UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Final

CyberSentinel
Provedor de CSIRT Acessível

Autor:
Henrique Rotsen Santos Ferreira

Orientador:
Antônio Alfredo Ferreira Loureiro

Email: henriqueferreira@dcc.ufmg.br

Belo Horizonte, Minas Gerais
1 de dezembro de 2025

Sumário

Resumo

Este relatório apresenta uma metodologia prática e faseada para a implementação de um Computer Security Incident Response Team (CSIRT) eficaz e de baixo custo, denominado "CSIRT Acessível". A proposta se fundamenta na utilização estratégica de um conjunto de ferramentas de código aberto, integradas e orquestradas para fornecer capacidades robustas de resposta a incidentes de segurança.

A solução arquitetural proposta é composta por Wazuh para monitoramento de segurança (SIEM/XDR), TheHIVE para gestão de casos e incidentes (SIRP), Cortex como motor de análise e enriquecimento de observáveis, MISP como plataforma de inteligência sobre ameaças e Shuffle como a plataforma de orquestração e automação (SOAR) que integra todo o ecossistema.

O objetivo é demonstrar um caminho viável para que organizações, incluindo pequenas e médias empresas (PMEs) com recursos limitados, possam construir uma função de segurança resiliente. A metodologia detalha desde o planejamento estratégico e a seleção de ferramentas até a implementação, a automação de fluxos de trabalho e a maturação contínua das operações, visando transformar a segurança de uma função puramente reativa para uma postura estratégica e proativa.

Por fim, é apresentada uma Prova de Conceito (PoC) realizada em parceria com uma empresa real, detalhando a arquitetura utilizada, as configurações técnicas dos componentes (com destaque para Wazuh, TheHive, Cortex e Shuffle) e as principais dificuldades encontradas, tanto técnicas quanto de licenciamento e uso das ferramentas.

1 Introdução

1.1 Caracterização do Problema e Motivação

O cenário contemporâneo de cibersegurança é marcado por um aumento contínuo no volume, na sofisticação e no impacto dos ciberataques. Tais ameaças podem causar danos severos às organizações, incluindo perdas financeiras, prejuízos à reputação, vazamento de informações sensíveis e interrupção das operações. Neste contexto, a implementação de uma Equipe de Resposta a Incidentes de Segurança Computacional (Computer Security Incident Response Team – CSIRT) tornou-se um componente indispensável para uma postura de segurança resiliente. Um CSIRT oferece uma estrutura organizada e processos definidos para gerenciar incidentes, mitigar seus impactos e restaurar a normalidade o mais rápido possível [1, 2].

Contudo, a criação e manutenção de um CSIRT são tradicionalmente associadas a altos custos, tanto de aquisição de ferramentas quanto de contratação de pessoal especializado, tornando essa iniciativa um desafio para organizações com orçamentos limitados, como pequenas e médias empresas (PMEs). A crescente democratização de ferramentas de ataque significa que mesmo organizações menores são alvos viáveis, o que acentua a necessidade de soluções de segurança que sejam não apenas eficazes, mas também financeiramente acessíveis.

A motivação deste trabalho é o conceito de "CSIRT Acessível", que propõe que uma capacidade robusta de resposta a incidentes não deve ser um privilégio de grandes corporações. Esta acessibilidade é alcançada através do uso estratégico de software de código aberto (open source) e de serviços gratuitos ou de baixo custo em nuvem. Agências governamentais, como a CISA, inclusive, catalogam diversas ferramentas gratuitas para este fim [3]. O desafio, portanto, não é apenas tecnológico, mas também metodológico: como combinar essas ferramentas de forma coerente, segura e sustentável, dentro das restrições típicas de uma PME.

1.2 Objetivos

O principal objetivo deste relatório é apresentar o caminho prático e a metodologia detalhada para a implementação de um CSIRT acessível e eficiente. Para isso, os seguintes objetivos específicos são traçados:

- **Apresentar um *stack* tecnológico sinérgico:** Detalhar a função de cada uma das cinco ferramentas de código aberto escolhidas: Wazuh[4], Shuffle[5], TheHIVE[6], Cortex[7] e MISP[8], e como elas se complementam para fornecer capacidades de monitoramento, análise, gestão de incidentes, compartilhamento de inteligência e automação.
- **Propor uma metodologia de implementação faseada:** Oferecer um guia estruturado em quatro fases: Planejamento, Implementação, Integração e Automação, e Operacionalização e Maturação, permitindo que organizações caminhem gradualmente em direção a um CSIRT completo.
- **Demonstrar a integração e automação:** Ilustrar como a plataforma SOAR (Shuffle) atua como o componente central que orquestra as demais ferramentas, automatizando fluxos de trabalho (*playbooks*) e multiplicando a força da equipe de segurança.

- **Montar o modelo em um ambiente real:** Apresentar uma Prova de Conceito (PoC) da implementação desse modelo em um ambiente real de uma empresa, descrevendo a arquitetura, as configurações, as restrições de infraestrutura e as dificuldades práticas encontradas.

1.3 Estrutura do Trabalho

Este relatório está estruturado da seguinte forma: o Capítulo 2 estabelece o referencial teórico, definindo o que é um CSIRT segundo frameworks consagrados e discutindo o conceito de acessibilidade através do open source. O Capítulo 3 descreve em detalhe o guia de implementação do *stack* tecnológico e discute o encadeamento de serviços entre as ferramentas. O Capítulo 4 apresenta a implementação da PoC e aprofunda os aspectos práticos de configuração dos componentes Wazuh, TheHive, Cortex e Shuffle na infraestrutura da empresa. O Capítulo 5 apresenta as conclusões do trabalho, limitações identificadas e recomendações para trabalhos futuros e para a sustentabilidade do CSIRT a longo prazo. Por fim, são listadas as referências bibliográficas utilizadas.

2 Referencial Teórico

2.1 Vertentes de Serviços de um CSIRT

A criação de um CSIRT, mesmo um com foco em acessibilidade, deve ser pautada por boas práticas estabelecidas pela comunidade de segurança. O **FIRST (Forum of Incident Response and Security Teams) CSIRT Services Framework** é um documento de referência que descreve os serviços e funções que um CSIRT pode oferecer ao longo de seu ciclo de vida. As principais áreas de serviço definidas pelo framework e atendidas, total ou parcialmente, pela solução proposta podem ser vistas como [9]:

- **Serviços Reativos:** Focam na identificação, análise e tratamento de incidentes de segurança já em curso ou recém-detectados, incluindo investigação, contenção, erradicação, recuperação e lições aprendidas. Nessa vertente, Wazuh, TheHive, Cortex e Shuffle atuam de forma integrada: o Wazuh gera alertas, o TheHive estrutura esses alertas em casos, o Cortex enriquece observáveis (IPs, domínios, hashes) e o Shuffle automatiza ações de contenção.
- **Serviços Proativos:** Incluem a descoberta, análise e tratamento de vulnerabilidades nos ativos da organização, bem como a disseminação de boas práticas e recomendações de mitigação. O módulo de detecção de vulnerabilidades do Wazuh[4] contribui diretamente para este serviço, enquanto a integração planejada com o MISP permite incorporar inteligência de ameaças e indicadores de compromisso (IoCs) de fontes externas.
- **Serviços de Gerenciamento da Qualidade de Segurança:** Esta vertente engloba serviços que asseguram e melhoram a qualidade geral da postura de segurança da organização, indo além da resposta a incidentes individuais. Inclui revisão de políticas, auditorias, métricas de desempenho, conformidade e lições aprendidas, que podem ser apoiadas por relatórios e *dashboards* construídos a partir dos dados gerados pelo Wazuh, TheHive e MISP.

2.2 O Conceito de "CSIRT Acessível" via Open Source

O "CSIRT Acessível" se baseia na premissa de que a capacidade de resposta a incidentes não deve ser exclusiva de organizações com grandes recursos. A acessibilidade, neste contexto, possui múltiplas dimensões:

- **Custo-efetividade:** O principal caminho para a custo-efetividade é a utilização de software open source, que elimina os altos custos iniciais de licenciamento de ferramentas comerciais e, muitas vezes, permite começar com infraestrutura modesta, ampliando-a conforme a maturidade e a necessidade.
- **Implementabilidade:** Refere-se à viabilidade de implementar e manter o CSIRT com os recursos humanos e materiais disponíveis. Isso é facilitado por ferramentas com documentação robusta, comunidades de suporte ativas e, quando possível, imagens de contêiner prontas para uso (*Docker images*).
- **Flexibilidade e Personalização:** A natureza do open source permite que as ferramentas sejam modificadas para se adequarem aos processos específicos de uma organização, em vez de forçá-la a se adaptar a uma solução proprietária rígida. Por exemplo, no caso da PoC, foi possível ajustar o código e a configuração de *analyzers* do Cortex e de *apps* do Shuffle para necessidades específicas.

- **Transparência:** O acesso ao código-fonte permite a verificação da ausência de vulnerabilidades ocultas e comportamentos indesejados, o que é vital para garantir a integridade das ferramentas de segurança.

Contudo, a abordagem open source apresenta uma dualidade. A redução de custos diretos de software é contrabalançada pela maior exigência de conhecimento técnico interno para configuração, integração e manutenção contínua. A ausência de suporte comercial dedicado significa que a responsabilidade pela resolução de problemas recai sobre a equipe interna, que precisa lidar com questões de *tuning*, escalabilidade, atualizações de versão e mudanças de modelo de licenciamento (como a transição parcial para modelos comerciais observada em TheHive e Shuffle). Portanto, "acessível" não significa "isento de esforço", mas sim uma realocação de investimento de capital financeiro para capital humano e tempo.

3 Guia de Implementação

Esta seção apresenta uma das principais contribuições do trabalho: um guia prático para implementação de um CSIRT acessível. O guia completo foi organizado em um documento à parte, acessível em: CyberSentinel – Guia de Implementação. A seguir, sintetizam-se as principais fases deste guia e o papel de cada ferramenta no *stack*.

3.1 Fases da Metodologia

A metodologia proposta é composta por quatro fases principais:

1. **Fase 1 – Planejamento e Definição de Escopo:** Nesta etapa são definidos o escopo de atuação do CSIRT, os serviços prioritários (reativos, proativos, de qualidade), os tipos de ativos a serem monitorados, a criticidade dos sistemas e as restrições de orçamento e de pessoal. Também são mapeadas as integrações existentes (por exemplo, firewalls, proxies, sistemas de autenticação) que podem gerar eventos de segurança para o Wazuh.

2. **Fase 2 – Implementação Básica das Ferramentas:** Inclui a instalação e configuração inicial de Wazuh, TheHive, Cortex, MISP e Shuffle. Prioriza-se o estabelecimento de um fluxo mínimo de alerta → caso → resposta manual, antes da automação avançada.

3. **Fase 3 – Integração e Automação:** Concentra-se na integração entre as ferramentas via APIs, webhooks e conectores específicos. Nesta fase, o Shuffle é configurado para consumir alertas (por exemplo, do Wazuh), criar ou atualizar casos no TheHive, acionar *analyzers* no Cortex e compartilhar IoCs com o MISP. Também são desenvolvidos os primeiros *playbooks* de automação.

4. **Fase 4 – Operacionalização e Maturação:** Foca na monitoração contínua, ajuste de regras, redução de falsos positivos, melhoria dos *playbooks*, definição de métricas de desempenho (SLAs/SLOs) e na documentação de processos. É nesta fase que o CSIRT começa a se consolidar como uma função permanente da organização.

3.2 Matriz de Integração entre Ferramentas

Para clarear ainda mais os papéis de cada ferramenta, a Tabela 1 resume o fluxo de dados principal entre os componentes. É importante destacar que essa matriz não esgota todas as possibilidades de integração, mas evidencia o fluxo mínimo viável para um CSIRT acessível.

Tabela 1: Matriz de Fluxo de Dados de Integração de Ferramentas

Origem	Destino	Dados Transmitidos	Propósito	Configuração Chave
Wazuh	Shuffle	Alerta em formato JSON	Ingestão de alertas para automação	Webhook ou API do Shuffle
Shuffle	TheHive	Dados do caso/alerta	Criação e atualização de casos	API Key do TheHive e URL do serviço
TheHive (via Shuffle)	Cortex	Observáveis (IP, URL, hash etc.)	Análise e enriquecimento de evidências	API Key e URL do Cortex
Cortex (via Shuffle/TheHive)	TheHive	Relatórios e artefatos de análise	Atualização do caso com resultados	Configuração de analisadores e conectores
TheHive (via Shuffle)	MISP	Observáveis (IoCs)	Compartilhamento e consulta de TI	API Key do MISP e *events*
Shuffle	Ferramentas de resposta	Comandos (ex.: bloquear IP)	Execução de ações de contenção	APIs de firewalls, EDRs, proxies, etc.

Na Prova de Conceito descrita no Capítulo 4, o foco recaiu sobre o fluxo Wazuh → TheHive → Cortex → Shuffle, sendo a integração com MISP apontada como etapa futura de amadurecimento do ambiente.

4 Prova de Conceito (PoC)

Esta seção descreve a Prova de Conceito (Proof of Concept – PoC) realizada para validar, em um ambiente real, a metodologia de um CSIRT acessível proposta neste trabalho. A PoC foi desenvolvida em parceria com a empresa RadioMemory, que dispõe de infraestrutura baseada em serviços da Amazon Web Services (AWS) e políticas rígidas de segurança de rede. O objetivo principal foi demonstrar a viabilidade de implantar, com ferramentas majoritariamente open-source e de baixo custo, um conjunto mínimo de capacidades de detecção, orquestração e resposta a incidentes de segurança integrado ao ambiente corporativo existente.

A PoC teve como foco a integração de quatro componentes centrais: o Wazuh, como plataforma de SIEM/SOC; o TheHive, como solução de gestão de incidentes; o Cortex, como mecanismo de enriquecimento de evidências; e o Shuffle, como plataforma de automação e orquestração (SOAR). A combinação dessas ferramentas permitiu simular o funcionamento de um CSIRT em condições próximas ao uso produtivo, respeitando restrições técnicas e de segurança da infraestrutura da RadioMemory.

4.1 Objetivos e escopo da PoC

A PoC foi planejada com os seguintes objetivos específicos:

- Verificar, na prática, se a metodologia de CSIRT acessível pode ser aplicada em um ambiente corporativo baseado em AWS, com restrições reais de rede e segurança.
- Avaliar o nível de esforço necessário para integrar ferramentas open-source de monitoramento, gestão e automação de incidentes, incluindo ajustes finos de configuração e de infraestrutura.
- Identificar limitações técnicas, de *performance* e de licenciamento das soluções utilizadas, principalmente no contexto de versões gratuitas ou comunitárias, e como essas limitações impactam um CSIRT acessível.
- Levantar desafios recorrentes na implantação de um CSIRT com base em ferramentas open-source, gerando insumos para recomendações no capítulo de conclusões.

O escopo da PoC foi propositalmente limitado a um conjunto controlado de casos de uso de segurança, que envolviam: geração de alertas, ingestão desses eventos pelo Wazuh, criação de casos de incidente no TheHive, enriquecimento automatizado via Cortex e acionamento de *playbooks* de automação no Shuffle. Assim, não se tratou de uma implantação completa de CSIRT em produção, mas de um ambiente de testes suficientemente realista para validar a metodologia e evidenciar pontos de atenção.

4.2 Arquitetura geral da PoC

A arquitetura da PoC foi desenhada para respeitar as diretrizes de segurança de rede da RadioMemory, em especial as políticas de *security groups* da AWS e a separação entre recursos locais e de nuvem. De forma resumida:

- O **Wazuh** foi instalado em um servidor local (*on-premises*), em vez de ser executado diretamente em instâncias na nuvem. Essa decisão facilitou o controle do ambiente de testes, reduziu a exposição do gerenciador de eventos à internet pública e atendeu à exigência da empresa de manter a coleta e correlação de eventos sob maior controle.
- As ferramentas **TheHive** e **Cortex** foram executadas em contêineres Docker em ambiente local, utilizando um *docker-compose* dedicado, com rede isolada (`thehive-cortex-network`), volumes persistentes para dados e *logs* e limites explícitos de memória. Um contêiner Nginx adicional atuou como *reverse proxy* HTTPS, concentrando o acesso externo em uma única porta.
- A plataforma **Shuffle** foi a única componente executada em uma instância AWS dedicada, do tipo `c6a.4xlarge`, devido ao alto consumo de memória e à lentidão extrema observada quando executada em recursos mais modestos. Essa instância foi utilizada essencialmente para suportar o backend do Shuffle e seu banco de dados baseado em OpenSearch.

Do ponto de vista de recursos, o arquivo `docker-compose` do TheHive e do Cortex define limites de aproximadamente 2 GB de memória para cada serviço principal (Cassandra, Elasticsearch, TheHive e Cortex) e 512 MB para o Nginx reverso, por meio da diretiva `deploy.resources.limits.memory` e da configuração `memswap_limit`. Esse limite protege a máquina hospedeira contra consumo excessivo de memória por um único serviço.

No caso do Shuffle, o componente de banco de dados (*OpenSearch*) foi configurado com `OPENSEARCH_JAVA_OPTS=-Xms8g -Xmx8g`, reservando 8 GB de *heap* Java. Somando-se a isso o backend, o frontend, o Orborus e os contêineres de *workers* que são criados sob demanda para execução de *apps*, torna-se clara a necessidade de uma instância de maior porte na AWS, com memória e CPU suficientes para suportar picos de carga.

Outro ponto relevante é que, embora o Shuffle estivesse auto-hospedado em contêineres, o ambiente utilizado está sujeito a uma cota de uso de aproximadamente 2000 *app runs* por mês. Esta limitação foi atingida durante os experimentos, impedindo a execução de cenários adicionais de automação e constituindo-se em mais um fator limitante da PoC.

4.3 Topologia lógica e fluxos de dados

A Figura conceitual (não exibida aqui) que representa a topologia lógica da PoC pode ser descrita da seguinte forma:

- **Camada de Coleta e Detecção**: Servidores e serviços monitorados enviam *logs* e eventos para o Wazuh Manager (por meio de agentes Wazuh ou protocolos suportados). O Wazuh aplica regras de correlação e gera alertas estruturados.
- **Camada de Gestão de Incidentes**: Alertas selecionados são encaminhados ao TheHive (diretamente ou por intermédio do Shuffle), onde são transformados em casos (*cases*) e tarefas (*tasks*). Nesta camada ocorre o registro de contexto, classificação, priorização e documentação do incidente.
- **Camada de Enriquecimento**: A partir de observáveis extraídos dos casos (IPs, domínios, hashes, URLs), o TheHive e/ou o Shuffle acionam *analyzers* no Cortex, que consulta fontes externas (por exemplo, AbuseIPDB, VirusTotal, WHOIS) e retorna informações de reputação e contexto.
- **Camada de Automação e Resposta**: O Shuffle orquestra as interações entre Wazuh, TheHive e Cortex, além de conectar-se a outros sistemas de resposta (como firewalls, EDRs ou serviços de notificação) para executar ações automáticas de contenção, quando apropriado.

Mesmo sem a integração completa com MISP na PoC, essa arquitetura permitiu validar o fluxo mínimo de ponta a ponta: evento → alerta → caso → enriquecimento → automação.

4.4 Configuração do Wazuh

O Wazuh foi escolhido como plataforma de monitoramento e correlação de eventos por ser gratuito, de código aberto e amplamente adotado em contexto corporativo. O ambiente foi configurado seguindo o guia de início rápido oficial (*quickstart*), com as seguintes características gerais:

- **Pré-requisitos de hardware**: servidor com pelo menos 2 vCPUs, 4 GB de RAM e 50 GB de armazenamento, além de conectividade com a internet para instalação de pacotes.
- **Instalação**: utilização do script oficial `wazuh-install.sh`, que automatiza a instalação do Wazuh Server, do Filebeat e do Wazuh Dashboard em uma única máquina. O comando executado foi, de forma simplificada:

```
curl -sO https://packages.wazuh.com/4.12/wazuh-install.sh \
  && sudo bash ./wazuh-install.sh -a
```

- **Componentes**: *Wazuh Manager* como núcleo de correlação e aplicação de regras, Filebeat para envio de *logs* ao mecanismo de indexação e Wazuh Dashboard (baseado em OpenSearch/Kibana) como interface web de visualização.

Após a instalação, o script gera um arquivo comprimido com as credenciais iniciais. As senhas foram extraídas via:

```
sudo tar -O -xvf wazuh-install-files.tar \
  wazuh-install-files/wazuh-passwords.txt
```

e armazenadas em local seguro. Em seguida, os serviços foram verificados via `systemctl`:

```
sudo systemctl status wazuh-manager
sudo systemctl status filebeat
sudo systemctl status wazuh-dashboard
```

garantindo que Wazuh Manager, Filebeat e Dashboard estavam ativos.

4.4.1 Ajuste de portas e *security groups*

Uma adaptação importante diz respeito à porta de acesso ao Wazuh Dashboard. Por padrão, a interface é exposta na porta 443, mas essa porta encontrava-se bloqueada pelas políticas de *firewall/security groups* do ambiente da RadioMemory. Foi necessário:

1. Ajustar as regras do *security group* da instância para permitir uma porta TCP alta dentro do intervalo definido pela equipe de infraestrutura (por exemplo, um intervalo de portas dinâmicas liberadas para testes).
2. Alterar a configuração do Dashboard no arquivo `/etc/wazuh-dashboard/opensearch_dashboards.yml`, substituindo a porta 443 pela porta escolhida.
3. Reiniciar o serviço do Wazuh Dashboard para aplicar as mudanças:

   ```
   sudo systemctl restart wazuh-dashboard
   ```

Após esses ajustes, o Dashboard passou a ser acessado via:

```
https://<ip_publico>:<porta_escolhida>
```

permitindo a visualização dos alertas que, posteriormente, seriam integrados ao fluxo de incidentes no TheHive.

4.5 Configuração do TheHive e do Cortex

O TheHive e o Cortex foram implantados conjuntamente utilizando um arquivo `docker-compose` que orquestra cinco serviços principais: Cassandra, Elasticsearch, TheHive, Cortex e Nginx. Abaixo são detalhados os principais pontos dessa configuração.

4.5.1 Cassandra como banco de dados do TheHive

O serviço Cassandra foi definido com a imagem `cassandra:${cassandra_image_version}`, com as seguintes características:

- **Ambiente:** `CASSANDRA_CLUSTER_NAME=TheHive`, `CASSANDRA_AUTHENTICATOR=PasswordAuthenticator` (habilitando autenticação), `CASSANDRA_NUM_TOKENS=4` e parâmetros de *heap* Java (`HEAP_NEWSIZE=1280M`, `MAX_HEAP_SIZE=200M`).
- **Volumes:** mapeamento de `./cassandra/data` para `/var/lib/cassandra` (dados) e `./cassandra/logs` para `/var/log/cassandra` (*logs*).
- **Recursos:** limite de memória de 2 GB e `memswap_limit: 2G`, prevenindo uso de *swap*.
- **Rede:** conexão à rede interna `thehive-cortex-network`.
- ***Healthcheck*:** comando periódico `cqlsh -u cassandra -p cassandra -e 'describe keyspaces'` para garantir que o Cassandra está pronto antes de inicializar o TheHive.

4.5.2 Elasticsearch como mecanismo de indexação

O serviço Elasticsearch foi configurado com a imagem `elasticsearch:${elasticsearch_image_version}`, sendo utilizado tanto pelo TheHive quanto pelo Cortex:

- **Ambiente:** `http.host=0.0.0.0`, `discovery.type=single-node`, `cluster.name=hive`, aumento das filas de busca e escrita (`thread_pool.search.queue_size=100000`, `thread_pool.write.queue_size=100000`), `bootstrap.memory_lock=true` e `xpack.security.enabled=true`. A senha do usuário `elastic` é definida em `ELASTIC_PASSWORD=${elasticsearch_password}`.
- **Heap Java:** `ES_JAVA_OPTS=-Xms1G -Xmx1G`, reservando 1 GB de memória para o processo Java.
- **Volumes:** mapeamento de `./elasticsearch/data` e `./elasticsearch/logs` para os diretórios de dados e *logs* do Elasticsearch.
- ***Ulimits*:** aumento do número máximo de arquivos abertos (`nofile` soft e hard: 65536), requisito comum para Elasticsearch em produção.
- **Portas:** mapeamento de `8200:9200` (API HTTP) e `8300:9300` (transporte interno).
- ***Healthcheck*:** uso de `curl` com autenticação básica `elastic:${elasticsearch_password}` para consultar `_cat/health`.

4.5.3 TheHive como aplicação principal de gestão de incidentes

O TheHive foi implantado a partir da imagem `strangebee/thehive:${thehive_image_version}` com:

- **Comando de inicialização:** `-no-config -no-config-secret`, indicando que a configuração será lida exclusivamente a partir dos arquivos montados em `/etc/thehive`.
- **Parâmetros Java:** `JAVA_OPTS` com `-Xms1280M`, `-Xmx1280M`, `-XX:MaxMetaspaceSize=400m`, `-XX:ReservedCodeCacheSize=250m`.
- **Volumes:** mapeamento de `./thehive/config` para `/etc/thehive:ro` (somente leitura), `./thehive/data/files` para `/opt/thp/thehive/files` (anexos e artefatos) e `./thehive/logs` para `/var/log/thehive`.
- **Portas:** exposição em `0.0.0.0:9000:9000`, permitindo acesso direto à API e interface web, além do acesso via Nginx.
- ***Healthcheck*:** verificação de `/thehive/api/status` para garantir que o serviço está pronto.
- **Dependências:** inicialização condicionada à saúde do Elasticsearch e do Cassandra.

4.5.4 Cortex como motor de enriquecimento

O Cortex foi implantado com a imagem `thehiveproject/cortex:${cortex_image_version}`:

- **Parâmetros Java:** `JAVA_OPTS` com `-Xms1000M`, `-Xmx1000M`, além de limites de *metaspace* e cache de código.
- **Integração com Elasticsearch:** variável `es_uri=http://elasticsearch:9200`, reutilizando o mesmo cluster do TheHive para armazenamento de resultados.
- **Diretórios de trabalho:** `job_directory=/tmp/cortex-jobs` (dentro do contêiner) e `docker_job_directory=/aidata/.../cortex/cortex-jobs` (no host), permitindo persistir resultados de análises.
- **Volumes:** montagem do `/var/run/docker.sock` para que o Cortex possa executar *analyzers* como contêineres; diretórios para *jobs*, configuração, *logs* e *neurons* (pacotes de *analyzers*).
- **Portas:** mapeamento `0.0.0.0:8091:9001`, expondo a API do Cortex na porta 8091 do host.
- ***Healthcheck*:** verificação de `/cortex/api/status`.

4.5.5 Nginx como *reverse proxy*

O serviço Nginx, com imagem `nginx:${nginx_image_version}`, cumpre o papel de *reverse proxy* e terminação TLS:

- **Ambiente:** `SERVER_NAME=${nginx_server_name}` e `NGINX_SSL_TRUSTED_CERTIFICATE=${nginx_ssl_trusted_certificate}`, permitindo parametrizar certificados e nome do host.
- **Volumes:** `./nginx/templates` para `/etc/nginx/templates` (configurações dinâmicas) e `./nginx/certs` para `/etc/nginx/certs` (certificados SSL).
- **Portas:** publicação da porta `443` interna na porta `8443` do host, concentrando acessos HTTPS externos em um único ponto.
- **Dependência:** inicialização posterior aos serviços TheHive e Cortex.

4.6 Configuração do Shuffle e limitações de uso

Para a plataforma SOAR, foi utilizado o Shuffle em modo auto-hospedado, também via `docker-compose`. Os principais serviços configurados foram:

4.6.1 Frontend e backend do Shuffle

- **shuffle-frontend**: interface web do Shuffle, baseada em `ghcr.io/shuffle/shuffle-frontend:latest`, exposta nas portas `${FRONTEND_PORT}:80` (HTTP) e `${FRONTEND_PORT_HTTPS}:443` (HTTPS). O frontend utiliza a variável de ambiente `BACKEND_HOSTNAME` para se comunicar com o serviço backend.
- **shuffle-backend**: API principal do Shuffle, com imagem `ghcr.io/shuffle/shuffle-backend:latest`, expondo internamente a porta 5001 e externamente a porta `${BACKEND_HOST_PORT}` no host.
  - **Volumes:** `/var/run/docker.sock` para orquestrar contêineres de apps, `${SHUFFLE_APP_HOTLOAD_LOCATION}:/shuffle-apps` para armazenamento e *hot reload* de apps, e `${SHUFFLE_FILE_LOCATION}:/shuffle-files` para arquivos associados a *workflows*.
  - **Configuração:** uso de arquivo `.env` para parametrização e variáveis como `SHUFFLE_APP_HOTLOAD_FOLDER=/shuffle-apps`, `SHUFFLE_FILE_LOCATION=/shuffle-files`.
  - **Usuário inicial:** definido por `SHUFFLE_DEFAULT_USERNAME`, `SHUFFLE_DEFAULT_PASSWORD` e `SHUFFLE_DEFAULT_APIKEY`, permitindo acesso imediato à interface e às APIs.

4.6.2 Orborus e *workers*

O componente **shuffle-orborus** é responsável por agendar e executar os *workflows*:

- **Imagem:** `ghcr.io/shuffle/shuffle-orborus:latest`.
- **Volumes:** `/var/run/docker.sock`, permitindo criar contêineres de *workers* com base na imagem `ghcr.io/shuffle/shuffle-worker:latest`.
- **Variáveis de ambiente:**
  - `SHUFFLE_APP_SDK_TIMEOUT=300`: tempo limite padrão para execuções de apps.
  - `SHUFFLE_ORBORUS_EXECUTION_CONCURRENCY`: limite de execuções concorrentes (configurado para 7 no `docker-compose` e ajustável também no `.env` para 5).
  - `ENVIRONMENT_NAME=Shuffle` e `ORG_ID=Shuffle`: identificadores lógicos do ambiente e da organização.
  - `SHUFFLE_STATS_DISABLED=true` e `SHUFFLE_LOGS_DISABLED=true`: redução de coleta de estatísticas e logs detalhados para aliviar carga.

- `SHUFFLE_SKIPSSL_VERIFY=true`: desabilita a verificação de certificados SSL em integrações internas, útil em ambientes de teste.
- `SHUFFLE_WORKER_IMAGE`: define a imagem padrão dos *workers*.

4.6.3 OpenSearch como banco de dados do Shuffle

O serviço **shuffle-opensearch**, baseado em `opensearchproject/opensearch:3.2.0`, cumpre o papel de banco de dados e mecanismo de busca do Shuffle:

- **Heap Java:** `OPENSEARCH_JAVA_OPTS=-Xms8g -Xmx8g`, reservando 8 GB de memória para o processo Java.
- **Configuração de cluster:** `cluster.name=shuffle-cluster`, `node.name=shuffle-opensearch`, `discovery.seed_hosts=shuffle-opensearch`, desativação de limiares de disco e do *Performance Analyzer* para simplificar o ambiente.
- **Segurança:** definição de `OPENSEARCH_INITIAL_ADMIN_PASSWORD=${SHUFFLE_OPENSEARCH_PASSWORD}` para o usuário `admin`;
  variáveis no backend apontam para `SHUFFLE_OPENSEARCH_URL=https://shuffle-opensearch:9200`, com opção de ignorar verificação SSL (`SHUFFLE_OPENSEARCH_SKIPSSL_VERIFY=true`).
- **Portas:** mapeamento da porta `9200` interna para `8051` no host, facilitando depuração.
- **Volumes:** armazenamento persistente em `shuffle-database`, mapeado para `${DB_LOCATION}` no host.

4.6.4 Limitações práticas observadas

Na prática, o Shuffle mostrou-se o componente mais exigente em termos de recursos. Mesmo com ajustes de concorrência e desativação de recursos não essenciais, foi observada lentidão significativa em execuções de *workflows* mais complexos, o que motivou a migração desse componente para uma instância `c6a.4xlarge` na AWS, com mais CPU e memória disponíveis.

Além disso, o ambiente estava sujeito a uma cota de aproximadamente 2000 *app runs* mensais. Durante a PoC, esse limite foi atingido, impedindo a realização de novos testes. Esse fato evidenciou que, mesmo em cenários auto-hospedados, podem existir restrições de uso impostas pela plataforma ou pelo modelo de licenciamento, exigindo planejamento de consumo e eventual contratação de planos pagos, caso a automação seja intensiva.

Outro desafio observado foi a qualidade e atualidade dos *webhooks* e integrações pré-configuradas do Shuffle. Diversos conectores apresentavam parâmetros desatualizados ou inconsistentes com as APIs atuais, sendo necessário revisar manualmente os campos de autenticação, URLs e formatos de payload para que os *playbooks* funcionassem como esperado. Esse esforço adicional necessita ser considerado em qualquer estimativa de esforço para implantação de um CSIRT baseado em ferramentas open source.

4.7 Resumo de requisitos de infraestrutura e portas

A Tabela 2 sintetiza, de forma comparativa, os principais requisitos de infraestrutura e portas utilizados na PoC para cada componente.

Tabela 2: Requisitos de infraestrutura e portas por componente na PoC

Ferramenta	Componente	CPU/RAM (referência)	Armazenamento	Portas principais (host → contêiner/serviço)	Observações
Wazuh	Manager + Dashboard	≥2 vCPUs / 4 GB	≥50 GB	443 → 443 (alterada para porta alta)	Instalação via script oficial
TheHive/Cortex	Cassandra	∼2 GB RAM (limite contêiner)	Volume `./cassandra/data`	Somente rede interna Docker	Autenticação habilitada
TheHive/Cortex	Elasticsearch	∼2 GB RAM / 1 GB heap	Volume `./elasticsearch/data`	8200 → 9200; 8300 → 9300	`xpack.security` habilitado
TheHive/Cortex	TheHive	∼2 GB RAM	Volume `./thehive/data/files`	9000 → 9000	API e UI de casos
TheHive/Cortex	Cortex	∼2 GB RAM / 1 GB heap	Volume `./cortex/cortex-jobs`	8091 → 9001	Usa Docker para analyzers
TheHive/Cortex	Nginx	∼512 MB RAM	Sem dados persistentes	8443 → 443	*Reverse proxy* TLS para TheHive/Cortex
Shuffle	Frontend	Compartilha instância	Sem dados críticos	${FRONTEND_PORT}:80; ${FRONTEND_PORT_HTTPS}:443	Interface web do SOAR
Shuffle	Backend	Alta, dependente de carga	`${SHUFFLE_FILE_LOCATION}`	${BACKEND_HOST_PORT}:5001	API e orquestração de *workflows*
Shuffle	Orborus/Workers	CPU variável / RAM variável	Sem dados permanentes	Rede interna Docker	Executa apps sob demanda
Shuffle	OpenSearch	≥8 GB heap Java	Volume `shuffle-database`	8051 → 9200	Banco de dados principal

4.8 Lições aprendidas

A PoC realizada na RadioMemory permitiu extrair algumas lições importantes:

- **Planejamento de rede e portas é crítico**: a necessidade de adaptar portas do Wazuh Dashboard, do TheHive/Cortex (via Nginx) e do backend do Shuffle mostrou que as restrições de *security groups* e demais políticas de segurança da nuvem devem ser consideradas desde o desenho inicial da arquitetura. Pequenos detalhes, como a escolha de portas altas e não usadas por outros serviços, podem evitar retrabalho.
- **Ferramentas open-source exigem margem para ajustes**: configurações de *webhooks* e integrações predefinidas, como no caso do Shuffle, podem não funcionar conforme o esperado e demandar um esforço de customização e correção. Da mesma forma, parâmetros padrão de banco de dados e de *heap* precisam frequentemente de *tuning*.
- **Consumo de recursos pode ser elevado**: mesmo em um cenário de PoC, o Shuffle exigiu uma instância de grande porte (`c6a.4xlarge`) devido ao consumo de memória do OpenSearch e à necessidade de suportar múltiplas execuções concorrentes. Isso reforça que um CSIRT acessível não significa necessariamente um CSIRT "leve" em termos de infraestrutura, principalmente quando se pretende automatizar de forma intensiva.
- **Limitações de licenciamento e cotas impactam a PoC**: o limite de 2000 *app runs* mensais no Shuffle restringiu a quantidade de experimentos possíveis e demonstra como restrições comerciais podem afetar a adoção de ferramentas em ambientes reais. O planejamento de consumo e a avaliação de planos pagos devem ser parte da estratégia de longo prazo.
- **Separação entre ambiente local e nuvem é viável, mas complexa**: manter Wazuh, TheHive e Cortex localmente, enquanto o Shuffle roda na nuvem, mostra que arquiteturas híbridas podem ser implementadas com ferramentas open source. Porém, essa abordagem implica maior atenção a latência, exposição de serviços e configuração de *firewalls*/*security groups*.
- **Apesar dos desafios, a metodologia é viável**: mesmo com as limitações encontradas, foi possível montar um ambiente funcional que integra detecção, gestão de incidentes, enriquecimento de evidências e automação, demonstrando na prática a viabilidade da metodologia proposta para um CSIRT acessível.

Em síntese, a PoC atingiu seu propósito de validar, em um cenário real e com restrições concretas de infraestrutura, a aplicabilidade da abordagem de CSIRT acessível, ao mesmo tempo em que revelou pontos de atenção importantes para futuras implantações em ambientes produtivos.

5 Avaliação de Custos

Além dos aspectos técnicos e de viabilidade operacional, a adoção de um CSIRT acessível precisa ser analisada sob a ótica econômica. Nesta seção são estimados os custos mensais de operação do modelo proposto neste trabalho e comparados com uma alternativa mais tradicional baseada em serviços de *SOC as a Service* (SOC terceirizado). Os valores são aproximados e servem como ordem de grandeza para apoiar a tomada de decisão.

Para facilitar a leitura, os custos são apresentados em dólares americanos (USD) e em reais (BRL), assumindo uma taxa de câmbio média de referência de aproximadamente

$$1\ \text{USD} \approx 5{,}34\ \text{BRL},$$

condizente com o período de elaboração deste trabalho.

5.1 CSIRT Acessível: decomposição de custos

No modelo de CSIRT acessível proposto, os principais componentes de custo recorrente são:

1. **Infraestrutura em nuvem** para suportar o componente mais pesado do ecossistema (Shuffle + OpenSearch).
2. **Plano pago do Shuffle** para ampliar o limite de execuções mensais de *apps*.
3. **Mão de obra especializada** para operar, manter e evoluir o ambiente.

Ferramentas como Wazuh, TheHive, Cortex e MISP são utilizadas em suas versões *open source* ou comunitárias, de modo que não geram custos diretos de licenciamento, apenas custos indiretos de infraestrutura e operação.

5.1.1 Infraestrutura: instância c6a.4xlarge

Na Prova de Conceito foi necessário utilizar uma instância do tipo `c6a.4xlarge` na AWS para suportar o Shuffle e o seu banco de dados OpenSearch, devido ao consumo elevado de memória e à necessidade de executar múltiplos contêineres (backend, frontend, Orborus, *workers* e OpenSearch).

Segundo a precificação *on-demand* da AWS para a região `us-east-1`, uma instância `c6a.4xlarge` (16 vCPUs, 32 GiB de RAM) custa aproximadamente US$ 0,61 por hora, o que corresponde a cerca de US$ 446,76 por mês para uma utilização contínua (24 h por dia, 30/31 dias) [10]. Arredondando:

- **EC2 c6a.4xlarge**: $\approx$ US$ 447/mês $\approx$ R$ 2,400/mês.

Esse valor representa o custo mensal de manter o Shuffle e o OpenSearch em funcionamento ininterrupto na nuvem.

5.1.2 Plano pago do Shuffle

O trabalho considerou a necessidade de migrar do plano gratuito do Shuffle (limitado a aproximadamente 2 000 *App Runs* por mês) para um plano pago com maior capacidade. Para fins de estimativa, adota-se o plano informado de:

- **Shuffle**: US$ 58/mês para 20 000 *App Runs*.

Convertendo:

- $\approx$ R$ 310/mês.

Este custo garante espaço para automatizar fluxos de trabalho com maior intensidade, sem esbarrar rapidamente em limites de execução.

5.1.3 Ferramentas open source

As seguintes ferramentas são utilizadas em suas edições gratuitas ou comunitárias:

- **Wazuh** (SIEM/XDR),
- **TheHive** (gestão de incidentes),
- **Cortex** (enriquecimento de observáveis),
- **MISP** (inteligência de ameaças).

Nesse cenário, assume-se:

- **Licenciamento Wazuh/TheHive/Cortex/MISP**: US$ 0/mês (apenas custos de infraestrutura e operação).

5.1.4 Mão de obra: analista responsável pelo CSIRT

Embora as ferramentas sejam gratuitas, a operação de um CSIRT acessível exige pelo menos um profissional dedicado para:

- manter regras de correlação e alertas;
- criar e ajustar *playbooks* no Shuffle;
- integrar novas fontes de dados;
- acompanhar e investigar incidentes;
- realizar manutenção e atualizações periódicas das ferramentas.

Pesquisas de mercado indicam que profissionais de cibersegurança no Brasil têm uma faixa ampla de remuneração. Há relatos de salários de entrada em torno de R$ 3,000 a R$ 6,000/mês para posições juniores [11, 12], enquanto analistas seniores podem atingir valores próximos a R$ 15,000/mês ou mais em grandes centros urbanos [13, 14, 15].

Para este trabalho, assume-se um valor intermediário e conservador para um analista de nível pleno/sênior responsável pelo CSIRT:

- **Analista de segurança (pleno/sênior)**: $\approx$ R$ 10,000/mês $\approx$ US$ 1,870/mês.

5.1.5 Resumo numérico do CSIRT Acessível

A Tabela 3 resume os principais componentes de custo mensais estimados para o CSIRT acessível.

Tabela 3: Componentes de custo mensais do CSIRT Acessível (estimativa)

Componente	**Custo (USD/mês)**	**Custo (BRL/mês)**
Instância EC2 c6a.4xlarge	$\approx$ 447	$\approx$ 2,400
Shuffle (20k App Runs)	58	$\approx$ 310
Ferramentas open source	0	0
1 analista de segurança	$\approx$ 1,870	$\approx$ 10,000
Total	$\approx$ **2,375**	$\approx$ **12,710**

Para efeitos de comunicação, pode-se arredondar esse total para:

- **CSIRT Acessível**: $\approx$ US$ 2,400/mês $\approx$ R$ 12,700/mês.

5.2 CSIRT sem essas tecnologias: alternativa via SOC terceirizado

Sem recorrer às ferramentas open source e à automação construída internamente, muitas organizações optam por contratar um *Security Operations Center* terceirizado (*SOC as a Service* ou MDR). Nesses modelos, o provedor oferece:

- SIEM gerenciado,
- monitoramento contínuo (frequentemente 24x7),
- correlação de eventos,
- resposta a incidentes,
- relatórios de conformidade,
- e, em alguns casos, integração com soluções de *endpoint*, *firewalls* e nuvem.

Diversas fontes indicam que o custo de um SOC como serviço para pequenas e médias empresas toma, em geral, a forma de um valor mensal fixo ou baseado em número de ativos monitorados. Faixas típicas reportadas são, por exemplo:

- Pequenas empresas: cerca de US$ 1,000 a US$ 3,000/mês para serviços básicos;
- Empresas de médio porte: cerca de US$ 3,000 a US$ 7,000/mês;
- Ambientes mais complexos ou com coberturas avançadas: valores que podem atingir US$ 10,000/mês ou mais [16, 17, 18].

Para fins de comparação com o CSIRT acessível proposto, adota-se um valor intermediário razoável para uma PME com cobertura 24x7:

- **SOC terceirizado (24x7, PME)**: $\approx$ US$ 7,500/mês $\approx$ R$ 40,000/mês.

Esse valor inclui, em regra, o licenciamento das ferramentas utilizadas pelo provedor (SIEM, eventualmente SOAR, *endpoints*, etc.) e a equipe especializada que opera o SOC. Mesmo assim, muitas organizações ainda mantêm algum profissional interno para fazer a ponte entre o negócio e o provedor, tratar exceções e alinhar prioridades.

Tabela 4: Comparação de custos mensais entre o CSIRT Acessível e um SOC terceirizado

Cenário	**Custo (USD/mês)**	**Custo (BRL/mês)**
CSIRT Acessível (open source + 1 analista)	$\approx$ 2,400	$\approx$ 12,700
SOC terceirizado 24x7 (MDR/SOCaaS)	$\approx$ 7,500	$\approx$ 40,000

5.3 Comparação dos custos mensais

A Tabela 4 resume a comparação entre os dois cenários:

Em termos relativos, o CSIRT acessível proposto neste trabalho apresenta um custo mensal:

- aproximadamente **2 a 3 vezes menor** do que a contratação de um SOC terceirizado de porte comparável;
- com a contrapartida de não oferecer, por si só, **cobertura 24x7** e de exigir **maior investimento em conhecimento técnico interno**.

Por outro lado, o SOC terceirizado tende a oferecer:

- uma solução mais próxima de "*turn-key*", com menor esforço interno de implantação;
- equipe multidisciplinar já formada, capaz de cobrir diferentes fusos horários e especialidades;
- SLAs formais de monitoramento e resposta.

Já o CSIRT acessível proporciona:

- forte redução de custos diretos de licenciamento;
- maior controle e flexibilidade sobre ferramentas e fluxos de trabalho;
- uma base sólida para amadurecer a função de segurança internamente, aproveitando melhor o conhecimento do ambiente específico da organização.

Dessa forma, a análise de custos reforça o argumento central deste trabalho: é possível construir uma capacidade relevante de resposta a incidentes com um orçamento substancialmente menor do que o exigido por soluções comerciais de SOC como serviço, desde que a organização esteja disposta a investir em pessoal qualificado e em um processo contínuo de implantação, ajuste e automação.

6 Conclusão

Este relatório apresentou uma metodologia prática para implementação de um CSIRT acessível, com base em ferramentas de código aberto e em uma abordagem faseada que permite evolução gradual de capacidades. Foram discutidos os fundamentos teóricos que orientam a definição de serviços de um CSIRT, a seleção de ferramentas (Wazuh, TheHive, Cortex, MISP e Shuffle) e o encadeamento de fluxos de dados entre elas.

A Prova de Conceito realizada em parceria com a RadioMemory mostrou que é possível, mesmo em um ambiente com restrições de segurança de rede e de orçamento, construir um *stack* integrado de detecção, gestão de incidentes, enriquecimento e automação. A implantação local de Wazuh, TheHive e Cortex, combinada com a execução do Shuffle em uma instância dedicada na AWS, constituiu um cenário realista para validação da proposta.

Por outro lado, a PoC também evidenciou limitações importantes: o alto consumo de recursos do Shuffle e do OpenSearch, as cotas de *app runs*, as dificuldades com integrações pré-configuradas e a tendência de algumas ferramentas open-source adotarem modelos híbridos ou comerciais ao longo do tempo. Esses fatores reforçam a necessidade de planejamento de longo prazo, tanto em termos de infraestrutura quanto de licenciamento.

A "acessibilidade" desta solução vem, portanto, com a contrapartida de uma maior necessidade de conhecimento técnico interno e um investimento significativo de tempo para configuração, integração, manutenção e, crucialmente, para o desenvolvimento e refinamento de fluxos de trabalho de automação. O sucesso do CSIRT Acessível depende de um planejamento estratégico sólido, do engajamento da direção da organização e de um compromisso contínuo com a aprendizagem e a melhoria de processos.

Como trabalhos futuros, destacam-se:

- A integração completa com o MISP, permitindo o consumo e o compartilhamento de inteligência de ameaças de forma automatizada.

- A migração do ambiente para orquestração em Kubernetes, com o objetivo de melhorar escalabilidade, isolamento e observabilidade.
- A realização de avaliações sistemáticas de desempenho e custo da solução, comparando cenários de infraestrutura local, nuvem e híbrida.
- O desenvolvimento de um catálogo de *playbooks* reutilizáveis para diferentes tipos de incidentes (phishing, ransomware, exfiltração de dados, entre outros), adaptados à realidade de PMEs.

Ao seguir esta metodologia e abraçar estas recomendações, as organizações podem não apenas construir um CSIRT acessível, mas garantir que ele permaneça uma componente vital, eficaz e adaptável da sua estratégia de cibersegurança.

Referências

[1] Sprinto, "Cybersecurity incident response plan: Steps & guide." https://sprinto.com/blog/cybersecurity-incident-response-plan/, 2025.

[2] M. Malatji, "A cybersecurity ai agent selection and decision support framework," 2025.

[3] Cybersecurity and Infrastructure Security Agency (CISA), "Free cybersecurity services & tools." https://www.cisa.gov/resources-tools/resources/free-cybersecurity-services-and-tools.

[4] Wazuh, "Wazuh - the open source security platform. unified xdr and siem protection for endpoints and cloud workloads." https://wazuh.com/, 2025.

[5] Shuffle, "Shuffle: A general purpose security automation platform. our focus is on collaboration and resource sharing." https://github.com/Shuffle/Shuffle, 2025.

[6] StrangeBee, "Thehive: a scalable, open source and free security incident response platform." https://github.com/TheHive-Project/TheHive, 2025.

[7] Cortex, "Cortex: a powerful observable analysis and active response engine." https://github.com/TheHive-Project/Cortex, 2025.

[8] MISP Project, "Misp open source threat intelligence platform & open standards for threat information sharing." https://www.misp-project.org/, 2025.

[9] R. Ruefle, A. Dorofee, D. Mundie, A. D. Householder, M. Murray, and S. J. Perl, "Computer Security Incident Response Team Development and Evolution," *IEEE Security & Privacy*, vol. 12, pp. 16–26, sep 2014.

[10] Amazon Web Services, "Amazon eks pricing," 2025. Exemplos oficiais de custo incluem instância c6a.4xlarge com preço de US$ 0,612 por hora em cenários de EKS Auto Mode.

[11] Quero Bolsa, "Quanto ganha analista de segurança da informação?," 2025. Informa salário médio nacional para Analista de Segurança da Informação em torno de R$ 9.998,79 mensais e destaca variação por estado e especialidade.

[12] IBSEC, "Cibersegurança (ou cyber segurança): o guia completo para dominar a profissão mais promissora de 2025," 2025. Aponta que salários em cibersegurança no Brasil em 2025 variam de R$ 5,000 a R$ 20,000 conforme nível de experiência, reforçando a percepção de alto custo de mão de obra qualificada.

[13] Trybe, "Salário de analista de segurança da informação," 2024. Cita medianas salariais a partir de dados do Glassdoor: cerca de R$ 3,5 mil para nível júnior e cerca de R$ 6,9 mil para profissionais seniores, ressaltando que salários variam por experiência e região.

[14] Forbes Brasil, "Brasil lidera com os maiores salários de tecnologia da américa latina," 2025. Relatório citado indica que funções de IA e cibersegurança pagam, em média, 20% a 25% acima da média global, evidenciando a pressão de custo por talentos especializados.

[15] IT Forum, "Salários de cibersegurança no brasil para 2026," 2025. Mostra faixas salariais como Analista de Segurança Júnior com valores na casa de R$ 6,200 a R$ 10,400 e Analista Sênior alcançando até cerca de R$ 19,000 mensais, reforçando a alta remuneração da área.

[16] Vectra AI, "Soc as a service: How to achieve 70% cost savings in 30 days," 2025. Discute SOCaaS para pequenas e médias empresas e cita faixa típica de custo entre US$ 1,000 e US$ 10,000 mensais, dependendo do porte e requisitos.

[17] L. Kontseva, "Socaas pricing, hidden fees, and investment value," 2025. Artigo detalha modelos de precificação de SOCaaS, incluindo abordagem por ativo/dispositivo/mês e intervalo típico de valores para diferentes portes.

[18] Ascendant Technologies, "Soc as a service pricing: How costs are calculated," 2025. Discute fatores que influenciam o custo de SOCaaS e compara com a construção de um SOC interno, citando que montar um SOC próprio pode ultrapassar US$ 2 milhões em custos anuais, enquanto SOCaaS pode sair por cerca da metade.