UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mateus Ribeiro Vaz Pereira

CowrieSniffer: Extensão do Honeypot Cowrie para coleta e monitoramento de URLs

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mateus Ribeiro Vaz Pereira

CowrieSniffer: Extensão do Honeypot Cowrie para coleta e monitoramento de URLs

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Agradecimentos

Agradeço à minha família, Jefferson, Silvia, Felipe e Marcela pelo apoio incondicional e pela oportunidade de seguir em frente mesmo nos momentos mais desafiadores. Sem eles, esta conquista não seria possível. São exemplos de dedicação, incentivo e suporte.

Ao meu orientador, Rodrigo Sanches Miani, pela orientação valiosa e pela confiança depositada em mim desde o início da graduação.

Ao Laboratório de Segurança Cibernética (LSG) da FACOM/UFU, conduzido pelos professores Rodrigo Sanches Miani e Ivan da Silva Sendin, pelo ambiente de aprendizado e pesquisa que contribuiu imensamente para minha formação.

Aos amigos que fiz na faculdade e que levo para a vida, por compartilharem essa jornada comigo.

E à minha companheira, Julia, que esteve ao meu lado em todas as etapas dessa conquista, com apoio e motivação ao longo de toda a caminhada acadêmica.

Resumo

Este trabalho apresenta o desenvolvimento do *CowrieSniffer*, uma ferramenta para análise e monitoramento de endereços capturados no *Honeypot Cowrie*. O sistema tem como objetivo verificar a disponibilidade das *URLs* utilizadas por atacantes, o que contribui para a análise de ameaças e o aprimoramento da segurança cibernética. A ferramenta desenvolvida atua na coleta das *URLs* armazenadas nas tabelas *downloads* e *input* do banco de dados do *Cowrie*, transferindo-as para um banco de monitoramento específico. Em seguida, a ferramenta realiza a verificação periódica da disponibilidade desses endereços e atualiza a tabela *urls* do banco de monitoramento de acordo com as mudanças na disponibilidade das *URLs* identificadas. A implementação foi realizada com o uso de *Python*, devido à sua versatilidade e facilidade na manipulação de dados, além da utilização de *Docker* para garantir a portabilidade do ambiente e *MySQL* para o armazenamento das informações coletadas. Os testes realizados demonstraram que a ferramenta é capaz de processar as informações de maneira eficiente, o que possibilita a análise da persistência e do tempo de vida de *URLs* utilizadas por atacantes. Os resultados obtidos durante os experimentos propostos validaram a eficácia do sistema.

Palavras-chave: *Honeypots*, *Cowrie*, Segurança da Informação, Análise de disponibilidade em *URLs*, endereços maliciosos.

Lista de ilustrações

Lista de abreviaturas e siglas

APT	*Advanced Persistent Threats*
C&C	Comando e Controle
DNS	*Domain Name System*
DPI	*Deep Packet Inspection*
ELK	*Elastic Stack*
FTP	*File Transfer Protocol*
HTTP	*Hypertext Transfer Protocol*
IP	*Internet Protocol*
IoT	*Internet of things*
MySQL	Banco de dados relacional de código aberto
NIST	*National Institute of Standards and Technology*
SQL	*Structured Query Language*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
TTPs	Táticas, técnicas e procedimentos
URL	*Uniform Resource Locator*

Sumário

1 Introdução

A indústria de crimes cibernéticos expande-se de maneira acelerada de acordo com o estudo de Viraja e Purandare (2021). O crescimento desse tipo de atividade ilícita observa-se ano após ano, impulsionado pelo desenvolvimento de novas técnicas e tecnologias cada vez mais sofisticadas. Em resposta, empresas e governos enfrentam a necessidade contínua de aprimorar suas defesas para superar os desafios impostos por essas ameaças emergentes. O avanço do cibercrime relaciona-se intrinsecamente a fatores econômicos, pessoais e ideológicos, o que contribui para a ocorrência de prejuízos financeiros significativos e danos irreparáveis à reputação de organizações e indivíduos.

Para mitigar essas ameaças e prevenir possíveis ataques, é essencial que as organizações compreendam e analisem os dados coletados a partir de eventos em suas redes, como endereços *Internet Protocol* (*IP*), domínios, ferramentas e técnicas utilizadas, e, em casos de ataques direcionados, credenciais como usuário e senha (AL-MOHANNADI; AWAN; HAMAR, 2020). No entanto, a coleta desses dados em ambientes de produção apresenta riscos consideráveis, o que exige a utilização de ferramentas específicas para esse fim. Nesse contexto, os *honeypots* surgem como uma solução eficaz: sistemas que simulam ambientes de produção com o objetivo de atrair, detectar e monitorar atividades maliciosas. Conforme descrito por Provos e Holz (2007), os *honeypots* podem ser classificados em dois tipos: de baixa interatividade, onde o sistema simulado oferece interações limitadas ao atacante; e de alta interatividade, que disponibiliza um ambiente realista, o qual permite que o invasor explore livremente o sistema operacional e seus serviços.

Para Viraja e Purandare (2021), o cibercrime desenvolve-se de maneira contínua e acelerada. Nesse cenário, empresas e organizações enfrentam vulnerabilidades emergentes e técnicas inovadoras, exploradas por indivíduos e grupos organizados, como as *APTs* (*Advanced Persistent Threats*). Esses grupos caracterizam-se por elevados níveis de especialização e pela disponibilidade de recursos significativos, o que lhes permite explorar diversos vetores de ataque cibernético para alcançar seus objetivos. Um grupo *APT* define-se como uma ameaça sofisticada e persistente, composta por agentes geralmente patrocinados por Estados ou motivados por interesses financeiros ou ideológicos, capazes de conduzir campanhas prolongadas e altamente direcionadas contra alvos estratégicos, com vistas à infiltração, manutenção do acesso e exfiltração de dados sensíveis de maneira furtiva. Entre seus principais objetivos, destaca-se o estabelecimento inicial e a expansão contínua de pontos de apoio dentro da infraestrutura das entidades-alvo, o que consiste em acessos privilegiados, dispositivos comprometidos ou persistência em sistemas críticos. Esses pontos permitem que o grupo mantenha o controle sobre a rede invadida, facilite movimentos laterais, colete informações sensíveis de forma contínua e, eventualmente,

conduza novas fases do ataque com maior eficácia e discrição. Em muitos casos, grupos *APT* mantêm a perseguição a seus alvos durante longos períodos e propagam-se pela rede interna por meio do emprego de *malwares*. Para a obtenção de informações relacionadas a ataques específicos, pode-se empregar redes de *honeypots* que simulam ambientes de produção, desde que apresentem alta similaridade com o ambiente real da organização, o que potencializa a eficácia na coleta de dados. Dessa forma, os *honeypots* possibilitam a identificação de informações valiosas sobre novos *malwares* e sobre tentativas inéditas de ataques direcionados a instituições.

O estudo de Mendes (2023) destaca a importância do uso de *honeypots* no contexto da *Internet of Things* (*IoT*). A adoção dessa tecnologia é particularmente relevante, uma vez que dispositivos *IoT* conectam-se constantemente à Internet, muitas vezes transmitem dados sensíveis ou apresentam configurações inadequadas de segurança. A exploração de um *honeypot* em ambientes *IoT*, conforme abordado no referido trabalho, possibilita a coleta de dados valiosos, como *endpoints*, a quantidade de requisições provenientes de cada endereço *IP* e as tentativas de download de *malwares*. Quando aplicado ao ambiente corporativo, um *honeypot* pode gerar informações significativas, semelhantes às apresentadas no estudo, o que auxilia na identificação precoce de ataques. De maneira complementar, Rodrigues (2017) propõe a utilização de *Deep Packet Inspection* (*DPI*) em conjunto com *honeypots* para capturar ataques direcionados às camadas de Rede, Transporte e Aplicação, o que analisa os logs obtidos e fornece uma série de informações críticas para a segurança das redes.

Nesse contexto, destaca-se a utilização do *Cowrie*[1], uma ferramenta amplamente empregada na construção de *honeypots*, especialmente para a simulação de serviços *Secure Shell* (*SSH*) e *Telnet*. O *Cowrie* é um *honeypot* que permite capturar as credenciais utilizadas por atacantes, registra comandos executados e coleta arquivos transferidos durante tentativas de intrusão, o que oferece dados valiosos para a análise de comportamento malicioso. Apesar de sua relevância e robustez na coleta de informações, a ferramenta apresenta limitações no que se refere à análise e visualização dos dados gerados, o que dificulta a extração de informações relevantes de forma sistemática. Nesse cenário, o presente trabalho propõe o desenvolvimento de uma solução que aprimora a interpretação e a visualização das *URLs* coletadas pelo *Cowrie*, o que torna-as mais acessíveis e úteis para a identificação de padrões de ataque e para a tomada de decisões em segurança da informação.

[1] Disponível em https://github.com/cowrie/cowrie

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma ferramenta para a análise de informações capturadas por uma rede de *honeypots*. A partir da coleta de dados expostos por atacantes, como *URLs*, propõe-se a implementação de mecanismos capazes de verificar a disponibilidade dessas informações.

Para alcançar os objetivos deste trabalho, serão utilizadas as seguintes tecnologias: *Python*, *Docker*, o *honeypot Cowrie*, o banco de dados *MySQL* e conhecimentos de redes. A escolha da linguagem *Python* justifica-se por sua versatilidade e pela facilidade na criação de *scripts*. O *honeypot Cowrie* foi selecionado em razão de sua simplicidade de configuração, das funcionalidades de integração e dos recursos de registro de *logs*. Já o banco de dados *MySQL* foi escolhido devido à sua eficiência nas operações e à facilidade de utilização.

1.1.2 Objetivos Específicos

Os objetivos específicos incluem:

- Criar uma ferramenta em *Python* para a verificação da disponibilidade das informações obtidas através dos *honeypots*.
- Armazenar a *URL* em um banco de dados, juntamente com as datas de primeiro e último registro de acesso.
- Conduzir estudos de caso para validar o funcionamento do *CowrieSniffer*.

1.2 Organização da Monografia

O presente trabalho está estruturado da seguinte maneira: O Capítulo 2 apresenta uma revisão bibliográfica que fundamenta a teoria utilizada para o desenvolvimento desta pesquisa. Nele, é realizada uma investigação sobre cibersegurança, conceitos relativos a *honeypots*, a linguagem de programação *Python*, conceitos sobre *MySQL*, *Docker* e os trabalhos correlatos. O Capítulo 3 caracteriza as etapas do desenvolvimento de forma modularizada, referenciando a visão geral do *CowrieSniffer*, a configuração inicial do banco de dados de monitoramento, o desenvolvimento do arquivo de configuração, o desenvolvimento do componente *Main*, do componente de configuração, do componente de manipulação do banco de dados do *honeypot Cowrie*, do componente de manipulação do banco de dados de monitoramento e do componente *URLMonitor*. O Capítulo 4 discorre sobre os resultados da pesquisa, descrevendo o caso 1, no qual ambos os bancos de dados

estão vazios; o caso 2, que corresponde à inicialização da ferramenta com o banco de dados *Cowrie* populado e o de monitoramento vazio; o caso 3, em que ambos os bancos de dados estão populados; e o caso 4, cujo objetivo é analisar o comportamento da ferramenta em relação à adição de novas *URLs*. Por fim, o Capítulo 5 apresenta as considerações finais elaboradas a partir dos resultados obtidos, além das propostas para trabalhos futuros.

2 Revisão Bibliográfica

Neste capítulo, serão apresentados os conceitos teóricos necessários para a compreensão dos principais conceitos abordados neste trabalho. Além disso, será realizada uma breve análise de trabalhos correlatos.

2.1 Fundamentação Teórica

2.1.1 Cibersegurança

De acordo com o *NIST* (*National Institute of Standards and Technology*), a cibersegurança consiste no conjunto de práticas, tecnologias e processos destinados à proteção de computadores, sistemas de informação e serviços de comunicação contra ataques cibernéticos (NIST, 2019). Essa área abrange diversos aspectos de segurança, com o objetivo de garantir a confidencialidade, a integridade e a disponibilidade dos dados, além de prevenir o vazamento de informações, o acesso não autorizado e a interrupção de serviços.

Além disso, Stallings (2014) apresenta três conceitos fundamentais que orientam a prática da cibersegurança. Esses princípios formam uma tríade, composta por confidencialidade, integridade e disponibilidade. Cada um desses elementos representa um objetivo essencial para a proteção de dados, sistemas de informação e ambientes computacionais. De acordo com Stallings (2014), esses conceitos são definidos como:

Confidencialidade: assegura que todo acesso deve ser restrito para que informações confidenciais não estejam disponíveis para indivíduos não autorizados;

Integridade: assegura que uma informação só poderá ser modificada ou acessada por aquele que detém a autorização específica;

Disponibilidade: garante que um sistema não fique disponível para que ocorra o acesso e uso rápido da informação.

Esses conceitos estabelecem os objetivos fundamentais da segurança em aplicações e sistemas. Dessa forma, os três pilares constituem a base essencial para a proteção eficaz dos ativos informacionais de qualquer organização.

2.1.2 Honeypots

Um *honeypot* é uma tecnologia associada a um sistema computacional que simula um ativo cibernético. Esse sistema possui vulnerabilidades de segurança conhecidas ou

configurações padrão. Seu objetivo é auxiliar na detecção de novos ataques e rastrear a atividade de um usuário não autorizado. O valor de um *honeypot* está relacionado à capacidade de simular cenários reais para capturar novas evidências e fornecer alertas e previsões (SPITZNER, 2003). *Honeypots* reduzem os falsos positivos em segurança, assim, isso permite que cada ambiente seja customizado para identificar os riscos de ataques dos grupos *APTs* (do inglês *Advanced Persistent Threats*). Além disso, de acordo com Javadpour et al. (2024), o *honeypot* é uma tecnologia flexível que pode ser adaptada para diferentes segmentos da computação. Destacam-se duas classificações principais de *honeypots*:

***Honeypot* de baixa interatividade:** esse tipo de *honeypot* é associado a sistemas altamente controlados que demandam poucos recursos. A interação com o usuário é limitada a comandos e controles básicos, simulando apenas serviços e protocolos.

***Honeypot* de alta interatividade:** essa modalidade de *honeypot* utiliza sistemas reais, geralmente estruturados para um ativo cibernético específico. Ele demanda maior capacidade computacional, permitindo que o atacante tenha controle total sobre o sistema, uma vez que simula um sistema operacional completo. Para implantar essa modalidade, é essencial adotar técnicas rigorosas de isolamento para proteger a rede principal.

No âmbito das tecnologias de *honeypots*, o *honeypot Cowrie* destaca-se como uma ferramenta de código aberto relevante, desenvolvida em *Python*, que emula serviços *SSH* e *Telnet* para atrair e registrar tentativas de invasão (OOSTERHOF, 2024). O estudo de Cabral et al. (2021) evidencia sua eficácia na coleta de dados sobre vetores de ataque contemporâneos, permitindo aos pesquisadores analisar técnicas maliciosas sem comprometer sistemas reais. No que concerne à sua integração com o sistema de gerenciamento de banco de dados *MySQL*, o *Cowrie* utiliza um módulo específico que, quando configurado no arquivo de configuração principal, estabelece conexão com o banco de dados e automaticamente cria tabelas estruturadas para armazenamento sistemático dos dados capturados. Entre as principais tabelas, representadas na Figura 1, geradas pelo sistema, destacam-se: *sessions* (registra informações básicas de cada sessão estabelecida), *auth* (armazena tentativas de autenticação), *input* (registra comandos inseridos pelos atacantes), *downloads* (contém informações sobre arquivos baixados durante as sessões) e *ttylog* (armazena registros completos da interação do terminal). Esta estruturação em banco de dados relacional facilita significativamente análises quantitativas e qualitativas posteriores dos padrões de ataque observados.

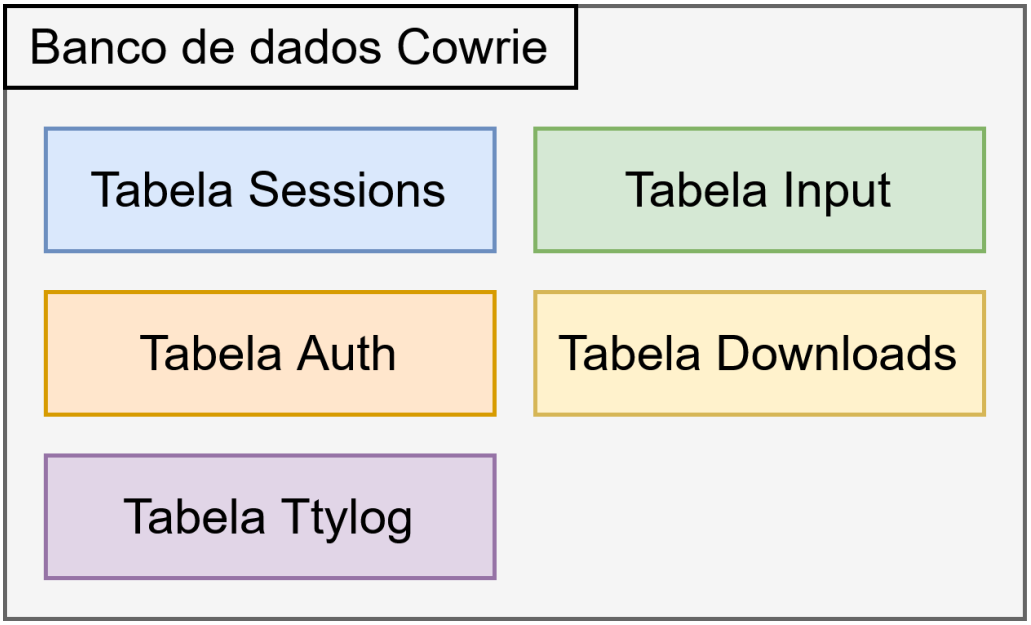


Figura 1 – Diagrama das principais tabelas do banco de dados do *Cowrie*. Fonte: Elaborado pelo autor (2025)

2.1.3 Python

Python é uma linguagem de programação amplamente reconhecida por sua facilidade de escrita e leitura de códigos. Essa característica possibilita que desenvolvedores atuem de maneira mais eficaz na criação e integração de sistemas. Sua sintaxe simples oferece uma ampla variedade de tipos e estruturas de dados, favorecendo a construção de soluções eficientes. Além disso, a linguagem destaca-se pelo extenso conjunto de bibliotecas desenvolvidas por colaboradores ao longo dos anos, o que amplia as possibilidades de criação de *scripts* rápidos e flexíveis (MALLOY; POWER, 2017). Portanto, devido a essas vantagens, a linguagem de programação escolhida para este trabalho será *Python*.

2.1.4 MySQL

O *MySQL* é um sistema de gerenciamento de banco de dados relacional de código aberto. Segundo Suehring (2002), desempenha um papel fundamental no armazenamento, gerenciamento e recuperação de dados, sendo amplamente utilizado em diversos tipos de aplicações.

Baseado em um modelo relacional, o *MySQL* organiza os dados em tabelas separadas com relacionamentos entre elas, o que proporciona maior eficiência no processamento das informações. A linguagem utilizada é o *Structured Query Language (SQL)*, padrão amplamente adotado para bancos de dados relacionais. Seu principal diferencial reside na facilidade e flexibilidade de implementação. O *MySQL* opera eficientemente no modelo cliente-servidor e destaca-se pela alta velocidade, confiabilidade e facilidade de uso.

2.1.5 Docker

Segundo Bernstein (2014), o *Docker* é uma plataforma de código aberto que fornece um método para automatizar a criação e execução de aplicações em containers portáteis. Essencialmente, o *Docker* isola completamente o ambiente da aplicação do sistema operacional hospedeiro, o que garante maior controle sobre a execução e a portabilidade do software.

Os containers do *Docker* são instâncias criadas a partir de imagens, que podem conter desde sistemas operacionais completos até partes específicas de uma aplicação já configuradas para execução. Essa abordagem possibilita a replicação precisa de ambientes computacionais. A construção dessas imagens é realizada por meio de arquivos chamados *Dockerfiles*, que consistem em *scripts* que possuem conjuntos de instruções e argumentos. Esses arquivos automatizam a criação de novas imagens a partir de uma base predefinida, permitindo a configuração e a customização do ambiente conforme necessário. Dessa forma, os *Dockerfiles* desempenham um papel fundamental na organização dos artefatos de implantação e na simplificação do processo de distribuição das aplicações.

De acordo com Ibrahim, Sayagh e Hassan (2021), grande parte das aplicações modernas compõe-se de múltiplos componentes, o que exige a coordenação de diversos containers para seu funcionamento. Por exemplo, uma aplicação pode necessitar simultaneamente de um servidor web e de um banco de dados. O *Docker Compose* surge como uma extensão natural do *Docker*, o qual permite a definição e a orquestração de aplicações multicontainer. Essa configuração descreve-se em um arquivo de composição (*Docker Compose file*), no qual especificam-se as imagens utilizadas, suas configurações e as interações entre os componentes. Além disso, o *Docker Compose* possibilita a definição de ações a serem executadas em caso de falha de um dos componentes, o que torna a administração da infraestrutura mais eficiente.

2.2 Trabalhos Correlatos

O trabalho de Mehta et al. (2021) utilizou o *honeypot Cowrie*, cujo principal objetivo é atrair ataques direcionados aos serviços *SSH* e *FTP*, para coletar e analisar informações sobre invasores. O *honeypot* adota técnicas que iludem os atacantes ao oferecer respostas aparentemente reais permitindo a interação do usuário. Além de apresentar uma análise abrangente dos dados coletados, o estudo propôs uma abordagem probabilística capaz de calcular a probabilidade de padrões de comportamento durante a navegação de diretórios pelos invasores. Essa abordagem, integrada às funcionalidades do *honeypot*, contribui para uma melhor compreensão da dinâmica dos ataques e auxilia na previsão do próximo movimento do atacante. O trabalho também utilizou o framework *ELK* para visualizar os dados extraídos dos *logs*. Para a análise probabilística, os autores compilaram

uma lista contendo todas as *strings* de comandos executados pelos usuários e estabeleceram dois dicionários. O primeiro registra a frequência de ocorrência de cada nome de diretório, enquanto o segundo, um dicionário bidimensional que armazena as transições entre os diretórios e suas respectivas frequências. A partir desses dados, a probabilidade de padrões de transição foi calculada por meio do método frequentista, dividindo o número de acessos a um diretório específico pelo total de ocorrências. Os resultados revelaram que os países com maior número de ataques originados são China, Estados Unidos, Rússia, Taiwan e Japão. Além disso, o usuário *admin* foi utilizado em 67% das tentativas, sendo que 54% delas não envolveram o uso de senha. Na análise probabilística, destacou-se que o diretório */bin/busybox* apresentou uma probabilidade de 10% de ser acessado, configurando-se como o destino mais provável dentro dos padrões analisados.

Outro trabalho relacionado é o de Năstase et al. (2024), o qual tem como objetivo aprimorar o *honeypot Cowrie*. O estudo apresenta contribuições significativas tanto na compreensão da arquitetura do *Cowrie* quanto na implementação de melhorias no código-fonte. Ele descreve dois modos de operação do *Cowrie*: o *Emulated Shell Mode*, o qual permite ao atacante executar comandos básicos com interações limitadas, e o *Proxy Mode*, o qual oferece interação completa com a máquina, dessa forma permite ao atacante assumir o controle total do sistema. Além disso, o trabalho propõe uma camada de *frontend*, a qual atua como um *proxy* para interceptar mensagens do protocolo *SSH*, enquanto a camada de *backend* recebe as conexões e opera como servidor. Entre os problemas identificados, destaca-se o uso excessivo de recursos computacionais, causado pela inicialização simultânea de todas as máquinas virtuais pelo *backend*, o que sobrecarrega os núcleos do processador. Para mitigar esse problema, implementou-se um intervalo de 30 segundos na função de criação de máquinas virtuais. Outro problema enfrentado foi a limitação de uma conexão por máquina virtual. Quando todas as máquinas estavam em uso, o *Cowrie* interpretava incorretamente o estado do sistema e tentava desligar todas as máquinas virtuais. A solução consistiu em remover a verificação de uma variável que verificava a disponibilidade das máquinas durante a sequência de inicialização. Por fim, o trabalho também aprimorou a análise dos artefatos gerados, e assim introduziu um evento de *log* que associa o *snapshot* a uma sessão, o qual facilita o processo de correlação que anteriormente não era suportado.

O experimento conduzido por Cabral et al. (2021) explorou o uso de três *honeypots Cowrie* configurados de maneiras distintas. *Honeypots* com configurações padrão mostram-se mais suscetíveis à detecção por atacantes, os quais frequentemente utilizam *scripts* personalizados e ferramentas amplamente conhecidas, como *Nmap* e *Shodan*, para identificá-los, especialmente devido ao fato de o *Cowrie* ser um projeto de código aberto. A pesquisa buscou, portanto, desenvolver uma estrutura a qual permitisse a personalização das configurações do *Cowrie*, com o objetivo de aumentar seu grau de realismo e eficácia. No experimento, o *Cowrie A* configurou-se de forma padrão, o qual utilizou as

portas *SSH* 22 e 2222, bem como versões e algoritmos conhecidos. Em contrapartida, os *Cowries* B e C configuraram-se de maneira personalizada, o que dificultou a detecção por meio da inclusão de outros protocolos, como *HTTP* e *DNS*, e o uso de algoritmos alternativos para *SSH*. Após 28 dias de coleta de dados, o *Cowrie* A registrou 49.340 conexões *SSH*, enquanto os *Cowries* B e C apresentaram um incremento de 356% e 400%, respectivamente, no número de conexões em comparação com o *Cowrie* A. Por fim, a pesquisa desenvolveu um *script* em *Python* para avaliar se um *host* específico era um *honeypot* *Cowrie*. Durante os testes, o *script* detectou o *Cowrie* A como *honeypot* com 100% de precisão, mas não conseguiu identificar os *Cowries* B e C como *honeypots*, o que reforçou a eficácia das configurações personalizadas na camuflagem desses sistemas.

3 Desenvolvimento

Neste Capítulo, apresenta-se o processo de desenvolvimento do *CowrieSniffer*[1]. Na Seção 3.1, é fornecida uma visão abrangente da aplicação. Em seguida, a Seção 3.2 detalha os aspectos relacionados à configuração do banco de dados de monitoramento, enquanto a Seção 3.3 descreve o arquivo de configuração utilizado para o armazenamento de parâmetros essenciais. A Seção 3.4 trata do desenvolvimento do componente principal, responsável pela orquestração das funcionalidades, e a Seção 3.5 aborda a construção do módulo de configuração.

Além disso, as Seções 3.6 e 3.7 explicam os componentes responsáveis pela manipulação dos bancos de dados do *Cowrie* e de monitoramento, respectivamente. Por fim, a Seção 3.8 apresenta o desenvolvimento do componente *URLMonitor*, encarregado da execução periódica dos demais componentes.

3.1 Visão Geral

A ferramenta é constituída por componentes que interagem entre si com o propósito de monitorar a disponibilidade da *URL* associada ao atacante, partindo do pressuposto de que este já se encontra conectado ao *honeypot*. A motivação para essa monitoração contínua decorre da volatilidade característica das infraestruturas cibernéticas maliciosas, as quais frequentemente são desativadas ou modificadas em curtos períodos após sua utilização inicial. A coleta de *URLs*, no ambiente controlado do *honeypot*, possibilita a identificação e análise de repositórios de *malwares*, servidores de comando e controle (*C&C*) ou plataformas de exfiltração de dados antes que sejam desativadas, constituindo, assim, um recurso valioso para pesquisadores de segurança e analistas forenses. Além disso, esse monitoramento ininterrupto permite o estabelecimento de correlações temporais entre diferentes campanhas de ataque, contribuindo para a compreensão da evolução tática dos agentes maliciosos e para o desenvolvimento de contramedidas mais eficazes no âmbito da segurança cibernética.

A Figura 2 ilustra o funcionamento da coleta de informações realizada pelo *honeypot Cowrie*. Inicialmente, quando o atacante realiza o *download* de um arquivo por meio de um comando, como *wget* ou *curl*, a *URL* correspondente é armazenada nas tabelas *downloads* e *input*. A tabela *downloads* armazena informações como identificadores, identificadores de sessões, *timestamps*, *URL* do servidor de origem, caminho do arquivo e *hash*. Por sua vez, a tabela *input* registra dados como identificadores, identificadores de

[1] Disponível em https://github.com/r1beirin/CowrieSniffer

sessões, *timestamps*, *realm*, *success* e o campo *input*, que contém os *keystrokes* do usuário.

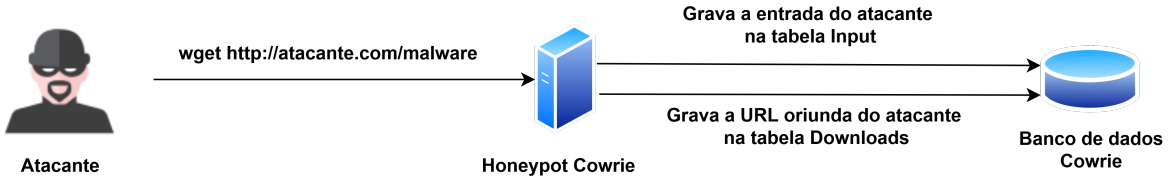


Figura 2 – Diagrama de coleta *Cowrie*. Fonte: Elaborado pelo autor (2025)

O diagrama apresentado na Figura 3 ilustra a visão geral da ferramenta desenvolvida, destacando seu fluxo de funcionamento e as principais decisões implementadas para o monitoramento das *URLs* capturadas pelo *Cowrie*. A ferramenta inicia sua execução e coleta as *URLs* armazenadas nas tabelas *downloads* e *input* do banco de dados do *Cowrie*. Em seguida, o componente *URLMonitor* processa essas *URLs* e insere as novas entradas no banco de dados de monitoramento. Após a inserção, o mesmo componente verifica a disponibilidade das *URLs* e atualiza as informações no banco de dados correspondente. Por fim, o ciclo reinicia-se, o que assegura a continuidade do monitoramento.

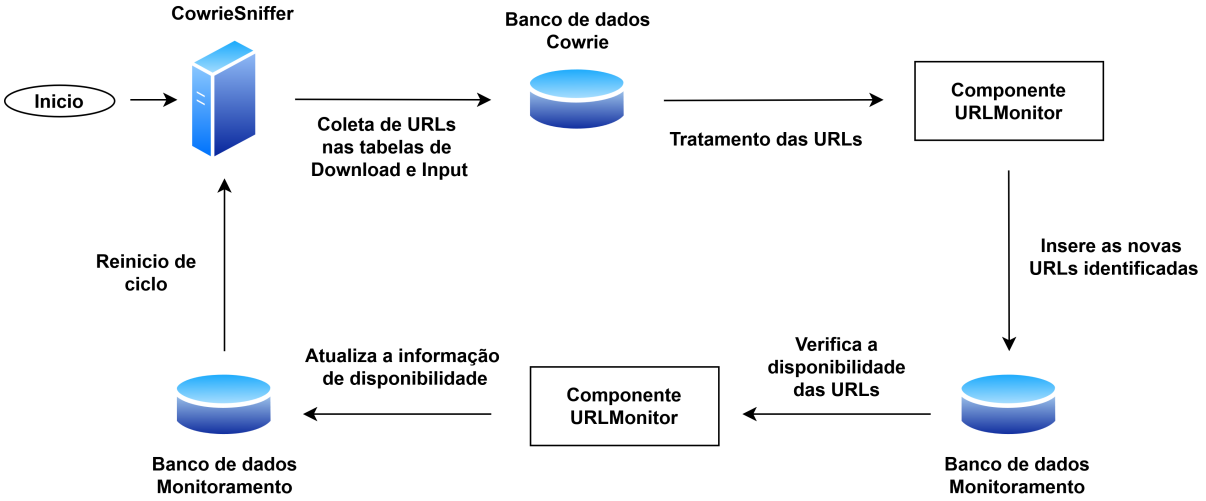


Figura 3 – Diagrama geral do *CowrieSniffer*. Fonte: Elaborado pelo autor (2025)

3.2 Configuração do banco de dados de monitoramento

O funcionamento do banco de dados de monitoramento é baseado na tecnologia *Docker*. Para sua criação, é necessária a definição de dois arquivos essenciais:

- ***schema.sql***: Responsável pela estrutura do banco de dados, este arquivo contém os comandos *SQL* necessários para definir as tabelas e seus respectivos atributos.

Conforme indicado na Listagem 3.1, o arquivo *schema.sql* define a estrutura do banco de monitoramento. No contexto do monitoramento de *URLs*, ele cria a tabela *urls*, que armazena informações essenciais sobre os acessos registrados.

```
1   CREATE TABLE urls (
2       id          INT AUTO_INCREMENT PRIMARY KEY,
3       url         TEXT NOT NULL,
4       first_view  DATETIME NULL,
5       last_view   DATETIME NULL
6   );
```

Listagem 3.1 – Estrutura para a tabela *urls* no banco de dados de monitoramento.

- ***docker-compose.yaml***: Responsável por especificar a configuração e orquestração dos serviços no ambiente *Docker*. Como ilustrado na Listagem 3.2, o arquivo *docker-compose.yaml* orquestra a configuração e o funcionamento do banco de monitoramento. Neste caso, ele define um serviço chamado *db*, que utiliza a imagem oficial do *MySQL* e configura um contêiner chamado *mysql_monitor*. O *docker-compose* também define:
  - **Reinicialização automática** do banco de dados com a opção *restart: always*.
  - **Configuração de credenciais** (*MYSQL_ROOT_PASSWORD*, *MYSQL_DATABASE*, *MYSQL_USER* e *MYSQL_PASSWORD*) que devem ser trocadas.
  - **Mapeamento de portas**, permitindo o acesso ao banco de dados via porta *3306*.
  - **Volumes persistentes**, garantindo que os dados armazenados no *MySQL* não sejam perdidos entre reinicializações.
  - **Montagem do arquivo *schema.sql***, que possibilita a criação da estrutura do banco de dados durante a inicialização do contêiner.
  - **Definição de redes *Docker***, assegurando a comunicação entre containers dentro de um ambiente isolado.

```
1   services:
2     db:
3       image: mysql:latest
4       container_name: mysql_monitor
5       restart: always
6       environment:
7         MYSQL_ROOT_PASSWORD: {CHANGE_HERE}
8         MYSQL_DATABASE: {CHANGE_HERE}
9         MYSQL_USER: {CHANGE_HERE}
10        MYSQL_PASSWORD: {CHANGE_HERE}
```

```
11        ports:
12          - "3306:3306"
13        volumes:
14          - db_data:/var/lib/mysql
15          - ./schema.sql:/docker-entrypoint-initdb.d/schema.sql
16        networks:
17          - my_network
18
19    volumes:
20      db_data:
21        driver: local
22
23    networks:
24      my_network:
25        driver: bridge
```

Listagem 3.2 – Configuração do container no *docker-compose.yaml* para inicialização do banco de dados *MySQL* de monitoramento.

Essa abordagem baseada em container facilita a implantação e manutenção do banco de dados. Dessa forma, garante-se a existência de um ambiente controlado para o monitoramento das *URLs* coletadas pelo *Cowrie*.

3.3 Desenvolvimento do arquivo de configuração

O arquivo de configuração é responsável por guardar as informações do banco de dados do *Cowrie* e do monitoramento. Este arquivo contém as configurações necessárias para a conexão com os bancos de dados utilizados no monitoramento. Ele é dividido em duas seções:

- ***CowrieDB***: Gerencia as informações de acesso ao banco de dados do *Cowrie*. Esse banco é utilizado para armazenar as informações, como *downloads*, *inputs*, *keystrokes*, do *honeypot Cowrie*.
- ***MonitoringDB***: Guarda as informações de acesso ao banco de dados. Esse banco é utilizado para armazenar as informações de monitoramento.

Como ilustrado na Listagem 3.3, o arquivo *config.ini* contém as informações para ambas as bases de dados. Dessa forma, garante a conexão adequada entre o *CowrieSniffer* e os bancos de dados.

```
1   [CowrieDB]
2   host = YOUR_HOST
3   user = YOUR_USER
```

```
4     password = YOUR_PASSWORD
5     database = YOUR_DATABASE
6
7     [MonitoringDB]
8     host = YOUR_HOST
9     user = YOUR_USER
10    password = YOUR_PASSWORD
11    database = YOUR_DATABASE
```

Listagem 3.3 – Arquivo de configuração *config.ini* para as conexões com os bancos de dados.

3.4 Desenvolvimento do componente *Main*

O arquivo principal do sistema é responsável por inicializar as configurações e instâncias das classes necessárias para o funcionamento do sistema de monitoramento. O diagrama de pacotes ilustrado na Figura 4 evidencia a organização estrutural hierárquica do sistema, destacando a modularização e as relações de dependência entre os diversos pacotes da ferramenta. A arquitetura apresenta uma configuração em camadas onde o módulo principal *main.py* desempenha função orquestradora, realizando a importação e integração dos pacotes especializados: *Config* para o gerenciamento de configurações; *database*, que encapsula os manipuladores de persistência específicos, como *CowrieDBHandler* e *MonitoringDBHandler*; e *monitoring*, contendo o componente *URLMonitor*, responsável pelo ciclo de execução.

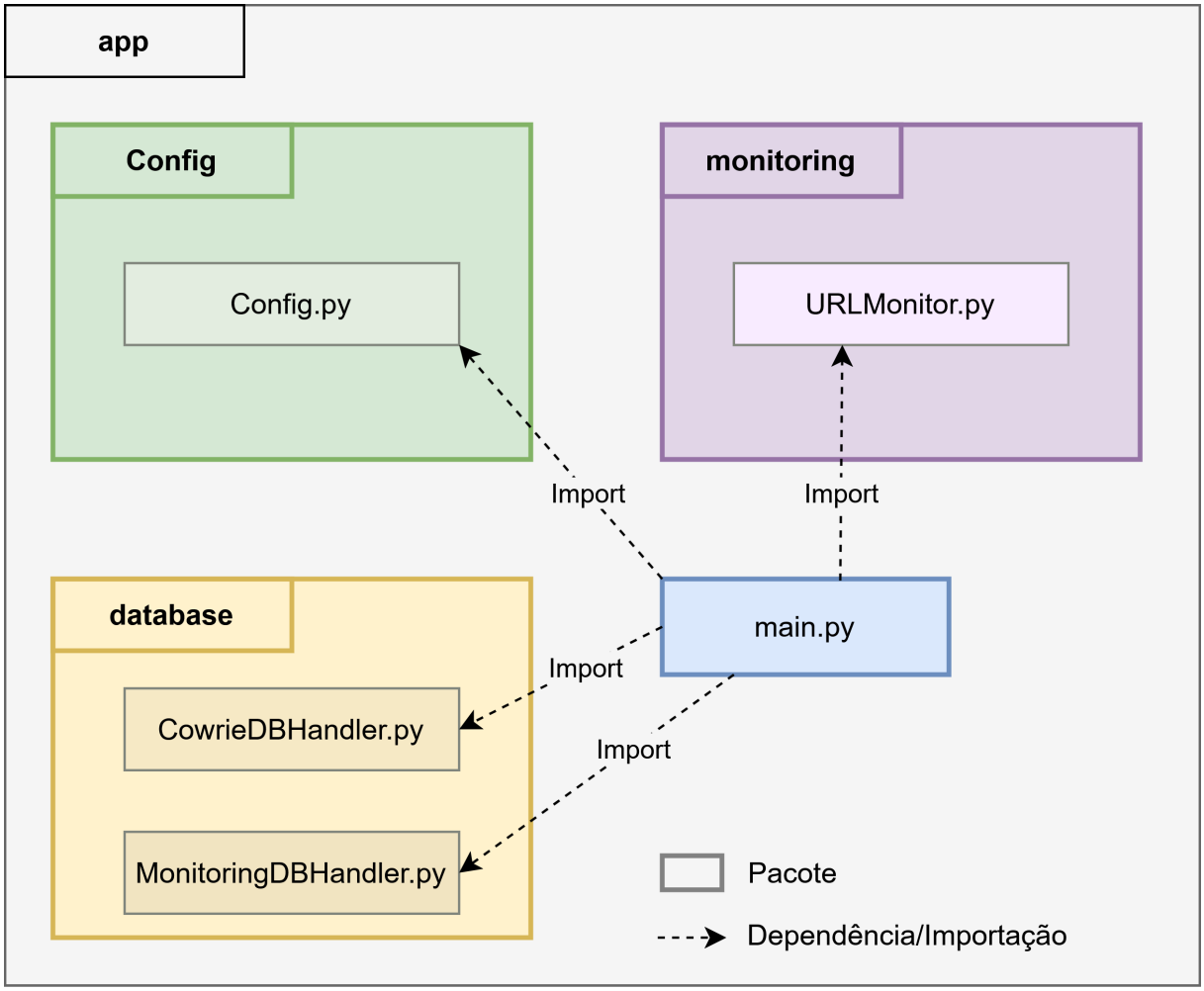


Figura 4 – Diagrama de pacotes da classe *Main*. Fonte: Elaborado pelo autor (2025)

A Listagem 3.4 apresenta o componente principal, que tem como objetivo inicializar e orquestrar os processos essenciais para o funcionamento do sistema. Inicialmente, a classe *Config* é instanciada, seu objeto acessa configurações do sistema, especificamente as credenciais e detalhes de conexão dos dois bancos de dados. Com essas informações, o fluxo de execução instancia objetos das classes *CowrieDBHandler* e *MonitoringDBHandler*, responsáveis por gerenciar as interações com os respectivos bancos de dados. Em seguida, a classe *URLMonitor*, que gerencia o ciclo de execução dos componentes é instanciada e inicia sua tarefa de coleta e monitoramento. Esse fluxo garante que o sistema de monitoramento seja configurado corretamente, com as conexões adequadas aos bancos de dados, e que o *URLMonitor* inicie suas atividades de monitoramento conforme o esperado.

```
1    def main():
2        config = Config()
3        cowrie_db_config = config.get_cowrie_db_config()
```

```
4         monitoring_db_config = config.get_monitoring_db_config()
5
6         cowrie_db = CowrieDBHandler(cowrie_db_config)
7         monitoring_db = MonitoringDBHandler(monitoring_db_config)
8
9         monitor = URLMonitor(cowrie_db, monitoring_db)
10        monitor.start()
11
12    if __name__ == '__main__':
13        main()
```

Listagem 3.4 – Componente principal (*main.py*) do sistema de monitoramento.

3.5 Desenvolvimento do componente de configuração

A classe do componente de configuração, *Config*, é responsável por ler e fornecer as configurações, extraídas do arquivo *config.ini*, para a conexão com os bancos de dados. Como mostrado na Listagem 3.5, a classe *Config* contém dois métodos *get_cowrie_db_config()* e *get_monitoring_db_config()*, que retornam um dicionário com as configurações de cada banco de dados, como o *host*, usuário, senha e nome do banco.

```
1     class Config:
2         def get_cowrie_db_config(self):
3             return {
4                 'host': self.config.get('CowrieDB', 'host'),
5                 'user': self.config.get('CowrieDB', 'user'),
6                 'password': self.config.get('CowrieDB', 'password'),
7                 'database': self.config.get('CowrieDB', 'database')
8             }
9
10        def get_monitoring_db_config(self):
11            return {
12                'host': self.config.get('MonitoringDB', 'host'),
13                'user': self.config.get('MonitoringDB', 'user'),
14                'password': self.config.get('MonitoringDB', 'password'),
15                'database': self.config.get('MonitoringDB', 'database')
16            }
```

Listagem 3.5 – Componente *Config.py*, responsável pela leitura das configurações do sistema.

3.6 Desenvolvimento do componente de manipulação do banco de dados do *Cowrie*

O componente de manipulação do banco de dados do *Cowrie* é responsável por gerenciar a conexão e execução de consultas no banco de dados *Cowrie*. Esse componente utiliza a biblioteca *mysql.connector* para conectar-se ao banco de dados *MySQL* e realizar as operações necessárias.

A Listagem 3.6 apresenta a classe *CowrieDBHandler*, responsável por estabelecer a conexão com o banco de dados e executar consultas para recuperar dados específicos, como *URLs* e entradas do banco de dados do *Cowrie*. Os métodos *get_urls_cowrie()* e *get_inputs_cowrie()* realizam consultas *SQL* para retornar dados das tabelas *downloads* e *input*, respectivamente.

```
1class CowrieDBHandler:
2    def __init__(self, configDB):
3        self.configDB = configDB
4
5    @contextlib.contextmanager
6    def get_connection(self):
7        connection = None
8        cursor = None
9        try:
10           max_retries = 5
11           for retry in range(max_retries):
12               try:
13                   connection = mysql.connector.connect(**self.configDB
    )
14                   cursor = connection.cursor(dictionary=True, buffered
    =True)
15                   break
16               except Error as e:
17                   print(f"Error connecting to database (attempt {retry
    +1}/{max_retries}): {e}")
18                   if retry == max_retries - 1:
19                       raise
20                   time.sleep(2)
21
22           yield cursor, connection
23
24       finally:
25           if cursor:
26               cursor.close()
27           if connection and connection.is_connected():
28               connection.close()
```

```
29
30     def get_urls_cowrie(self):
31         try:
32             with self.get_connection() as (cursor, connection):
33                 query = "SELECT DISTINCT url FROM downloads ORDER BY url
    "
34                 cursor.execute(query)
35                 return cursor.fetchall()
36
37         except Error as error:
38             print(f"Error executing query on CowrieDB: {error}")
39             return []
40
41     def get_inputs_cowrie(self):
42         try:
43             with self.get_connection() as (cursor, connection):
44                 query = "SELECT DISTINCT input FROM input ORDER BY input
    "
45                 cursor.execute(query)
46                 return cursor.fetchall()
47
48         except Error as error:
49             print(f"Error executing query on CowrieDB: {error}")
50             return []
```

Listagem 3.6 – Componente *CowrieDBHandler.py*, responsável pela conexão e execução de consultas no banco de dados Cowrie.

3.7 Desenvolvimento do componente de manipulação do banco de dados de Monitoramento

A classe *MonitoringDBHandler*, representada na Listagem 3.7, é responsável pela interação com o banco de dados de monitoramento. Ela estabelece a conectividade, insere novas *URLs* para monitoramento, atualiza as informações de disponibilidade e realiza consultas para obter as *URLs* que estão sendo monitoradas. A classe contém os seguintes métodos:

- ***url_monitoring_is_empty()***: verifica se o banco de dados está vazio. Retorna *True* em caso afirmativo e *False* caso contrário.
- ***update_last_view(url)***: atualiza o campo *last_view* para uma determinada *URL*. Esse campo indica a última vez em que a *URL* foi checada.
- ***insert_url(url)***: insere uma nova *URL* no banco de dados. A inserção ocorre apenas caso ela ainda não exista.

- ***get_urls_monitoring()***: retorna uma lista contendo as *URLs* monitoradas. Essa lista reúne todas as *URLs* que estão sendo acompanhadas pelo sistema.

Essa classe é essencial para o gerenciamento das *URLs* monitoradas, pois realiza inserções e atualizações no banco de dados. As operações de inserção e atualização asseguram que os dados de monitoramento permaneçam atualizados, enquanto a consulta das *URLs* monitoradas facilita a análise do comportamento no âmbito do *CowrieSniffer*.

```
class MonitoringDBHandler:
    def __init__(self, configDB):
        self.configDB = configDB

    @contextlib.contextmanager
    def get_connection(self):
        connection = None
        cursor = None
        try:
            max_retries = 5
            for retry in range(max_retries):
                try:
                    connection = mysql.connector.connect(**self.configDB
)
                    cursor = connection.cursor(dictionary=True, buffered
=True)
                    break
                except Error as e:
                    print(f"Error connecting to database (attempt {retry
+1}/{max_retries}): {e}")
                    if retry == max_retries - 1:
                        raise
                    time.sleep(2)

            yield cursor, connection

        finally:
            if cursor:
                cursor.close()
            if connection and connection.is_connected():
                connection.close()

    def url_monitoring_is_empty(self):
        try:
            with self.get_connection() as (cursor, connection):
                cursor.execute("SELECT COUNT(*) AS count FROM urls")
                result = cursor.fetchone()
                return result['count'] == 0
```

```
36
37          except Error as error:
38              print(f"Error checking if urls table is empty: {error}")
39              return False
40
41      def update_last_view(self, url):
42          try:
43              with self.get_connection() as (cursor, connection):
44                  cursor.execute("SELECT url FROM urls WHERE url = %s", (
    url,))
45                  if not cursor.fetchone():
46                      print(f"URL '{url}' does not exist in monitoring.")
47                      return
48
49                  currentTime = datetime.now()
50                  sql = "UPDATE urls SET last_view = %s WHERE url = %s"
51                  cursor.execute(sql, (currentTime, url))
52                  connection.commit()
53
54                  print(f"Updated URL: {url}")
55
56          except Error as error:
57              print(f"Error updating URL: {error}")
58
59      def insert_url(self, url):
60          try:
61              with self.get_connection() as (cursor, connection):
62                  cursor.execute("SELECT url FROM urls WHERE url = %s", (
    url,))
63                  if cursor.fetchone():
64                      print(f"URL '{url}' already exists in monitoring.")
65                      return
66
67                  currentTime = datetime.now()
68                  sql = "INSERT INTO urls (url, first_view, last_view)
    VALUES (%s, %s, %s)"
69                  cursor.execute(sql, (url, currentTime, currentTime))
70                  connection.commit()
71
72                  print(f"Inserted URL: {url}")
73
74          except Error as error:
75              print(f"Error inserting URL: {error}")
76
77      def get_urls_monitoring(self):
78          try:
79              with self.get_connection() as (cursor, connection):
```

```
80                 query = "SELECT DISTINCT url FROM urls ORDER BY url"
81                 cursor.execute(query)
82                 return cursor.fetchall()
83
84         except Error as error:
85             print(f"Error getting URLs from monitoring: {error}")
86             return []
```

Listagem 3.7 – Componente *MonitoringDBHandler.py* responsável pela interação com o banco de dados de monitoramento.

3.8 Desenvolvimento do componente *URLMonitor*

A Figura 5 apresenta a visão geral do componente *URLMonitor*. Esse componente é responsável por todo o funcionamento da estratégia de coleta e verificação das informações no sistema.

Inicialmente, o sistema verifica se o banco de dados de monitoramento está vazio. Caso essa condição seja verdadeira, o banco de monitoramento é preenchido com as *URLs* coletadas na tabela *downloads* do banco de dados do *Cowrie*, e a variável *last_known_downloads_urls_cowrie_db* é atualizada. Se o banco de monitoramento não estiver vazio, realiza-se a sincronização dessa variável com os endereços presentes na tabela *downloads* do banco de dados do *Cowrie*.

Em sequência, o *CowrieSniffer* ingressa em um estado condicional de iteração contínua (*while true*) do qual só será possível sair mediante interrupção manual realizada pelo usuário. A etapa subsequente consiste na verificação da existência de novas *URLs* tanto na tabela *downloads* quanto na tabela *input*, ambas pertencentes ao *honeypot Cowrie*. Caso sejam identificados novas *URLs*, estas são inseridas no banco de dados de monitoramento, e as variáveis responsáveis pelo armazenamento das *URLs* são devidamente atualizadas para refletir o estado atual do sistema.

Por fim, o sistema realiza uma verificação de conexão, atualizando os endereços ativos no banco de dados de monitoramento. Essa estrutura cíclica garante que a ferramenta desenvolvida mantenha um monitoramento contínuo e atualizado das *URLs* registradas pelo *Cowrie*, facilitando a identificação e o acompanhamento dos servidores que hospedam *malwares*.

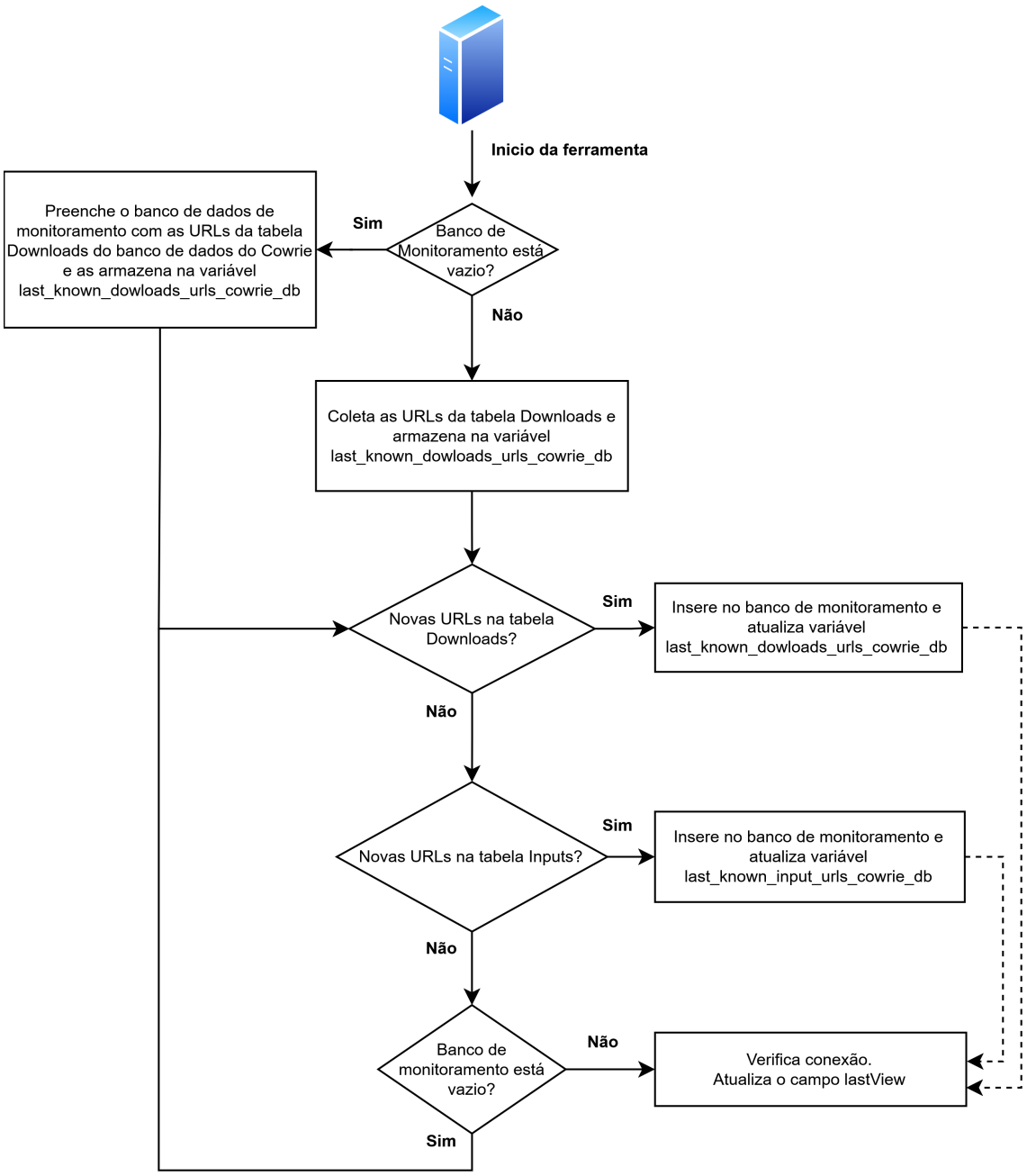


Figura 5 – Diagrama da visão geral do componente *URLMonitor*. Fonte: Elaborado pelo autor (2025)

O código representado na Listagem 3.8 é responsável pelas tarefas de extração, verificação e monitoramento. Esse componente constitui o núcleo funcional do *CowrieSniffer*, assegurando a coleta e a análise contínua das *URLs* acessadas por agentes maliciosos no ambiente de *honeypot*.

A classe *URLMonitor* mantém conexão com dois bancos de dados principais: o banco de dados do *Cowrie*, que armazena os registros de eventos do *honeypot*, e o banco de dados de monitoramento, que concentra os endereços identificados para acompanhamento. Além disso, a classe utiliza duas variáveis da estrutura de dados do tipo *set* para armazenar

os últimos registros conhecidos, evitando redundâncias nas novas entradas.

Os principais métodos implementados nesse componente são descritos a seguir. Cada um deles desempenha funções específicas que contribuem para o funcionamento integrado da ferramenta.

- ***extract_urls_from_log(log_entry)***: responsável pelo processamento e extração de *URLs* a partir dos *logs* do *Cowrie*. Para isso, emprega expressões regulares que identificam padrões de endereços na internet, garantindo uma extração precisa.
- ***verify_connections()***: verifica a conectividade das *URLs* armazenadas no banco de dados de monitoramento. Para isso, realiza tentativas de conexão *TCP* nos endereços extraídos, utilizando as portas padrão (443 para *HTTPS* e 80 para *HTTP*) ou uma porta específica. Caso a conexão seja bem-sucedida, a última data de verificação do endereço é atualizada no banco de dados.
- ***verify_from_input()***: monitora novos registros da tabela *input* no banco de dados do *Cowrie*, extraindo *URLs* presentes nesses comandos e inserindo-as no banco de dados de monitoramento para análise posterior.
- ***verify_download()***: analisa a tabela *downloads* do *Cowrie* para identificar *URLs* associadas a arquivos baixados durante sessões interativas dos atacantes. Caso novos endereços sejam encontrados, são adicionados à base de monitoramento.
- ***populate_urls_monitoring()***: executado durante a inicialização da ferramenta, esse método popula a base de monitoramento com as *URLs* previamente registradas no *Cowrie*, garantindo que a análise inicie com um conjunto inicial de endereços.
- ***run_periodic_tasks()***: principal responsável pela execução contínua das tarefas do *CowrieSniffer*. Em ciclos periódicos, o método verifica a presença de novos endereços nos registros do *Cowrie*, monitora mudanças nas tabelas *input* e *downloads*, testa a conectividade dos endereços armazenados e repete todo o processo, garantindo a atualização constante das informações.

Com essa abordagem, o *CowrieSniffer* proporciona um monitoramento eficiente e automatizado das *URLs* acessadas em um ambiente de *honeypot*. Tal funcionalidade possibilita a realização de análises mais aprofundadas sobre o comportamento de agentes maliciosos.

```
1    class URLMonitor:
2        def extract_urls_from_log(self, log_entry):
3            url_pattern = r'https?://[^\s<>\"]+|www\.[^\s<>\"]+'
4            urls = set(re.findall(url_pattern, log_entry))
5            cleaned_urls = set(url.rstrip(';') for url in urls)
```

```
6             return cleaned_urls
7
8         def verify_connections(self):
9             urls = self.monitoring_db.get_urls_monitoring()
10
11            # Regex for domain/IP and port (if present)
12            addrPattern = (
13                r'https?:\/\/'  # http or https
14                r'((?:\d{1,3}\.){3}\d{1,3}'  # IPv4 address
15                r'|'  # or
16                r'(?:[a-zA-Z0-9-]+\.)+[a-zA-Z]{2,})'  # domain name
17                r'(?::(\d+))?'  # optional port
18            )
19
20            for url in urls:
21                if url['url']:
22                    match = re.search(addrPattern, url['url'])
23                    if match:
24                        host = match.group(1)
25
26                        if match.group(2):
27                            port = int(match.group(2))
28                        else:
29                            if url['url'].startswith('https'):
30                                port = 443
31                            else:
32                                port = 80
33
34                        print(f"[VERIFYING CONNECTION] Testing {host}:{
    port}")
35
36                        try:
37                            with socket.create_connection((host, port),
    timeout=5) as s:
38                                print(f"[VERIFYING CONNECTION]
    Connection {host}:{port} successful")
39                                self.monitoring_db.update_last_view(url[
    'url'])
40                        except socket.error:
41                            print(f"[VERIFYING CONNECTION] Connection to
     {host}:{port} failed")
42                    else:
43                        print(f"[VERIFYING CONNECTION] Invalid URL
    format: {url['url']}")
44
45        def verify_from_input(self):
46            current_inputs = set(input['input'] for input in self.
```

```
    cowrie_db.get_inputs_cowrie())
47
48          try:
49              if current_inputs != self.
    last_known_input_urls_cowrie_db:
50                  print(f"[VERIFYING URL] Change Detect on Input on
    Cowrie!")
51
52                  for input in current_inputs:
53                      urls = set(self.extract_urls_from_log(input))
54                      if urls:
55                          for url in urls:
56                              try:
57                                  self.monitoring_db.insert_url(url)
58                              except Exception as e:
59                                  print(f"[VERIFYING URL] Error adding
     URL {url}: {str(e)}")
60
61                  self.last_known_input_urls_cowrie_db =
    current_inputs
62
63              else:
64                  print("[VERIFYING URL] No change on Input on Cowrie!
    ")
65
66          except Exception as e:
67              print(f"[VERIFYING URL] Error on verifying: {str(e)}")
68
69      def verify_download(self):
70          try:
71              current_cowrie_urls = set(url['url'] for url in self.
    cowrie_db.get_urls_cowrie())
72
73              if current_cowrie_urls != self.
    last_known_dowloads_urls_cowrie_db:
74                  print(f"[VERIFYING URL] Change Detect on Downloads
    on Cowrie!")
75                  monitoring_urls = set(url['url'] for url in self.
    monitoring_db.get_urls_monitoring())
76                  new_urls = current_cowrie_urls - monitoring_urls
77
78                  if new_urls:
79                      for url in new_urls:
80                          try:
81                              self.monitoring_db.insert_url(url)
82                          except Exception as e:
83                              print(f"[VERIFYING URL] Error adding URL
```

```
 {url}: {str(e)}")
84
85                      self.last_known_dowloads_urls_cowrie_db =
    current_cowrie_urls
86
87                  else:
88                      print("[VERIFYING URL] No change on Downloads on
    Cowrie!")
89
90              except Exception as e:
91                  print(f"[VERIFYING URL] Error on veryfying: {str(e)}")
92
93          def populate_urls_monitoring(self):
94              print("[INIT] Initializing population of monitoringDB")
95              urls = self.cowrie_db.get_urls_cowrie()
96              initial_urls = set(url['url'] for url in urls)
97              self.last_known_dowloads_urls_cowrie_db = initial_urls
98
99              for url in initial_urls:
100                 try:
101                     self.monitoring_db.insert_url(url)
102                 except Exception as e:
103                     print(f"[INIT] Error adding URL {url}: {str(e)}")
104
105         def run_periodic_tasks(self):
106             if self.monitoring_db.url_monitoring_is_empty():
107                 print("[INIT] MonitoringDB empty")
108                 self.populate_urls_monitoring()
109             else:
110                 self.last_known_dowloads_urls_cowrie_db = set(url['url']
     for url in self.cowrie_db.get_urls_cowrie())
111                 print(f"[INIT] State loaded: {self.
    last_known_dowloads_urls_cowrie_db}")
112
113             while True:
114                 print("[PERIODIC TASK] Running...")
115                 self.verify_download()
116                 self.verify_from_input()
117
118                 if not self.monitoring_db.url_monitoring_is_empty():
119                     self.verify_connections()
120
121                 print("[PERIODIC TASK] Finished!", end="\n\n")
122                 time.sleep(5)
123
124         def start(self):
```

```
125             self.run_periodic_tasks()
```

Listagem 3.8 – Componente *URLMonitor.py* responsável pelo ciclo de monitoramento

4 Resultados

Neste capítulo, são apresentados os estudos de caso realizados para avaliar o funcionamento do *CowrieSniffer* em diversos contextos operacionais. Os cenários selecionados foram metodicamente definidos com base no processo iterativo de desenvolvimento da ferramenta, sendo que cada caso representa um desafio técnico específico identificado e solucionado durante a implementação do *CowrieSniffer*.

A Seção 4.1 descreve o Caso 1, no qual ambos os bancos de dados, *Cowrie* e de monitoramento, encontram-se vazios. A Seção 4.2 aborda o Caso 2, que corresponde à inicialização da ferramenta com o banco de dados do *Cowrie* populado e o banco de monitoramento vazio. A Seção 4.3 trata do Caso 3, em que ambos os bancos de dados já estão populados no momento da inicialização. Por fim, a Seção 4.4 apresenta o Caso 4, cujo objetivo é analisar o comportamento da ferramenta diante da adição de novas *URLs*.

4.1 Caso 1: Inicialização da ferramenta com bancos de dados vazios

Este cenário considera a situação em que tanto o banco de dados do *Cowrie* quanto o banco de dados de monitoramento estão vazios. O objetivo desta análise é verificar o comportamento do *CowrieSniffer* durante sua inicialização em um ambiente sem registros prévios.

A Figura 6 apresenta a execução da ferramenta nesse contexto. Inicialmente, convém ressaltar que a implementação foi realizada em um ambiente controlado de rede local, caracterizando um cenário simulado, sem a necessidade de implantação em infraestrutura de nuvem. Neste ambiente, a ferramenta detecta que o banco de dados de monitoramento está vazio e inicia sua população, garantindo que, assim que novos registros forem adicionados ao *Cowrie*, eles possam ser capturados e monitorados. Esse cenário demonstra que, mesmo em uma configuração experimental circunscrita a uma rede local, a ferramenta mantém seu funcionamento contínuo, aguardando a inserção de novas *URLs*.

```
(env) PS C:\Users\ribeirin\Documents\DirectoryMonitor> python.exe .\main.py
[INIT] MonitoringDB empty
[INIT] Initializing population of monitoringDB
[PERIODIC TASK] Running...
[VERIFYING URL] No change on Downloads on Cowrie!
[VERIFYING URL] No change on Input on Cowrie!
[PERIODIC TASK] Finished!

[PERIODIC TASK] Running...
[VERIFYING URL] No change on Downloads on Cowrie!
[VERIFYING URL] No change on Input on Cowrie!
[PERIODIC TASK] Finished!

[PERIODIC TASK] Running...
[VERIFYING URL] No change on Downloads on Cowrie!
[VERIFYING URL] No change on Input on Cowrie!
[PERIODIC TASK] Finished!
```

Figura 6 – Caso 01: *Log* do funcionamento da ferramenta. Fonte: Elaborado pelo autor (2025)

Na Figura 7, verifica-se que ambos os bancos de dados apresentam suas respectivas tabelas sem registros. As duas primeiras consultas evidenciam que as tabelas *input* e *downloads* pertencentes ao banco de dados do *Cowrie* não contêm nenhum registro. Adicionalmente, observa-se que a tabela *urls* do banco de dados de monitoramento, representada pela terceira consulta, também se encontra desprovida de dados.

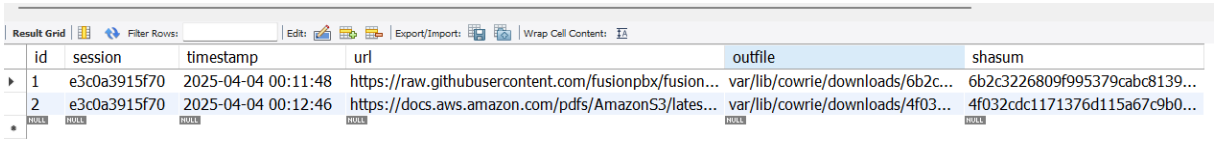
```
mysql> select * from cowrie.input;
Empty set (0.01 sec)

mysql> select * from cowrie.downloads;
Empty set (0.00 sec)

mysql>
mysql>
mysql>
mysql> select * from monitoringdb.urls;
Empty set (0.00 sec)

mysql>
mysql>
mysql>
```

Figura 7 – Caso 01: Tabelas *input* e *downloads* do *honeypot Cowrie* sem registros e tabela *urls* do banco de monitoramento desprovida de dados. Fonte: Elaborado pelo autor (2025)

4.2 Caso 2: Inicialização da ferramenta com o banco de dados do *Cowrie* populado e o banco de monitoramento vazio

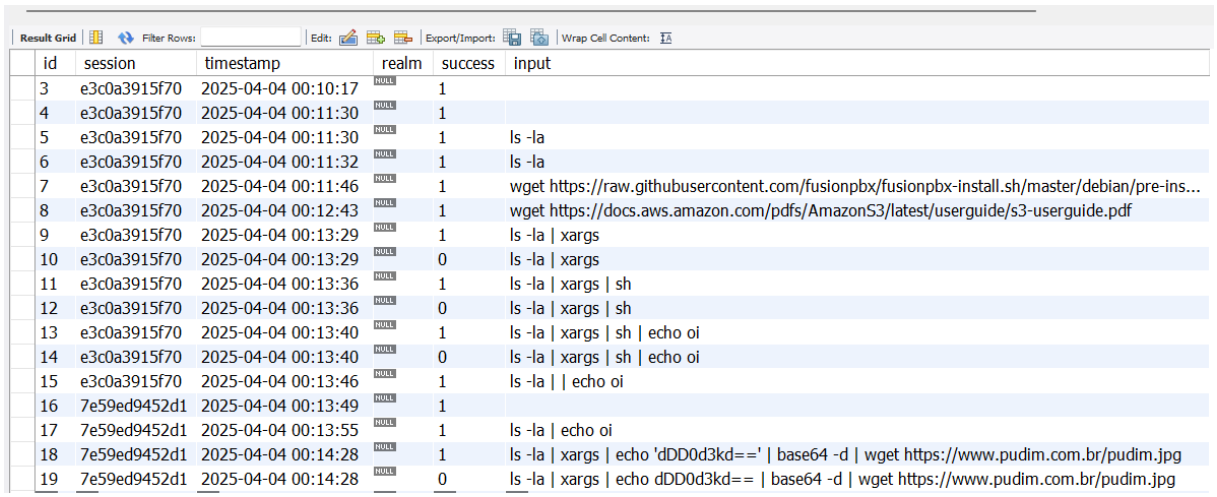
Neste estudo de caso, o banco de dados do *Cowrie* já contém *URLs* coletadas de sessões anteriores, enquanto o banco de dados de monitoramento permanece vazio. O foco desta análise é compreender como o *CowrieSniffer* processa os dados históricos do *Cowrie*, populando a base de monitoramento e assegurando que nenhuma informação relevante seja perdida durante a transição inicial.

No início do Caso 2, observa-se que as tabelas do banco de dados do *Cowrie* já possuem registros, conforme ilustrado nas Figuras 8 e 9. Em contrapartida, a Figura 10 apresenta o estado inicial da tabela *urls*, que se encontra vazia.



id	session	timestamp	url	outfile	shasum
1	e3c0a3915f70	2025-04-04 00:11:48	https://raw.githubusercontent.com/fusionpbx/fusion...	var/lib/cowrie/downloads/6b2c...	6b2c3226809f995379cabc8139...
2	e3c0a3915f70	2025-04-04 00:12:46	https://docs.aws.amazon.com/pdfs/AmazonS3/lates...	var/lib/cowrie/downloads/4f03...	4f032cdc1171376d115a67c9b0...

Figura 8 – Caso 02: Tabela *downloads* do *Cowrie* com entradas. Fonte: Elaborado pelo autor (2025)



id	session	timestamp	realm	success	input
3	e3c0a3915f70	2025-04-04 00:10:17		1	
4	e3c0a3915f70	2025-04-04 00:11:30		1	
5	e3c0a3915f70	2025-04-04 00:11:30		1	ls -la
6	e3c0a3915f70	2025-04-04 00:11:32		1	ls -la
7	e3c0a3915f70	2025-04-04 00:11:46		1	wget https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-ins...
8	e3c0a3915f70	2025-04-04 00:12:43		1	wget https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
9	e3c0a3915f70	2025-04-04 00:13:29		1	ls -la | xargs
10	e3c0a3915f70	2025-04-04 00:13:29		0	ls -la | xargs
11	e3c0a3915f70	2025-04-04 00:13:36		1	ls -la | xargs | sh
12	e3c0a3915f70	2025-04-04 00:13:36		0	ls -la | xargs | sh
13	e3c0a3915f70	2025-04-04 00:13:40		1	ls -la | xargs | sh | echo oi
14	e3c0a3915f70	2025-04-04 00:13:40		0	ls -la | xargs | sh | echo oi
15	e3c0a3915f70	2025-04-04 00:13:46		1	ls -la | | echo oi
16	7e59ed9452d1	2025-04-04 00:13:49		1	
17	7e59ed9452d1	2025-04-04 00:13:55		1	ls -la | echo oi
18	7e59ed9452d1	2025-04-04 00:14:28		1	ls -la | xargs | echo 'dDD0d3kd==' | base64 -d | wget https://www.pudim.com.br/pudim.jpg
19	7e59ed9452d1	2025-04-04 00:14:28		0	ls -la | xargs | echo dDD0d3kd== | base64 -d | wget https://www.pudim.com.br/pudim.jpg

Figura 9 – Caso 02: Tabela *input* do *Cowrie* com entradas. Fonte: Elaborado pelo autor (2025)

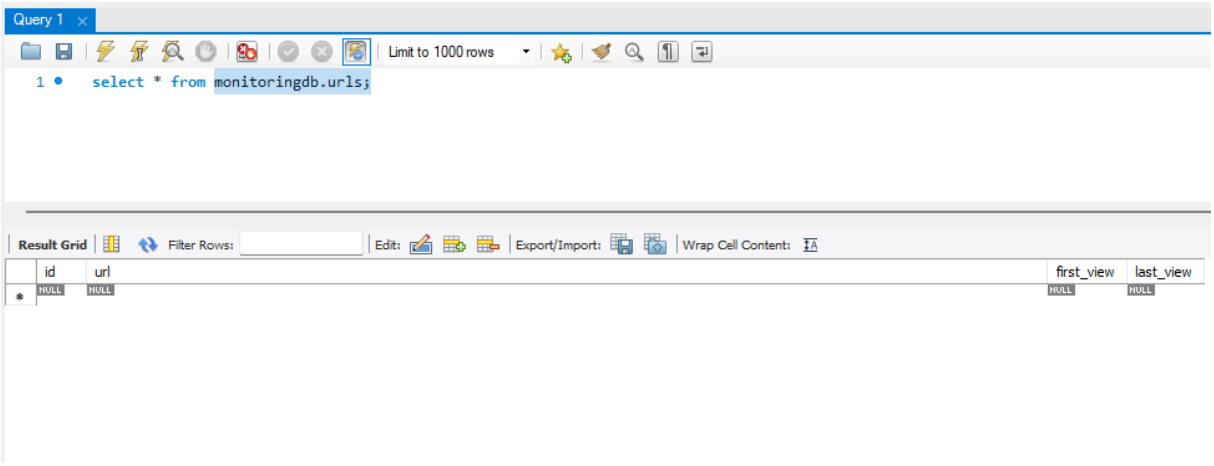


Figura 10 – Caso 02: Tabela *urls* do monitoramento inicialmente vazia. Fonte: Elaborado pelo autor (2025)

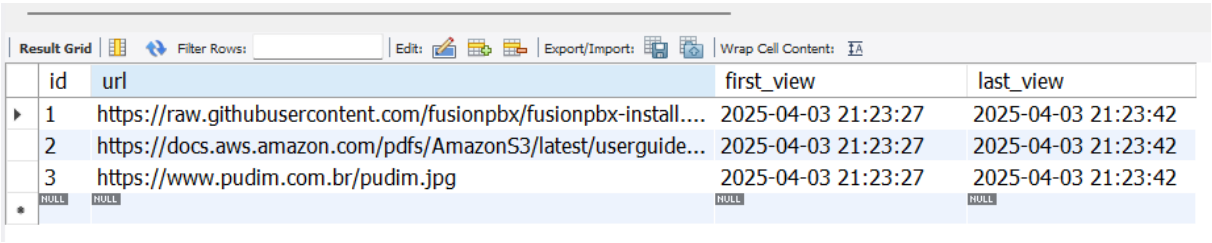
Após o início da execução do *CowrieSniffer*, conforme ilustrado na Figura 11, a ferramenta inicia a população da tabela *urls* do banco de dados de monitoramento a partir dos registros existentes na tabela *downloads* do banco de dados do *Cowrie*. Em seguida, ao realizar sua primeira verificação na tabela *inputs*, a ferramenta realiza a identificação de novas entradas e procede com sua inserção na tabela *urls*. Estas entradas constituem os comandos digitados durante a sessão via *shell*, os quais são submetidos a um processamento específico para a extração e captura das *URLs* neles contidas. Concluída essa etapa inicial, são realizadas as verificações de disponibilidade das *URLs*, e o sistema prossegue com seu ciclo contínuo de monitoramento.

```
(env) PS C:\Users\ribeirin\Documents\DirectoryMonitor> python.exe .\main.py
[INIT] MonitoringDB empty
[INIT] Initializing population of monitoringDB
Inserted URL: https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh
Inserted URL: https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
[PERIODIC TASK] Running...
[VERIFYING URL] No change on Downloads on Cowrie!
[VERIFYING URL] Change Detect on Input on Cowrie!
Inserted URL: https://www.pudim.com.br/pudim.jpg
URL 'https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf' already exists in monitoring.
URL 'https://www.pudim.com.br/pudim.jpg' already exists in monitoring.
URL 'https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh' already exists in monitoring.
[VERIFYING CONNECTION] Testing docs.aws.amazon.com:443
[VERIFYING CONNECTION] Connection docs.aws.amazon.com:443 successful
Updated URL: https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
[VERIFYING CONNECTION] Testing raw.githubusercontent.com:443
[VERIFYING CONNECTION] Connection raw.githubusercontent.com:443 successful
Updated URL: https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh
[VERIFYING CONNECTION] Testing www.pudim.com.br:443
[VERIFYING CONNECTION] Connection www.pudim.com.br:443 successful
Updated URL: https://www.pudim.com.br/pudim.jpg
[PERIODIC TASK] Finished!
```

Figura 11 – Caso 02: *Log* do funcionamento da ferramenta. Fonte: Elaborado pelo autor (2025)

A Figura 12 apresenta o estado final da tabela *urls* do banco de dados de monitoramento após a inicialização da ferramenta. Esse resultado evidencia a capacidade da

aplicação de identificar e registrar corretamente as *URLs* extraídas do banco de dados do *Cowrie*.



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

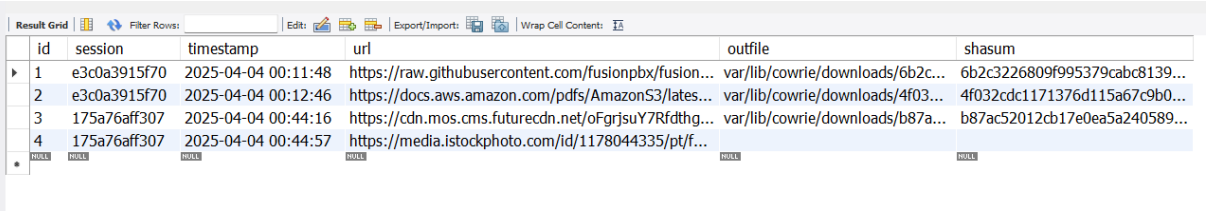
id	url	first_view	last_view
1	https://raw.githubusercontent.com/fusionpbx/fusionpbx-install....	2025-04-03 21:23:27	2025-04-03 21:23:42
2	https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide...	2025-04-03 21:23:27	2025-04-03 21:23:42
3	https://www.pudim.com.br/pudim.jpg	2025-04-03 21:23:27	2025-04-03 21:23:42
NULL	NULL	NULL	NULL

Figura 12 – Caso 02: Tabela *urls* do monitoramento com entradas. Fonte: Elaborado pelo autor (2025)

4.3 Caso 3: Inicialização da ferramenta com ambos os bancos de dados populados

Este cenário tem como objetivo avaliar o funcionamento do *CowrieSniffer* quando tanto o banco de dados do *Cowrie* quanto o banco de monitoramento já possuem registros. A proposta consiste em analisar o processo de inicialização da ferramenta, com ênfase no carregamento das variáveis com as *URLs* existentes e na execução dos testes de disponibilidade no ambiente monitorado.

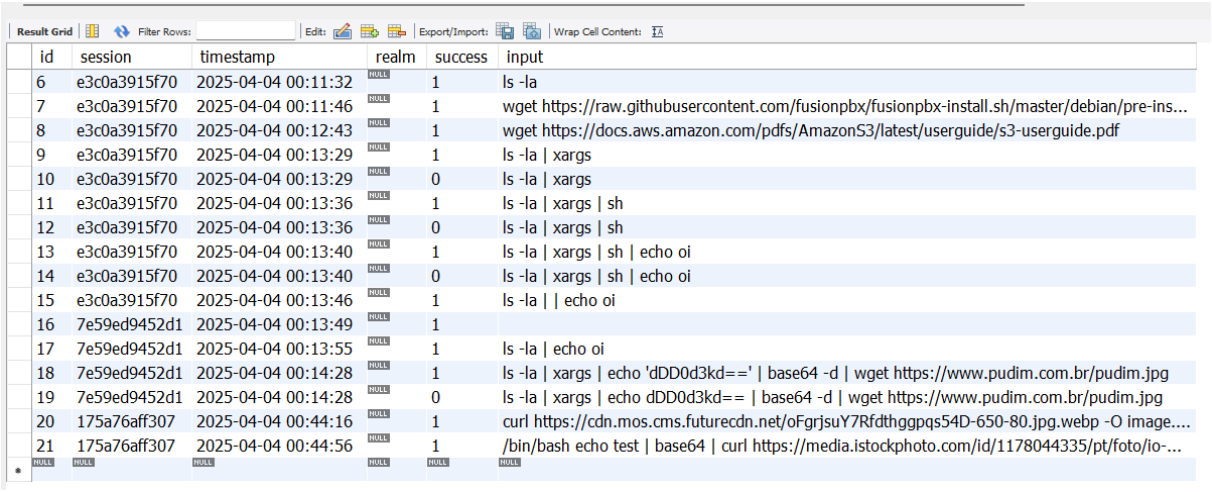
As Figuras 13, 14 e 15 apresentam, respectivamente, o estado das tabelas *downloads* e *input*, pertencentes ao banco de dados do *Cowrie*, e da tabela *urls*, localizada no banco de dados de monitoramento. A partir dessas condições iniciais, a execução do *CowrieSniffer*, ilustrada na Figura 16, inicia o carregamento dos endereços provenientes da tabela *downloads*. Em seguida, é realizada a verificação da tabela *input*, que também contém *URLs*; entretanto, como essas já estão registradas no banco de monitoramento, não é necessária sua reinserção. Por fim, a ferramenta procede com os testes de conectividade utilizando as *URLs* previamente armazenadas.



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

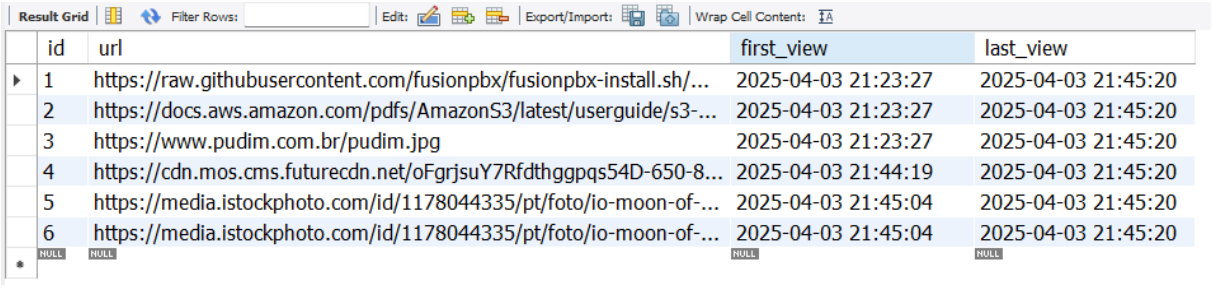
id	session	timestamp	url	outfile	shasum
1	e3c0a3915f70	2025-04-04 00:11:48	https://raw.githubusercontent.com/fusionpbx/fusion...	var/lib/cowrie/downloads/6b2c...	6b2c3226809f995379cabc8139...
2	e3c0a3915f70	2025-04-04 00:12:46	https://docs.aws.amazon.com/pdfs/AmazonS3/lates...	var/lib/cowrie/downloads/4f03...	4f032cdc1171376d115a67c9b0...
3	175a76aff307	2025-04-04 00:44:16	https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthg...	var/lib/cowrie/downloads/b87a...	b87ac52012cb17e0ea5a240589...
4	175a76aff307	2025-04-04 00:44:57	https://media.istockphoto.com/id/1178044335/pt/f...		
NULL	NULL	NULL	NULL	NULL	NULL

Figura 13 – Caso 03: Tabela *downloads* do *Cowrie* com entradas. Fonte: Elaborado pelo autor (2025)



id	session	timestamp	realm	success	input
6	e3c0a3915f70	2025-04-04 00:11:32	NULL	1	ls -la
7	e3c0a3915f70	2025-04-04 00:11:46	NULL	1	wget https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-ins...
8	e3c0a3915f70	2025-04-04 00:12:43	NULL	1	wget https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
9	e3c0a3915f70	2025-04-04 00:13:29	NULL	1	ls -la | xargs
10	e3c0a3915f70	2025-04-04 00:13:29	NULL	0	ls -la | xargs
11	e3c0a3915f70	2025-04-04 00:13:36	NULL	1	ls -la | xargs | sh
12	e3c0a3915f70	2025-04-04 00:13:36	NULL	0	ls -la | xargs | sh
13	e3c0a3915f70	2025-04-04 00:13:40	NULL	1	ls -la | xargs | sh | echo oi
14	e3c0a3915f70	2025-04-04 00:13:40	NULL	0	ls -la | xargs | sh | echo oi
15	e3c0a3915f70	2025-04-04 00:13:46	NULL	1	ls -la | | echo oi
16	7e59ed9452d1	2025-04-04 00:13:49	NULL	1	
17	7e59ed9452d1	2025-04-04 00:13:55	NULL	1	ls -la | echo oi
18	7e59ed9452d1	2025-04-04 00:14:28	NULL	1	ls -la | xargs | echo 'dDD0d3kd==' | base64 -d | wget https://www.pudim.com.br/pudim.jpg
19	7e59ed9452d1	2025-04-04 00:14:28	NULL	0	ls -la | xargs | echo dDD0d3kd== | base64 -d | wget https://www.pudim.com.br/pudim.jpg
20	175a76aff307	2025-04-04 00:44:16	NULL	1	curl https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp -O image....
21	175a76aff307	2025-04-04 00:44:56	NULL	1	/bin/bash echo test | base64 | curl https://media.istockphoto.com/id/1178044335/pt/foto/io-...

Figura 14 – Caso 03: Tabela *input* do *Cowrie* com entradas. Fonte: Elaborado pelo autor (2025)



id	url	first_view	last_view
1	https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/...	2025-04-03 21:23:27	2025-04-03 21:45:20
2	https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-...	2025-04-03 21:23:27	2025-04-03 21:45:20
3	https://www.pudim.com.br/pudim.jpg	2025-04-03 21:23:27	2025-04-03 21:45:20
4	https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-8...	2025-04-03 21:44:19	2025-04-03 21:45:20
5	https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-...	2025-04-03 21:45:04	2025-04-03 21:45:20
6	https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-...	2025-04-03 21:45:04	2025-04-03 21:45:20

Figura 15 – Caso 03: Tabela *urls* do monitoramento com entradas. Fonte: Elaborado pelo autor (2025)

```
(env) PS C:\Users\ribeirin\Documents\DirectoryMonitor> python.exe .\main.py
[INIT] State loaded: {'https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024': 'https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf', 'https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh': 'https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp'}
[PERIODIC TASK] Running...
[VERIFYING URL] No change on Downloads on Cowrie!
[VERIFYING URL] Change Detect on Input on Cowrie!
URL 'https://www.pudim.com.br/pudim.jpg' already exists in monitoring.
URL 'https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh' already exists in monitoring.
URL 'https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp' already exists in monitoring.
URL 'https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024&w=is&k=20&c=VfzZA6Ncrt8I6J7bdPtR-Kv0qiwBAi10rBGsFLz5kZQ=' already exists in monitoring.
URL 'https://www.pudim.com.br/pudim.jpg' already exists in monitoring.
URL 'https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf' already exists in monitoring.
[VERIFYING CONNECTION] Testing cdn.mos.cms.futurecdn.net:443
[VERIFYING CONNECTION] Connection cdn.mos.cms.futurecdn.net:443 successful
Updated URL: https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp
[VERIFYING CONNECTION] Testing docs.aws.amazon.com:443
[VERIFYING CONNECTION] Connection docs.aws.amazon.com:443 successful
Updated URL: https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
[VERIFYING CONNECTION] Testing media.istockphoto.com:443
[VERIFYING CONNECTION] Connection media.istockphoto.com:443 successful
Updated URL: https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024
[VERIFYING CONNECTION] Testing media.istockphoto.com:443
[VERIFYING CONNECTION] Connection media.istockphoto.com:443 successful
Updated URL: https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024&w=is&k=20&c=VfzZA6Ncrt8I6J7bdPtR-Kv0qiwBAi10rBGsFLz5kZQ=
[VERIFYING CONNECTION] Testing raw.githubusercontent.com:443
[VERIFYING CONNECTION] Connection raw.githubusercontent.com:443 successful
Updated URL: https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh
[VERIFYING CONNECTION] Testing www.pudim.com.br:443
[VERIFYING CONNECTION] Connection www.pudim.com.br:443 successful
Updated URL: https://www.pudim.com.br/pudim.jpg
[PERIODIC TASK] Finished!
```

Figura 16 – Caso 03: *Log* do funcionamento da ferramenta. Fonte: Elaborado pelo autor (2025)

4.4 Caso 4: Entradas e análises das *URLs*

Este estudo de caso tem como objetivo analisar o comportamento do *CowrieSniffer* diante da adição de novas *URLs*. A análise visa verificar se a ferramenta é capaz de detectar essas novas entradas e avaliar a disponibilidade de *URLs*, tanto indisponíveis quanto disponíveis na porta 80 do protocolo *http*. É importante notar que a ferramenta também pode coletar e verificar *URLs* provenientes do protocolo *HTTPS*, utilizando a porta 443 como ponto de conexão inicial ou ainda em outras portas. A ideia do Caso 4 é simplesmente mostrar o funcionamento de tal requisito.

4.4.1 *URL* indisponível

Este caso apresenta uma situação específica em que o endereço monitorado se encontra indisponível na porta 3000. Conforme ilustrado na Figura 17, a ferramenta detecta a inserção da *URL* por meio da tabela *downloads* e, em seguida, realiza o teste de conectividade, que resulta em falha devido à indisponibilidade. Posteriormente, a Figura 18 apresenta o registro correspondente no banco de dados de monitoramento, identificado pela chave primária *id* de valor 7. Observa-se que, em razão da indisponibilidade da *URL* na porta 3000, a coluna *last_view* não foi atualizada, indicando a ausência de sucesso na tentativa de conexão.

```
[PERIODIC TASK] Running...
[VERIFYING URL] Change Detect on Downloads on Cowrie!
Inserted URL: http://pudim.com.br:3000/script.sh
[VERIFYING URL] No change on Input on Cowrie!
[VERIFYING CONNECTION] Testing pudim.com.br:3000
[VERIFYING CONNECTION] Connection to pudim.com.br:3000 failed
[VERIFYING CONNECTION] Testing cdn.mos.cms.futurecdn.net:443
[VERIFYING CONNECTION] Connection cdn.mos.cms.futurecdn.net:443 successful
Updated URL: https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp
[VERIFYING CONNECTION] Testing docs.aws.amazon.com:443
[VERIFYING CONNECTION] Connection docs.aws.amazon.com:443 successful
Updated URL: https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
[VERIFYING CONNECTION] Testing media.istockphoto.com:443
[VERIFYING CONNECTION] Connection media.istockphoto.com:443 successful
Updated URL: https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024
[VERIFYING CONNECTION] Testing media.istockphoto.com:443
[VERIFYING CONNECTION] Connection media.istockphoto.com:443 successful
Updated URL: https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024&w=is&k=20&c=VfzZA6Ncrt8I6J7bdPtR-Kv0qiwBAi10rBGsFLz5kZQ=
[VERIFYING CONNECTION] Testing raw.githubusercontent.com:443
[VERIFYING CONNECTION] Connection raw.githubusercontent.com:443 successful
Updated URL: https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh
[VERIFYING CONNECTION] Testing www.pudim.com.br:443
[VERIFYING CONNECTION] Connection www.pudim.com.br:443 successful
Updated URL: https://www.pudim.com.br/pudim.jpg
[PERIODIC TASK] Finished!
```

Figura 17 – Caso 04: *Log* 1 do funcionamento da ferramenta. Fonte: Elaborado pelo autor (2025)

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	url	first_view	last_view
1	https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/...	2025-04-03 21:23:27	2025-04-03 22:12:52
2	https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-...	2025-04-03 21:23:27	2025-04-03 22:12:52
3	https://www.pudim.com.br/pudim.jpg	2025-04-03 21:23:27	2025-04-03 22:12:52
4	https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-8...	2025-04-03 21:44:19	2025-04-03 22:12:52
5	https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-...	2025-04-03 21:45:04	2025-04-03 22:12:52
6	https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-...	2025-04-03 21:45:04	2025-04-03 22:12:52
7	http://pudim.com.br:3000/script.sh	2025-04-03 22:08:19	2025-04-03 22:08:19
8	http://testphp.vulnweb.com/images/logo.gif	2025-04-03 22:11:33	2025-04-03 22:12:52
NULL	NULL	NULL	NULL

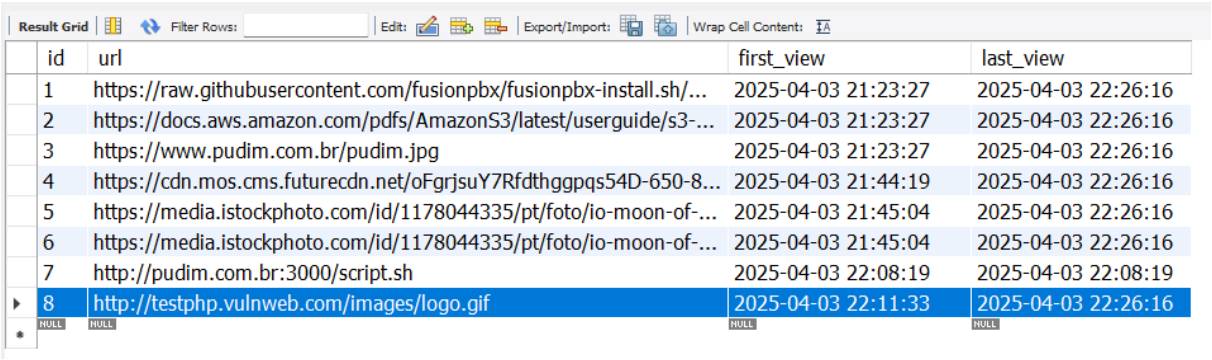
Figura 18 – Caso 04: Registro 1 na tabela *urls*. Fonte: Elaborado pelo autor (2025)

4.4.2 *URL* sem a utilização do protocolo *HTTPS*

Nesta situação, observa-se o processo integral de detecção e monitoramento de uma *URL* maliciosa quando apresentada sem a utilização do protocolo *HTTPS*. Inicialmente, o sistema identifica a presença da *URL* mediante análise dos registros na tabela *downloads*. Após a identificação deste elemento, a ferramenta executa, automaticamente, três ações sequenciais: (i) realiza a inserção dos dados correspondentes no banco de dados de monitoramento; (ii) verifica a disponibilidade da *URL* através da porta 80 (ressalta-se que, caso fosse utilizada outra porta aleatória, o *CowrieSniffer* a identificaria); e (iii) processa essas informações, procedimento que ocorre automaticamente quando o sistema reconhece o protocolo *HTTP* no início da *URL*, conforme demonstrado na Figura 19. Como resultado final deste fluxo, a Figura 20 apresenta a *URL* registrada na ferramenta, juntamente com sua atualização referente à disponibilidade, sendo este registro identificado univocamente pela chave primária *id* de valor 8.

```
[PERIODIC TASK] Running...
[VERIFYING URL] Change Detect on Downloads on Cowrie!
Inserted URL: http://testphp.vulnweb.com/images/logo.gif
[VERIFYING URL] Change Detect on Input on Cowrie!
URL 'https://www.pudim.com.br/pudim.jpg' already exists in monitoring.
URL 'https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf' already exists in monitoring.
URL 'https://www.pudim.com.br/pudim.jpg' already exists in monitoring.
URL 'https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh' already exists in monitoring.
URL 'https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp' already exists in monitoring.
URL 'http://testphp.vulnweb.com/images/logo.gif' already exists in monitoring.
URL 'https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024&w=is&k=20&c=VfzZA6Ncrt8I6J7bdPtR-Kv0qiwBAi10rBGsFLz5kZQ=' already exists in monitoring.
[VERIFYING CONNECTION] Testing pudim.com.br:3000
[VERIFYING CONNECTION] Connection to pudim.com.br:3000 failed
[VERIFYING CONNECTION] Testing testphp.vulnweb.com:80
[VERIFYING CONNECTION] Connection testphp.vulnweb.com:80 successful
Updated URL: http://testphp.vulnweb.com/images/logo.gif
[VERIFYING CONNECTION] Testing cdn.mos.cms.futurecdn.net:443
[VERIFYING CONNECTION] Connection cdn.mos.cms.futurecdn.net:443 successful
Updated URL: https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-80.jpg.webp
[VERIFYING CONNECTION] Testing docs.aws.amazon.com:443
[VERIFYING CONNECTION] Connection docs.aws.amazon.com:443 successful
Updated URL: https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-userguide.pdf
[VERIFYING CONNECTION] Testing media.istockphoto.com:443
[VERIFYING CONNECTION] Connection media.istockphoto.com:443 successful
Updated URL: https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024
[VERIFYING CONNECTION] Testing media.istockphoto.com:443
[VERIFYING CONNECTION] Connection media.istockphoto.com:443 successful
Updated URL: https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-jupiter.webp?s=1024x1024&w=is&k=20&c=VfzZA6Ncrt8I6J7bdPtR-Kv0qiwBAi10rBGsFLz5kZQ=
[VERIFYING CONNECTION] Testing raw.githubusercontent.com:443
[VERIFYING CONNECTION] Connection raw.githubusercontent.com:443 successful
Updated URL: https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/master/debian/pre-install.sh
[VERIFYING CONNECTION] Testing www.pudim.com.br:443
[VERIFYING CONNECTION] Connection www.pudim.com.br:443 successful
Updated URL: https://www.pudim.com.br/pudim.jpg
[PERIODIC TASK] Finished!
```

Figura 19 – Caso 04: *Log* 2 do funcionamento da ferramenta. Fonte: Elaborado pelo autor (2025)



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	url	first_view	last_view
1	https://raw.githubusercontent.com/fusionpbx/fusionpbx-install.sh/...	2025-04-03 21:23:27	2025-04-03 22:26:16
2	https://docs.aws.amazon.com/pdfs/AmazonS3/latest/userguide/s3-...	2025-04-03 21:23:27	2025-04-03 22:26:16
3	https://www.pudim.com.br/pudim.jpg	2025-04-03 21:23:27	2025-04-03 22:26:16
4	https://cdn.mos.cms.futurecdn.net/oFgrjsuY7Rfdthggpqs54D-650-8...	2025-04-03 21:44:19	2025-04-03 22:26:16
5	https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-...	2025-04-03 21:45:04	2025-04-03 22:26:16
6	https://media.istockphoto.com/id/1178044335/pt/foto/io-moon-of-...	2025-04-03 21:45:04	2025-04-03 22:26:16
7	http://pudim.com.br:3000/script.sh	2025-04-03 22:08:19	2025-04-03 22:08:19
8	http://testphp.vulnweb.com/images/logo.gif	2025-04-03 22:11:33	2025-04-03 22:26:16
NULL	NULL	NULL	NULL

Figura 20 – Caso 04: Registro 2 na tabela *urls*. Fonte: Elaborado pelo autor (2025)

5 Conclusão

Este trabalho teve como principal objetivo desenvolver uma ferramenta capaz de coletar, processar e verificar a disponibilidade de *URLs* capturadas pelo *honeypot Cowrie*, contribuindo para o aprimoramento de sua estrutura na área de segurança da informação. Com base nos estudos de caso realizados, os resultados obtidos permitiram validar o funcionamento da ferramenta em cenários específicos de uso do *CowrieSniffer*, demonstrando sua eficácia na coleta, tratamento e verificação da disponibilidade de *URLs*. Portanto, conclui-se que a ferramenta atingiu plenamente seu principal objetivo.

O desenvolvimento do *CowrieSniffer* foi estruturado de maneira modular. A arquitetura do sistema *CowrieSniffer* foi concebida sob uma perspectiva estritamente modular. Os principais componentes compreendem: o módulo *Main*, responsável pela orquestração sistêmica; o módulo de configuração, encarregado da gestão dos parâmetros de conexão; os módulos de manipulação de banco de dados, tanto do *Cowrie* quanto do sistema de monitoramento; e, fundamentalmente, o módulo *URLMonitor*, que implementa a lógica central de verificação e atualização das *URLs*. Durante o processo de desenvolvimento, diversos desafios técnicos emergiram, tais como: a necessidade de extrair *URLs* das tabelas *input* e *downloads* do banco de dados do *Cowrie*; a implementação de mecanismos de verificação de disponibilidade em diferentes protocolos (*HTTP* e *HTTPS*) e portas; e a garantia de persistência e consistência dos dados monitorados ao longo do tempo.

O *CowrieSniffer* constitui ferramenta de notável relevância em análise forense digital, permitindo a coleta de *URLs* que podem ser analisadas para a identificação de padrões de persistência em infraestruturas maliciosas. Em ambientes corporativos com múltiplos *honeypots*, possibilita a detecção de campanhas direcionadas mediante correlação entre *URLs* utilizadas por diferentes agentes maliciosos. O monitoramento contínuo destas *URLs* fornece dados substanciais sobre o ciclo vital de infraestruturas nocivas, como servidores de comando e controle (*C&C*) ou repositórios de *malware*, elucidando táticas, técnicas e procedimentos dos atacantes (*TTPs*).

Além disso, o código da ferramenta foi desenvolvido de forma modular, permitindo fácil manipulação e a inserção de novas funcionalidades. Dessa maneira, diversas melhorias podem ser implementadas em trabalhos futuros, tais como a incorporação do *CowrieSniffer* como um componente nativo do *Cowrie*, a implementação de uma coluna no banco de dados de monitoramento para indicar os endereços ativos e o desenvolvimento de um mecanismo para exibição ou ocultação de *logs*.

Referências

AL-MOHANNADI, H.; AWAN, I.; HAMAR, J. A. Analysis of adversary activities using cloud-based web services to enhance cyber threat intelligence. **Serv. Oriented Comput. Appl.**, Springer-Verlag, Berlin, Heidelberg, v. 14, n. 3, p. 175–187, sep 2020. ISSN 1863-2386. Disponível em: <https://doi.org/10.1007/s11761-019-00285-7>. Citado na página 8.

BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. **IEEE Cloud Computing**, v. 1, n. 3, p. 81–84, 2014. Citado na página 15.

CABRAL, W. Z.; VALLI, C.; SIKOS, L. F.; WAKELING, S. G. Advanced cowrie configuration to increase honeypot deceptiveness. In: JØSANG, A.; FUTCHER, L.; HAGEN, J. (Ed.). **ICT Systems Security and Privacy Protection**. Cham: Springer International Publishing, 2021. p. 317–331. ISBN 978-3-030-78120-0. Citado 2 vezes nas páginas 13 e 16.

IBRAHIM, M. H.; SAYAGH, M.; HASSAN, A. E. A study of how docker compose is used to compose multi-component systems. **Empirical Softw. Engg.**, Kluwer Academic Publishers, USA, v. 26, n. 6, nov. 2021. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-021-10025-1>. Citado na página 15.

JAVADPOUR, A.; JA'FARI, F.; TALEB, T.; SHOJAFAR, M.; BENZAïD, C. A comprehensive survey on cyber deception techniques to improve honeypot performance. **Computers Security**, v. 140, p. 103792, 2024. ISSN 0167-4048. Citado na página 13.

MALLOY, B. A.; POWER, J. F. Quantifying the transition from python 2 to 3: An empirical study of python applications. In: **2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)**. [S.l.: s.n.], 2017. p. 314–323. Citado na página 14.

MEHTA, S.; PAWADE, D.; NAYYAR, Y.; SIDDAVATAM, I.; TIWART, A.; DALVI, A. Cowrie honeypot data analysis and predicting the directory traverser pattern during the attack. In: **2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)**. [S.l.: s.n.], 2021. p. 1–4. Citado na página 15.

MENDES, L. G. Trabalho de Conclusão de Curso, **Construção de infraestrutura de Honeypots IoT usando computação em nuvem**. 2023. Universidade Federal de Uberlândia. Citado na página 9.

NIST. **Cybersecurity definition**. 2019. Acesso em: 30 de out. 2024. Disponível em: <https://csrc.nist.gov/glossary/term/cybersecurity>. Citado na página 12.

NăSTASE, V.-I.; MIHăILESCU, M.-E.; WEISZ, S.; DAGILIS, L. V.; MIHAI, D.; CARABAS, M. Cowrie ssh honeypot: Architecture, improvements and data visualization. In: **2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet)**. [S.l.: s.n.], 2024. p. 1–7. Citado na página 16.

OOSTERHOF, M. **Cowrie Documentation**. 2024. Acesso em: 28 de abril de 2025. Disponível em: <https://docs.cowrie.org/en/latest/>. Citado na página 13.

PROVOS, N.; HOLZ, T. **Virtual honeypots from botnet tracking to intrusion detection**. Upper Saddle River, N.J: Addison-Wesley, 2007. (Safari Books Online.). ISBN 9780321336323. Citado na página 8.

RODRIGUES, G. A. P. Trabalho de Conclusão de Curso, **Análise de tráfego malicioso direcionado a uma Honeynet com inspeção profunda de pacotes**. 2017. Universidade de Brasília. Citado na página 9.

SPITZNER, L. Honeypots: Catching the insider threat. In: **Proceedings of the 19th Annual Computer Security Applications Conference**. USA: IEEE Computer Society, 2003. (ACSAC '03), p. 170. ISBN 0769520413. Citado na página 13.

STALLINGS, W. **Criptografia e Segurança de Redes: Princípios e Práticas. 6. ed**. [S.l.]: Pearson Education, 2014. Citado na página 12.

SUEHRING, S. **MySQL Bible**. Wiley, 2002. (Bible). ISBN 9780764518614. Disponível em: <https://books.google.com.br/books?id=bY01hYV3r-gC>. Citado na página 14.

VIRAJA, V. K.; PURANDARE, P. A qualitative research on the impact and challenges of cybercrimes. **Journal of Physics: Conference Series**, IOP Publishing, v. 1964, n. 4, p. 042004, jul 2021. Disponível em: <https://dx.doi.org/10.1088/1742-6596/1964/4/042004>. Citado na página 8.

Anexos

ANEXO A – Repositório *CowrieSniffer*

O código desenvolvido no decorrer deste trabalho será disponibilizado publicamente no *GitHub*[1], permitindo a transparência dos resultados e possibilitando futuras contribuições da comunidade acadêmica e profissional.

O repositório, intitulado *CowrieSniffer*, conterá os *scripts* utilizados para o monitoramento das *URLs* extraídas do *honeypot Cowrie*, bem como a documentação detalhada sobre a instalação, configuração e uso da ferramenta.

[1] Disponível em https://github.com/r1beirin/CowrieSniffer