;
2. Paravirtualização (*paravirtualization*), ou virtualização assistida pelo SO;
3. Virtualização assistida pelo hardware (*hardware assisted virtualization*).

Virtualização completa

A virtualização completa usa uma combinação de tradução binária com execução direta no hardware. Esta aproximação, ilustrada na Figura 2.6, substitui instruções não virtualizáveis com novas sequências de instruções que têm o efeito pretendido no hardware virtual. Por outro lado, o código em modo utilizador é executado diretamente no hardware, para manter o desempenho nativo. Cada Hypervisor providencia a cada VM os serviços do sistema físico. Desta forma o SO é completamente abstraído pela camada de virtualização; o SO hospedado não se apercebe que está a ser virtualizado, não precisando portanto de nenhuma modificação, o que é excelente para lidar com os SO que não possam ser modificados.

A virtualização completa é a única alternativa das três que não precisa de nenhum tipo de assistência para virtualizar instruções sensíveis ou privilegiadas. Tal como os emuladores, o Hypervisor traduz as instruções e armazenas em cache para uso futuro.

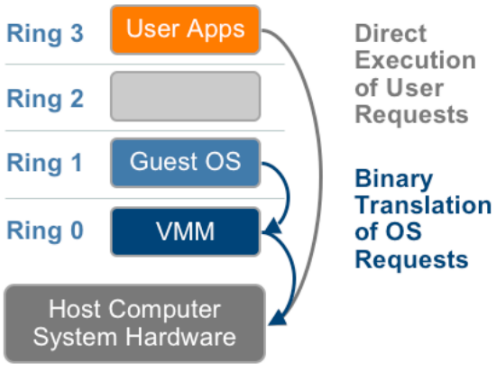

Figura 2.6: Virtualização completa da arquitetura x86_64 (imagem extraída de [23]). O SO hospedado corre em nível de privilégio Ring 1. É usando um mecanismo de tradução binária dos pedidos do SO, enquanto as chamadas das aplicações do utilizador correm diretamente no hardware.

Paravirtualização

A paravirtualização explora uma comunicação direta mínima entre o SO hospedado e o Hypervisor de forma a melhorar o desempenho e eficiência. Como mostrado na Figura 2.7, este processo envolve modificar o núcleo do SO para substituir instruções não virtualizáveis com chamadas ao diretas ao Hypervisor (*hypercalls*). A vantagem da paravirtualização face à virtualização completa, onde o SO não modificado não sabe que está numa VM, é que o peso computacional das *hypercalls* é muito menor do que a tradução binária [23]. Contudo, uma desvantagem é não ter acesso a SO fechados, de forma a poder modificar estes.

Virtualização assistida por hardware

A primeira geração de virtualização assistida por hardware inclui a Virtualization Technology (VT-x) da Intel e o AMD Virtualization (AMD-V) da AMD. Ambas permitem a execução de instruções privilegiadas pelo Hypervisor com um novo modo de privilégio raiz (*root mode*), também conhecido por vezes como Ring -1.

Como ilustrado na Figura 2.8, as chamadas privilegiadas e sensíveis são automaticamente intercetadas pelo Hypervisor, removendo a necessidade da tradução binária ou das *hypercalls*. O estado do SO hospedado é armazenado numa área especial denominada Virtual-Machine Control data Structure (VMCS) (VT-x, ver 2.4.5) ou Virtual Machine Control Blocks (VMBC) (AMD-V). Os processadores com suporte a VT-x e AMD-V são cada vez mais comuns.

Virtualização da memória

Um SO mantém tabelas com mapeamentos dos números das páginas da memória virtual para páginas físicas. Todos os CPU x86_64 incluem uma Memory Management Unit (MMU) e uma Translation Lookaside Buffer (TLB) para otimizar o desempenho da memória virtual.

Para correr múltiplas VM num só sistema computacional é necessário outro nível de virtualização de memória, nomeadamente é preciso virtualizar a MMU para suportar o SO hospedeiro. A virtualização de memória realizada pelas VM é muito semelhante à memória

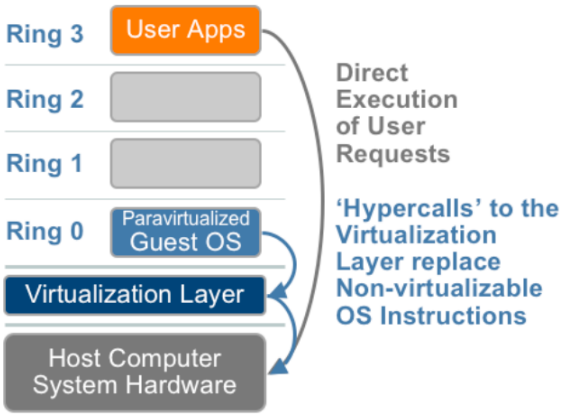

Figura 2.7: Paravirtualização da arquitetura x86_64 (imagem extraída de [23]). O SO hospedeiro em vez de chamar diretamente o hardware, chama uma camada intermédia de virtualização através de *hypercalls*, que por sua vez exploram diretamente o hardware.

---

virtual dos SO: o Hypervisor é responsável por mapear a memória física do SO hospedado para a memória física do sistema. Para acelerar os mapeamentos, usa *shadow page tables*.

Como ilustrado na Figura 2.9, o Hypervisor usa hardware TLB para mapear a memória virtual diretamente para a memória da máquina para evitar dois níveis de tradução em cada acesso. Quando o SO hospedeiro altera o mapeamento da memória virtual para física, o Hypervisor atualiza as *shadow page tables* para permitir a procura direta.

A virtualização da MMU cria peso computacional em todas as aproximações de virtualização do CPU. A segunda geração de virtualização assistida por hardware pretende resolver este problema [23].

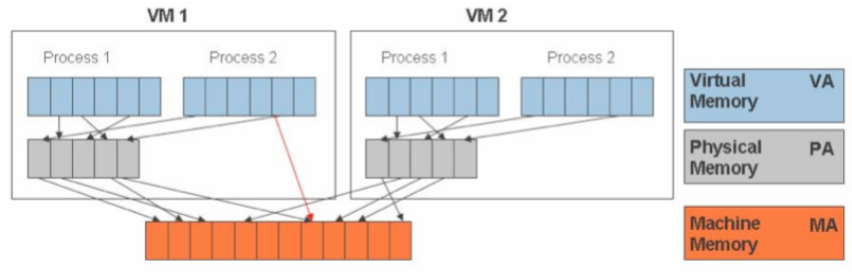

Figura 2.9: Virtualização da memória (imagem extraída de [23]). O mapeamento da memória virtual para a memória física seria indireta como ilustrado pelas setas a preto. A memória virtual é traduzida para a memória física da máquina virtual e que por sua vez traduz para a memória física do hardware. Para otimizar este processo o Hypervisor traduz diretamente a memória virtual para a memória física do hardware como ilustrado na linha a vermelho.

Virtualização de dispositivos de E/S

O Hypervisor apresenta a cada VM com um conjunto padrão de dispositivos virtuais. Estes dispositivos virtuais simulam hardware bem conhecido e traduzem os pedidos das VM para o hardware nativo.

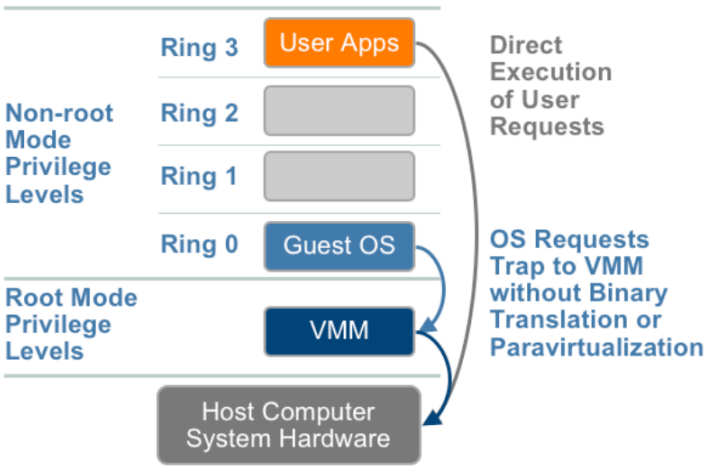

Figura 2.8: Virtualização assistida por hardware na arquitetura x86_64 (imagem extraída de [23]). O Hypervisor corre num nível de privilégio raiz (*root mode*, ou Ring -1), conseguindo assim intercetar as chamadas do núcleo do SO (que corre em nível Ring 0) ao hardware nativo.

2.4.5 Virtualization Technology

Uma instrução sensível é uma instrução que ao ser executada por uma VM pode alterar o estado do Hypervisor. Portanto o Hypervisor não pode permitir a execução destas instruções diretamente no hardware. Para que um processador suporte virtualização, um Hypervisor deve cumprir os seguintes requisitos [22, 7]:

1. O método para executar instruções não privilegiadas deve ser equivalente em modo privilegiado ou de utilizador.
2. Deve existir um método de proteção como um sistema de tradução de endereços que protege o sistema físico e outras máquinas virtuais da máquina virtual ativa.
3. Deve existir um método de sinalizar o Hypervisor quando uma VM tenta executar uma instrução sensível. Deve ser também possível ao Hypervisor simular o efeito da instrução sensível.

Na arquitetura x86_64, se uma instrução privilegiada for executada a um nível superior a Ring 0 é acionada uma exceção de Proteção Geral (General Protection (GP)), ou #GP. Se uma VM tentar executar uma instrução privilegiada é acionada uma #GP que é apanhada e tratada da forma adequada pelo Hypervisor. Se todas as instruções sensíveis forem privilegiadas, então diz-se que o processador é virtualizável, i.e., o Hypervisor pode apanhar as exceções geradas pelas instruções sensíveis. Inicialmente este não era o caso da arquitetura x86_64, que inclui instruções sensíveis não privilegiadas (ver 2.4.4).

De forma a resolver o problema anterior, a Intel introduziu na sua arquitetura uma extensão para a virtualização assistida por hardware, o VT-x. A ideia desta extensão é providenciar um modo de operação separado para os Hypervisors, o *root mode* (ver 2.4.4), e o *non-root mode* para as VM. O código executado em *root mode* possui privilégios mais altos que o código executado em *non-root mode*. Quando uma VM, mesmo quando no Ring 0, tenta executar código que afetaria as restantes VM ou Hypervisors, o processador reconhece

a instrução e notifica o Hypervisor. Este, consequentemente, pode reagir à instrução e simular o seu comportamento na VM sem interferir com outros hypervisors ou VM. A execução de uma instrução sensível causa um sinal #VMEXIT, que leva imediatamente a uma interrupção que força a mudança de contexto da VM para o Hypervisor. Este vai por sua vez simular a execução da instrução em causa.

Operações VMX

A extensão VT-x da Intel introduziu um conjunto de operações para suportar este modo de virtualização, as operações VMX. Estas encontram-se divididas em dois tipos: as operações em *root mode* e as operações em *non-root mode*. Como seria de esperar, as primeiras são executadas pelo Hypervisor, enquanto as restantes são executadas pela própria VM. Estas foram constituídas com o propósito de manipular a estrutura de dados de controlo de máquinas virtuais VMCS. A VMCS controla as transições para dentro e fora do modo de operação *non-root* (*VM entries* e *VM exits*). Esta estrutura é manipulada com as instruções VMCLEAR, VMPTRLD, VMREAD e VMWRITE. O Hypervisor pode usar diferentes VMCS para cada VM. Para uma VM com múltiplos processadores lógicos, o Hypervisor pode usar diferentes VMCS por cada processador virtual [26].

2.5 Tópicos de segurança em sistemas operativos Linux

No processo de montagem de uma imagem reforçada para o exame é necessário realizar configurações em vários níveis do SO Linux, de forma a limitar as ações do aluno durante a prova de exame. Nesta secção introduzimos as principais tecnologias usadas para a montagem da imagem reforçada: (i) a infraestrutura Pluggable Authentication Modules (PAM) que lida com as operações de autenticação do Linux; (ii) o módulo nativo *firewall* do Linux, as *iptables*; (iii) os mecanismos de elevação de privilégios Set-UID e Get-UID, que podem ser usados para contornar as políticas estabelecidas na imagem distribuída.

2.5.1 *Pluggable Authentication Modules*

O PAM é um *middleware* integrador de metodologias de autenticação que permite a coexistência de vários mecanismos de autenticação e a alteração fácil dos mecanismos a usar por cada aplicação. O PAM faculta a autenticação de uma entidade perante um autenticador, permitindo uma separação clara entre a necessidade de uma aplicação realizar a autenticação e o modo como essa é efetivamente realizada. O autenticador usa uma interface padrão para realizar a autenticação, independentemente de como a mesma é realizada; tal compete ao *middleware* [27].

As aplicações que precisam de realizar operações de autenticação chamam a biblioteca PAM, que abstrai toda a metodologia efetivamente usada através da orquestração de um conjunto de módulos. Os módulos PAM são bibliotecas dinâmicas que possuem uma interface padrão e que são carregados dinamicamente de acordo com as instruções fornecidas por ficheiros de configuração de políticas de aplicação dos módulos. As políticas podem ser para grupos de aplicações ou aplicações específicas. A infraestrutura PAM encontra-se ilustrada na Figura 2.10 [27].

[5] `https://docs.oracle.com/cd/E19082-01/819-2145/ch3pam-01/index.html`

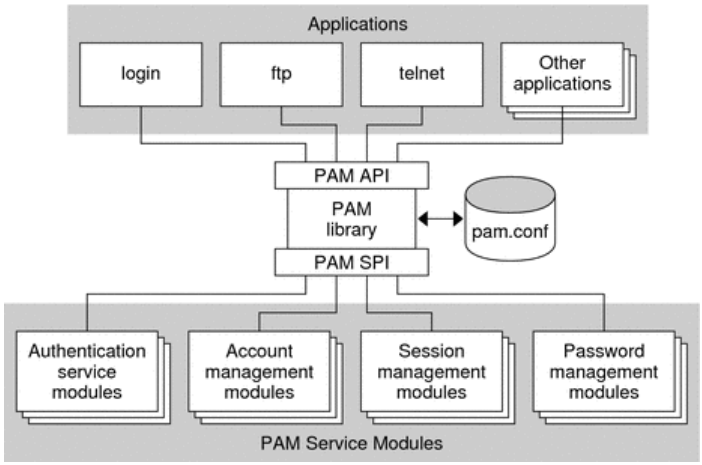

Figura 2.10: Infraestrutura PAM do Linux[5]. As aplicações usam a PAM Application Programming Interface (API) da biblioteca PAM para autenticar os utilizadores. As políticas que definem os módulos a usar encontram-se configurados no ficheiro `pam.conf`.

Na infraestrutura PAM há quatro vertentes que importa no âmbito das operações relacionadas com autenticação:

1. **Serviços de autenticação**. Estes módulos lidam com as operações de autenticação.
2. **Gestão de contas**. Lida com processos de autorização após a autenticação.
3. **Gestão de sessões**. Lida com processos de gestão de uma sessão após o processo de autenticação e autorização.
4. **Gestão de senhas**. Lida com as operações de alteração de senhas dos utilizadores do sistema.

2.5.2 IPTables

O *iptables* é um módulo Linux que recebe todos os pacotes IP que chegam à máquina ou que são enviados por esta. O *iptables* permite realizar uma *firewall* do tipo filtro de pacotes, mas também permite comutar fluxos de informação enviando os pacotes para qualquer aplicação local. Também permite realizar filtragem a nível aplicacional [27].

O *iptables* usa o conceito de cadeias para analisar os datagramas. Uma cadeia é uma sequência de regras, cada uma com zero ou mais condições e uma decisão (ACCEPT, DROP, etc). O *iptables* possui cinco cadeias padrão como ilustrado na Figura 2.11. A cadeia INPUT aplica-se aos pacotes recebidos destinados à própria máquina. A cadeia OUTPUT aplica-se a pacote destinados para o exterior provenientes da própria máquina. A cadeia FORWARD aplica-se a pacotes recebidos que não são destinados à máquina (e.g. podem ser reencaminhados para uma interface de rede alternativa). As cadeias PREROUTING e POSTROUTING aplicam-se a todos os pacotes recebidos e a serem enviados pela máquina, respetivamente.

O *iptables* também possui as chamadas políticas, que são o comportamento por omissão das cadeias de INPUT, OUTPUT e FORWARD, que deverá ser seguido quando não existe nenhuma regra na cadeia que possa ser aplicado ao datagrama.

Para que as políticas e as regras de caga cadeia sejam restauradas no arranque da imagem é necessário colocar o comando `iptables-restore` no ficheiro `init.d` que é executado após o arranque do sistema.

A vantagem das *iptables* é a flexibilidade e a generalidade. Por outro lado tem a desvantagem que é preciso ter um conhecimento a mais baixo nível do mecanismo e alguns conhecimentos das camadas protocolares Transport Control Protocol (TCP)/IP.

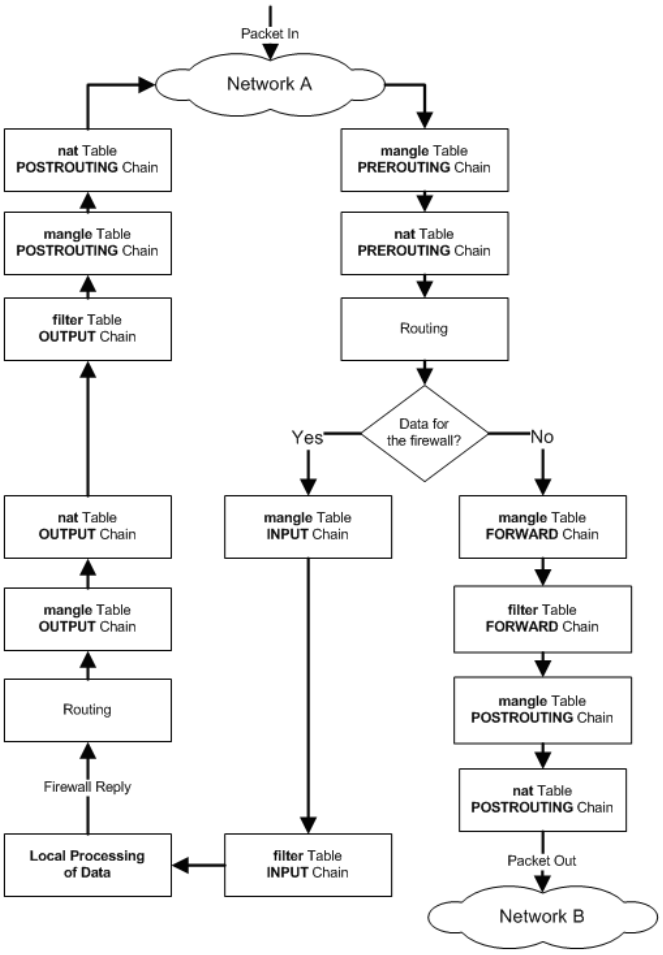

Figura 2.11: Fluxo de encaminhamento através das cadeias e tabelas das *iptables*[6]. Todos os pacotes que entram na máquina são processados pela cadeia PREROUTING. Se um pacote é destinado à própria máquina, é processado pela cadeia INPUT e enviado para o processo destinatário. Caso contrário é enviado para a cadeia FORWARD. Os pacotes provenientes das cadeias FORWARD e OUTPUT (pacotes enviados pelos processo da máquina), são enviados para a rede correspondente e processado antes disso pela cadeia POSTROUTING.

[6] `http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch14_:_Linux_Firewalls_Using_iptables`

2.5.3 Set-UID e Set-GID

No Linux, os privilégios de acesso de um processo são definidos pelo seu UID e GIDs, estes herdados do processo pai. No arranque de sessão a consola de comandos (*shell*) ou o gestor de ecrã (*window Manager*) obtêm o UID e os GID do utilizador extraídos dos ficheiros `/etc/passwd` e `/etc/group`.

Existem, contudo, situações que criam cenários contraditórios. Por exemplo, se um utilizador desejar alterar a sua senha terá que editar o ficheiro `/etc/shadow/passwd`, e utilizadores ordinários (não administradores) não devem ter um acesso de escrita genérico a esse ficheiro, pois podem alterar as senhas de outros utilizadores. Esta situação não pode ser apenas resolvida pelos direitos `rwx` de acesso de um processo a um ficheiro.

Portanto, é necessário um mecanismo especial para que certas aplicações possam realizar operações que tenham implicações de segurança, como alterar a senha de um utilizador, mas que estejam acessíveis a qualquer utilizador em termos da sua execução. Isto é conseguido com as *flags* Set-UID e Set-GID, que quando ativas, permitem ao programa correr com a identidade (UID e GID) do dono do executável, em vez do UID e GID de quem o executa. O objetivo destas *flags* é permitirem a um utilizador realizar operações que necessitam de privilégios especiais (não necessariamente privilégios mais altos), como mudar a senha, montar ou desmontar sistemas de ficheiros, enviar pings para a rede, etc.

Cada processo tem um UID e GID real e efetivo. O real é o que herdou do processo pai. O efetivo é que aquele que efetivamente conta para o controlo de acesso. As flags Set-UID e Set-GID alteram temporariamente o UID e GID efetivo.

Capítulo 3

Authenticated File Broadcast Protocol

O sistema DETIboot usa um protocolo próprio, designado por File Broadcast Protocol (FBP). A primeira versão do FBP não continha nenhum mecanismo de segurança, o que não é aconselhável para garantir a distribuição correta da imagem Linux pretendida para todos os recetores. Neste capítulo propomos um protocolo de autenticação para o FBP que garante aos clientes FBP a descarga de um ficheiro do servidor FBP pretendido. Este protocolo foi publicado em [21].

O nosso protocolo de autenticação mantém o comportamento base do FBP. A autenticação é realizada com o uso de mensagens extras (autenticadores) intercalados de forma não previsível dentro do fluxo de transmissão das tramas originais FBP. Estes autenticadores permitem que um conjunto de tramas futuras possam ser autenticadas pelos recetores. Os autenticadores são enviados antes das tramas alvo de forma a prevenir que os recetores acumulem tramas que podem nunca a vir a ser autenticadas.

Os autenticadores usam tecnologia padrão: sínteses Secure Hashing Algorithm (SHA)-1 de tramas, assinadas com uma chave privada RSA. Para o ambiente de exploração visionado usam uma configuração RSA de alto desempenho (módulo de 1024 bits e expoente público fixo e pequeno) sem comprometer a segurança. A configuração de um sessão FBP é condicionada por parâmetros obtidos da chave pública do servidor FBP, o que garante que os recetores não estão a ser enganados por redes instaladas por atacantes.

Para avaliar o desempenho do protocolo proposto, foram feitas medições em vários cenários operacionais, considerando diferentes condições de receção, diferentes distribuições de envio de autenticadores e a presença de um atacante.

3.1 *File Broadcast Protocol*

Nesta secção o FBP é descrito de forma a clarificar o cenário que precisa de ser protegido.

O FBP [6, 8, 9] usa códigos Fountain [5, 14] para difundir um ficheiro. Os códigos Fountain criam uma sequência de *codewords* que podem ser gerados de um conjunto de símbolos fonte de forma a que esses símbolos possam ser recuperados de qualquer subconjunto de códigos de tamanho igual ou ligeiramente maior que o número de símbolos fonte.

O FBP começa por segmentar o ficheiro a ser transmitido num conjunto de segmentos (símbolos fonte), todos eles do mesmo tamanho. Combinações pseudoaleatórias de somas

em módulo 2 (XOR) desses segmentos são então calculadas, produzindo as *codewords*. O FBP difunde de forma repetitiva e infinita *codewords* de um servidor para múltiplos clientes. O transmissor codifica os segmentos do ficheiros em *codewords* e os clientes descodificam as *codewords*, obtendo os segmentos originais do ficheiro.

Os cliente FBP podem entrar em qualquer momento na sessão de difusão, estes não precisam de estar presentes no início da transmissão FBP. Além disso, estes são tolerantes a perda de pacotes visto que, em teoria, não interessa o conjunto exato de *codewords* que um cliente recebe (todas as *codewords* são igualmente boas para recuperar os símbolos originais). Após um certo limiar de *codewords* recebidas, deve ser altamente provável que o cliente consiga completar a descodificação e recuperar todos os símbolos originais.

FBP é um protocolo de rede (camada 3 do modelo OSI), identificado através do código Ethernet 0x1986 em tramas 802.11, embora não limitado a esta. A difusão pode ser realizada também em redes com fios.

Cada trama 802.11 transmitida pelo FBP contém não são a própria *codeword* mas também os índices de todos os símbolos usados para gerar a *codeword*. Os índices não são transmitidos diretamente, estes são derivados pelos recetores através de parâmetros incluídos na trama: o número total de símbolos (K), o grau da *codeword* (número de símbolos usados na sua geração), e uma semente aleatória (Figura 3.3). O grau e a semente em conjunto com um gerador pseudoaleatório universal, são usados para gerar o conjunto de índices (sem repetições) dos símbolos contidos na *codeword*.

Cada trama possui também incluí um índice da *codeword* (um número de sequência) para propósitos de avaliação de performance (nomeadamente, para avaliar perdas de *codeword* nos recetores). Este campo foi usado para definir uma janela de autenticação.

3.1.1 Vulnerabilidades de segurança / modelo de ataque

O FBP usa um meio de comunicação sem fios, através de redes *ad hoc* 802.11, para difundir as *codewords* de uma imagem de arranque de uma fonte para muitos computadores portáteis destino. Assim, um atacante pode tentar personificar uma fonte legitima de forma a providenciar a sua própria imagem. Alternativamente, o atacante pode fornecer apenas algumas *codewords* que atuariam como Cavalos de Troia, i.e., poderiam alterar o comportamento final da imagem ao mesmo que manteriam a maior parte das suas funcionalidades inalteradas. Não é fácil de o realizar, mas não é certamente impossível.

Além desses ataques, onde o atacante pode tentar obter o controlo da imagem descarregada pelos clientes, um atacante pode realizar ataques de Negação de Serviço Denial of Service (DoS). Neste caso, ele pode (i) repetir *codewords* anteriores ou (ii) bloquear emissões ou receções legitimas de *codewords*.

No primeiro caso, que é um típico ataque de repetição, o ataque poderia aumentar a ocupação de memória nos recetores para armazenar *codewords* inúteis, mas não arruinaria o processo de descodificação (as *codewords* podem ser repetidas). Em qualquer caso, é aconselhado a descartar *codewords* repetidas no caso de ser possível detetar tal situação.

No segundo caso, que envolve a interferência com uma emissão e receção legítimas, não existe uma solução definitiva, porque *jamming* ou ocupação abusiva do médio 802.11 é sempre possível. Apesar disso, podemos tornar a tarefa do atacante mais difícil obrigando este a interferir permanentemente com o transmissor legítimo. Visto que as *codewords* do FBP não necessitam de chegar por ordem e podem ser perdidas, enquanto o recetor conseguir receber *codewords*, mesmo a uma taxa mais baixa, ele irá evoluir continuamente no sentido

de completar a descodificação de todos os símbolos originais. A única forma de evitar tal é prevenir toda a receção de *codewords*.

Outro tipo de ataque envolve a resolução de nomes usados numa rede *ad hoc*. Estas redes, formalmente referidas como Independent Basic Service Set (IBSS), podem ser identificados por nomes, que estão associados a valores de 48 bits, os Basic Service Set Identifier (BSSID). Normalmente a associação entre o nome da rede e o BSSID é feito por qualquer nó que tenta resolver um nome entre os vizinhos e não obtém resposta. Portanto, é perfeitamente possível ter um transmissor FBP legítimo e um atacante com o mesmo nome de rede, cada um associado a um BSSID distinto. Neste caso, recetores FBP não devem resolver nomes para valores BSSID, porque estes podem obter o BSSID do atacante, entrando assim na sua rede posteriormente. Em tal caso, o atacante pode personificar o transmissor FBP legítimo, fornecendo a sua imagem, ou manter-se em silêncio, desta forma realizando um ataque DoS conhecido por buraco negro. Uma vez que a resolução de nomes é uma característica básica do 802.11, que não pode ser alterada ou protegida, a solução óbvia passa por forçar o BSSID do transmissor nos recetores FBP para um valor conhecido como sendo um usado por um transmissor FBP legítimo.

Finalmente, assumindo que o FBP necessita de algum tipo de mecanismo que permita os recetores verificar a validade das *codewords* que recebem, este mecanismo deve estar desenhado de tal forma que não permita o atacante interferir com um esforço mínimo. Caso contrário, seria fácil para um atacante forçar os recetores a descartar todas as *codewords* legitimas.

A autenticação automática para cada *codeword* poderia ser uma solução, mas os custos em termos transmissão de informação e processamento de CPU poderia ser excessivo. Por outro lado, a transmissão das tramas de autenticação, em que cada uma pode ser usada para autenticar muitas *codewords*, não podem ser previstas (i.e., ninguém deve poder prever o seu instante de transmissão). Caso contrário, o atacante poderia simplesmente estragar a transmissão das tramas de controlo para interferir e arruinar completamente o processo de receção.

3.2 Requisitos de autenticação e alternativas

Como referido em [18], as soluções para a comunicação ponto-a-ponto de pacotes confiável não escalam bem em ambientes de difusão. Em comunicações ponto-a-ponto é comum que os recetores peçam a retransmissão de dados em caso de falha na transmissão ou receção. O FBP resolve este problema para comunicação por difusão usando os códigos Fountain, que não precisam de confirmação de receção. Além disso, a autenticação em comunicações ponto-a-ponto pode ser conseguida com uma solução puramente simétrica, usando um um MAC, por exemplo. Ambos os intervenientes partilham um segredo comum, e quando uma mensagem com um MAC correto é recebida, o recetor é assegurado que foi o transmissor correto a enviar. Contudo, como referido na secção 2.3, num ambiente de difusão com clientes não confiáveis o uso de um MAC não é seguro. Se todos os intervenientes partilham o mesmo segredo, qualquer um pode personificar a fonte genuína e tomar controlo da difusão. A aproximação óbvia passa pelo uso de um mecanismo assimétrico, tal como a assinatura digital. Este mecanismo contém a propriedade fundamental necessária ao FBP: cada fonte gera assinaturas com a sua chave privada e os recetores podem verificar a assinatura com a chave pública da fonte pretendida.

Um recetor FBP deve processar as *codewords* imediatamente após a sua receção, para maximizar o número de ciclos CPU para descodificação entre a receção de tramas consecutivas.

Consequentemente, é aconselhável (i) autenticar cada *codeword* independentemente das outras ou (ii) transmitir uma trama autenticadora de múltiplas tramas previamente à transmissão das *codewords* respetivas.

A solução natural para a primeira opção é incluir em cada *codeword* uma assinatura, produzida pelo transmissor FBP, que pode ser verificada por todos os recetores. Contudo, assinaturas ordinárias, como as produzidas com RSA, podem ocupar um espaço relevante na trama da *codeword*. Para uma *codeword* típica com aproximadamente 1500 octetos, uma assinatura RSA com módulo de 1024 bits iria ocupar 128 octetos na trama. Isto significa uma perda de 9% dos dados transmitidos por cada trama. Visto que o tempo de descodificação total é uma função do tempo que demora a receber o numero mínimo de *codewords* necessárias, o tempo de descodificação total aumentaria em pelo menos 9%. Apesar de não ser um decréscimo de desempenho dramático, consegui-se encontrar uma solução melhor.

A segunda opção passa por transmitir tramas de autenticação com campos de autenticação da própria trama e para a validação de um conjunto de *codewords* que se seguem nas tramas seguintes. Uma estratégia simples para implementar esta política de autenticação é a de incluir um conjunto de referências e sínteses de tramas de *codewords* futuras. Todas as tramas de autenticação são assinadas pelo transmissor FBP. Com tramas de 1500 octetos e uma assinatura RSA de 128 octetos, temos espaço para 60 sínteses SHA-1 de 20 octetos cada. Sem perdas de tramas, esta estratégia tem um peso menor que 2% relativamente ao total de dados transmitidos, porque só precisamos de transmitir um autenticador (e a chave pública correspondente para o validar) antes de uma sequência de 60 *codewords*. Contudo, com perdas de tramas o peso torna-se maior, porque se o recetor perder um autenticador terá de descartas todas as tramas com *codewors* seguintes até que receba um novo autenticador.

Quanto a ataques, a segunda opção é potencialmente mais fraca que a primeira contra ataques de DoS. Se um atacante conseguir prever os instantes de quando os autenticadores são transmitidos, ele pode interferir durante a sua transmissão e assim, com esforço mínimo, poderia impedir a validação de todas as *codewords* recebidas. Contudo, esta fraqueza pode ser mitigada adicionando alguma aleatoriedade aos instantes de transmissão dos autenticadores. Por exemplo, o transmissor pode inserir um número variável de *codewords* entre autenticadores, variando de um até ao número de sínteses presentes em cada autenticador. Com esta estratégia, um atacante nunca pode antecipar o instante de transmissão de um autenticador, o que invalida interferências cirúrgicas em instantes bem definidos.

3.3 *Authenticated File Broadcast Protocol*

Esta secção apresenta a extensão de autenticação desenvolvida para o protocolo FBP, resultando num novo protocolo denominado Authenticated File Broadcast Protocol (AFBP). Nesta extensão usamos a última solução apresentada na secção anterior: pacotes autenticadores especiais, intercalados por um número variável de *codewords*, que autenticam um número fixo de *codewords* que se seguem ao autenticador (ver Figura 3.1).

3.3.1 Assunções e opções de design

O nosso protocolo de autenticação foi concebido para um ambiente operacional onde um novo par de chaves assimétricas pode ser criado e usado durante o tempo de vida de uma sessão de difusão de um ficheiro. Por exemplo, pode ser usado para a difusão de uma distribuição live Linux no inicio de uma aula, não demorando essa difusão mais do que uns minutos. Ou

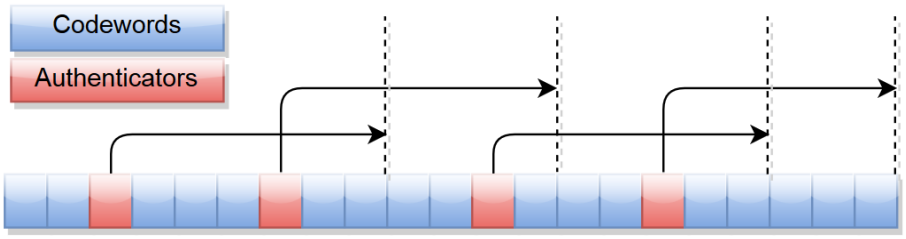

Figura 3.1: Emissão de autenticadores em intervalos aleatórios, que autenticam as próximas tramas por emitir. Cada autenticador (a vermelho) protege as próximas 5 *codewords* (a azul).

---

pode ser usado numa base diária para computadores de uma organização (e.g. distribuição de uma imagem de demonstração para todos os computadores a serem apresentados nas montras de uma loja de computadores). Este será também essencial para a realização de exames onde os alunos querem estar certos que estão a receber a imagem que os vai autorizar a entregar o exame e a disponibilizar todas as ferramentas de trabalho num ambiente protegido.

Nos vários casos, assumimos que (i) o par de chaves pode ser alterado frequentemente, numa base de sessões, e não precisa de ficar estável por um período longo; e (ii) os recetores podem obter, de uma fonte confiável, alguns elementos que lhes permite verificar que a chave pública é a que devem usar. Neste último caso, não considerámos nenhuma estratégia automática de validação, como certificados de chave pública ou cadeias de certificação. Em vez disso, optámos por usar algum tipo de mecanismo facilmente aplicável por um humano, como a validação da igualdade entre sínteses. Como veremos mais à frente, a chave pública vai permitir também aos utilizadores destingir sessões num ambiente em que existem múltiplas difusões simultâneas, funcionalidade que não era possível com o protocolo original.

Tomando em consideração a primeira assunção, não há necessidade de usar chaves assimétricas longas; escolhemos, por isso, chaves RSA de 1024 bits. Além disso, usamos o primo de Fermat (3) como expoente público, o que reduz a computação da validação das assinaturas ao mínimo sem trazer vulnerabilidades de segurança conhecidas. Escolhemos o SHA-1 como o algoritmo de cálculo da síntese de cada *codeword*. Não existem vulnerabilidades conhecidas do uso do algoritmo como função de síntese e os valores produzidos não são muito longas.

3.3.2 Distribuição e validação de chaves

Cada autenticador transporta o módulo da chave pública do seu gerador, e transporta também a assinatura produzida pela chave privada correspondente. Quando um recetor arranca, espera pela receção de um autenticador, apresenta uma síntese da chave pública ao utilizador e espera por uma decisão de aceitação/rejeição. Esta decisão deve ser feita confrontando, de alguma forma, se a síntese é a esperada. Por exemplo, no ambiente de uma aula ou exame, o docente que controla a máquina transmissora obtém a síntese da mesma chave e pode escreve-la ou projeta-la no quadro.

Para separar comunicações envolvidas em sessões FBP simultâneas, a montagem do esquemas relacionados com a chave tornam-se ligeiramente mais complexos:

1. O emissor inicia uma rede *ad hoc* com um BSSID aleatório gerado no momento. O BSSID pode já estar em uso ou não, é irrelevante para o FBP.

2. O emissor inicia um servidor FBP, que por sua vez gera um par de chaves para a sessão de transmissão. Depois, calcula (e mostra) a síntese do módulo da chave pública e mostra o BSSID da sua rede *ad hoc*.

3. O recetor inicia o cliente FBP com o BSSID a ser usado pelo servidor pretendido, de forma a entrar na rede *ad hoc*. Para facilitar a entrada do BSSID, o recetor faz uma pesquisa nos vários canais Wi-Fi, e a lista dos BSSID disponíveis são apresentadas ao utilizador. Este pode escolher usar um dos BSSID encontrados ou repetir a pesquisa.

4. O cliente FBP espera por um autenticador, que apresentará a síntese do módulo público. Se o utilizador aprovar o valor, o módulo é registado para verificação de autenticadores futuros.

5. O cliente FBP espera por uma *codeword* válida (e devidamente autenticada) para extrair os parâmetros operacionais: número de símbolos K e o tamanho de cada símbolo. Tendo estes, o processo de descodificação pode começar (usando esta *codeword* e as que se seguem válidas).

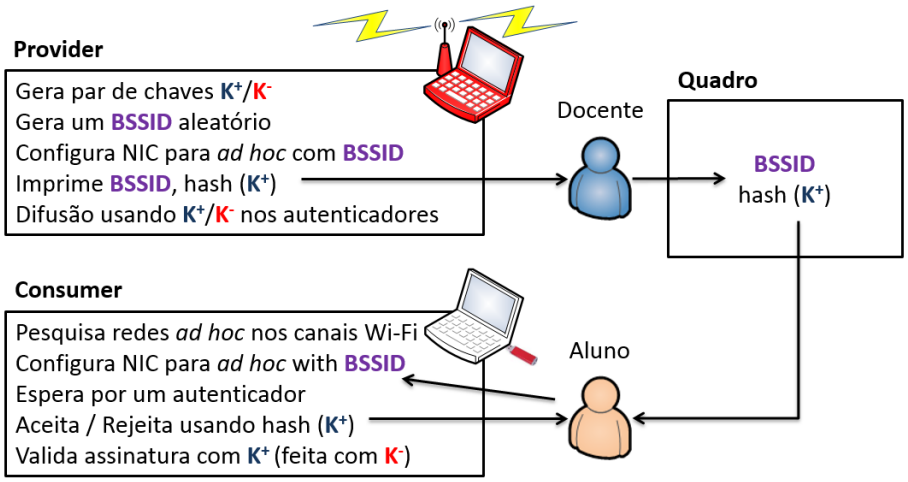

Figura 3.2: Distribuição e validação de chaves. O docente configura uma rede *ad hoc* com um BSSID aleatório. O servidor FBP gera um par de chaves e imprime a síntese do módulo público. O docente escreve os dados no quadro da sala de aula. O aluno usa os dados do quadro para entrar sucessivamente na rede *ad hoc* e na sessão FBP.

3.3.3 Estrutura física de um autenticador

As tramas Ethernet estão otimizadas nos sistemas Linux para serem enviados com uma carga de tamanho até 1500 octetos, máximo suportado nas redes com fios. Subtraindo o tamanho do módulo da chave pública RSA (128 octetos), o tamanho da assinatura correspondente (128 octetos) e o tamanho dos campos operacionais restantes, o espaço restante pode comportar 60 sínteses SHA-1 de 20 octetos. Assim, cada autenticador pode autenticar 60 *codewords* consecutivas. Considerando que os autenticadores são de tamanho igual ao tamanho das *codewords* (na prática são ligeiramente mais pequenos), pelo menos $\frac{1}{61}$ ($\sim$1.6%) de todos os octetos transmitidos são exclusivamente para autenticação. Aumentando a frequência de

índice	(4 octetos)
K	(4 octetos)
semente	(4 octetos)
grau	(4 octetos)
codeword	(1484 octetos)

índice mais recente: $x + 59$	(4 octetos)
0	(4 octetos)
identificador de sessão	(4 octetos)
sínteses SHA-1: $d_x, d_{x+1}, \cdots, d_{x+59}$	(1200 octetos)
assinatura RSA & módulo público	(256 octetos)

Figura 3.3: Trama FBP de uma *codeword* (esquerda) e autenticador a transmitir antes da *codeword* x (direita).

transmissão dis autenticadores vai aumentar a largura de banda gasta ao longo do tempo com dados relativos a autenticação. Apesar disso, o custo pode ser baixo quando as perdas de transmissão aumentam, afetando menos tramas pois são enviados mais autenticadores em intervalos mais curtos. Este problema será mais detalhado na secção 3.7.

A Figura 3.3 mostra o mapeamento físico completo dos campos de um autenticador. O segundo campo, correspondente em termos de localização ao campo K de uma *codeword*, é sempre 0. Visto que o campo K (número de símbolos) nunca tem o valor 0 nas *codewords*, este campo pode ser usado pelos clientes para distinguir *codewords* de autenticadores. O campo identificador de sessão é usado para descartar de forma eficiente autenticadores pertencentes a uma sessão de transmissão diferente (partilhando, por uma coincidência improvável, o mesmo canal 802.11 e o mesmo valor BSSID). Este campo é escolhido pelo servidor FBP no início da sessão de transmissão e adotado pelo recetor na aceitação do autenticador que transporta o módulo público que será usado para autenticar o tráfego. Desta forma os autenticadores não desejados podem ser rapidamente excluídos sem verificar a sua assinatura.

Antes de enviar um conjunto de $1 \leq N \leq 60$ *codewords* geradas previamente, o servidor FBP gera um autenticador para proteger as próximas 60 *codewords* $(C_x, \cdots, C_{x+59})$. Por cada *codeword* C_i, com $i \in [x, x + 59]$, uma síntese $d_i = h(C_i)$ é calculada e inserida na posição respetiva do autenticador. O autenticador também transporta o índice $x + 59$ da *codeword* usada para calcular a última síntese (d_{x+59}), definido como índice mais recente. Por exemplo, se o índice do autenticador for 2000, apenas as *codewords* com índice entre 1941 e 2000 são validadas; caso contrário são imediatamente descartadas. Este índice não só é usado para identificar a gama de índices transportados mas é também usado como índice do próprio autenticador. Este irá se usado para combater ataques de repetição de autenticadores anteriores (ver Secção 3.4.2). Por último transporta consigo a assinatura RSA do transmissor e o módulo da chave pública.

3.4 Avaliação da segurança

3.4.1 Ataques de negação de serviço contra os clientes

Se o recetor FBP tem um tampão de receção com muita capacidade, ataques de inundação não causam grandes estragos. O tamanho do tampão apenas precisa de ter capacidade igual ao produto da largura de banda da rede e o tempo de atraso de processamento das tramas.

3.4.2 Ataques de repetição contra os clientes

Para prevenir ataques de repetição, onde se reenviam *codewords* ou autenticadores, é possível fazer uma verificação prévia rápida que permite detetar facilmente as réplicas. Assim,

os autenticadores com o um índice inferior ao do último autenticador armazenado podem ser descartados pelos clientes sem mais nenhuma validação. O índice das *codewords* também pode validado da mesma forma: os clientes guardam o índice da última *codeword* (válida) e descartam *codewords* com índice igual ou inferior ao último recebido sem mais nenhuma validação. É de notar que nos encontramos num ambiente de comunicações sem fios direta, sem retransmissores, onde em principio as tramas nunca podem chegar fora da ordem por que foram emitidas.

3.4.3 Ataques contra o servidor

Não podem existir ataques de repetição ou DoS ao servidor visto que toda a transmissão é unidirecional (apenas do servidor para os clientes).

3.5 Filtragem ao nível do endereçamento de grupo

Nesta secção descrevemos uma otimização que foi contemplada no desenho mas não foi concretizada por razões que mais adiante se verá.

De forma a que o cliente FBP posso tomar a decisão de descartar tramas pertencentes a outras emissões mais rapidamente, o servidor FBP calcula um endereço Ethernet destino de grupo (*multicast*) a partir do módulo da chave pública, tal como no esquema de cálculo do BSSID. Os autenticadores seriam na mesma enviados em difusão para que a sessão pudesse ser detetada por qualquer máquina. O cliente, após aceitar o módulo da chave pública, calcularia o endereço de grupo e juntar-se-ia ao grupo. Desta forma, as *codewords* destinadas a outras sessões seriam imediatamente descartadas ao nível da receção da interface Wi-Fi.

As interfaces de rede Wi-Fi em modo *ad hoc* interpretam as tramas de difusão (*broadcast*) e grupo que enviam da mesma forma: verificam se o bit de difusão está ativo e transmitem as tramas para todos os destinatários na rede em caso afirmativo, não esperando por confirmações de receção. Apenas as interfaces recetoras é que diferenciam e filtram as tramas por endereço de destino. A única diferença entre um endereço de destino de difusão ou de grupo é que o primeiro (que possui todos os bits a 1) é sempre aceite por todos os recetores, enquanto os outros são filtrados se os recetores não se juntarem ao grupo.

Este otimização acabou por não ser explorada porque o comportamento acima referido não é na prática aplicado em redes *ad hoc*. Com efeito, os recetores Linux não filtram endereços de grupo nas interfaces associadas a uma rede *ad hoc*, aceitam todos os grupos como se de difusão se tratassem. Como as redes *ad hoc* são o ambiente alvo principal do DETIboot, e por outro lado um resultado similar já é conseguido com o uso direto do BSSID sem tradução de nomes, o mecanismo de grupos não foi mantido para não adicionar complexidade desnecessária.

3.6 Concretização

3.6.1 Gestão de chaves

O cliente FBP espera por um autenticador, mostra a síntese SHA-1 do módulo da chave pública e leva o utilizador a escolher se é a chave pretendida a ser usada. O utilizador tem a opção de (i) usá-la; (ii) não a usar apenas esta vez; (iii) não a usar para sempre; ou (iv) inserir a síntese manualmente. Portanto, um ataque por inundação que explore módulos constantes ou que estejam em permanente mutação podem ser ultrapassados por (i) escolher não usar

um módulo para sempre ou (ii) providenciar a síntese do módulo correto. O segundo tem a vantagem de apenas necessitar de memória para uma síntese, enquanto a primeira pode necessitar de memória indefinida para armazenar uma quantidade absurda de sínteses para ignorar.

3.6.2 Produção de autenticadores

O servidor FBP foi modificado para retirar o melhor aproveitamento do CPU dividindo as suas tarefas em três fios de execução (*threads*): (i) produção de *codewords*, (ii) geração de autenticadores e (iii) emissão de autenticadores e *codewords* já geradas. A maior parte dos sistemas computacionais atuais são multiprocessador, portanto a cada uma das tarefas Producer, Signer e Sender pode ser associada a um CPU distinto. Estas tarefas gerem dois tampões circulares que são usados para armazenar de forma ordenada as *codewords* e os autenticadores (ver Figura 3.4).

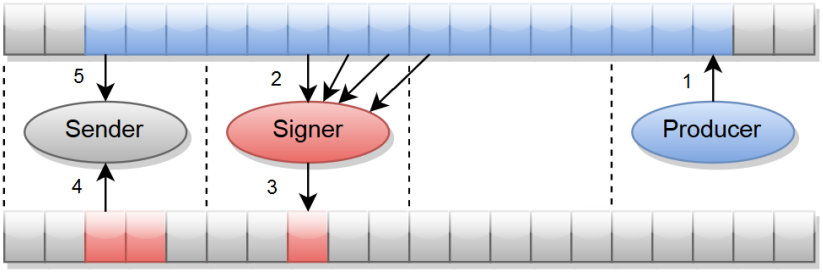

Figura 3.4: Tarefas, tampões e ações usadas para coordenar a produção e transmissão de *codewords* e autenticadores. Autenticadores (a vermelho), *codewords* (a azul) e pontos de sincronização a tracejado.

A tarefa Producer é a primeira a começar. Produz *codewords* enquanto for possível (ação 1 na Figura 3.4). Assim que são geradas *codewords* suficiente (pelo menos 60) para encher um autenticador com as suas sínteses, a tarefa Signer começa. Tendo pelo menos um bloco de assinaturas, a tarefa Sender começa.

Existem quatro pontos de sincronização, implementados com semáforos: dois entre o Producer e o Signer; um entre o Signer e o Sender e um entre o Sender e o Producer. Estes estão representados pelas linhas verticais na Figura 3.4. Teste preliminares indicaram que o uso de pontos de sincronização entre qualquer par de posições dos tampões causava demasiado peso computacional e não se retirava nenhum proveito do multiprocessamento. Portanto, estes quatro pontos de sincronização são feitos em blocos de 60 posições. O Producer apenas precisa de um bloco; enquanto dentro do bloco, produz tantas *codewords* o quanto possível. O Signer precisa de dois blocos, porque cada autenticador deve sempre proteger as próximas 60 *codewords*, e algumas podem estar no próximo bloco adjacente. O Sender só precisa de um bloco. Portanto no total são necessários quatro blocos de 60 posições.

Como podemos ver nas ações 2 e 3 na Figura 3.4, o Signer armazena sempre o autenticador no mesmo índice que a primeira *codeword* que autentica. Nas ações 4 e 5 na Figura 3.4, o Sender no índice i de um bloco b verifica se na fila de autenticadores existe um autenticador nessa posição. Se existir, o Sender envia-o, e depois envia a *codeword* na mesma posição. Caso contrário, apenas envia a *codeword* nessa posição. Desta forma é garantido que os autenticadores são enviados antes das *codewords* que autenticam.

3.7 Avaliação de desempenho

A avaliação de desempenho teve como objetivo obter respostas práticas para duas perguntas fundamentais: (i) qual é acréscimo no tempo de descodificação, dentro de condições normais (i.e., quando o sistema não está a ser atacado), devido a inclusão de autenticação no FBP e (ii) qual é a frequência preferível para enviar autenticadores. Para obter estas respostas realizámos uma série de medidas considerando todos os cenários possíveis e combinando os seguintes parâmetros:

- Um recetor próximo ou distante do transmissor. Para o primeiro caso usámos um recetor com um Received Signal Strength Indication (RSSI) médio de -25 dBm, enquanto no segundo caso usámos um recetor com um RSSI de -75 dBm. A avaliação do RSSI foi feita externamente com uma aplicação móvel Android, o Wifi Analizer.
- Um número variável C de *codewords* entre dois autenticadores, distribuídos de forma uniforme nos intervalos $[1, 30]$ e $[1, 60]$. O primeiro intervalo leva a uma maior frequência de transmissão de autenticadores.
- A presença ou ausência de outro transmissor, próximo do recetor, usando a mesma rede sem fios (mesmo canal 802.11, mesmo BSSID).

Para transmissões sem autenticação obtivemos os seguintes indicadores: (i) tempo de descodificação (**Time**); (ii) o número de *codewords* efetivamente usadas para alcançar a descodificação total do ficheiro (**Used**); (iii) o número de *codewords* recebidas pelo descodificador mas não descodificadas (**Unused**); e (iv) o número de *codewords* perdidas na transmissão, devido a problemas físicos de transmissão ou lotação nos tampões de receção (**Lost**). Este último valor é calculado a partir dos índices da primeira e última *codeword* (F e L, respetivamente) recebidos pelo descodificador e o número total de *codewords* recebidas pelo descodificador (R):

$$\text{Lost} = L - F + 1 - R$$

Visto que $R = \text{Used} + \text{Unused}$, então

$$\text{Lost} = L - F + 1 - (\text{Used} + \text{Unused})$$

A percentagem de perda de *codewords* no processo de descodificação (Loss) é dada por:

$$\text{Loss} = \frac{\text{Lost}}{R + \text{Lost}} = \frac{\text{Lost}}{\text{Used} + \text{Unused} + \text{Lost}}$$

Para transmissões com autenticação obtivemos todos os indicadores anteriores mais os seguintes: (i) o número de *codewords* da fonte pretendida que falharam na autenticação (**Invalid**); e (ii) o número de *codewords* de outras fontes que também falharam na autenticação (**Others**). Para distinguir *codewords* da fonte correta ou incorreta usamos o parâmetro K de cada *codeword*.

Com autenticação o cálculo de Lost é diferente, sendo dado por

$$\text{Lost} = L - F + 1 - (\text{Used} + \text{Unused} + \text{Invalid})$$

e a percentagem de perda de *codewords* no processo de descodificação é dada por

$$\text{Loss} = \frac{\text{Invalid} + \text{Lost}}{\text{Used} + \text{Unused} + \text{Invalid} + \text{Lost}}$$

Nas medições usámos os seguintes sistemas e informação:

Transmissor legítimo: Toshiba Portégé 830-10R, com um Intel Core i7-2620M a 2.7 GHz, 8 GiB de RAM, interface Wi-Fi externa (USB) Thomson TG123g WiFi (com a opção TxOP [13] para transmissão rápida[1]), a executar um Linux Lubuntu de 64 bits.Foi usado para transmitir um ficheiro com 104.792.660 octetos (∼100 MiB, 70615 símbolos, cada com 1484 octetos).

Recetor: Asus K55VM-SX083V, com um Intel Core i5-3210M Dual Core a 2.5 GHz, 8 GiB de RAM, interface Wi-Fi Atheros AR9485 interna, a executar um Linux Lubuntu de 64 bits em runlevel 1 (*single user administration mode*).

Atacante: Asus F3SC-AP260C, com um Intel Core 2 Duo T5450 a 1.67 GHz, 1 GiB of RAM, a executar um Linux Lubuntu de 32 bits.

É de notar que o atacante pode ser tão potente quanto desejar, pois pode ser instalado em múltiplas máquinas próximas de um recetor. No nosso caso, o atacante foi feito intencionalmente menos poderoso do que a fonte FBP legítima FBP.

Em todas as transmissões usámos difusão 802.11g à velocidade máxima permitida pelas drivers de rede. Quando combinado com o TxOP, o servidor FBP sem autenticação consegue alcançar por volta dos 45 Mbit/s de velocidade útil de transmissão, próximo do máximo do 802.11g que é 54 Mbit/s. Sem esta opção, o desempenho máximo de transmissão cai para volta dos 25 Mbit/s.

As tabelas 3.1, 3.2 e 3.3 apresentam a média e o desvio padrão dos valores observados pelos elementos referidos nos vários cenários considerados. Todos os valores foram calculados após 10 experiências nas mesmas circunstâncias.

3.8 Análise de resultados

Os resultados mostram um resultado típico da nossa política de codificação com códigos Fountain: o número de *codewords* necessárias para completar a descodificação do ficheiro (variável Used) é bastante estável em todos os casos. Contudo, o tempo necessário para completa a descodificação do ficheiro (variável Time) varia muito consoante o cenário em causa.

Quanto ao nosso primeiro objetivo, o cálculo da degradação de desempenho introduzida pela autenticação em circunstâncias normais (sem ser atacado), vemos que o decréscimo de desempenho é muito pequeno. A tabela 3.4 mostra o acréscimo, em termos de tempo de descodificação e em *codewords* perdidas, introduzido pela autenticação. Em termos de tempo de descodificação, o incremento variou de 3.5 a 4.8% quando a frequência dos autenticadores é alta ($C \in [1, 30]$), e 1.3 a 72.2% quando a frequência é mais baixa ($C \in [1, 60]$). Em termos de perdas de *codewords*, este valor aumenta devido ao descarte de *codewords* inválidas. O aumento foi entre 0.5 e 1780.1% quando os autenticadores são mais frequentes, e entre 773.9 e 2086.5% quando os autenticadores são menos frequentes.

[1] TxOP permite um transmissor enviar lotes de tramas separadas pelo intervalo de tempo mínimo, um *Short Interframe Space* (SIFS).

Tabela 3.1: Resultados ao fim da descodificação do ficheiro sem autenticação

	Time (s)		Used		Unused		Lost (%)	
Distância	Média	σ	Média	σ	Média	σ	Média	σ
Próximo	20.8	0.9	72102.0	171.3	913.6	540.2	0.09	0.09
Longe	70.6	6.4	72238.4	137.3	151.3	111.6	3.72	0.12

Tabela 3.2: Resultados ao fim da descodificação do ficheiro com autenticação e sem atacantes

		Time (s)		Used		Unused		Invalid		Lost		Loss (%)	
Distância	C	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ
Próximo	[1, 30]	21.5	0.1	72141.1	212.5	493.6	328.9	25.6	5.4	42.6	28.2	0.09	0.04
	[1, 60]	21.0	0.1	72212.7	236.4	591.3	343.0	161.2	53.5	437.7	87.4	0.82	0.12
Longe	[1, 30]	74.0	14.5	71989.3	132.0	228.0	310.6	8107.3	2069.3	168894.7	46626.2	69.95	6.27
	[1, 60]	121.5	40.3	71943.1	156.3	147.2	270.7	30170.0	6624.8	314448.3	132831.6	81.34	4.94

Tabela 3.3: Resultados ao fim da descodificação do ficheiro com autenticação e com um atacante

		Time (s)		Used		Unused		Invalid		Lost		Loss (%)		Others	
Distância	C	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ
Próximo	[1, 30]	35.5	0.4	71949.3	145.4	616.8	262.8	820.2	101.5	23006.3	681.2	24.7	0.6	20436.3	23.9
	[1, 60]	36.0	2.1	72155.2	409.9	548.2	281.4	7144.1	1329.5	20811.5	4384.4	27.6	4.0	20917.0	20.5
Longe	[1, 30]	87.8	32.7	72061.8	363.9	238.3	330.3	9925.3	3728.8	209600.6	105077.0	72.3	9.3	2610.0	68.9
	[1, 60]	135.8	37.2	72033.8	174.8	38.8	68.2	35275.0	5758.2	358082.8	126205.5	83.7	4.1	3265.3	75.8

Tabela 3.4: Acréscimo no tempo de descodificação e perdas de *codewords* devido a autenticação

Distância	C	Δ Time (%)	Δ Loss (%)
Próximo	1-30	3.5	0.5
	1-60	1.3	773.9
Longe	1-30	4.8	1780.1
	1-60	72.2	2086.5

Existe um resultado aparentemente estranho, que é o facto de, apesar do grande aumento das perdas com autenticação, o tempo de descodificação não aumenta na mesma proporção. Note-se que, tendo taxas de perdas (totais) α_n e α_f para transmissões próximas (near) e distantes (far) respetivamente, para chegar a um limiar X de *codewords* no descodificador, permitindo a este completar a descodificação, precisamos de transmitir N_n e N_f *codewords*, em cada caso, de forma a que

$$\begin{aligned} X &= N_n \times (1 - \alpha_n) \\ X &= N_f \times (1 - \alpha_f) \end{aligned}$$

o que significa que

$$N_f \times (1 - \alpha_f) = N_n \times (1 - \alpha_n) \quad \Leftrightarrow \quad \frac{N_f}{N_n} = \frac{1 - \alpha_n}{1 - \alpha_f}$$

Sabendo que $\alpha_n = 0.0009$ e $\alpha_f = 0.6995$ (valores observados no caso do envio de autenticadores com uma frequência elevada, ver Tabela 3.2), obtemos $\frac{N_f}{N_n} \approx 3.325$. Visto que existe alguma correlação linear entre o tempo de descodificação T e o número total de *codewords* transmitidas durante a descodificação (N), podemos também antecipar que $\frac{T_f}{T_n}$ deve dar um

valor similar, que o dá: $\frac{74.0}{21.0} = 3.442$. Esta demonstração também é aplicável aos resultados obtidos para transmissões menos frequentes de autenticadores.

Em relação ao nosso segundo objetivo, a de encontrar uma frequência preferível para enviar autenticadores, os testes permitem-nos concluir, exceto num caso, que é preferível usar uma frequência mais alta ($C \in [1, 30]$) do que uma mais baixa ($C \in [1, 60]$). Com alta frequência, os indicadores de Time e Loss, aquelas que são relevantes para avaliar a eficiência de transmissão útil de *codewords*, são normalmente mais baixos do que com baixa frequência. O aumento das Losses é parcialmente devido ao crescimento do indicador Inválido, que cresce quando os autenticadores são enviados com menos frequência.

Uma exceção ocorre quando o recetor está muito próximo do emissor e não há nenhum ataque. Para além de ser um cenário raro (não é possível ter tantos recetores tão próximos, especialmente quando há muitos ou quando estão espalhados ao longo de uma sala), a diferença entre o tempo médio de descodificação é negligenciável ($\sim 2\%$) para baixa frequência.

Quando um atacante está presente e compete pelo meio de comunicação, vai naturalmente conseguir reduzir o desempenho do FBP. Isto é evidente por comparação de resultados nas Tabelas 3.2 e 3.3. Contudo, os resultados mostram um comportamento curioso: quando o atacante e a vítima estão próximos um do outro, e ambos longe da fonte genuína, o indicador Others cai quando comparado com o cenário com os três próximos uns dos outros. A causa provável será a colisão de transmissões entre a fonte genuína e o atacante, que têm dificuldade em ouvir o tráfego um do outro.

Nós usámos SHA-1 tanto para calcular as sínteses das *codewords* e as assinaturas dos autenticadores. Visto que o SHA-1 foi declarado como obsoleto para assinaturas digitais [3], deveríamos usar funções de síntese mais fortes como o SHA256 ou SHA512, para manusear as assinaturas. Realizámos algumas experiências com o SHA256 e os resultados de desempenho foram semelhantes aos observados com o SHA-1, o que significa que podemos aumentar a segurança sem comprometer o desempenho.

3.9 Trabalhos relacionados

Em relação à autenticação na transmissão de códigos Fountain, em [4] os autores desenvolveram uma solução para autenticar *codewords* Fountain usadas na distribuição de uma nova imagem em redes multi-salto de sensores sem fios. A solução, porém, é totalmente diferente da nossa: eles recodificam os símbolos originais para incluir a síntese de outros símbolos, formando uma cadeia de sínteses até a um novo símbolo raiz que necessita de ser transmitido autenticado e sem códigos Fountain. As sínteses só podem ser recuperadas quando os símbolos originais (recodificados) são recuperados, e para construir a árvore completa de sínteses é necessário de recuperar os símbolos (recodificados) numa ordem particular. Durante o processo, símbolos recuperados que não podem ser verificados são descartados. Apesar do uso de códigos Fountain, existe um tempo inicial para a transmissão, onde o símbolo raiz é transmitido.

Em relação à autenticação de outras transmissões em difusão, existem várias contribuições usando várias estratégias. Não vamos detalhar todas as contribuições individuais, mas sim destacar as suas estratégias com alguma referências relevantes.

Os métodos de amortização de assinaturas são similares á nossa aproximação: eles calculam a assinatura para um conjunto de tramas para reduzir a geração de assinaturas e o peso das verificações. Esta aproximação pode ser complexa de implementar se não for possível ve-

rificar uma assinatura ao perder uma trama relacionada (como em [16]) ou se não podermos verificar uma assinatura até receber um conjunto de tramas (como em [24]). Nós resolvemos estes problemas, visto que toleramos perdas e podemos verificar imediatamente a validade de uma *codeword* assim que é recebida pelo descodificador, com uma taxa de falso negativos que é função da taxa de transferência de autenticadores.

Esquemas com chaves simétricas são usadas nalgumas aproximações de difusão segura, como no protocolo Timed Efficient Stream Loss-tolerant Authentication (TESLA) [17]. Mas o TESLA requer uma sincronização iniciada por todos os recetores (que nós não precisamos) e as tramas não podem ser imediatamente autenticadas, apenas após receber outras tramas (o que nós não queremos e não requeremos).

Em [25] o autor desenvolveu um mecanismo, Rapid Authentication, que permite o uso de dados pré-calculados na criação de assinaturas RSA. O seu objetivo foi o de acelerar a assinatura individual de mensagens de Comando & Controlo para uma transmissão eficiente em tempo-real. Isto não é um problema para nós, visto que nós não precisamos de assinar todas as *codewords*, apenas os autenticadores, e isto pode ser feito em paralelo com a produção e transmissão de *codewords*.

3.10 Conclusões

Neste capítulo apresentámos a solução para adicionar autenticação de origem às *codewords* transmitidas por um servidor FBP. O objetivo foi o de prevenir que um atacante próximo pudesse comprometer a receção das *codewords* adicionando *codewords* inválidas.

A solução apresentada usa tecnologia conhecida e aceite – sínteses SHA-1 e par de chaves e assinaturas RSA – para produzir e verificar autenticadores de *codewords*. Estes são transmitidos antes da transmissão das *codewords* que autenticam, o que permite aos recetores validar as *codewords* imediatamente após sua receção. Os autenticadores são transmitidos a uma frequência variável e imprevisível, o que evita que os atacantes de fazer interferências seletivas contra os autenticadores. Usando frequências de envio de autenticadores mais altas conseguimos resultados de desempenho muito bons, com um acréscimo no tempo total de descodificação inferior a 5%. Note-se que este acréscimo já inclui a distribuição da chave pública, que é feita por todos os autenticadores.

A distribuição do módulo público do par de chaves RSA usado para autenticar uma sessão FBP foi adaptada para cenários operacionais onde o FBP é explorado no contexto do DETIboot: para a transmissão de um ficheiro (normalmente uma distribuição Live de Linux) para uma população arbitrariamente grande de recetores próximos (como uma sala de aula). Visto que estes recetores estão o suficientemente próximos da fonte de transmissão para terem contacto visual com o seu administrador, a informação crítica relativa ao módulo da chave pública (a sua síntese) e o BSSID da rede *ad hoc* podem ser fornecidos de uma forma simples e direta: escrevendo ou projetando nalgum sitio que todos os recetores possam ver, como o quadro da sala de aula.

Capítulo 4

Virtual Machine Halt

Num exame um aluno pode executar a imagem distribuída pelo DETIboot sobre uma VM, tendo desse modo a possibilidade de correr no seu computador dois sistemas em paralelo: o sistema hospedeiro da VM e o sistema distribuído (o hospedado da VM), no qual é suposto realizar o exame. Caso isso fosse permitido o aluno poderia beneficiar de facilidades facultadas pelo hospedeiro que não são desejadas durante o exame (acesso à Internet ou a ficheiros do seu computador) mas que não estão sob a alçada do sistema hospedado (imagem difundida que, supostamente, deveria ter um controlo total sobre o computador do aluno).

Neste capítulo iremos apresentar uma solução que concebemos para terminar a execução de um SO quando o mesmo for executado sobre uma VM. O Virtual Machine Halt (VMHalt), nome atribuído a esta solução, foi concretizado num módulo do núcleo Linux, e é transportado dentro da imagem Linux descarregada. Sendo um módulo Linux, o VMHalt pode ser executado durante o arranque do SO e com privilégios de *kernel mode*. O VMHalt vai executar vários testes de deteção de uma VM subjacente; caso um deles dê um resultado positivo, o SO é parado. O nosso alvo principal é a deteção de VM e Emuladores desconhecidos que permitem virtualizar a arquitetura x86_64, ou seja, em que a deteção é feita sem a procura de dados estáticos bem conhecidos. Muitas ferramentas foram desenvolvidas para ocultar e alterar estes dados retirando a disponibilidade dos dados para os sistemas hospedados.

4.1 Enquadramento

Para implementar o VMHalt, começámos por estudar as principais tecnologias para simulação de hardware sobre a arquitetura x86_64 (ver Secção 2.4.4), e o trabalho anterior realizado na área, ou seja, métodos já estudados para a deteção de virtualização.

Contudo existe aqui uma agravante. O estudo cientifico das técnicas para a deteção de VM é orientado para a proteção de VM que controlam os sistemas na nuvem. Ou seja, um tipo especial de *malware*, os *rootkits*, pretendem tomar o controlo de um SO. Um subconjunto destes especializa-se no controlo do Hypervisor da VM hospedeira, que no fundo tem privilégios superiores aos do SO hospedado e portanto controlo total sobre este. Se é possível para nós detetar o Hypervisor, então é também possível para o *rootkit*. Isto significa que o estudo é feito maioritariamente para VM e não para Emuladores que possuem um desempenho muito inferior, e mesmo dentro do reino das VM, por exemplo, um vendedor de serviços na nuvem só precisa de proteger a sua VM. Conseguindo encontrar métodos para detetar as VM, é possível corrigir o Hypervisor de forma a oculta-lo de um *rootkit*.

O nosso caso é exatamente o oposto e muito mais exigente. Um aluno numa prova de exame pode correr qualquer VM ou Emulador, e portanto temos que ter meios de detetá-los a todos. Ou seja, pretendemos detetar Emuladores desconhecidos e VM desconhecidas que usam qualquer um dos três principais paradigmas para a virtualização da arquitetura x86_64:

1. Virtualização completa usando tradução binária;
2. Paravirtualização, ou virtualização assistida pelo SO;
3. Virtualização assistida pelo hardware.

4.2 Trabalho relacionado – deteção de virtualização

Nesta secção enunciamos o elenco das técnicas de deteção de virtualização estudadas. Algumas das técnicas foram testadas, outras (que usam softwares para plataformas antigas) são apenas referidas. Como veremos muitas destas técnicas tornaram-se ineficazes ao longo de tempo devido à evolução das tecnologias de virtualização. Por outro lado outras técnicas foram desenhadas para VM mais específicas, enquanto outras podem ser aplicadas a de uma forma mais geral. Visto que na atualidade todos os computadores portáteis comerciais têm instalado a arquitetura x86_64 é sobre esta que nos vamos debruçar.

4.2.1 RedPill

A RedPill[1] baseia-se em verificar a posição na memória da Interrupt Descriptor Table (IDT) com a instrução Store IDT (SIDT) codificada como `0x0F010D[addr]`.

A instrução SIDT pode ser executada em modo não privilegiado (Ring 3), mas retorna um valor de um registo sensível, sendo portanto uma instrução sensível. Logo, numa arquitetura não virtualizável como o x86_64, o Hypervisor não consegue intercetar a execução do SIDT no sistema hospedado, porque a execução do SIDT não gera uma #GP. Como só há um IDT Register (IDTR), mas há pelo menos dois sistemas (hospedeiro e hospedado) em execução, o Hypervisor tem que realojar a IDT do sistema hospedado noutra zona da memória, de forma a não entrar em conflito com a IDT do sistema hospedeiro. Nas máquinas físicas o endereço da IDT encontra-se num endereço inferior a `0xd0`, enquanto nas VM (como o VirtualPC e VMware) encontra-se num endereço superior.

O programa original desenvolvido por Joanna Rutkowska encontra-se no Código Fonte 4.1. Um programa alternativo que usa *inline assembly* está no Código Fonte 4.2.

```
1 int swallow_redpill() {
2     unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
3     *((unsigned*)&rpill[3]) = (unsigned)m;
4     ((void(*)())&rpill)();
5     return (m[5]>0xd0) ? 1 : 0;
6 }
```

Código Fonte 4.1: RedPill

[1] `http://repo.hackerzvoice.net/depot_ouah/Red_%20Pill.html`

```
int idtCheck(void) {
        unsigned char idt_addr[(sizeof(long)==8)?10:6];
        asm ("SIDT (%[ptr])" : "=m" (idt_addr) : [ptr] "r" (&idt_addr));
        return idt_addr[(sizeof(long)==8)?9:5] > 0xd0;
}
```

Código Fonte 4.2: Red Pill com *inline assembly*

Atualmente as VM fazem o seu trabalho melhor com o aparecimento de novas tecnologias como o VT-x da Intel, que veio a resolver o problema da arquitetura x86_64 não ser virtualizável (ver Secção 2.4.5). Agora endereço da IDT das VM é o mesmo do hardware. Mesmo que o endereço variasse de máquina física para virtual, com o aparecimento dos CPU multi-processador, existe uma IDT por cada CPU, cada numa zona de memória distinta. Isto significa que num ambiente não virtual, quando se tem múltiplos CPU este teste devolve falsos positivos. Num sistema quad-core, assumindo uma distribuição uniforme entre todos os CPU, em 75% dos casos, resultará num falso positivo. Portanto esta estratégia é obsoleta.

4.2.2 NoPill

O NoPill [20] usa a instrução Store LDT (SLDT) alternativamente à SIDT do RedPill. Esta permite obter o endereço da Local Descriptor Table (LDT) usando a instrução SLDT. A ideia é a mesma da RedPill, o LDT Register (LDTR) é um registo sensível, mas a instrução SLDT não é uma instrução privilegiada e não pode ser apanhada através de uma exceção de hardware. Portanto o Hypervisor tem que manter uma cópia da LDT do sistema hospedado noutra zona da memória. O teste pretende determinar se o endereço da LDT tem os dois octetos mais significativos a zero, como no Código Fonte 4.3.

```
int ldtCheck(void) {
        unsigned char ldt_addr[(sizeof(long)==8)?10:6];
        asm ("SLDT (%[ptr])" : "=m" (ldt_addr) : [ptr] "r" (&ldt_addr));
        return (ldt_addr[0] != 0x00 && ldt_addr[1] != 0x00) ? 1 : 0;
}
```

Código Fonte 4.3: NoPill com *inline assembly*

Infelizmente esta técnica sofre um dos mesmos problemas da RedPill. As VM modernas já conseguem apanhar a execução de instruções sensíveis. Portanto esta estratégia é obsoleta.

4.2.3 Instrução SGDT

Num outro método semelhante aos dois anteriores o teste pretende determinar se o endereço da Global Descriptor Table (GDT) está acima ou abaixo do endereço `0xd0`, mais uma vez porque o GDT Register (GDTR) é um registo sensível, mas a instrução Store GDT (SGDT) não é uma instrução privilegiada e não pode ser apanhada através de uma exceção de hardware. Portanto o Hypervisor tem que manter uma cópia da GDT do sistema hospedado noutra zona da memória, como no Código Fonte 4.4.

```
int gdtCheck(void) {
        unsigned char gdt_addr[(sizeof(long)==8)?10:6];
        asm ("SGDT (%[ptr])" : "=m" (gdt_addr) : [ptr] "r" (&gdt_addr));
        return gdt_addr[(sizeof(long)==8)?9:5] > 0xd0;
}
```

Código Fonte 4.4: Deteção de virtualização com a instrução SGDT

Logo esta técnica sofre os mesmos problemas da RedPill e NoPill. As VM modernas já conseguem apanhar a execução de instruções sensíveis. Portanto esta estratégia é obsoleta.

4.2.4 Instrução SRT

Uma última técnica semelhante às três anteriores [15] usa a instrução Store Task Register (STR) para obter um Task State Segment (TSS) e verificar se o octeto 0 do endereço da memória do TSS é igual a `0x00` e se o octeto 1 do endereço da memória do TSS é igual a `0x40`, como no Código Fonte 4.5.

```
int trCheck(void) {
        unsigned char tr_addr[(sizeof(long)==8)?10:6];
        asm ("STR (%[ptr])" : "=m" (tr_addr) : [ptr] "r" (&tr_addr));
        return (tr_addr[0] == 0x00 && tr_addr[1] == 0x40) ? 1 : 0;
}
```

Código Fonte 4.5: Deteção de virtualização com a instrução STR

Esta técnica sofre dos mesmos problemas das três técnicas anteriores, sendo, por isso, igualmente obsoleta.

4.2.5 Backdoor da VMware

Esta estratégia é específica para o VMware, usa a *backdoor* do mesmo para extrair informação como a versão do produto, estado dos registos, etc. A ideia é que a comunicação é feita acedendo a um porto E/S específico em `0x5658`, como ilustrado no Código Fonte 4.6.

Já é possível desativar a *backdoor* do VMware. Logo, este método, para além de específico para o VMware, pode ser suprimido, sendo por isso ineficaz.

4.2.6 MAC OUI

Inspecionar o endereço Medium Access Control (MAC) das interfaces de rede virtuais, que deverão seguir a norma e conter uma componente de 24 bits que identifica o fabricante Organizationally Unique Identifier (OUI), é uma técnica simples para identificar VM específicas. Contudo, tal atualização não é isenta de problemas:

1. Novas atualizações nas VM podem trazer endereços MAC diferentes.
2. Nem todas as VM têm interfaces de rede virtuais.
3. Muitas VM permitem ao utilizador clonar o MAC da máquina física, como o Virtual Box e o VMware.

```
#define VMWARE_HYPERVISOR_MAGIC 0x564D5868
#define VMWARE_HYPERVISOR_PORT  0x5658

#define VMWARE_PORT_CMD_GETVERSION      10

#define VMWARE_PORT(cmd, eax, ebx, ecx, edx)          \
    __asm__("inl (%%dx)" :                            \
        "=a"(eax), "=c"(ecx), "=d"(edx), "=b"(ebx) :  \
        ""(VMWARE_HYPERVISOR_MAGIC),                  \
        "1"(VMWARE_PORT_CMD_##cmd),                   \
        "2"(VMWARE_HYPERVISOR_PORT), "3"(UINT_MAX) :  \
        "memory");

static inline int __vmware_platform(void) {
    uint32_t eax, ebx, ecx, edx;
    VMWARE_PORT(GETVERSION, eax, ebx, ecx, edx);
    return eax != (uint32_t)-1 && ebx == VMWARE_HYPERVISOR_MAGIC;
}
```

Código Fonte 4.6: Chamada ao porto de E/S da VMware (retirado do Código Fonte `/arch/x86/kernel/cpu/vmware.c` do núcleo do Linux).

---

4. Muitas VM permitem ao utilizador configurar manualmente o MAC da interface de rede virtual.

A manutenção desta estratégia é difícil e é facilmente iludível, sendo por isso ineficaz.

4.2.7 DMI BIOS strings

Tal como o MAC OUI, as strings da Desktop Management Interface (DMI) Basic Input/Output System (BIOS) podem indicar a presença duma VM, usando para isso um comando simples como `dmidecode`. Os problemas que apresenta são semelhantes à estratégia anterior:

1. Novas atualizações nas VM podem modificar as strings.
2. Nem todas as VM contêm tabelas da DMI BIOS.
3. Algumas VM, como a Virtual Box, permitem editar as strings.

Tal como no caso anterior, a manutenção desta estratégia é difícil e é facilmente iludível, sendo por isso ineficaz.

4.2.8 Dispositivos PCI

A lista de dispositivos Peripheral Component Interconnect) (PCI) devolvidos pelo hardware virtual pode conter indicadores que podem ser usados para determinar a execução de um ambiente virtual, usando para isso um comando simples como `lspci`. Contudo esta estratégia sofre os mesmos problemas das duas estratégias anteriores:

1. Novas atualizações nas VM podem modificar os identificadores PCI.

2. Algumas VM podem permitir que os identificadores sejam alterados pelos utilizadores.

4.2.9 *Device Drivers*

Semelhante aos três anteriores, a lista de *device drivers* instalados pode indicar a presença de uma VM. Contudo, não só pode a VM não usar *device drivers* específicos, como pode requerer software opcional como o "VMware Tools", ou "VirtualBox Guest Additions". Pode ser usado o comando `lsmod` para listar os módulos Linux.

A manutenção desta estratégia é difícil. Portanto esta estratégia é pouco eficaz.

4.2.10 Núcleo Linux

O núcleo do Linux contém rotinas de paravirtualização e configurações para colaborar com o Hypervisor no caso de estar correr sobre uma VM [11]. O núcleo deteta a presença das VM Xen, VMware, Hyper-V e KVM, através de uma *flag* do núcleo do Linux que é copiada para a variável global `cpu_has_hypervisor`. Esta procedimento é feito para o Linux poder preparar no seu arranque tecnologias de aceleração de virtualização.

Os Hypervisors "honestos' que ativam a variável `cpu_has_hypervisor`, normalmente têm também presente uma assinatura de CPU própria, que pode ser obtida através da instrução `cpuid` com o código de operação `0x40000000`. As assinaturas são strings de comprimento menor ou igual a 12:

1. Xen: `"XenVMMXenVMM"`

2. VMware : `"VMwareVMware"`

3. Hyper-V: `"Microsoft Hv"`

4. KVM: `"KVMKVMKVM"`

Apesar de ser uma técnica para detetar VM "honestas", ao contrários das estratégias anteriores, não tem problemas de manutenção, i.e., o método não muda ao longo do tempo, ao contrário das anteriores em que as assinaturas podem variar com as atualizações das VM, e por outro lado ao ocultar a presença do Hypervisor ao Linux pode comprometer o desempenho da VM. A assinatura do CPU pode ser usada para detetar a VM específica. Portanto é uma estratégia que vale a pena testar, muito embora só seja eficaz face a atacantes pouco sofisticados, i.e., incapazes de executar uma máquina virtual que não se anuncie ao sistema hospedado.

4.2.11 Model-Specific Registers reservados e inválidos

Esta estratégia baseia-se em procurar registos específicos do CPU, conhecidos por Model-Specific Register (MSR). Segundo [22], tentar aceder a um MSR reservado ou não implementado causa uma #GP. Contudo nos seus testes, os autores verificaram que o QEMU não levantou nenhuma exceção ao tentar aceder a um MSR reservado ou não implementado. Portanto é uma estratégia eficaz para detetar o QEMU.

```
1 asm volatile("WRMSR;" : : "c" (0x18b)); //input: IA32_MCG_RESERVED1
```

Código Fonte 4.7: Escrita num Model-Specific Register reservado

No seu teste, os autores de [22] tentaram escrever sobre o MSR `IA32_MCG_RESERVED1` (número `0x18B`), com a instrução Write MSR (WRMSR) apropriada através de *inline assembly* como no Código Fonte 4.7.

Os autores executaram o Código Fonte 4.7 num módulo do núcleo do Linux, pois os MSR são registos privilegiados que só podem ser acedidos em *kernel space*. Por último, os autores testam a existência de um MSR com a macro segura `rdmsr_safe` definida no núcleo Linux, que no fundo lida com a exceção causada. Isto é ótimo porque significa que podemos executar de forma segura este código em máquina virtual e física. Portanto é uma estratégia que vale a pena testar.

4.2.12 Comprimento das instruções

Segundo [22], só processadores da Intel colocam um limite de 15 octetos no comprimento das instruções. A única forma de violar o limite é colocando prefixos redundantes antes da instrução. Uma #GP é acionada se o limite da instrução é violado. Os autores descobriram que os CPU emulados pelo QEMU não colocam este limite.

Os autores usaram o prefixo Repeat String (REP). Este prefixo pode ser adicionado a uma instrução string para repetir a instrução várias vezes. Se este prefixo for repetido mais de 14 vezes consecutivamente antes de uma operação de movimento de strings, o CPU aciona uma #GP. Isto porque por exemplo 15×REP (1 octeto) + 1×MOVSB (1 octeto) > 15 octetos [22].

A estratégia é eficaz para detetar o QEMU, só que nos exames não podemos causar a falha do arranque do SO nas máquinas físicas. Portanto é uma estratégia eficaz mas não apropriada para o nosso caso.

4.2.13 Verificação de alinhamento

A arquitetura x86_64 suporta verificação de alinhamento. Quando código com privilégios baixos é executado, o alinhamento dos endereços de memória pode ser verificado pelo CPU. Quando verificação de alinhamento é ativada, através da ativação das *flags* Alignment Mask (AM) do registo Control Register 0 (CR0) e Alignment Check (AC) do registo EFLAGS, os acessos a memória não alinhada gera uma exceção (Alignment Exception, #AC). O QEMU não suporta esta verificação, e pode portanto ser distinguido de um CPU físico [22].

Para testar esta estratégia, os autores de [22] desenvolveram um módulo para o núcleo que ativa as duas *flags* AM e um programa em *user space* que activa a *flag* AC e este último tenta aceder um endereço que é tanto inválido como desalinhado. Se verificação de alinhamento estiver ativa, então uma #AC é acionada, caso contrário é gerada uma exceção de acesso a memória não reservada, a qual redunda num *segmentation fault* da aplicação. No QEMU só esta última situação ocorre.

Tal como a estratégia anterior, não podemos causar exceções no arranque do SO (seja com um #AC, seja com um *segmentation fault*) durante a prova de exame e não podemos depender de uma aplicação que corre em *user space*, visto que o VMHalt vai ser executado em tempo de arranque do núcleo, onde ainda não foram lançadas aplicações em *user space*. Portanto é uma estratégia eficaz mas não apropriada para o nosso caso.

4.2.14 Desempenho relativo – comparação de instruções

O desempenho absoluto de execução de certas instruções sobre uma VM ou Emulador é menor do que sobre o hardware real. Contudo, usar o desempenho absoluto é difícil, pois a heterogeneidade do hardware dos computadores é grande.

Portanto esta estratégia pretende medir o tempo absoluto de execução de uma instrução privilegiada e comparar com o tempo absoluto de execução de uma instrução não privilegiada. A esta aproximação é chamámos de desempenho relativo. Com esta, o sistema é caracterizado pelo rácio de execução entre duas ou mais operações executadas no sistema. A ideia é que uma instrução privilegiada em princípio necessita de tarefas diferentes por parte do CPU para a correr diretamente sobre o hardware e para uma VM ou Emulador a simular. Por outro lado uma instrução não privilegiada não irá diferenciar muito, principalmente nas VM, onde as instruções não privilegiadas executam diretamente sobre o hardware.

Para além das instruções privilegiadas, podemos usar as instruções que devem ser simuladas pelo Hypervisor, as instruções sensíveis, que acionam um #VMEXIT quando é usada virtualização assistida por hardware. O Virtual Machine Exit (VMEXIT) é uma operação que muda o contexto atual da VM para o Hypervisor para que o mesmo possa simular a instrução sensível (ver Secção 2.4.4). Ora esta mudança de contexto leva milhares de ciclos de relógio. Portanto, no caso dos Hypervisores que usam virtualização assistida por hardware podemos usar como instruções alvo as instruções sensíveis da arquitetura Intel.

Para além do atraso causado pelo #VMEXIT causado pela virtualização assistida por hardware, as *hypercalls* da paravirtualização e a tradução binária da virtualização completa também devem diminuir o desempenho das instruções sensíveis. Portanto esta estratégias tem a potencialidade de detetar os três grandes paradigmas de virtualização e os dois grandes paradigmas de emulação (interpretação e tradução dinâmica). Naturalmente este facto é de esperar, visto que a característica que mais distingue máquinas reais de virtuais é o seu desempenho.

Os autores de [22, 7] recomendam a instrução No OPeration (NOP) como instrução base. Em [22] recomendam a leitura e escrita dos registos de controlo CR0 e CR3. Em [7] recomendam o uso da instrução CPU IDentification (CPUID). Em ambos os textos os autores concentram-se em detetar tanto a presença de um emulador (QEMU) como de uma VM (VMware). Para a medição do tempo pode ser usado o Time Stamp Counter (TSC), que pode ser lido através da instrução Read TSC (RDTSC). Os autores de [22] testaram a estratégia num módulo do núcleo Linux para poderem desativar as interrupções e o escalonador, de forma a diminuir a variância das medições. Portanto é uma estratégia que vale a pena testar.

Contudo, o uso desta na prática levanta alguns problemas. Nomeadamente, este é vulnerável aos ataques RDTSC *offset cheating* e BlueChicken, e algumas VM estão preparadas para os realizar.

Ataque RDTSC *offset cheating*

De forma a ocultar a ocorrência dos #VMEXIT, as tecnologias de virtualização assistida por hardware AMD-V e VT-x permitem a modificação do TSC dos CPU virtuais. Pode ser definido um TSC_OFFSET que é subtraído ao TSC depois de cada interceção do Hypervisor. Este encontra-se na VMCS (ver Secção 2.4.5).

As transições entre VM e Hypervisor não têm período fixo, o que torna difícil definir um

offset constante para corrigir o tempo. Uma alternativa foi proposta para corrigir o TSC que não obriga o cálculo do *offset*, o RDTSC *offset cheating*, ilustrada na Figura 4.1. Antes da execução da instrução RDTSC, a sequência de instruções entre duas operações RDTSC é traçada e o número de ciclos de relógio para executar toda a sequência é calculado. Na execução do segundo RDTSC a execução é intercetada e os valores dos registos de leitura [EDX:EAX] são substituídos por valores falsos, iguais ao último TSC mais o número de ciclos calculados durante a Virtual Machine Entry (VMENTRY) [1].

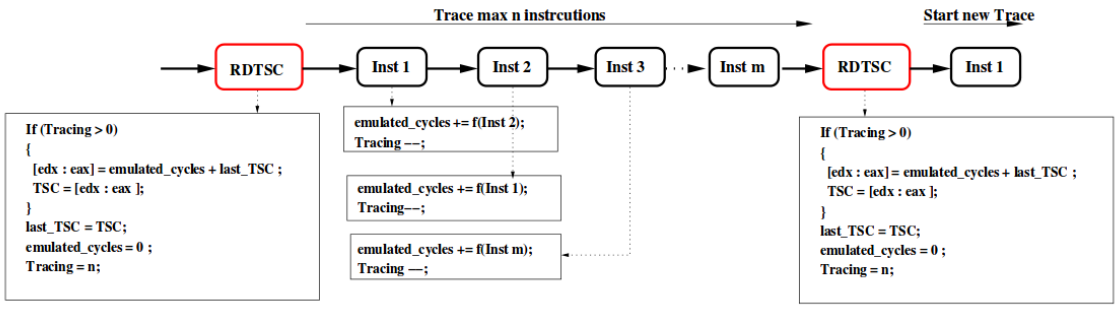

Figura 4.1: Ataque RDTSC *offset cheating* (imagem extraída de [1]).

Relógios externos

Uma alternativa estudada passa por usar fontes de tempo que o Hypervisor não pode alterar. Um exemplo é usar um relógio remoto através do Network Time Protocol (NTP). Visto que a leitura de um relógio apresenta um atraso, a execução de apenas uma instrução não providência a resolução suficiente para destingir um hardware real de um Hypervisor. Portanto a instrução sensível é executada múltiplas vezes dentro de um ciclo para compensar o atraso da leitura do relógio remoto [1].

```
1  int vmm_present_ntp(int threshold) {
2      int t1 = GetTimeViaNTP();
3      for (i = 0; i < BIGNUM; i++)
4          rdmsr(EFER); // #VMEXIT
5      int t2 = GetTimeViaNTP();
6      return t2 - t1 > threshold;
7  }
```

Código Fonte 4.8: Medição do desempenho com o NTP (código extraído de [1]).

Ataque BlueChicken

Não há razão para uma aplicação correr um ciclo longo de instruções sensíveis como o Read MSR (RDMSR) num intervalo de tempo curto, como é necessário para a medição do desempenho com o NTP. Um Hypervisor malicioso pode detetar tais ciclos e prevenir as medições de tempo longas causadas pelo #VMEXIT. A técnica usada pelo Hypervisor *rootkit* BluePill é chamada de BlueChicken e consiste no Hypervisor se desinstalar ao se aperceber de uma tentativa de deteção. O Hypervisor mantém um temporizador que quando termina

a contagem, volta a instalar-se. Enquanto este está desinstalado o tempo de execução das instruções será o tempo de execução direta sobre o hardware [10, 2].

4.2.15 Desempenho relativo – instruções em cache

As caches de instruções e de dados têm uma enorme influência no desempenho de execução de um programa. Esta estratégia tenta verificar se há alguma diferença no comportamento da cache de um processador físico ou simulado ao executar a mesma peça de código muitas vezes. A ideia é que o desempenho vai aumentar após a primeira execução.

Em [22] os autores concluem que o efeito da cache é mais acentuado em ambientes simulados (tanto em emuladores ou em máquinas virtuais) do que em ambiente real. Também notaram que os simuladores não rejeitam os pedidos para desligar a cache (QEMU e VMware). Os tempos antes e depois de desligarem a cache foram muito semelhantes (primeira execução muito mais lenta). Para desligar a cache os autores colocam o CPU em modo *no-fill* cache (ligando o bit CD no registo CR0 e desligando o bit NW). A cache é limpa com a instrução Write Back and Invalidate Cache (WBINVD). Finalmente desativam todos os Memory Type Range Register (MTRR) e colocam a memória em modo uncached ou todos os MTRR teriam que ser configurados também em modo uncached. Tal como na comparação de instruções, usam o TSC para medir o tempo de execução e desenvolvem um módulo para o núcleo Linux que permite desativar as interrupções e o escalonador para diminuir a variância das medições.

Por tudo isto, esta é uma estratégia que vale a pena testar. Contudo, é vulnerável às VM que falsifiquem o valor do TSC.

4.2.16 Return Stack Buffer

Esta estratégia foi proposta pela Intel. O Return Stack Buffer (RSB) como o nome indica, é uma pilha de hardware com à volta de 16 entradas. O endereço de retorno de uma chamada é previsto e armazenado neste *buffer*. São invocadas funções até encher o *buffer*. As primeiras 15 funções apenas invocam a função seguinte. A última função executa uma instrução sensível que implique um #VMEXIT. O processo de transição entre a VM e o Hypervisor vai remover certas entradas do RSB, substituindo por rotinas de transição do VMEXIT. Após isto é medido o TSC através do RDTSC. Após o retorno de todas as funções o TSC volta a ser lido e o intervalo de tempo é calculado. Daqui resulta que numa VM o intervalo de tempo do retorno das chamadas vai ser maior porque ocorreram várias faltas de RSB [10, 1].

Esta é também uma estratégia que vale a pena testar. Contudo, tal como a anterior, é vulnerável às VM que falsifiquem o valor do TSC.

4.2.17 Contador paralelo

Esta estratégia vem resolver o problema da medição do tempo com o TSC. Em vez de medir o TSC do CPU, usamos contador internos da aplicação de deteção [2, 10, 1].

São arrancadas duas *threads*. A primeira *thread* executa uma instrução sensível. A segunda *thread* conta o número de ciclos que a primeira *thread* demora a correr a instrução sensível (que aciona um #VMEXIT). Antes de executar a instrução sensível, a primeira *thread* notifica a segunda *thread* que pode começar a contar. Assim que acabar de executar a instrução sensível a primeira *thread* notifica a segunda que pode parar de contar. Não é preciso avaliar o desempenho relativo, porque nós contamos o número de ciclos e não o intervalo de tempo. O número de ciclos deverá ser invariante sempre que corremos uma instrução no mesmo sistema.

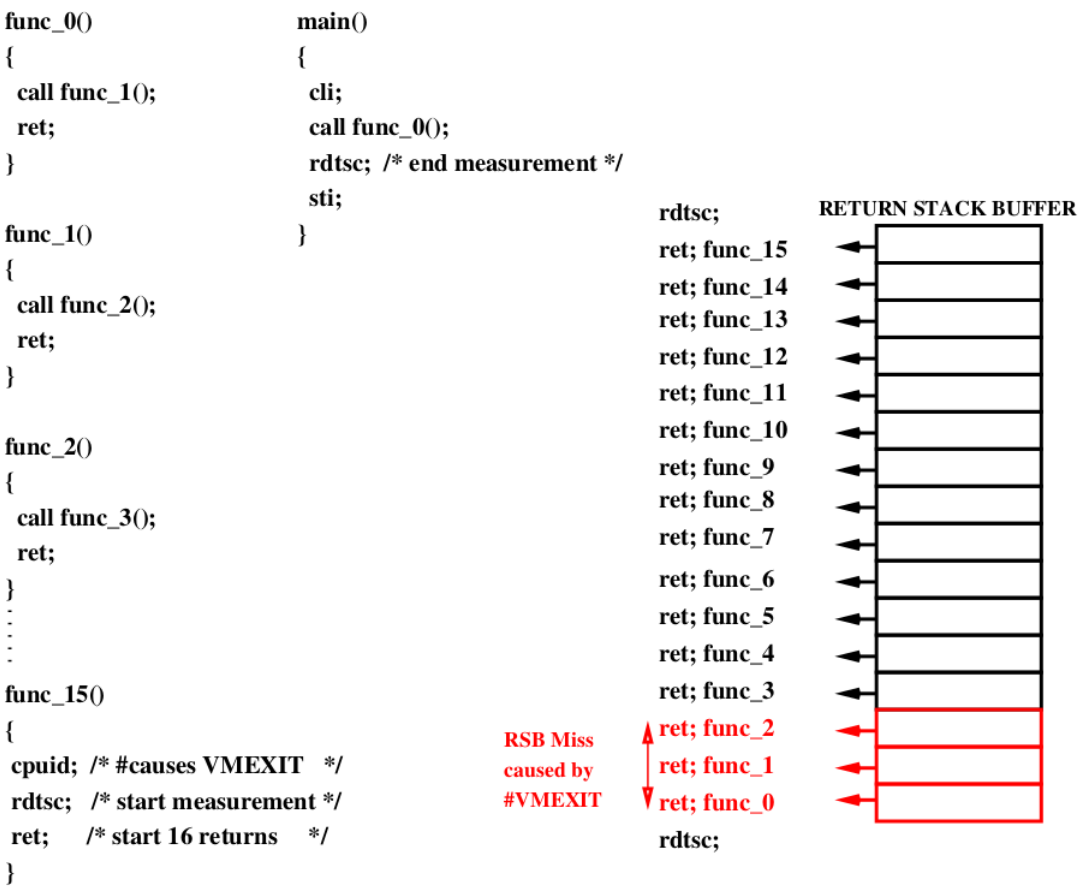

Figura 4.2: Return Stack Buffer (imagem extraída de [1]).

Esta técnica requer um processador multi-core, pois as duas *threads* devem correr simultaneamente em 2 CPU físicos e não apenas em concorrência. Isto seria uma desvantagem em sistemas mais antigos mas a maior parte dos computadores portáteis já possui 2 ou mais processadores. O esquema encontra-se ilustrado na Figura 4.3.

Ao contrário das estratégias anteriores, não existe contra-ataque (ou defesa) possível. Um *rootkit* teria que conhecer a semântica do contador de deteção para conseguir detetar a sua execução. No caso teórico do Hypervisor conhecer a semântica do detetor, para manipular este teria que usar mais medidas que eventualmente poderiam ser detetadas. Porém, a troca de contexto da VM para o Hypervisor com as rotinas do VMEXIT, já levou à queda de desempenho, tornando-se demasiado tarde para o Hypervisor responder, mesmo conhecendo a semântica do detetor.

Tal como as outras estratégias de medição do desempenho de instruções privilegiadas e sensíveis, esta é candidata a conseguir detetar todos os grandes paradigmas de virtualização e emulação. É, portanto, uma estratégia que vale a pena testar.

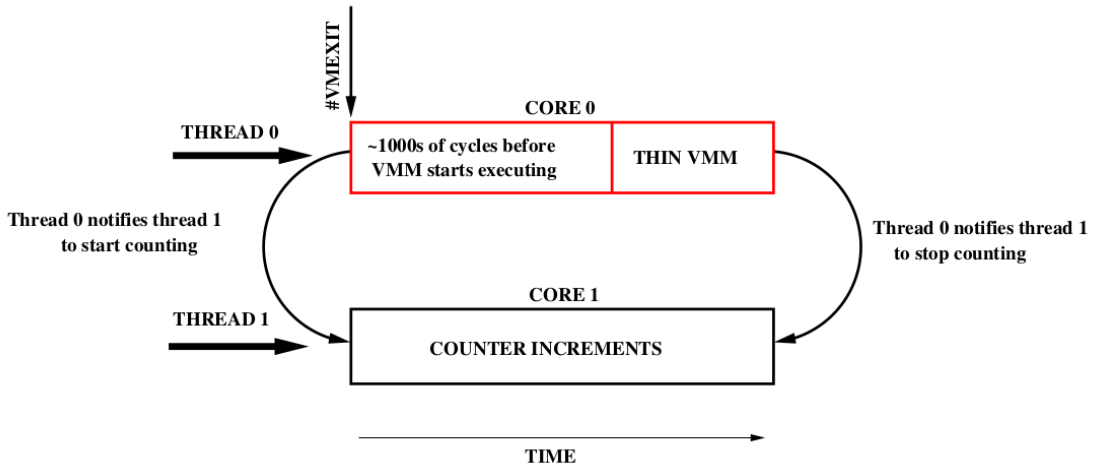

Figura 4.3: Contador Paralelo (imagem extraída de [1]).

4.2.18 TLB *profiling*

Uma estratégia enunciada em [2, 10, 1] consiste em preencher a TLB, causar um #VMEXIT que vai alterar a TLB e detetar as alterações nas páginas.

Os autores de [1] começam por alojar um número máximo de páginas de 4 KBytes, e preencher as páginas com um valor conhecido, por exemplo `0x44`. O número máximo é número de mapeamentos suportados pela TLB da cache L1. Uma nova página de 4 KBytes é alojada e preenchida com um valor diferente, por exemplo `0x55`. A TLB é preenchida na sua totalidade acedendo posteriormente a todas as páginas que foram preenchidas com `0x44`. As entradas da tabela de páginas correspondentes aos endereços destas páginas são arranjadas (*patched*) para apontar para uma nova página, a que contém o valor `0x55`. Depois é causado um #VMEXIT, que vai libertar a TLB no caso do VT-x, ou alterar algumas das entradas no caso do AMD-V. Agora voltamos a ler os valores de todas as páginas (exceto a extra) e se o valor `0x55` for encontrado é porque ocorreu um *page walk* devido a uma falta na TLB. Portanto podemos concluir que existe um Hypervisor.

Esta é uma estratégia que vale a pena testar. Contudo esta é vulnerável à manipulação das tabelas de páginas sombra (*shadow page tables*, ver Secção 2.4.4).

Shadow Page Tables

A estratégia de TLB *profiling* pode ser derrotada usando tabelas de páginas sombra. Qualquer tentativa de alterar as Page Table Entries (PTE) pelo sistema hospedeiro pode ser intercetada pelo Hypervisor, evitando a adulteração das PTE [10, 1]. No caso da BluePill, foi proposto que poderia usar este contra-ataque para se desinstalar assim que as PTE estivesse a ser adulteradas.

4.2.19 TLB *timing profiling*

A estratégia consiste em preencher a TLB de uma forma idêntica à estratégia anterior e medir o tempo de acesso às páginas em cache. Depois é executado uma instrução sensível que despoleta um #VMEXIT. Se existir, o Hypervisor deverá alterar algumas das entradas da

TLB [2, 1]. Depois voltamos a medir o tempo que leva a aceder às páginas. Se demorar mais tempo é porque houve que preencher algumas entradas na TLB (*table misses*), logo estamos sobre um Hypervisor.

Em [2] referem que existem Hypervisors baseados na arquitetura AMD que podem ordenar ao hardware que não limpe as entradas da TLB. Esta estratégia também é vulnerável aos ataques contra o TSC. Como existem atualmente mais computadores com processadores da Intel do que a AMD esta estratégia vale a pena ser testada.

4.2.20 Predição de saltos

Num processador pipelined, a predição de saltos é usada para continuar a execução de código nas próximas fases do pipeline quando uma instrução numa fase mais à frente do pipeline ainda não tomou a decisão de salto. Se a predição for correta aproveita-se o tempo que seria desperdiçado por esperar pela decisão da instrução de salto. Se a decisão for a errada, todo o pipeline que se segue é limpo. Em [2], os autores propõem a deteção do Hypervisor a partir do histórico dos saltos. A predição é feita através de registo dos resultados anteriores numa cache de história.

A estratégia baseia-se em encher as entradas das tabelas de histórico dos saltos usando as regras de preditores estáticos e dinâmicos e medir o tempo que o código demora a executar. Agora corremos uma instrução privilegiada que tem de ser intercetada pelo Hypervisor. O Hypervisor vai afetar as tabelas de histórico dos saltos. Voltamos a correr o código. Se o tempo da segunda execução for diferente, é porque um Hypervisor intercetou a execução da instrução privilegiada.

Este método requer conhecimento detalhado da implementação da Branch Prediction Unit do CPU, tornando difícil a sua implementação. Portanto não considerámos testar esta técnica.

4.3 *Virtual Machine Halt*

O VMHalt foi uma solução concebida para terminar a execução de um SO quando o mesmo for executado sobre uma VM ou Emulador. O VMHalt vai executar vários testes de deteção de uma VM subjacente; caso um deles dê um resultado positivo, o SO é parado. O objetivo principal do VMHalt é a deteção de VM e Emuladores desconhecidos que permitam virtualizar a arquitetura x86_64.

4.3.1 Assunções e opções de desenho

O VMHalt foi concretizado num módulo do núcleo Linux para poder ser executado durante o arranque do SO e com privilégios de *kernel space*. Isto deve-se ao facto de as instruções privilegiadas da arquitetura x86_64, como o nome indica só podem correr com privilégio elevados (*kernel space*) e de grande parte das instruções sensíveis serem também privilegiadas e só poderem serem executadas com esses privilégios. Uma exceção é a CPUID, que é uma instrução sensível mas não privilegiada, e daí o surgimento da virtualização assistida por hardware. Por outro lado, não queremos dar oportunidade de um aluno durante a prova de poder passar a fase de arranque do SO ao correr sobre uma VM.

Com O VMHalt queremos detetar todos os paradigmas de virtualização e emulação, mas queremos ainda mais não classificar um sistema físico como virtual (falso positivo) e impedir um aluno de realizar a prova de exame sobre o seu hardware. O ideal seria ter um conjunto

grande de testes e tomar a decisão baseada em votos. Só que, como veremos, infelizmente não é o caso; são muito poucos os testes que conseguimos implementar e que realmente funcionam na prática com as tecnologias de virtualização atuais. Portanto o critério usado foi: se um teste classifica o sistema como virtual, então classificamos o sistema como virtual. Por consequência da opção anterior, excluímos todos os vetores de teste que poderiam facilmente resultar em falsos positivos.

4.3.2 Testes implementados

Ao longo deste capítulo temos vindo a analisar as estratégias propostas por trabalhos anteriores e avaliámos a sua aplicabilidade ao nosso caso de uso. Contudo não implementámos todas as técnicas que poderíamos vir a usar. Nomeadamente, não implementámos a estratégia da TLB *profiling* porque não conseguimos encontrar documentação sobre o arranjo (*patching*) das PTE da TLB.

Ficámo-nos por quatro testes: (i) detetar VM "honestas" através dos dados fornecidos pelo núcleo Linux; (ii) contador paralelo; (iii) detetar MSR reservados ou inválidos; (iv) desempenho relativo baseado em comparação de instruções; (v) desempenho relativo baseado em efeitos na cache.

Testámos outros testes, como a *backdoor* do VMware, Return Stack Buffer e o TLB *timing profiling* mas não conseguimos observar os comportamentos esperados.

4.4 Concretização

Nesta secção descrevemos de forma detalhada a implementação do VMHalt. Quanto às técnicas, descrevemos apenas as técnicas usadas na nossa solução final.

Para parar o núcleo no caso de sobre uma VM, usamos a primitiva `kernel_halt`. A deteção de VM "honestas" e a deteção de MSR reservados ou inválidos é feito como descrito na Secção 4.2.10 e na Secção 4.2.11 respetivamente, com a leitura da variável `cpu_has_hypervisor` e com a leitura da exceção retornada pela primitiva `rdmsr_safe` do MSR `IA32_MCG_RESERVED1`. Se retornar erro ao ler um MSR reservado, é porque estamos a correr sobre hardware, se não retornar erro é porque estamos a correr sobre um sistema simulado.

Antes de medir o tempo nos testes de comparação de instruções, corremos a primitiva `local_irq_disable` para desativar todas as interrupções e o escalonador do núcleo. No final das medições as interrupções e escalonador são reativados com a primitiva `local_irq_enable`.

O mesmo foi feito nos testes sobre os efeitos cache. Como explicado na Secção 4.2.15, ativamos e desativamos os bits do registo CR0 para desligar e ligar a cache. Primeiro realizamos as medições com a cache ligada. Depois libertamos a cache e a TLB. Depois voltamos a fazer as mesmas medições com a cache desligada.

Na implementação do teste do contador paralelo são executadas duas *kernel threads* (*kthreads*). Usando a primitiva `kthread_run` a primeira *thread* cria uma segunda *thread*. Para identificar o CPU onde está a correr a primeira *thread* usa a primitiva `smp_processor_id`, e usando a primitiva `kthread_bind` a *thread* prende-se a um CPU. A segunda *thread* usa a mesma primitiva para se prender a outro CPU. Para diminuir a variância nas medições, antes da contagem começar as *threads* usam as primitivas `local_irq_disable` e `local_irq_enable` para desativar e reativar as interrupções e o escalonador. O protocolo de sincronização entre as duas *threads* é feita através de memória partilhada com variáveis globais voláteis, para

que cada *thread* leia sempre o valores mais atualizados da memória, e não da copia local nos registos do CPU.

Para tornar os testes baseados em desempenho mais robustos, estes são repetidos várias vezes usando várias instruções sensíveis. Escolhemos apenas instruções de leitura para não alterar o fluxo do programa nem estruturas sensíveis. Segundo [26], as instruções de leitura que implicam um #VMEXIT são:

1. CPUID - lê dados relacionado com o CPU;
2. RDMSR Extended Feature Enable Register (EFER) - um MSR que contém o bit Secure Virtual Machine Enable (SVME);
3. RDTSC - lê o registo TSC;
4. MOV from Control Register 3 (CR3) - o registo CR3 armazena dados relacionados com a memória virtual;
5. MOV from Control Register 8 (CR8) - o registo CR8 armazena dados relacionados com prioridade de interrupções externas e tarefas;
6. Read Performance Measurement Counter (RDPMC) - lê os contadores de monitorização de desempenho;
7. IN(B/W/L) - operação de leitura externa (byte, word, long).

Nos testes baseados em desempenho das instruções, o resultado de cada uma leva a um voto. Cada voto que classifica o sistema como virtual vale $+1$ e -1 caso contrário; começamos com 0. Se o número de votos for maior que zero, o teste classifica o sistema como virtual.

Para carregar o módulo VMHalt durante o arranque do núcleo é preciso: (i) adicionar o nome do módulo ao ficheiro `/etc/modules` sem a extensão `.ko` (kernel object); (ii) mover o módulo para o diretório `/lib/modules/‘uname -r‘/kernel/drivers`.

4.5 Limiares dos testes baseados em desempenho

Os testes de desempenho baseiam-se em comparar os valores medidos com um limiar predefinido. Se o valor medido estiver abaixo do limiar, encontramos-nos a correr sobre o hardware; caso contrário encontramos-nos a correr sobre um sistema virtual.

Realizámos testes empíricos com o objetivo de definir o limiar da contagem das várias instruções para o contador paralelo, para o desempenho relativo baseado na comparação de instruções e para o desempenho relativo baseado em efeitos da cache. Os testes foram realizados em computadores de vários colegas da UA. Por cada computador testado, corremos o VMHalt tanto diretamente no hardware como em várias VM. Teoricamente os valores deveriam ser invariantes de computador para computador, mas pode existir alguma variância devido à latência no acesso às memórias cache ou no caso do contador paralelo, alguma discrepância entre a frequência nos dois CPU. De forma a obter dados confiáveis, realizámos testes nas VM/Emuladores mais usados atualmente em ambiente académico: Virtual Box, VMware Player e QEMU (em modo Emulador e KVM).

O tempo de execução das instruções sensíveis é menor quando executadas diretamente sobre o hardware do que numa máquina virtual. Com o contador paralelo temos $C_H < C_V$,

Tabela 4.1: Limiar da contagem de instruções sensíveis

Sistema	CPUID	RDMSR EFER	RDTSC	RDPMC	INB
ASUS K55VM-SX045V Intel Core i7-3610QM Quad Core at 2.3 GHz, 8 GiB of RAM					
Hardware	73	24	21	21	827
VMWare	444	406	0	1085	6652
Virtual Box	891	1400	1041	931	1866
QEMU-KVM	405	548	0	-	10257
QEMU	0	0	0	-	0
Clevo W150-HRM Intel Core i7-2720QM Quad Core at 2.2 GHz, 8 GiB of RAM					
Hardware	51	24	24	22	329
VMWare	616	804	0	1541	9296
Virtual Box	901	1253	803	811	1311
ASUS G550JK Intel Core i7-4710HQ Quad Core at 2.5 GHz, 16 GiB					
Hardware	73	21	0	22	1159
VMWare	519	369	0	927	4675
Virtual Box	763	1244	884	746	1523
Lenovo z50-70 Intel Core i7-4510HQ Quad Core at 2.0 GHz, 8 GiB					
Hardware	82	26	30	24	218
VMWare	387	418	7	6291	4774
Virtual Box	767	1131	882	761	1300
HP Envy 14-1040EP Intel Core i7-720QM Quad Core at 1.6 GHz, 6 GiB					
Hardware	83	28	17	30	554
VMWare	429	470	3	6042	4821
Virtual Box	857	1244	1013	899	1490
QEMU-KVM	435	452	0	-	1170
Resultados					
$\max C_H$	83	28	30	30	1159
$\min C_V$	387	369	803	761	1311
L_C	235	199	416	401	1235

com C_H a contagem obtida durante a execução em hardware e C_V a contagem obtida durante a execução num sistema virtual. O nosso limiar de contagem L_C, para cada instrução sensível, deve ser um valor arbitrário entre a contagem máxima de execução direta no hardware e o tempo mínimo de execução em máquina virtual, $\max C_H < L_C < \min C_V$. Ou seja, o limiar deve ser maior que o tempo de execução máximo em hardware (verdadeiro negativo) e menor que o tempo de execução mínimo em ambiente virtual (verdadeiro positivo). Escolhemos como limiar arbitrário o valor intermédio entre ambos os limites:

$$L_C = \frac{\max C_H + \min C_V}{2}$$

Os resultados encontram-se na Tabela 4.1.

De forma semelhante, no desempenho relativo baseado em comparação de instruções realizámos várias medições de tempo. Usámos como base o tempo de execução da instrução NOP, e como tempo alvo as mesmas instruções testadas no contador paralelo. Ou seja o tempo de execução relativo no hardware é dado por

$$R_H = \frac{T_H}{T_H(\text{NOP})}$$

onde T_H é o tempo de execução no hardware de uma dada instrução e $T_H(\text{NOP})$ é o tempo

Tabela 4.2: Limiar do desempenho relativo de instruções sensíveis

Sistema	CPUID	RDMSR EFER	RDTSC	RDPMC	INB
ASUS K55VM-SX045V Intel Core i7-3610QM Quad Core at 2.3 GHz, 8 GiB of RAM					
Hardware	17	6	1	2	347
VMWare	109	91	1	663	1275
Virtual Box	1	1	2	1	2
QEMU-KVM	98	87	2	-	1451
QEMU	440	479	1	-	88
Clevo W150-HRM Intel Core i7-2720QM Quad Core at 2.2 GHz, 8 GiB of RAM					
Hardware	15	6	2	3	97
VMWare	97	62	1	182	654
Virtual Box	1	2	2	2	1
HP Envy 14-1040EP Intel Core i7-720QM Quad Core at 1.6 GHz, 6 GiB					
Hardware	28	5	1	3	63
VMWare	89	84	1	611	1024
Virtual Box	1	2	2	2	1
QEMU-KVM	108	99	2	-	1179
Resultados					
$\max R_H$	28	6	2	3	347
$\min R_V$	89	62	1	182	1024
L_R	59	34	2	93	686

de execução no hardware da instrução NOP. Da mesma forma para sistemas virtuais temos

$$R_V = \frac{T_V}{T_V(\text{NOP})}$$

Assumimos que $T_H(\text{NOP}) \approx T_V(\text{NOP})$. Portanto temos

$$T_H < T_V \equiv \frac{T_H}{T_H(\text{NOP})} < \frac{T_V}{T_V(\text{NOP})} \equiv R_H < R_V$$

e $\max R_H < L_R < \min R_V$, onde L_R é o limiar do tempo de execução relativo. Escolhemos como limiar arbitrário o valor intermédio entre ambos os limites:

$$L_R = \frac{\max R_H + \min R_V}{2}$$

Os resultados encontram-se na Tabela 4.2.

Para testar os efeitos de cache, medimos o tempo de execução de uma instrução com a cache ligada C_{on}, e com a cache desligada C_{off}. A ideia é que no hardware o desempenho de uma instrução com a cache desligada vai ser muito maior que com a cache desligada $C_{\text{off}}(H) > C_{\text{on}}(H)$, enquanto num sistema virtual deverão ser próximos ou mesmo iguais $C_{\text{off}}(V) \approx C_{\text{on}}(V)$. Tal como os testes anteriores, fazemos uma avaliação do desempenho relativo através do rácio

$$C_{\text{ratio}} = \frac{C_{\text{off}}}{C_{\text{on}}}$$

Mais uma vez, escolhemos como limiar arbitrário o valor intermédio entre ambos os limites:

$$L_E = \frac{\min C_{\text{ratio}}(H) + \max C_{\text{ratio}}(V)}{2}$$

A diferença para os testes anteriores, é que agora o hardware é que deve demonstrar um desempenho relativo mais baixo (tempo de execução relativo mais alto) que todos os sistemas virtuais. Os resultados encontram-se na Tabela 4.3.

Tabela 4.3: Limiar do desempenho relativo de efeitos de cache

Sistema	C_{on}	C_{off}	C_{ratio}
ASUS K55VM-SX045V Intel Core i7-3610QM Quad Core 2.3 GHz, 8 GiB of RAM			
Hardware	50	1960	39
VMWare	50	50	1
Virtual Box	7378	7422	1
QEMU-KVM	36	36	1
QEMU	30	30	1
Resultados			
$\min C_{\text{ratio}}(H)$			39
$\max C_{\text{ratio}}(V)$			1
L_E			20

4.6 Análise de resultados

O núcleo do Linux consegue detetar todas as VM que têm uma assinatura conhecida, as mostradas na Secção 4.2.10. Ou seja o VMware e o QEMU-KVM são detetados pelo Linux. O resultado interessante foi que o QEMU em modo emulador também se revela como um Hypervisor. É um resultado inesperado mas é uma mais-valia.

Como era esperado conseguimos detetar o QEMU com MSR reservados. O resultado inesperado foi que a VMware também não lança uma exceção quando é lido o mesmo MSR. É um resultado inesperado mas é também uma mais-valia.

Conseguimos obter vários resultados quanto ao contador paralelo. Primeiro, as instruções MOV from CR3 e MOV from CR8 não são virtualizadas, pois resultam sempre de uma contagem igual a 0 tal como no hardware, daí terem sido retiradas da Tabela 4.1. O RDTSC, ao contrário do que a documentação indicava, parece que não é virtualizado.

Descobrimos que o QEMU (tanto em modo Emulador como em KVM) não consegue executar a instrução RDPMC, causando o bloqueio da execução dos CPU onde o VMHalt se encontra a correr. Ou seja, ao correr o teste do contador paralelo são ocupados dois CPU, mas se o Emulador tem quatro CPU virtuais, os outros dois continuam em execução e como o VMHalt não termina, o núcleo Linux não é parado. Portanto esta instrução teve que ser retirada dos testes do VMHalt.

Verificámos que o contador paralelo não funciona de todo com nenhuma instrução no QEMU em modo emulador. Este resultado é explicado pelo seguinte facto: o QEMU tem vários CPU virtuais, mas executa apenas num CPU real. Ora o contador paralelo necessita que as duas *threads* executem em simultâneo, mas como o QEMU as executa em pseudo-concorrência, a *thread* que conta só começa a executar depois da outra *thread* ter terminado. Se observarmos os resultados do desempenho relativo, o mesmo já não ocorre e o QEMU tem um desempenho muito inferior nas instruções privilegiadas. Contudo podemos usar os resultados do contador paralelo como uma assinatura. Ou seja se a contagem com o contador paralelo resultar em zero, estamos sobre um emulador que usa apenas um CPU real.

Com o Virtual Box averiguámos que, tal como o contador paralelo indicava, os valores absolutos da medição do tempo com o RDTSC são de uma ordem de grandeza muito maior, mas o problema é que o mesmo acontece com a instrução NOP, ou seja

$$T_H(\text{NOP}) \ll T_V(\text{NOP}) \quad \text{e} \quad R_V = \frac{T_V}{T_V(\text{NOP})} \approx 1$$

Tabela 4.4: Resultados finais do VMHalt. Não conta com as assinaturas descobertas.

	VMware	Virtual Box	QEMU-KVM	QEMU
Núcleo Linux	✓	✗	✓	✓
Contador paralelo	✓	✓	✓	✗
MSRs reservados	✓	✗	✗	✓
Comparação de instruções	✓	✗	✓	✓
Efeitos de cache	✓	✓	✓	✓
	✓	✓	✓	✓

para todas as instruções, o que está contra as nossas assunções, e não conseguimos explicar este fenómeno. Mas, tal como os resultados do contador paralelo no QEMU, podemos usar este resultado para criar uma assinatura que identifique o Virtual Box. Se todos os desempenhos relativos estiveram na ordem de um, estamos a correr sobre o Virtual Box.

Não conseguimos explicar o comportamento do INB no QEMU, o seu desempenho relativo é muito inferior que o hardware.

O VMware executa por omissão com virtualização completa, o Virtual Box executa com virtualização assistida por hardware e o QEMU-KVM com paravirtualização. Isto demonstra que apesar do trabalho anterior se focar em instruções que perdem desempenho no tempo da troca de contexto do VMEXIT com tecnologia de virtualização assistida por hardware, os outros paradigmas também têm uma perda de desempenho a simular essas mesmas instruções.

Os valores $\min C_V$ e $\min R_V$ das tabelas excluíram os casos em que claramente as instruções não estavam a ser virtualizadas, ou que não corriam em simultâneo com o contador.

Quanto aos resultados dos efeitos de cache, os resultados estão de acordo com o esperado, ao ponto de ultrapassar as nossas expectativas, visto que é o único método para detetar sistemas virtuais desconhecidos que funciona em todos as VM/Emuladores testados. Também é o único método de avaliação de desempenho que não está dependente da execução de instruções privilegiadas e sensíveis.

A compilação dos resultados encontra-se na Tabela 4.4.

4.7 Conclusões

O módulo VMHalt que desenvolvemos para o núcleo do Linux conseguiu detetar todas as VM testadas (ou seja, sem falsos negativos), que abrangeram todos os paradigmas de virtualização e emulação, sem nunca classificar o hardware como sistema virtual (ou seja, sem falsos positivos).

O desenvolvimento deste módulo foi uma tarefa desafiante pois muito do trabalho relacionado era focado na deteção de Hypervisors *rootkits*, que muitas vezes envolvem estratégias muito distintas. E, como referimos anteriormente, a deteção de emuladores também é um trabalho escasso, visto que hoje em dia há mais interesse nas VM, porque possuem um desempenho muito superior.

Apesar das variâncias, provamos que os testes mantêm consistência entre várias VM e Emuladores a executarem em vários computadores com estruturas de hardware distintas.

Mas o trabalho não acaba por aqui. O VMHalt vai ter que ser adaptado ao longo do tempo às novas tecnologias que vão surgindo. A área da virtualização foi uma área que evoluiu muitíssimo nestes últimos 15 anos e não sabemos o que vai conseguir alcançar. O que podemos esperar é que vai sempre haver um processo de simulação que demora mais tempo a correr do que no hardware, esperando por isso que a deteção por desempenho permaneça.

Capítulo 5

DETIexam

Nos capítulos 3 e 4 descrevemos o AFBP, o protocolo para autenticação em comunicação por difusão, e o VMHalt, o sistema de deteção de máquinas virtuais. Estes dois mecanismos permitem a descarga e arranque seguro de uma imagem Linux distribuída pelo docente para realizar as provas de exame nos computadores pessoais dos alunos.

Neste capítulo exploramos o DETIexam, o ecossistema para a exploração segura de exames informáticos nos computadores pessoais dos alunos. Concretizámos uma imagem Linux reforçada para ser distribuída pelo DETIboot, que impede os alunos de acederem a recursos não autorizados durante a prova. O objetivo final é que os alunos apenas possam entregar a prova através do SO reforçado nos seus computadores pessoais, dentro da sala de exame, e que não possam trocar de sistema enquanto realizam a prova.

Neste capítulo não pretendemos desenvolver um mecanismo para produzir a imagem para o exame, mas sim definir as políticas e mecanismos que terão que ser exercidos pelo SO reforçado que corre nas máquinas dos alunos durante a prova, e por uma entidade central reguladora do exame.

5.1 Status quo: exames informáticos com PC *desktop*

O DETI já realiza alguns exames práticos de programação em PC *desktop* instalados nas salas de aula. Estes correm sistemas Linux com as ferramentas de apoio à realização das aulas práticas e exames práticos, e encontram-se preparados para barrar o acesso à maior parte dos sites da Web. Os alunos também não podem aceder a dispositivos de E/S. Atualmente os exames são realizados da seguinte forma:

1. Os alunos entram na sala e acedem a um qualquer computador disponível na sala;
2. Quando todos os intervenientes estão prontos para começar a prova, o docente revela o nome de uma conta e uma senha que permite aos alunos entrarem na conta Linux dedicada para a prova;
3. Os alunos realizam a prova durante o tempo estabelecido pelos docentes;
4. Os alunos entregam a prova através de um diretório remoto acedido via a versão 3 do Network File System (NFS). Os alunos colocam dentro desse diretório um diretório seu, identificado pelo seu número mecanográfico da UA.

O servidor NFSv3 pode limitar o seu acesso com base nos endereços IP dos clientes. Como os endereços IP das máquinas dentro da salas de aula estão bem definidos, os alunos apenas podem depositar conteúdo através dessas máquinas. O esquema encontra-se ilustrado na Figura 5.1.

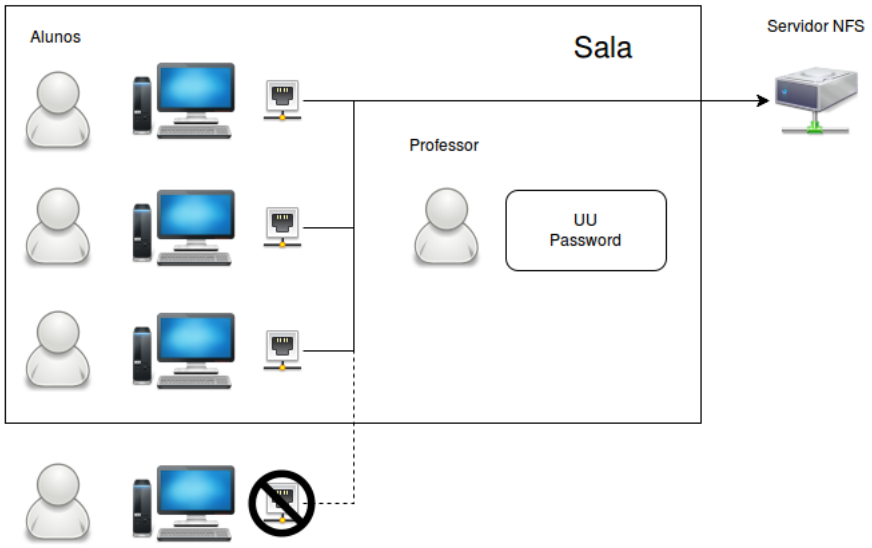

Figura 5.1: Exame com PC *desktop*. O docente revela o nome de Utilizador Universal (UU) e a senha para a entrada na conta Linux dedicada à prova. Apenas os alunos dentro da sala de aula podem depositar o trabalho produzido através de filtragem por endereços IP.

Não é necessário qualquer configuração de ligação à rede por parte dos alunos, visto que a ligação é por cabo.

5.2 Exames informáticos com os portáteis dos alunos

A realização das provas de exame nos computadores portáteis dos alunos traz novos problemas com que vamos ter de lidar.

O primeiro problema reside na possibilidade do aluno poder realizar a prova fora da sala de exame. Este apenas precisa de estar no raio de cobertura do servidor FBP, e eventualmente do AP da sala de exame para o acesso à Internet. O servidor de armazenamento do trabalho produzido pelos alunos pode residir tanto num ponto central fora das salas de exame, como na estação que criou a rede *ad hoc* para a distribuição da imagem Linux para o exame, que corre um servidor FBP. Em qualquer um dos dois casos anteriores, não podemos distinguir quais são os endereços IP ou endereços MAC que pertencem a máquinas que estão dentro ou fora da sala de exame.

O segundo problema (que é consequência do anterior) reside na possibilidade do aluno entregar a prova através de um sistema diferente do distribuído pelo docente. Nos exames com PC *desktop*, os docentes sabiam que sistemas estavam instalados nas máquinas. Ao usarmos os portáteis dos alunos, nada nos garante que estes usaram a imagem Linux reforçada criada para o exame.

O terceiro problema reside na definição das políticas de controlo de acesso a recursos não desejados durante a prova. O aluno pode aceder a material de apoio tanto através da rede, como através de dispositivos de armazenamento de dados (internos ou externos ao seu computador portátil). As políticas terão que ser exercidos pelo SO reforçado durante a prova.

O quarto problema é consequência do anterior. Com o bloqueio de toda a E/S dos computadores, o aluno não pode armazenar dados no seu computador. Portanto estes terão que ser obrigatoriamente armazenados num local central, e não só apenas na entrega final da prova. Se um PC *desktop* se desligar, é preciso apenas voltar a ligá-lo para retomar a prova. Nos portáteis dos alunos, a correr uma sessão Live de Linux, onde todo o trabalho é armazenado na RAM do computador, quando houver uma falha no mesmo todo o trabalho é perdido.

O último problema reside na possibilidade do aluno poder desligar o sistema a meio da prova, para ligar outro sistema que esteja instalado na sua máquina (onde pode consultar apontamentos ou outros recursos para beneficiar a sua prova) e voltar a arrancar o sistema do exame mais adiante.

Outros problemas envolvem a autenticação dos alunos e ligação à Internet com configuração mínima numa sessão Live de Linux.

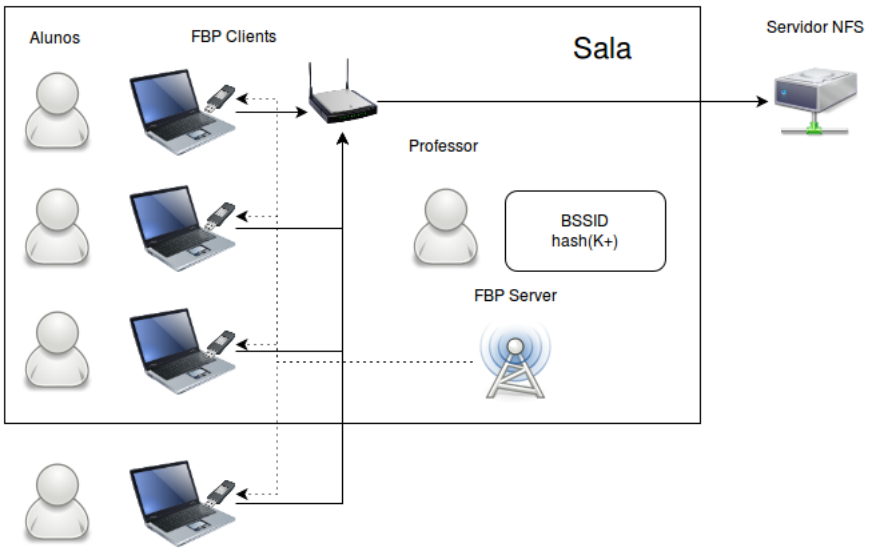

Figura 5.2: Exame com os portáteis dos alunos. Acesso aos servidor NFS (linha carregada), acesso ao servidor FBP (linha tracejada). O docente fornece o BSSID e a síntese da (K^+), como descrito na secção 3.3.2. Como a transmissão do SO através do protocolo FBP é feita em aberto, os alunos dentro e fora da sala de exame podem descarregar a imagem.

Para simplificar a descrição do texto restante, chamámos ao conjunto de entidades reguladoras pertencentes ao docente que controlam centralmente a prova o regulador da prova. Entendemos por conjunto de entidades: (i) o servidor FBP; (ii) o repositório de armazenamento (NFS ou outro); (iii) outros mecanismos a falar mais à frente.

5.3 Sistema operativo reforçado como raiz de confiança

Durante a prova, o SO reforçado é a raiz de confiança do regulador da prova nos computadores dos alunos. Portanto vai ser necessário identificar o SO a correr dentro das máquinas dos alunos. Numa prova, queremos que o aluno apenas possa entregar o trabalho que produziu a partir do SO reforçado. Para isso no repositório de entrega do exame o regulador tem que conseguir verificar a origem to trabalho entregue. Como o SO reforçado obtém controlo total da máquina do aluno, é neste que o regulador confia.

Para identificar o SO reforçado, é preciso que este possua algo que o diferencie de todos os outros. Uma solução imediata passa por adicionar-lhe um identificador, no fundo uma "impressão digital". Esta pode ser uma chave secreta simétrica que apenas as instâncias legítimas do SO reforçado e o docente conhecem. Desta forma, por exemplo, é possível o SO assinar todo o trabalho que o aluno entrega ao docente através do regulador da prova. Se a assinatura for válida é porque o trabalho foi entregue pelo SO reforçado. A chave deve estar armazenada num local que o utilizador do sistema (o aluno) não tenha permissão de leitura.

5.3.1 Navegador do exame como raiz de confiança

Através do mesmo princípio da "impressão digital" do SO, num exame online (ou seja realizado através de páginas da Web), o navegador pode identificar-se de forma a que os alunos fora da sala de exame ou que não usem o SO reforçado não estejam autorizados a aceder ao site da prova.

O protocolo Hyper Text Transfer Protocol Secure (HTTPS) ou HyperText Transfer Protocol (HTTP) sobre Secure Sockets Layer (SSL)/Transport Layer Security (TLS) permite a autenticação dos clientes para além da autenticação dos servidores. Estes usam autenticação assimétrica com certificados de chave pública X.509. Portanto o navegador pode transportar uma "impressão digital" sobre a forma de uma chave privada de um par de chaves assimétrica RSA que usa para se autenticar perante o servidor HTTPS do regulador da prova, que contém o certificado da chave pública do navegador.

Para facilitar a configuração do SO, a mesma chave assimétrica pode ser partilhada pelo próprio SO e pelo navegador. Desta forma, sejam quais forem as necessidades do docente, tanto o SO como o navegador podem usar a mesma chave para se identificarem.

5.4 Distinção dos alunos dentro e fora da sala do exame

Nos exames informáticos, os alunos realizam as provas de exame nos PC *desktop* instalados na sala de exame. Apenas nesses os alunos podem entregar a prova de exame depositando o trabalho que desenvolveram num repositório central do docente. Se o aluno sair da prova, não pode entregar mais o trabalho que produz, visto que a origem de entrega é filtrada através dos endereços IP das máquinas dentro da sala de exame. Por outro lado, o docente regista a saída do aluno, e mesmo que encontre trabalho produzido por este, anula a prova.

Com a realização da prova nos computadores portáteis dos alunos o mesmo já não acontece. Um aluno dentro da área de cobertura do servidor FBP, mesmo fora da sala de exame pode descarregar a imagem Linux difundida. Como vimos na Secção 5.3, o SO reforçado é a raiz de confiança do regulador da prova, mas se o aluno consegue usá-lo fora da sala de exame, pode ter acesso a outros recursos indesejados (colegas, livros, Internet através de outra máquina).

Apesar do aluno poder entregar a prova fora da sala de exame, como este não entrou e marcou a sua presença, ou então saiu a meio da prova, esta em princípio seria anulada. Só que, como veremos mais adiante (ver secção 5.6), não é possível realizar uma verdadeira autenticação do aluno: este pode dar as suas credenciais de acesso a um colega para realizar a prova por ele. Ou seja, um aluno dentro da sala de exame pode ter um cúmplice que está a fazer a prova em seu nome fora da sala.

5.4.1 Permissão de entrada no exame com um dispositivo de autorização

Seria desejável que a distinção dos computadores pessoais dos alunos que estão dentro e fora da sala de exame fosse tão automatizada o quanto possível. Os seres humanos distinguem o que é o interior e exterior da sala de exame através da sua perceção do mundo. Infelizmente um software que corre dentro dos computadores dos alunos, que não tem qualquer tipo de perceção sensorial, não consegue fazer essa distinção. Portanto, para garantir que apenas os alunos dentro da sala de exame podem realizar a prova, cada máquina deve ser autorizada por um agente humano, pois um software só por si não vai conseguir destingir se a máquina se encontra dentro ou fora da sala.

Na nossa primeira aproximação, o docente percorre as máquinas de todos os alunos e liga-as à vez a um dispositivo de autorização, que realiza autenticação mútua com o SO a correr na máquina do aluno. Se o SO reforçado comprovar a identidade do dispositivo do docente, é porque o docente classificou a máquina como estando dentro da sala de exame e o SO pode deixar o aluno pode entrar digitalmente na prova.

O dispositivo de autorização deve: (i) ser fácil de transportar; (ii) ser *writable-only*, ou seja, pode-se programar uma chave de identificação no dispositivo, mas esta não pode ser extraível; (iii) poder comunicar com os computadores dos alunos através de uma tecnologia padrão de E/S. Propomos um micro-controlador (μC) minimalista com uma interface USB para comunicar com o SO, e que tenha um conjunto de sensores que possam ser usado como fonte de ruído para gerar valores aleatórios.

O SO e o μC realizam autenticação mútua através de um protocolo de desafio-resposta. Este esquema é necessário para que o aluno não possa roubar a chave. Primeiro o SO autentica-se perante o dispositivo, para que o docente tenha a certeza que é o SO reforçado que está a correr na máquina. Depois o dispositivo autentica-se perante o SO reforçado. Se o SO verificar a identidade do dispositivo, sabe que o docente autorizou a realização da prova. O uso de chaves simétricas ou assimétricas é indiferente, pois o segredo é partilhado pelo μC e pelas várias instâncias do SO reforçado em execução nos sistemas computacionais dos alunos. O esquema encontra-se ilustrado na Figura 5.3.

5.4.2 Permissão de entrega da prova com um dispositivo de validação

O modelo anterior não resolve inteiramente o problema. O aluno a meio da prova pode decidir sair da sala de exame levando consigo o seu computador já autorizado. Mesmo que a sua prova seja anulada manualmente pelo docente, este pode manter o seu computador ligado e continuar o trabalho fora da sala de exame, no caso de se ter autenticado como um colega seu, e realiza a prova em nome desse colega.

A solução passa por inserir o dispositivo não durante o processo de autenticação do aluno (entrada no exame), mas sim durante a entrega final do trabalho que produziu. O dispositivo passa a fornecer um serviço de validação da entrega da prova e não de autorização de entrada

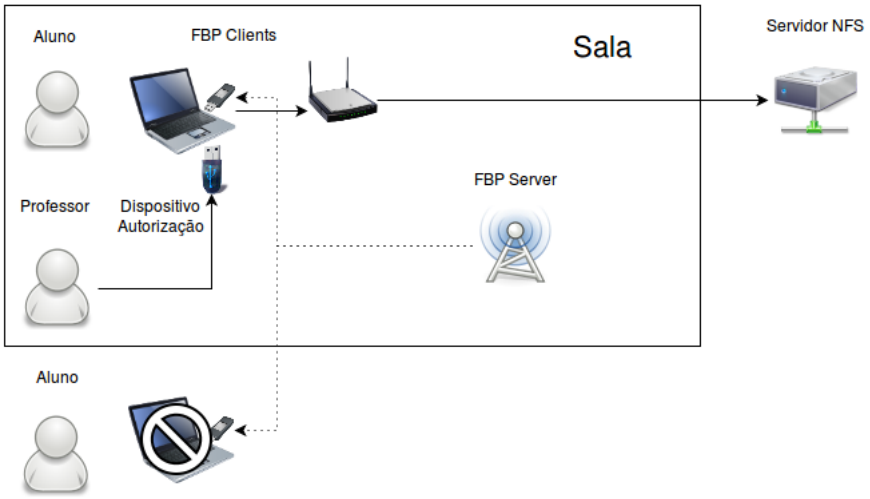

Figura 5.3: Dispositivo de autorização. O docente insere o dispositivo de autorização apenas nos computadores dentro da sala. Se o aluno está fora da sala de exame o SO bloqueia-lhe a entrada no exame.

---

na prova. O regulador da prova identifica não só que a fonte de origem do trabalho entregue é o SO reforçado, como também que este vem validado pelo dispositivo de validação do docente.

O aluno pode continuar a depositar trabalho que vai produzindo ao longo da prova fora da sala. Mas sem a entrega com validação, o trabalho não será avaliado pelo docente. Como precaução extra, após a entrega final com validação, após o SO ter confirmação a receção do trabalho por parte do regulador da prova, termina a sua execução.

5.4.3 Unicidade da sessão da prova

Infelizmente a validação da entrega da prova ainda não resolve inteiramente o problema. Dois alunos podem ainda entrar na prova usando as mesmas credenciais. Visto que o trabalho do aluno tem de ser salvaguardado centralmente porque não pode aceder aos seus dispositivos de armazenamento internos e externos, se os dois alunos partilharem uma conta, têm acesso ao mesmo conteúdo armazenado. Precisamos de garantir a unicidade das sessões de prova nos computadores dos alunos.

O regulador da prova confia no SO reforçado. O SO tem acesso a todos os dados da máquina onde se encontra a correr. O SO pode usar um dado estático do hardware para identificar a máquina onde está a correr. Desta forma pode associar a sessão de prova do aluno à maquina onde entrou na prova pela primeira vez. Ou seja, o email do aluno é vinculado à "impressão digital" da sua máquina. O aluno ao autenticar-se perante o regulador da prova, o regulador vincula o trabalho entregue pelo aluno à "impressão digital" do seu computador.

O valor usado deve ser estático e único no reino de todos os computadores portáteis, e o aluno não o deve poder alterar durante a prova através de instalação de novo hardware, como usar uma placa de rede externa, o que torna o uso do MAC da Network Interface Controller (NIC) uma fonte insegura. Um valor único que pode ser usado que o aluno não pode trocar a meio da prova é o uuid da motherboard, que no Linux pode ser obtido através do diretório `/sys/class/dmi/id/product_uuid`. Poderia até ser usada uma operação criptográfica de um

Trusted Platform Module (TPM). O TPM é um coprocessador de segurança que serve como raiz de confiança em vários protocolos de atestação. No fundo este funciona como um *smart card* embutido no hardware do sistema computacional. A correção desta estratégia deve-se ao facto de ser o SO a escolher os dados e não o aluno, e estes serem inalteráveis, e por outro lado temos a certeza que não estamos a correr sobre uma VM, onde os dados do hardware são virtuais e podem ser editados.

Resumindo, o regulador da prova associa cada computador a um aluno através de um identificador fixo e único do hardware (ou de uma operação criptográfica com um TPM), que é fornecido pelo SO reforçado. Se o aluno sair da sala o docente não valida a entrega da prova do aluno, e portanto não importa que este continue a realizar a prova fora da sala de exame. Nenhum aluno dentro da sala vai poder entrar com a mesma conta na prova e aceder ao trabalho do outro, porque na autenticação do aluno, o SO informa o regulador do valor único do hardware, e por sua vez o regulador da prova informa o SO que já existe uma conta associada ao aluno numa outra máquina.

5.4.4 Ligação à rede por cabo

Se o docente não estiver disposto a validar as máquinas dos alunos com o dispositivo de validação e se este dispõem de alguma infraestrutura na sala de exame para acesso à rede com ligação por cabo, a nossa segunda aproximação passa por usar os portáteis dos alunos como PC *desktop*. Ou seja, os alunos podem realizar a prova nos seus computadores pessoais e usarem uma interface de rede Ethernet para a ligação à Internet. As salas do DETI que possuem PC *desktop*, já estão devidamente equipadas com redes de acesso por cabo, tornando a migração fácil. Desta forma a distinção dos alunos dentro e fora da sala de exame é feita como nos PC *desktop*, através de filtragem dos endereços IP das salas. Tem a vantagem de não precisar da validação dos docentes, mas tem a desvantagem de requerer uma infraestrutura adicional. O esquema encontra-se ilustrado na Figura 5.4.

5.5 Monitorização dos portáteis dos alunos

Existindo uma forma de recuperação do trabalho realizado por um aluno, o mesmo poderia alternar entre o uso, no seu computador, do SO nativo do mesmo e do SO difundido. Para evitar que tal seja feito, é necessário assegurar que a falha na execução do sistema difundido seja detetada pelo docente logo após a sua ocorrência, e não apenas quando o aluno pretender recuperar trabalho salvaguardado.

A nossa proposta passa por um esquema de *heartbeat*, em que o SO corre um processo que envia, periodicamente, mensagens *keep alive* para o regulador da prova. Se o regulador detetar inatividade durante um certo intervalo de tempo alerta o administrador do sistema (o docente no cenário do exame). As mensagens *keep alive* são assinadas pelo SO e contêm um número de sequência para o atacante não poder usufruir de ataques por repetição, como ilustrado na Figura 5.5.

Se a máquina do aluno for reiniciada, o SO vai recomeçar a contagem do índice do *heartbeat*. Para que o regulador da prova possa verificar que uma nova contagem não se trata de um ataque de repetição, a mensagem contém um campo extra com um valor aleatório escolhido no início do *heartbeat* que permite destingir se estamos perante uma nova contagem. Naturalmente, todos os valores aleatórios usados anteriormente deverão ser memorizados pelo regulador da prova.

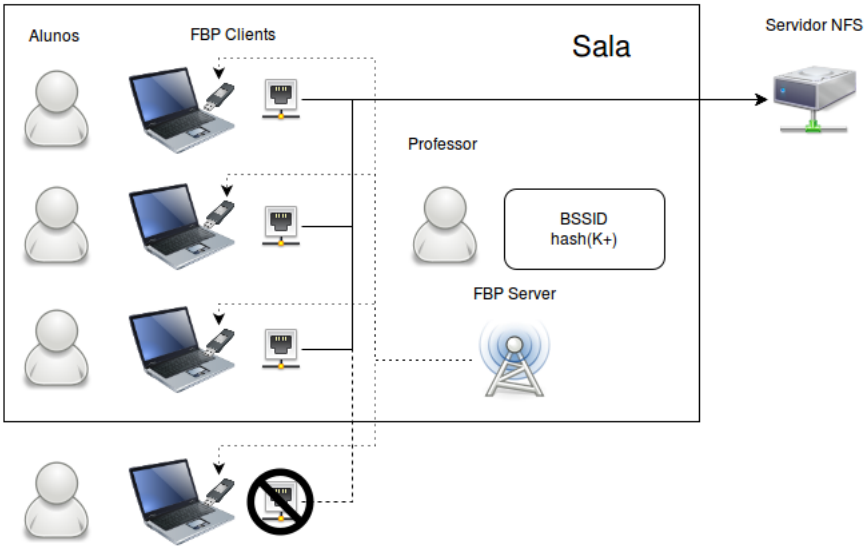

Figura 5.4: Exame com os portáteis dos alunos acedendo à rede por cabo. Os alunos fora da sala de exame podem ser filtrados por endereço IP como nos exames com PC *desktop*.

identificador da máquina/aluno
índice
valor aleatório
assinatura do SO reforçado

Figura 5.5: Mensagem heartbeat

5.6 Autenticação dos alunos

Como vimos na Secção 5.4, o aluno é associado à máquina onde entrou pela primeira vez, não podendo entrar mais noutras máquinas. Sem autenticação, um aluno que queira impedir a realização da prova a um colega pode entrar com a conta do primeiro caso não haja autenticação. Portanto a autenticação do aluno não só é necessária para a sua identificação nas entregas da prova, como é necessário para que um colega não o possa personificar e negar-lhe o acesso á prova, visto que a sessão de cada aluno é única.

Considerámos vantajoso o uso das credenciais universais da UA (email do aluno e senha) para a autenticação das provas de exame, porque é confortável para o aluno não ter de memorizar novas credenciais, e porque é vantajoso para o regulador da prova, porque diminui a sua complexidade, não sendo necessário implantar um sistema de informação para registar e armazenar os dados dos alunos para o âmbito de exames. As credenciais universais da UA são dados que os alunos usam com elevada frequência, justificando a opção. Contrariamente, credenciais dedicadas aos exames seriam usadas com baixa frequência, sendo fácil de os alunos as esquecerem. O docente, na preparação das provas, apenas tem que associar os endereços de email ao nome dos alunos das provas.

O regulador confia no SO do exame, que por sua vez confia no Identity Provider (IdP) da UA. Se a autenticação do aluno for válida, o SO reforçado regista no regulador da prova o du-

plo (email do aluno, "impressão digital" do hardware) assinado com a sua própria "impressão digital".

5.7 Ligação à Internet através da Eduroam

Quando um SO Linux é instalado pela primeira vez num computador, se este está ligado à Internet por cabo tem conectividade imediata. Caso contrário é preciso a pré-configuração da ligação. Normalmente a configuração é apenas precisa ser feita uma vez, ficando esta armazenada no computador do utilizador. Quando o aluno arranca uma sessão Live de Linux sem qualquer configuração prévia e onde não pode aceder os seus dispositivos de armazenamento (durante a prova) ele tem de ter a possibilidade de configurar a ligação à rede.

Atualmente o acesso à Internet na UA é feito através da Eduroam. As credenciais universais da UA são usadas também para a autenticação na Eduroam. No âmbito do projeto DETIboot foi desenvolvido anteriormente um novo módulo PAM (ver Secção 2.5.1) para minimizar a configuração necessária por parte do utilizador, o *pam_wpa*. Este conecta automaticamente o utilizador à rede durante a sua autenticação (não realiza propriamente operações no âmbito da autenticação). Visto que as credenciais universais da UA do aluno são usadas também para a ligação da Eduroam, podemos aproveitar os dados inseridos pelo aluno para simultaneamente ligá-lo à rede e autenticar o aluno, não tendo este que configurar mais tarde manualmente a ligação à rede. O *pam_wpa* faz uso do código fonte do comando `wpa_supplicant`, que inclui a implementação de um cliente 802.1X.

Após a ligação com a Eduroam, o endereço IP é configurado através de um cliente Dynamic Host Configuration Protocol (DHCP).

5.8 Imagem reforçada

Imagem reforçada é uma imagem que contém um conjunto de mecanismos e configurações que vão exercer políticas de confinamento sobre o utilizador. No exame estas têm que ser feitas ao nível da rede para impedir o acesso a sites na web não autorizados, a nível de E/S para impedir o acesso a dispositivos de armazenamento de dados e ao nível do congelamento de privilégios, de forma a que o aluno não possa ter privilégios que lhe permitam revogar as restantes configurações. A imagem reforçada traz instalada o VMHalt, executado em tempo de arranque e que verifica a presença de uma VM, e se for o caso para o arranque do SO.

5.8.1 Restrições de rede

Para a configuração do controlo da rede usamos o mecanismo *firewall* nativo do Linux, as IPTables (ver 2.5.2).

Como princípio base de segurança definimos como política base das IPTables a decisão DROP para todas as cadeias (*whitelist*). De forma a permitir que o aluno possa aceder a sites Web (exemplo, ao e-learning da UA), é permitido a entrada e saída de todo o tráfego HTTP/HTTPS para os endereços desejados pelos docentes. Caso exista backup remoto, é permitido tráfego de entrada e saída para o endereço do servidor de backup do docente através de um sistema de ficheiros de rede como o NFS. É também permitido o tráfego do protocolo de autenticação usado (Kerberos) para permitir a autenticação remota do aluno.

5.8.2 Restrições de E/S

O Linux é um SO modular. Este permite que facilmente sejam desenvolvidos e instalados (ou desinstalados) novos módulos no seu núcleo. Para bloquear todos os acessos relacionados com E/S optámos por remover os módulos de *device drivers* do Linux responsáveis pela comunicação com o exterior em tempo de arranque. Para tal é necessário a configuração base do ficheiro `/etc/modprobe.d/blacklist.conf` em que se listam os módulos que não pretendem instalar durante o arranque. Bloqueámos os seguintes módulos:

1. `usb_storage` - permite o acesso a dispositivos de armazenamento externos USB.
2. `bluetooth` - para impede a comunicação Bluetooth.
3. `ahci` - permite a comunicação SATA. Ao remover este módulo impedimos o acesso ao(s) HDD/SSD instalados no portátil do aluno e aos leitores CD/DVD/blu-ray e eventualmente de disquetes.
4. `floppy` - permite o acesso a leitores de disquetes que possam vir integradas no portátil do aluno. Alternativa ao `ahci` para impedir o acesso aos leitores de disquetes.
5. `isofs` - permite a leitura do formato ISO. Alternativa ao `ahci` para impedir o acesso aos leitores de CD/DVD/blu-ray.

Caso o docente deseje dar acesso a algum dos canais de comunicação é só adaptar a configuração. A vantagem desta abordagem é a flexibilidade e a generalidade que providencia.

A remoção do `usb_storage` não impede a comunicação série com μCs necessários para a realização de alguns exames práticos (como a unidade curricular de Arquitetura de Computadores II do DETI). Ao remover o `usb_storage` é impedido o `mount` de dispositivos manual ou automaticamente (carregamento automático de pen drives pelo gestor do ambiente gráfico).

5.8.3 Congelamento da elevação de privilégios

O modelo de controlo de acesso do Linux permite a definição das permissões a nível de leitura, escrita e execução. Também permite a promoção de privilégios durante o acesso ao ficheiro com o mecanismo Set-UID e Set-GID (ver Secção 2.5.3).

Numa prova de exame estes mecanismos podem permitir ao aluno a desativação das barreiras de proteção descritas até aqui. Portanto é necessário desativar o Set-UID e Get-UID em todos os ficheiros do SO durante a sua produção.

5.9 Problema em aberto – confidencialidade da imagem reforçada

O método de arranque do DETIboot foi construído e é operacionalizado com tecnologias padrão bem conhecidas (UEFI, GRUB, Casper). Por outro lado, a execução do *bootstrap system* do DETIboot é controlada por um guião *shell* com etapas bem definidas. Primeiro, o cliente FBP descarrega o núcleo e o *initramfs* do novo SO. No final da transferência é usado o comando `kexec` para reiniciar a máquina com o novo SO. Entre estes dois passos (descarga e arranque) é possível atacar a imagem reforçada.

5.9.1 Vulnerabilidades de segurança / modelo de ataque

O AFBP (ver Capítulo 3) evita os ataques possíveis durante a descarga da imagem Linux, permitindo ao aluno autenticar a imagem durante a descarga. Mas o AFBP não garante ao emissor que o aluno não modifica a imagem antes da executar. Portanto, seguidamente enunciamos os modelos de ataque que o aluno pode realizar na sua própria máquina para realizar o exame de forma ilícita.

Consideremos as seguintes entidades: o Servidor (**S**), uma máquina do docente que executa um servidor FBP; o Cliente (**C**), uma máquina do aluno com o *bootstrap system* do DETIboot (a executar um cliente FBP); e o Atacante (**A**), uma máquina do aluno a executar um servidor e cliente FBP. **A** poderá ser um dispositivo móvel de pequena dimensão (para passar despercebido pelos docentes) ou poderá ser uma máquina cúmplice fora da sala de exame.

S difunde a imagem reforçada $\mathcal{I}$. Tanto **C** como **A** podem descarregar $\mathcal{I}$. Desta forma o aluno (ou alguém em conluio) por usar **A** para "atacar" **C** (o seu computador portátil onde irá realizar a prova): só tem que receber $\mathcal{I}$ em **A**, descompactar $\mathcal{I}$, alterar $\mathcal{I}$ da forma mais conveniente, montar uma nova imagem $\mathcal{I}'$, eventualmente quase igual a $\mathcal{I}$, e difundir $\mathcal{I}'$ via FBP. Em **C** o aluno apenas tem que aceitar o seu próprio módulo da chave pública. Este ataque encontra-se ilustrado na figura 5.6.

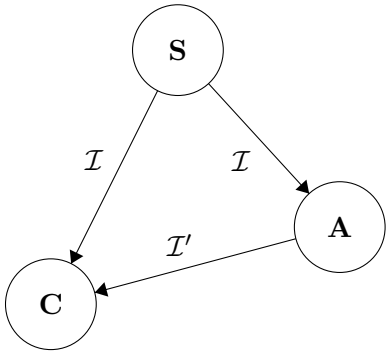

Figura 5.6: Ataque de manipulação da imagem $\mathcal{I}$. Todas as máquinas, clientes ou atacantes têm acesso à imagem difundida pelo servidor ($\mathcal{I}$). Um atacante **A** pode descarregar a imagem, alterá-la para $\mathcal{I}'$ e transmiti-la novamente, em difusão ou não. Se o utilizador possuir uma máquina atacante e cliente (**A** e **C**, respetivamente), pode transmitir a imagem manipulada $\mathcal{I}'$ da primeira para a segunda.

Para ajudar a ocultar a presença do seu servidor FBP em **A**, o aluno pode usar um canal 802.11 diferente do usado por **S** e transferir $\mathcal{I}'$ em *unicast* para o endereço Ethernet de **C**. Como o BSSID é gerado pelo servidor FBP, o aluno apenas tem que configurar o BSSID em **C** de acordo com o esquema descrito no Capítulo 3.

Uma situação idêntica é ter **C**=**A**, ou seja o cliente e o atacante são a mesma máquina. Desta forma a presença de **A** na rede é completamente ocultada. O DETIboot faz uso do comando `kexec` para arrancar o novo núcleo. Portanto o aluno pode descarregar a imagem $\mathcal{I}$ com o um cliente FBP, alterá-la para $\mathcal{I}'$ e executá-la com o comando `kexec` na mesma máquina, sem ter necessidade de fazer as sua própria difusão.

Mesmo que **C** possua um segredo partilhado com **S**, este pode ser facilmente comprometido pelo o aluno. Se o *bootstrap system* do DETIboot tiver embebida uma chave pública de validação da imagem, esta pode ser trocada pelo aluno para corresponder à chave privada

que usou para assinar a imagem modificada. Relembramos que o *bootstrap system* encontra-se num dispositivo genérico de armazenamento USB.

Até aqui abordámos apenas a questão da modificação da imagem do SO por parte do aluno. Mas, na realidade, o aluno pode atacar a imagem segundo duas linhas de ação distintas:

1. **Personificar o SO**. Como veremos mais à frente (ver Capítulo 5), o aluno apenas deverá realizar/entregar a prova através do SO distribuído pelo docente. Para isso a imagem do SO terá que transportar consigo uma "impressão digital" que a permita destingir de todas as outras imagens. Desta forma o SO consegue identificar-se com a "impressão digital" perante o repositório de entrega. Se o aluno conseguir extrair esse identificador mais as ferramentas para a realização do exame, pode realizar o exame noutro sistema.

2. **Modificar o SO**. Enquanto por arrancar, o SO não detém qualquer controlo sobre a máquina do aluno. Se o aluno conhecer a estrutura do SO pode realizar todas as modificações que pretende sobre a imagem antes de o executar de forma a remover as barreiras que exerce.

Para impedir a personificação e modificação do SO, temos que garantir a confidencialidade deste até o aluno o executar. Se a imagem do SO for difundida cifrada, com uma chave conhecida só pelo docente, então o aluno não tem acesso a esta até que lhe seja fornecida pelo docente. Só que para decifrar a imagem, a chave vai ter que estar eventualmente presente em aberto no computador do aluno, e portanto há também inúmeras maneiras do aluno obter a posse desta. Pior ainda, a chave serve não só para um aluno mas também para todos os seus colegas, visto que todos receberam o mesmo ficheiro por difusão através do DETIboot.

Este último aspeto, embora tenha sido estudado e até concebido um protocolo de atestação remota de software para o DETIboot, não é abordado nesta dissertação porque não cumpria todos os requisitos de segurança impostos, o que não excluí trabalho futuro.

Enquanto não encontrar-mos uma solução para o problema anterior, podem ser os docentes a arrancar as máquinas dos alunos. As suas *pens bootable* devem conter uma chave e um mecanismo de decifra para que a difusão seja confidencial.

5.10 Conclusões

Neste capítulo mostrámos que a realização de exames nos portáteis pessoais dos alunos é possível e viável, muito embora ainda subsistam alguns aspetos que gostaríamos de ver mais bem resolvidos. Nos exames com PC *desktop* o docente fornece a entrada do exame através do nome de uma conta e uma senha escritas no quadro da sala. Com o DETIboot escreve o BSSID e a síntese do módulo público da sessão FBP. Nos exames tradicionais o docente passa pelas mesas de forma a registar os alunos, onde estes assinam as folhas. Com o DETIboot circula na mesma pelas mesas dos alunos mas para inserir o dispositivo de validação da entrega final do trabalho que os alunos produziram.

Apresentámos um conjunto de políticas e configurações que permitem a construção de uma imagem Linux reforçada para a realização de uma prova de exame, através: (i) do bloqueio de *device drivers*, (ii) de configurações de *firewall* e (iii) da desativação de mecanismos de promoção de privilégios. O objetivo é bloquear o acesso aos recursos indesejados por parte o aluno durante a prova.

O grande desafio foi distinguir os alunos dentro e fora da sala de exame. Com o acesso à Internet através de uma rede sem fios, os alunos podem sair da sala e continuar a fazer a prova. A solução do dispositivo de validação cumpre os requisitos de segurança exigidos. A alternativa ao dispositivo passa pelo uso de uma rede de ligação por cabo previamente montada, em que os alunos apenas podem entregar a prova a partir das interfaces de rede disponibilizadas na sala. É uma solução viável, visto que a maior parte das salas do DETI com equipamento fixo já possui a infraestrutura montada necessária.

Por último o docente consegue monitorizar a atividade dos computadores dos alunos, e saber se estes estão a correr a imagem reforçada nas suas máquinas através de mensagem periódicas enviadas ao regulador da prova.

Ficamos apenas com uma questão em aberto, que é ainda o nosso maior desafio: como manter a imagem confidencial até esta ser executada pelo aluno?

Capítulo 6

Conclusão

O principal objetivo desta dissertação passava pelo desenvolvimento de um conjunto de mecanismos que estendessem o sistema DETIboot, de forma a que este pudesse ser explorado no âmbito de exames usando os computadores dos alunos, criando assim um cenário equivalente aos dos exames com PC *desktop*.

Concebemos o DETIexam, um ecossistema para a realização dos exames usando os computadores pessoais dos alunos. Este protege a prova de exame a vários níveis. Com o reforço da imagem distribuída pelo DETIboot, que ao tomar controlo do computador do aluno, impede o acesso do mesmo a recursos indesejáveis na rede e de dispositivos de E/S. Impede também a entrega da prova fora da sala de exame através da singularidade das sessões de prova; desta forma dois alunos não podem entram no exame em nome do mesmo. E também através de um dispositivo de validação, permite ao docente "marcar" as máquinas que estão dentro da sala. Por último, impede o aluno de trocar temporariamente de SO através de monitorização com mensagens de *heartbeat*, alertando o docente de qualquer inatividade das máquinas. Desta forma os alunos não podem consultar material não autorizado usando o sistema nativo da sua máquina.

Com o VMHalt alcançámos o arranque seguro, sobre uma máquina real, num exame usando o DETIboot. O mesmo impede que um aluno execute sobre uma VM, em paralelo ao seu sistema nativo, a imagem reforçada para o exame. A nossa solução usa uma série de testes de deteção de virtualização. Se um dos testes classificar o sistema como virtual, paramos o arranque do SO. Os resultados foram muito satisfatórios, conseguimos classificar corretamente um sistema como real e como virtual com 100% de precisão.

Com o AFBP alcançámos a transferência segura no DETIboot. Protegemos a integridade da imagem difundida, permitindo que os clientes FBP a possam validar. A solução envolveu a inclusão de um novo tipo de mensagem autenticadora que protege as próximas *codewords* por enviar. Estas são assinadas pelo emissor FBP. Para validar as *codewords* os autenticadores transportam a chave pública do emissor. No ambiente operacional de um exame, o professor pode escrever a síntese da chave no quadro da sala de aula. O aluno só tem que confrontar o valor impresso no seu ecrã com o do quadro. O AFBP penaliza o tempo de transferência de uma imagem de SO em aproximadamente 5% em relação ao protocolo base. Esta contribuição foi apresentada num artigo publicado num fórum internacional [21].

Em termos de usabilidade do sistema, é necessário apenas que o docentes(s) escrevam o BSSID da rede *ad hoc*, mais a síntese da chave pública da difusão AFBP, e que circulem no final das provas inserido o dispositivo de validação das provas nos computadores dos alunos.

Em contraste aos exames com PC *desktop*, o professor também escrevia no quadro o nome de utilizador universal e a password para aceder à sessão da prova. Por parte dos alunos, estes têm apenas que selecionar a rede e a chave pública que pretendem usar na sessão FBP.

Para além de resolver o problema proposto, a proteção dos exames, conseguimos alcançar a utilização segura do DETIboot em outros cenários de uso como aulas, laboratório, montras de lojas ou mesmo *workshops*, através da verificação de integridade da imagem com o AFBP, contribuindo assim para além do que nos propusemos a resolver.

Esta dissertação foi extremamente enriquecedora em termos de conhecimento, mais concretamente nas áreas de segurança em comunicação por difusão, *network coding*, atestação de software, máquinas virtuais e tópicos diversos de segurança em sistemas operativos Linux, bem como em experiência em investigação científica e também na conceção e desenvolvimento de sistemas de segurança complexos.

Considero que o trabalho desenvolvido nesta dissertação ofereceu o seu contributo científico, e penso que permitiu avançar o projeto DETIboot para o próximo nível. Penso que tanto o trabalho desenvolvido nesta dissertação como todo o projeto DETIboot é deveras original e inovador, e que vai contribuir para a resolução de problemas reais em ambientes académicos e empresariais.

6.1 Trabalho Futuro

Ainda existe trabalho que ficou por realizar no âmbito da exploração do DETIboot no âmbito de exames.

O primeiro passa pelo desenvolvimento de uma ferramenta de preparação automática dos exames. Esta deve gerar e configurar todas as ferramentas necessárias à realização da prova: imagem reforçada, mais todos os elementos do regulador da prova: (i) o repositório de entrega do trabalho produzido pelos alunos; (ii) cliente e servidor da monitorização com mensagens *heartbeat*; (iii) autenticação com o IdP da UA. O docente deverá poder configurar as políticas que pretende exercer na imagem e adicionar todas as ferramentas de apoio que desejar. O repositório deverá não só poder armazenar o trabalho dos alunos, mas também realizar as operações de autenticação, registo e anulamento das provas dos alunos.

Falta concretizar um dispositivo de hardware dedicado que cumpra os requisitos do dispositivo de validação. Este vai ser de tamanho reduzido e ter uma fonte de ruído que o permita gerar os *nonces* para a autenticação com desafio-resposta. Testámos que tal é possível com um módulo Analog to Digital Converter (ADC) em contacto com uma mão humana, mas outros sensores podem ser usado como de temperatura ou intensidade luminosa. Por último deverá permitir programar automaticamente a chave de identificação do dispositivo.

Por último, é preciso procurar soluções para o problema em aberto da confidencialidade da imagem reforçada. Recomendamos que a procura comece pelo estudo de atestação remota de software (de preferência baseada em software), esquemas de Digital Rights Managment (DRM) e ofuscação de código.

Bibliografia

[1] Manoj B. Athreya. Subverting Linux on-the-fly using Hardware Virtualization Technology. Master's thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology, United States, 2010.

[2] Edgar Barbosa. Detection of hardware virtualization rootkits. SyScan, 2007.

[3] Elaine B. Barker and Allen L. Roginsky. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. NIST SP - 800-131A, 2011.

[4] J.-M. Bohli, A. Hessler, O. Ugus, and D. Westhoff. Security enhanced multi-hop over the air reprogramming with fountain codes. In *IEEE 34th Conference on Local Computer Networks (LCN 2009)*, pages 850–857, Oct 2009.

[5] J.W. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1528–1540, 2002.

[6] João Cardoso. DETIboot: distribuição e arranque de sistemas Linux com redes WiFi. Master's thesis, University of Aveiro, Portugal, 2013.

[7] Chad Link Christopher Thompson, Maria Huntley. Virtualization detection: New strategies and their effectivness.

[8] Carlos Faneca, José Vieira, and André Zúquete. Fast image file distribution with Fountain Codes via a Wi-Fi Ad-Hoc network, using low power processors. In *16th Int. Telecommunications Network Strategy and Planning Symposium (NETWORKS 2014)*, Funchal, Madeira, Portugal, September 2014.

[9] Carlos Faneca, José Vieira, André Zúquete, and João Cardoso. DETIboot: A fast, wireless system to install operating systems on students laptops. In *2nd Int. Conf. on Advances in Computing, Electronics and Communication (ACEC 2014)*, Zurich, Switzerland, October 2014.

[10] Hagen Fritsch. Analysis and detection of virtualization-based rootkits. Master's thesis, Fakultät Für Informatik, Technische Universität München, Germany, 2008.

[11] Mohamad Gebai. Linux: virtualization and tracing, 2013.

[12] Shay Gueron and Jean-Pierre Seifert. On the Impossibility of Detecting Virtual Machine Monitors. In *24th IFIP TC 11 International Information Security Conference, SEC 2009, Pafos, Cyprus, May 18–20, 2009. Proceedings*, pages 143–151, May 2009.

[13] IEEE Std 802.11e. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 8: Medium Access Control (MAC) Enhancements for Quality of Service (QoS), 2005.

[14] D. J. C MacKay. Fountain codes. *IEEE Proceedings Communications*, 152(6):1062–1068, 2005.

[15] Alfredo Andrés Omella. Methods for virtual machine detection. 2006.

[16] Jung Min Park, Edwin K. P. Chong, and Howard Jay Siegel. Efficient Multicast Packet Authentication Using Signature Amortization. In *Proc. of IEEE Symposium on Security and Privacy*, Washington, DC, USA, 2002.

[17] A. Perrig, R. Canetti, J.D. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[18] Adrian Perrig and J. D. Tygar. *Secure Broadcast Communication in Wired and Wireless Networks*. Springer Science & Business Media, 2003.

[19] B. Preneel. Hash Functions and MAC Algorithms Based on Block Ciphers. In Michael Darnell, editor, *6th IMA Int. Conf. on Cryptography and Coding (LNCS 1355)*, Berlin, December 1997. Springer-Verlag.

[20] Danny Quist and Val Smith. Detecting the presence of virtual machines using the local data table.

[21] Simão Reis, André Zúquete, Carlos Faneca, and José M. N. Vieira. Authenticated file broadcast protocol. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 237–251. Springer, 2015.

[22] Christopher Kruegel Thomas Raffetseder and Engin Kirda. Detecting System Emulators. In *ISC'07 Proceedings of the 10th international conference on Information Security*, pages 1–18, October 2007.

[23] vmware. White paper: Understanding full virtualization, paravirtualization, and hardware assist. Technical report.

[24] Chung Kei Wong and Simon S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, August 1999.

[25] A.A. Yavuz. An Efficient Real-Time Broadcast Authentication Scheme for Command and Control Messages. *IEEE Transactions on Information Forensics and Security*, 9(10):1733–1742, Oct 2014.

[26] Matías Zabaljáuregui. Hardware assisted virtualization intel virtualization technology. 2008.

[27] André Zúquete. *Segurança em redes Informáticas, 4ª Edição*. FCA - Editora de Informática, Lda., 2013.