Relatório Sentinel IOC

Sentinel IOC: Analisador de Indicadores de Comprometimento

Disciplina: EDOO 2025.1

Docente: Francisco Paulo Magalhães Simões

Discentes: Beatriz Helena, Dayane Lima, Maria Antonia, Rafael Teles e Williams Andrade

1. INTRODUÇÃO

A cibersegurança tornou-se um pilar fundamental para proteger dados e sistemas. Uma das ferramentas mais eficazes na detecção e prevenção de ataques são os Indicadores de Comprometimento (IOCs). Estes são fragmentos de dados forenses, como endereços IP maliciosos, URLs suspeitas ou hashes de arquivos infectados, que atuam como "impressões digitais" de atividades maliciosas. Este projeto, intitulado "Sentinel IOC", simula um analisador de *IOCs*, oferecendo uma introdução prática à Programação Orientada a Objetos (POO) utilizando a linguagem C++. Ele permite o cadastro, consulta, edição e exclusão dessas informações, fornecendo uma base para a compreensão de conceitos de POO em um contexto de segurança da informação.

2. OBJETIVOS DO PROJETO

O projeto Sentinel IOC foi desenvolvido com múltiplos objetivos em mente, visando proporcionar uma experiência de aprendizado abrangente e prática:

- Aprender POO com C++: Implementação Herança, Polimorfismo e Encapsulamento em C++, aplicando-os na modelagem de um sistema real;
- Simular um analisador de *IOCs*: Criar uma aplicação que replique as funcionalidades básicas de um sistema de gerenciamento e análise de Indicadores de Comprometimento, familiarizando-os com o fluxo de trabalho de um analista de segurança;
- Trabalhar com arquivos como base de dados: Desenvolver a capacidade de persistir dados em arquivos de texto (.csv), simulando um banco de dados simples para armazenar e recuperar informações sobre os *IOCs*;
- Explorar boas práticas de código e versionamento: Incentivar o desenvolvimento de código limpo, modularizado, organizado e legível, além de praticar o uso de sistemas de controle de versão (como Git e GitHub) para colaboração e gerenciamento do projeto.

3. ARQUITETURA DO SISTEMA

3.1. Explicação das classes:

3.1.1. Classes base

- **Indicator:** Esta é a classe abstrata base do sistema. Ela define os atributos e métodos comuns a todos os tipos de *IOCs*, como o tipo do indicador (IP, URL, Hash) e seu valor. Ela serve como um "modelo" para as subclasses, garantindo que todos os indicadores compartilhem características essenciais;
- **IndicatorManager:** Esta classe atua como o controlador principal do sistema, sendo responsável pela lógica de *CRUD* (Create, Read, Update, Delete) dos *IOCs*. Contém um vetor de ponteiros para Indicador, o que permite armazenar objetos de qualquer uma das classes derivadas de forma polimórfica. A *IndicatorManager* também gerencia a persistência dos dados, incluindo métodos para salvar e carregar os *IOCs* de um arquivo de texto, buscar *IOCs* por tipo ou valor e simular a análise de um *IOC*;
- **FileManager:** Abstrai a lógica de leitura e escrita de arquivos. A sua responsabilidade é serializar a coleção de indicadores para um formato de arquivo e desserializar os dados do arquivo de volta para a memória, recriando os objetos. Isso mantém o *IndicatorManager* focado na lógica de negócio.

3.1.2. Classes derivadas

- **MaliciousIP:** Herda da classe Indicador e representa um endereço IP malicioso. Pode incluir atributos extras específicos, como o país de origem do IP;
- **MaliciousURL:** Também herda de Indicador e representa uma URL suspeita;
- **MaliciousHash:** Deriva de Indicador e representa o hash de um arquivo infectado. Pode ter atributos adicionais, como o algoritmo de hashing utilizado.

3.2. Auxiliares

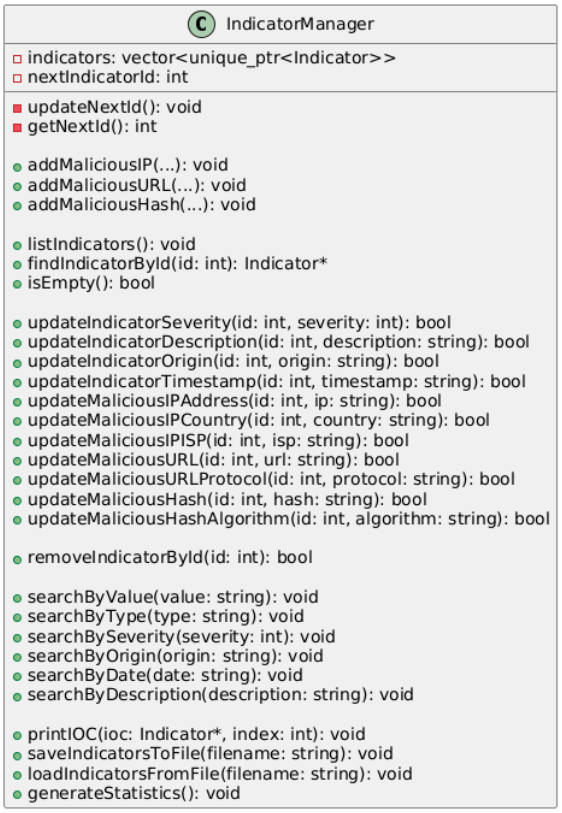
- **CLI:** Ela é uma classe responsável por interagir com o usuário através do terminal (linha de comando);
- **Utils:** Fornece funções de apoio para todas as outras partes.

3.3. Diagrama UML:

Figura 1 – Diagrama UML da classe Indicator e suas derivadas

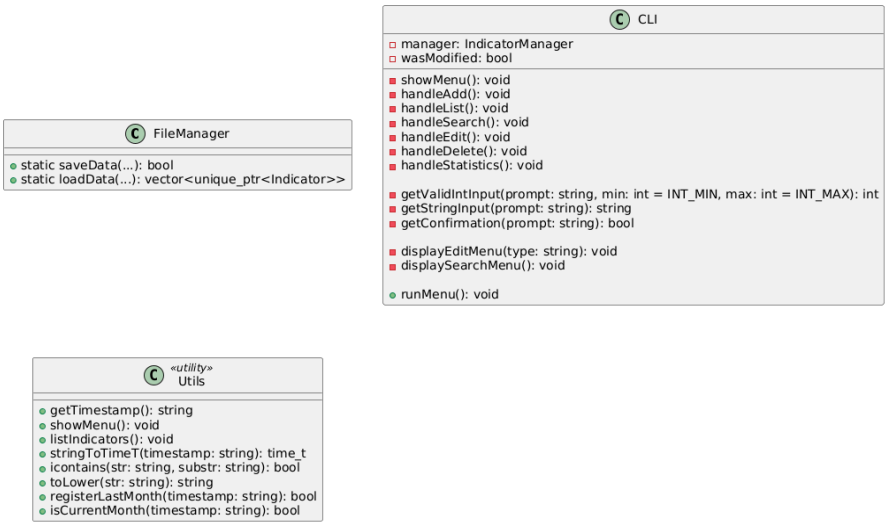
Fonte: Autoria própria, 2025.

Figura 2 – Diagrama UML da classe IndicatorManager



Fonte: Autoria própria, 2025.

Figura 3 – Diagrama UML do FileManager, CLI e Utils



Fonte: Autoria própria, 2025.

4. CONCEITOS DE POO UTILIZADOS

1. **Herança:** É fundamental na estrutura do projeto. As classes IPMalicioso, URLMaliciosa e HashMalicioso herdam da classe base abstrata Indicador. Isso permite reutilizar atributos e métodos comuns, evitando a duplicação de código e organizando a hierarquia de classes;
2. **Polimorfismo:** É intensamente utilizado no IndicatorManager. O vetor *std::vector<Indicador*>* pode armazenar ponteiros para objetos de qualquer uma das classes derivadas (IPMalicioso, URLMaliciosa, HashMalicioso). Isso também se aplica à simulação de análise, onde mensagens diferentes podem ser exibidas dependendo do tipo do IOC;
3. **Encapsulamento:** Os atributos das classes são protegidos utilizando modificadores de acesso *private*, e são acessados ou modificados apenas através de métodos públicos (getters e setters). Isso garante a integridade dos dados e impede o acesso direto e não controlado, promovendo a modularidade e a manutenibilidade do código;
4. **Abstração:** A classe IndicatorManager esconde toda a complexidade de gerenciar a coleção de indicadores. Isso inclui como os indicadores são armazenados, como um novo ID é gerado e atribuído, como a busca é implementada ou como a memória é gerenciada. Assim, a CLI usa o IndicatorManager como um "serviço", sem se preocupar com a sua implementação interna;
5. **Uso de ponteiros e referências:** São empregados para possibilitar o armazenamento polimórfico dos objetos Indicador* no vetor da classe IndicatorManagerme para manipular objetos de forma eficiente.

5. FUNCIONALIDADES IMPLEMENTADAS

O Sentinel IOC oferece um conjunto robusto de funcionalidades, todas acessíveis através de uma interface de linha de comando (CLI), permitindo ao usuário interagir de forma prática com o sistema de gerenciamento de *IOCs*:

- **CRUD:**
    - Create (Adicionar): Permite ao usuário cadastrar novos *IOCs*, escolhendo entre os tipos (IP Malicioso, URL Maliciosa, Hash Malicioso) e inserindo os dados;
    - Read (Listar/Buscar): Permite listar todos os *IOCs* cadastrados, exibindo suas informações detalhadas de forma polimórfica. Também é possível buscar um IOC específico por meio de suas informações;
    - Update (Atualizar): Oferece a funcionalidade de modificar os dados de um IOC existente, selecionando-o e fornecendo as novas informações;
    - Delete (Remover): Permite a exclusão de um IOC do sistema.
- **Simulação de análise:** A funcionalidade de análise permite simular a verificação dos *IOCs* cadastrados, fornecendo estatísticas gerais sobre os indicadores armazenados no sistema;
- **Interface por linha de comando:** Toda a interação com o usuário é feita através de um menu de texto simples no console, tornando o projeto acessível e focado nos conceitos de lógica e POO. Ao iniciar, o sistema carrega os dados de um arquivo (ioc.csv), e antes de sair, salva as alterações, garantindo a persistência dos dados.

6. TESTES REALIZADOS

Para garantir a funcionalidade e robustez do Sentinel IOC, foram realizados testes unitários, validando cada uma das funcionalidades implementadas. Os principais cenários testados incluem:

- Validação da Inicialização;
- Testes de CRUD;
- Testes de Persistência;
- Testes da Simulação de Análise.

Os testes foram conduzidos de forma a cobrir os fluxos principais e alternativos do sistema, garantindo que todas as funcionalidades se comportem conforme o esperado e que os conceitos de POO sejam aplicados de maneira correta e eficaz.

7. APRENDIZADOS E DESAFIOS

7.1. Desafios

- **Gerenciamento de Memória com Ponteiros e Polimorfismo:** Um dos maiores desafios foi o correto gerenciamento da memória ao lidar com std::vector<Indicador*>. Garantir que os objetos fossem alocados dinamicamente e deslocados adequadamente para evitar vazamentos de memória, especialmente ao remover *IOCs* ou ao finalizar o programa, exigiu atenção redobrada. Compreender como os ponteiros para a classe base se comportam com objetos das classes derivadas e a importância do destrutor virtual na classe base foi crucial;
- **Dificuldade para depurar Código:** O baixo nível do C++ e o uso extensivo de ponteiros e alocação dinâmica tornaram a depuração um desafio considerável. Foi necessária muitas vezes uma análise cuidadosa do fluxo de execução para identificar a causa raiz dos problemas, que muitas vezes não eram óbvios ou diretos;
- **Organizar a Ordem de Prioridade da Implementação das Funcionalidades:** Definir uma sequência lógica para o desenvolvimento das funcionalidades foi um ponto crítico. Decidir o que implementar primeiro – a hierarquia de classes, o *CRUD* básico, a persistência em arquivo ou a interface de usuário – e como essas partes se integrariam sem gerar bloqueios ou retrabalho excessivo, exigiu um bom planejamento e a capacidade de adaptar a estratégia à medida que os desafios surgiam;
- **Implementação Polimórfica Robusta:** Assegurar que o polimorfismo funcionasse de maneira consistente em todas as funcionalidades (listagem, busca, análise) foi complexo;
- **Manipulação de Arquivos para Persistência:** A lógica de salvar e carregar os dados dos *IOCs* em um arquivo de texto, mantendo a integridade e o formato correto para posterior leitura, apresentou seus próprios obstáculos. Lidar com a leitura de diferentes atributos para cada tipo de IOC e reconstruir os objetos polimorficamente a partir das strings do arquivo exigiu um design cuidadoso da lógica de serialização/desserialização.

7.2. Aprendizados

- **Aprofundamento em Herança e Polimorfismo:** O projeto reforçou o entendimento de como a herança permite a reutilização de código e a criação de hierarquias lógicas. O polimorfismo, por sua vez, demonstrou o poder de escrever código genérico que se adapta ao tipo específico do objeto em tempo de execução, tornando o sistema mais flexível e extensível;
- **Importância do Encapsulamento:** A prática de encapsular os dados e expor apenas métodos controlados para acesso e modificação (getters e setters) provou ser essencial para manter a integridade dos dados e facilitar a manutenção do código;
- **Gerenciamento de Ciclo de Vida de Objetos:** Houve um aprendizado significativo sobre o ciclo de vida dos objetos em C++, desde a alocação dinâmica com *new* até a liberação com implementação dos *destructors*. A necessidade de um destrutor virtual na classe base abstrata para garantir a chamada correta dos destrutores das classes derivadas foi um aprendizado fundamental;

- **Tratamento de Exceções e Erros Básicos:** Embora não explicitamente detalhado, a implementação de robustez mínima para lidar com entradas inválidas do usuário e erros de arquivo foi um aprendizado prático, mostrando a importância da resiliência do sistema;

8. CONCLUSÃO

O projeto Sentinel IOC representa aprendizado prático de Programação Orientada a Objetos em C++. Ao simular um analisador de Indicadores de Comprometimento, os participantes não apenas aplicaram conceitos fundamentais como herança, polimorfismo, abstração e encapsulamento em um cenário simulado, mas também desenvolveram habilidades essenciais em manipulação de arquivos para persistência de dados e interação via linha de comando.

Os desafios enfrentados, especialmente no gerenciamento de memória com ponteiros e na implementação polimórfica robusta, foram cruciais para aprofundar a compreensão do C++ e suas particularidades. As soluções encontradas para esses desafios não só fortaleceram o conhecimento técnico, mas também a capacidade de depuração e resolução de problemas.

9. CONTRIBUIÇÕES

- Beatriz Helena da Silva Melo : Focou na lógica de busca de *IOCs* , na implementação de formulários de cadastro visual e na criação de testes unitários para a aplicação;
- Dayane Camile Bezerra de Lima : Ficou na estruturação do projeto, definindo a base das classes e arquitetura do projeto. Ela criou as classes para IPs, URLs e Hashes maliciosos, além de implementar funcionalidades como edição e remoção. Também foi responsável por criar o IOC.csv, o diagrama UML básico e o Readme.md;
- Maria Antonia Monteiro da Silva : Dedicou-se à criação da função de cadastro de *IOCs* na interface de linha de comando (CLI). Ela também trabalhou na criação de relatórios de estatísticas e estudou a integração com o Qt;
- Rafael Teles : Foi encarregado de criar a tabela com listagem interativa dos *IOCs* na interface gráfica (GUI). Ele também assumiu a tarefa de criar a tela inicial da GUI com Qt. A implementação da leitura e escrita de arquivos para persistência dos dados também ficou sob sua responsabilidade;
- Williams Andrade de Souza Filho : Teve como função principal a finalização do relatório técnico. Ele também contribuiu para a criação de relatórios de estatísticas e implementou a listagem de *IOCs*.