CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE INDAIATUBA

DR. ARCHIMEDES LAMMOGLIA

ANTHONY SANTOS DE OLIVEIRA

Análise de Alterações em BIOS: Plataforma Automatizada para Extração e Comparação de Módulos

INDAIATUBA
2025

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE INDAIATUBA

DR. ARCHIMEDES LAMMOGLIA

ANTHONY SANTOS DE OLIVEIRA

Análise de Alterações em BIOS: Plataforma Automatizada para Extração e Comparação de Módulos

Trabalho de Graduação apresentado por Anthony Santos de Oliveira como pré-requisito para a conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Faculdade de Tecnologia de Indaiatuba, elaborado sob a orientação do Prof. Jose Augusto Dias Mome

INDAIATUBA
2025

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE INDAIATUBA

DR. ARCHIMEDES LAMMOGLIA

ANTHONY SANTOS DE OLIVEIRA

Banca Avaliadora:

Prof. Jose Augusto Dias Mome	**Orientador**

Data da defesa: //2025

RESUMO

Este trabalho propôs e desenvolveu uma plataforma automatizada para análise e auditoria de *dumps* de *BIOS/UEFI*, focando na extração e comparação de módulos e *hashes* de versões submetidas pelos usuários. A solução visou atender à necessidade crescente de ferramentas especializadas para análise de segurança de *firmwares*, especialmente no contexto brasileiro, onde há uma escassez de iniciativas acadêmicas e práticas voltadas para o tema. A plataforma automatizou o processo de análise, utilizando ferramentas como UEFITool, Binwalk e Chipsec para extrair, organizar e comparar informações relevantes, como módulos e seus respectivos *hashes*. O objetivo foi oferecer uma interface acessível, permitindo que usuários submetam *dumps* de BIOS para auditoria sem a necessidade de conhecimento técnico profundo. A plataforma também permite a comparação entre duas ou mais versões de BIOS, auxiliando na identificação de alterações e possíveis vulnerabilidades. Este trabalho justifica-se pela necessidade de consolidar um repositório de versões de BIOS e facilitar a revisão de segurança, contribuindo com o desenvolvimento de soluções eficazes no campo da segurança de sistemas embarcados.

Palavras-chave: BIOS, UEFI, Análise de *Firmware*, Auditoria de Segurança, Comparação de Versões, Engenharia Reversa, Extração de Módulos, Segurança de Sistemas Embarcados.

ABSTRACT

This work proposed and developed an automated platform for the analysis and auditing of BIOS/UEFI dumps, focusing on the extraction and comparison of modules and hashes from versions submitted by users. The solution addressed the growing need for specialized tools for firmware security analysis, especially in the Brazilian context, where there is a scarcity of academic and practical initiatives focused on the topic. The platform automates the analysis process, using tools such as UEFITool, Binwalk, and Chipsec to extract, organize, and compare relevant information, such as modules and their respective hashes. The goal was to offer an accessible interface, allowing users to submit BIOS dumps for auditing without the need for deep technical knowledge. The platform also enables comparison between two or more BIOS versions, helping to identify changes and potential vulnerabilities. This work is justified by the need to consolidate a repository of BIOS versions and facilitate security reviews, contributing to the development of effective solutions in the field of embedded system security.

Keywords: BIOS, UEFI, Firmware Analysis, Security Auditing, Version Comparison, Reverse Engineering, Module Extraction, Embedded Systems Security.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

API - Application Programming Interface

CRUD – Create, Read, Update & Delete

BIOS – *Basic Input/Output System*

CSS – *Cascading Style Sheets*

DXE – *Driver Execution Environment*

ETL – *Extract, Transform, and Load*

Ghidra – *Software Reverse Engineering Tool*

GUID – *Globally Unique Identifier*

NSA – *National Security Agency*

OS – *Operating System*

SMM – *System Management Mode*

UEFI – *Unified Extensible Firmware Interface*

XSS – *Cross-Site Scripting*

GPT - *GUID Partition Table*

PEI - *Pre-EFI Initialization*

SPI - *Serial Peripheral Interface*

HII - *Human Interface Infrastructure*

EDK2 - EFI *Development Kit*

LVFS - *Linux Vendor Firmware Service*

CVE – Common Vulnerabilities and Exposures

SUMÁRIO

INTRODUÇÃO

Nos últimos anos, a segurança e a confiabilidade de componentes que operam antes do carregamento do sistema operacional, conhecidos como pré-OS, como a BIOS (Basic Input/Output System) e outros firmwares, tornaram-se questões de crescente relevância, especialmente em ambientes que demandam alta confiabilidade e segurança cibernética. Alterações em módulos de BIOS, muitas vezes invisíveis para o usuário comum, podem representar brechas de segurança ou impactar o desempenho de sistemas. Especialmente considerando que o código da BIOS pode ser extremamente extenso, o que exige grande esforço e tempo para uma análise manual completa por parte dos revisores de segurança. Apesar da importância desse tema, foi identificada uma carência significativa de ferramentas e estudos aprofundados sobre o comportamento de diferentes versões de BIOS e suas alterações. No Brasil, essa lacuna é ainda mais evidente, com pouca produção acadêmica voltada à análise técnica de BIOS e firmware. Esse contexto tornou o desenvolvimento de ferramentas que automatizam a análise dessas plataformas uma necessidade urgente para o aprimoramento da segurança.

A principal questão abordada por este trabalho foi a dificuldade em verificar o que foi alterado de uma versão de BIOS para outra, considerando o tamanho dos arquivos e a falta de detalhes técnicos nas releases notes, que geralmente apresentam informações genéricas voltadas ao público leigo. Isso dificulta a revisão técnica por especialistas e aumenta o risco de vulnerabilidades passarem despercebidas. O problema foi sintetizado nas seguintes questões norteadoras: 1. Como identificar de forma precisa e automatizada as alterações entre diferentes versões de BIOS, principalmente no que tange a aspectos de segurança? 2. De que forma a criação de um banco de dados estruturado contendo as informações de BIOS poderia facilitar futuras análises e pesquisas sobre segurança de firmware? 3. Quais ferramentas e metodologias seriam mais adequadas para automatizar a coleta e organização dos dados extraídos de arquivos de BIOS?

Diante disso, partiu-se da hipótese de que o desenvolvimento de uma plataforma que realizasse a desagregação e análise de arquivos de BIOS, por meio da extração automatizada de informações binárias e geração de hashes para cada

módulo, poderia facilitar o processo de revisão de versões de BIOS. Considerou-se que, com uma ferramenta eficaz, revisores de segurança e pesquisadores conseguiriam concentrar-se nas alterações mais significativas entre diferentes versões, o que, por sua vez, poderia agilizar o processo de auditoria e aumentar a segurança de sistemas embarcados. Adicionalmente, foi levantada a hipótese de que um banco de dados estruturado, contendo informações detalhadas sobre módulos e alterações, permitiria a futura aplicação de algoritmos de inteligência artificial para prever vulnerabilidades e identificar padrões em atualizações de firmware.

Sendo assim, o objetivo geral deste projeto foi desenvolver uma plataforma automatizada que recebesse arquivos de BIOS, realizasse a extração e análise de seus módulos e drivers, gerando hashes únicos para cada um. A plataforma foi projetada para armazenar de forma estruturada as informações extraídas em um banco de dados, permitindo a comparação eficiente entre diferentes versões de BIOS. O objetivo final foi facilitar a auditoria de segurança, identificando alterações críticas de forma automatizada e contribuindo para a análise sistemática de firmware, além de fornecer uma base de dados organizada para ser utilizada em estudos futuros.

Para atingir tais objetivos, este trabalho foi estruturado da seguinte forma: o Capítulo I apresenta a fundamentação teórica sobre segurança em baixo nível. O Capítulo II detalha a metodologia aplicada. O Capítulo III descreve os resultados obtidos com o desenvolvimento da plataforma, e nas considerações finais, é apresentado as avaliações do sistema e sugestões para trabalhos futuros.

CAPÍTULO I

1 Fundamentação teórica

1.1 Do BIOS ao UEFI

O firmware de sistema, historicamente conhecido como BIOS (Basic Input/Output System), serviu por décadas como a ponte fundamental entre o hardware de um computador e o software que ele executa. Operando em modo real de 16 *bits*, o BIOS legado era responsável pela inicialização dos componentes (*POST - Power-On Self-Test*) e pelo carregamento do sistema operacional. Contudo, suas limitações intrínsecas, como o endereçamento de memória restrito e a falta de modularidade, tornaram-se um gargalo significativo diante da crescente complexidade do hardware moderno.

Como resposta a essas limitações, a indústria desenvolveu a Interface de Firmware Extensível Unificada (UEFI). O UEFI não é uma simples atualização do BIOS; é uma substituição completa de arquitetura. Trata-se de um "mini-sistema operacional" que opera em 32 ou 64 bits, trazendo consigo capacidades avançadas como interfaces gráficas, suporte nativo a rede, gerenciamento de partições GPT (*GUID Partition Table*) e, o mais importante para o contexto deste trabalho, um design modular.

Diferente do código monolítico do BIOS, o firmware UEFI é composto por um ecossistema de *drivers* e aplicações (*Drivers DXE*, Aplicações PEI, etc.), onde cada componente é um arquivo executável (.efi) identificável por um Identificador Único Global (GUID). É precisamente essa arquitetura modular, que se assemelha a um sistema de arquivos, que permite — e exige — a análise granular de componentes individuais. Ferramentas de análise são capazes de decompor a imagem do firmware em seus módulos constituintes, permitindo a verificação de cada GUID e seus respectivos hashes, uma capacidade central explorada pela ferramenta desenvolvida neste projeto.

1.2 Por que Atacar o Firmware?

Na arquitetura de segurança de um sistema computacional, o *firmware* é a "raiz da confiança". Ele é o primeiro código executado quando o processador é energizado, sendo responsável por inicializar e validar todos os componentes subsequentes, incluindo o kernel do sistema operacional. Do ponto de vista de um atacante, comprometer essa raiz de confiança representa a forma mais elevada de controle sobre um sistema.

Em termos de privilégio, o código do firmware opera em níveis de execução mais profundos que o próprio sistema operacional. Para gerenciar a segurança, os processadores modernos utilizam um modelo de "anéis" (*Rings*), que funcionam como camadas hierárquicas de permissão. O "Anel 3" (Ring 3) é a camada mais externa e menos privilegiada, onde rodam as aplicações comuns do usuário, como navegadores e editores de texto. O "Anel 0" (Ring 0) é reservado para o núcleo (*kernel*) do sistema operacional, possuindo acesso direto ao hardware. O firmware, no entanto, opera em um nível ainda mais fundamental. Certos modos de execução do firmware, como o SMM (System Management Mode), são tão privilegiados que são frequentemente descritos como "Anel -2". Um código malicioso (implante) operando neste nível (Anel -2) é completamente invisível para softwares de segurança, como antivírus, que rodam nos anéis superiores (Anel 0 e Anel 3).

O segundo pilar é a persistência. A maioria dos malwares reside no disco rígido e é eliminada com a formatação do disco ou a reinstalação do sistema operacional. Um implante de firmware, no entanto, reside na memória flash SPI (Serial Peripheral Interface) soldada à placa-mãe. Ele é independente do disco rígido e, portanto, sobrevive a qualquer tentativa de limpeza do sistema, reinfectando-o a cada inicialização.

O modelo de ameaça também se expande para ataques à cadeia de suprimentos (*Supply Chain Attacks*), onde um dispositivo pode ser comprometido antes mesmo de chegar ao cliente final. Diante desse cenário, a única defesa viável não é apenas a prevenção, mas a verificação contínua. A capacidade de auditar o firmware de um dispositivo, comparando seus componentes e hashes contra uma "linha de base" limpa e conhecida, torna-se uma contramedida indispensável.

1.3 Vetores de Ataque e Vulnerabilidades Comuns

O comprometimento do firmware UEFI, embora complexo, é um vetor de ataque bem documentado e utilizado em ameaças avançadas. Esses ataques, conhecidos como *bootkits* ou *rootkits UEFI*, exploram vulnerabilidades na configuração da plataforma para obter acesso de escrita à memória flash SPI.

O vetor de ataque mais comum explora configurações de segurança de hardware incorretas. A memória flash SPI é protegida por mecanismos de travamento definidos no chipset, como o *bit* BIOS_CNTL e os registradores de Faixa Protegida (PRx). Estes mecanismos são projetados para impedir que o software (incluindo o sistema operacional) possa modificar o firmware após a fase de boot. No entanto, é comum que fabricantes de placas-mãe implementem essas travas de forma incorreta, deixando o firmware vulnerável a modificações maliciosas por um malware que já tenha obtido privilégios de administrador (Ring 0) no sistema operacional.

Ferramentas de auditoria de código aberto, como o Chipsec, são projetadas especificamente para inspecionar essas configurações de baixo nível. Elas geram relatórios detalhados, como as estruturas de árvore ou os mapas de componentes analisados neste projeto, que expõem falhas de configuração nas travas de SMM ou SPI.

A solução proposta neste trabalho se insere diretamente neste contexto: ela não apenas automatiza a análise desses relatórios complexos, mas também provê a defesa de segunda camada. Mesmo que uma vulnerabilidade desconhecida seja explorada, a ferramenta de comparação de hashes desenvolvida é capaz de detectar a *consequência* dessa exploração — a alteração não autorizada de módulos no firmware —, servindo como um sistema de detecção de intrusão vitalício para o componente mais crítico do sistema.

1.4 Segurança em baixo nível

O cenário de segurança de sistemas embarcados, como BIOS e UEFI, apresentou desafios crescentes à medida que novas vulnerabilidades foram descobertas e exploradas por atacantes. A UEFI, que substituiu a tradicional BIOS, trouxe uma maior flexibilidade e modernização ao processo de inicialização de sistemas operacionais. No entanto, essa complexidade adicional introduziu novos

riscos, ampliando a superfície de ataque e tornando a auditoria e análise de segurança do firmware um campo essencial, mas também desafiador.

O artigo "Analysis of the Security of UEFI BIOS Embedded Software in Modern Intel-Based Computers" (Pankov, Konoplev e Chernov 2019) ofereceu um panorama claro sobre a fragilidade do firmware UEFI em relação a ataques que exploram vulnerabilidades presentes em componentes críticos, como o System Management Mode (SMM) e o Intel Management Engine (ME). Esses componentes desempenham papéis vitais no controle de hardware e, se comprometidos, podem permitir a execução de código malicioso que afeta todo o sistema operacional, logo no início de sua execução. A análise detalhou como a falta de monitoramento contínuo e eficaz de versões de firmware pode permitir que essas falhas passem despercebidas, destacando a necessidade de ferramentas que permitam a análise profunda e contínua de diferentes versões de BIOS e UEFI para prevenir ataques a sistemas críticos.

Em paralelo, o artigo "Delving Deep into Reverse Engineering of UEFI Firmwares via Human Interface Infrastructure" (Chen et al. 2023) revelou a complexidade envolvida na engenharia reversa de firmwares UEFI. No estudo, os autores exploraram como a Human Interface Infrastructure (HII), uma camada de firmware usada para configurar opções de hardware, pode ser manipulada para comprometer as políticas de segurança de um sistema. Vulnerabilidades nas políticas de senha ou no armazenamento de dados críticos configurados por essa interface podem ser exploradas por atacantes, destacando a dificuldade que os revisores de segurança enfrentam ao tentar auditar esses sistemas manualmente. A pesquisa enfatizou a importância de um processo de auditoria que consiga acompanhar as mudanças frequentes nos módulos e seções do firmware, especialmente com a introdução constante de novos drivers e funcionalidades

O trabalho de Lebedev, Kogos e Vasilenko (2020) em "On Way to Simplify the Reverse Engineering of UEFI Firmwares" também reforçou a complexidade de auditar firmwares UEFI, focando na distinção entre módulos open-source e proprietários. Muitas empresas utilizam a base open-source da UEFI, conhecida como EDK2, para desenvolver suas próprias versões proprietárias, incluindo módulos e drivers customizados. No entanto, essa customização cria desafios adicionais para a engenharia reversa, pois os módulos proprietários tendem a ser menos estudados

pela comunidade de segurança e, portanto, são mais suscetíveis a vulnerabilidades não descobertas. O estudo descreveu como ferramentas de engenharia reversa podem ser usadas para identificar essas partes proprietárias, mas reconheceu que o processo de extração e análise desses módulos ainda era manual e demorado, especialmente em contextos em que há uma grande quantidade de dados a serem analisados.

A partir desses estudos, ficou claro que a auditoria de segurança de BIOS e UEFI ainda enfrenta muitos obstáculos, especialmente em relação à complexidade e ao volume de dados envolvidos. A engenharia reversa de firmwares, embora crucial, continua sendo uma tarefa demorada e técnica, que carece de ferramentas que possam automatizar a análise e a identificação de vulnerabilidades em larga escala. Os artigos mencionados destacaram a importância de esforços contínuos para melhorar a capacidade de auditoria e análise de segurança nesses sistemas, de modo que as vulnerabilidades possam ser identificadas e corrigidas antes de serem exploradas, garantindo maior segurança para sistemas embarcados e infraestruturas críticas.

1.5 Trabalhos relacionados

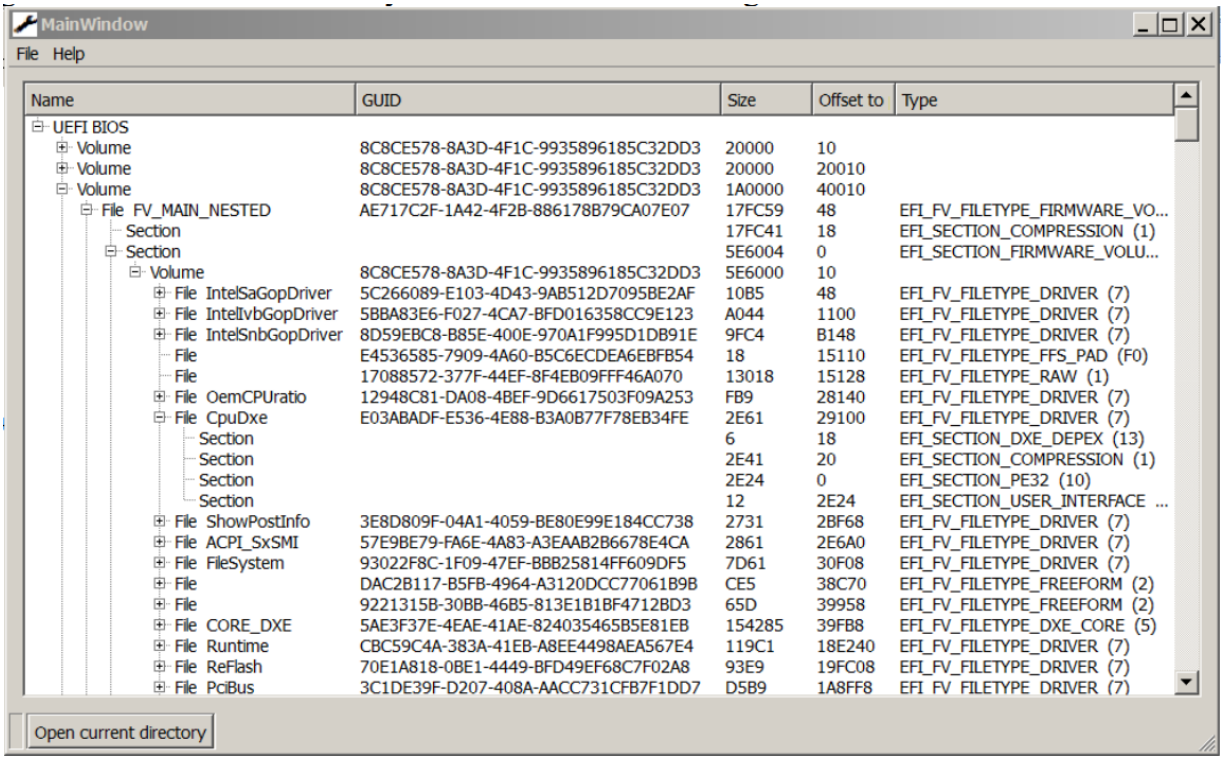
A crescente necessidade de ferramentas especializadas para a análise e auditoria de BIOS e UEFI era evidente, especialmente no contexto brasileiro, que carece de materiais acadêmicos e práticos sobre o tema. Embora existam iniciativas internacionais, como o desenvolvimento de bancos de dados de GUIDs (Globally Unique Identifier) relacionados ao UEFI, esses esforços ainda estavam em fase inicial, predominantemente conduzidos por pesquisadores e engenheiros fora do Brasil. Um exemplo é a thread "Building the best UEFI GUID database using LVFS", que buscou criar um banco de dados mais abrangente de GUIDs, evidenciando a relevância de projetos que organizem informações sobre BIOS e firmware (Schlej 2023).

Adicionalmente, a escassez de ferramentas open-source que oferecessem um repositório consolidado de versões de BIOS era uma lacuna importante. Este trabalho, portanto, teve por objetivo o desenvolvimento de uma plataforma que não apenas armazena versões de BIOS submetidas, mas também extrai módulos e gera hashes, permitindo que revisores de segurança se concentrem em alterações significativas

entre diferentes versões. Esse processo de automação, aliado ao uso de ferramentas como UEFITool, Binwalk e Ghidra, criou uma base sólida para a análise de segurança de sistemas embarcados.

Além disso, o artigo "UEFI Firmware Storage System Analysis" (Wu et al. 2016) forneceu uma análise detalhada sobre a organização interna de firmwares UEFI e o uso de identificadores GUID para seções e módulos. Este estudo ofereceu uma base teórica fundamental para a identificação e extração de módulos UEFI, possibilitando a geração de hashes para verificação de integridade e segurança. A proposta deste trabalho seguiu uma linha similar, ampliando o escopo ao compilar essas informações em um banco de dados que, no futuro, poderá se expandir para um data lake, facilitando a coleta contínua de dados e insights.

Figura 1 - Análise de arquivo de firmware



fonte: Wu et al. 2016/08

A análise de segurança de UEFI também foi um ponto fundamental para justificar o desenvolvimento desta plataforma. O artigo "SoK: Security Below the OS" (Surve et al. 2023) apontou vulnerabilidades em firmwares UEFI que podem comprometer a segurança do sistema logo no estágio inicial de boot. Essa problemática reforçou a importância de uma plataforma capaz de identificar rapidamente alterações em versões de BIOS e UEFI, auxiliando na detecção precoce

de vulnerabilidades que, se exploradas, podem comprometer todo o sistema operacional.

O estudo "On Way to Simplify the Reverse Engineering of UEFI Firmwares" (Lebedev, Kogos e Vasilenko 2020) descreveu métodos para facilitar a engenharia reversa de firmwares UEFI, identificando a importância da distinção entre módulos open-source e proprietários. No desenvolvimento da plataforma proposta, esses métodos foram particularmente relevantes para a extração automatizada de módulos DXE e SMM, que podem conter vulnerabilidades em módulos proprietários. O armazenamento desses dados no banco de dados MongoDB também permitiu uma análise histórica e a identificação de padrões de vulnerabilidades entre diferentes versões de BIOS.

Finalmente, o artigo "Analysis of the Security of UEFI BIOS Embedded Software in Modern Intel-Based Computers" (Pankov, Konoplev e Chernov 2019) demonstrou como falhas em firmwares podem ser exploradas por atacantes, especialmente em componentes críticos como o System Management Mode (SMM). A criação de uma plataforma que permite a auditoria e comparação automatizada de versões de BIOS respondeu diretamente a essa necessidade, ao fornecer uma solução eficaz para a detecção de alterações em módulos sensíveis, ajudando a mitigar os riscos discutidos no artigo.

Quadro 1 *- Análise de Artigos sobre engenharia reversa de UEFI*

Título	Autores	Foco Principal	Metodologia Utilizada	Relevância para Engenharia Reversa	Segurança	Prototipagem	Vulnerabilidades
UEFI Firmware Storage System Analysis	Wu,Weiming et al.	Organização interna de firmware UEFI	Análise de arquivos UEFI	x		x	
SoK: Security Below the OS	Surve, PriyankaP. Et al.	Análise de segurança da UEFI	Testes e experimentações	x	x		x
Analysis of the Security of UEFI BIOS Embedded Software	Pankov,I.D. et al.	Vulnerabilidades do firmware UEFI/BIOS	Revisão e análise de segurança	x	x		x
On Way to Simplify the Reverse Engineering of UEFI	Lebedev, Philip et al.	Engenharia reversa de firmwares UEFI	Engenharia reversa e scripts	x	x	x	
Delving Deep into Reverse Engineering of UEFI Firmwares	Chen,Siyi et al.	Engenharia reversa de módulo de infraestrutura	Engenharia reversa	x	x		x

Fonte – Autoria própria

CAPÍTULO II

2 Metodologia

2.1 Natureza da Pesquisa

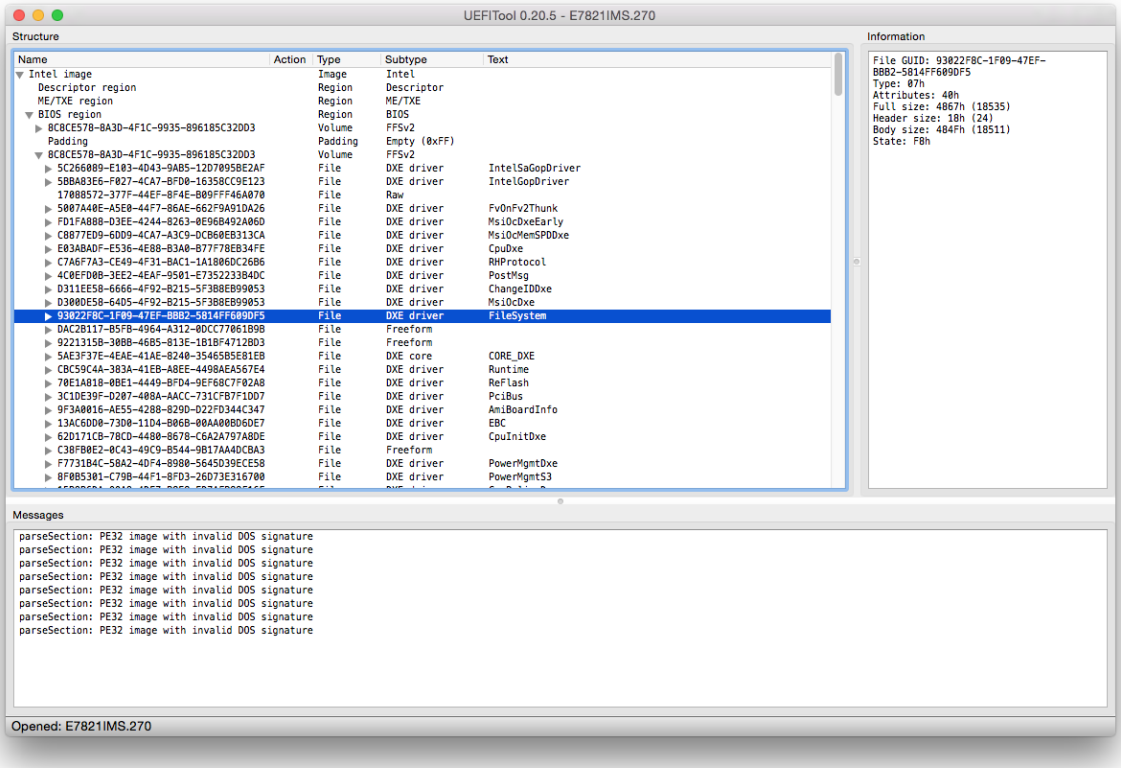
Este projeto enquadrou-se como uma **pesquisa aplicada**, pois foi voltado para o desenvolvimento de uma plataforma que automatiza a análise e auditoria de dumps de BIOS/UEFI. A pesquisa aplicada focou na resolução de problemas práticos e, neste caso, abordou uma lacuna específica no mercado em relação à auditoria automatizada de firmwares, especialmente no contexto da segurança cibernética. O sistema proposto permitiu ao revisor realizar análises detalhadas de BIOS/UEFI, extraindo módulos, gerando hashes e comparando diferentes versões. Com isso, o projeto objetivou oferecer uma ferramenta eficiente e precisa para profissionais de segurança, ampliando o alcance de auditorias e melhorando a detecção de vulnerabilidades em firmwares.

2.2 Ferramentas de Análise de Firmware

A plataforma desenvolvida baseou-se em um conjunto de ferramentas especializadas e consolidadas no mercado para realizar a extração e análise de dados de BIOS e UEFI. Cada uma dessas ferramentas desempenhou um papel específico no processamento dos dumps de BIOS, desde a extração de binários até a geração de hashes para comparação. As principais ferramentas utilizadas foram:

- **UEFITool**: Ferramenta central para a visualização e extração de módulos e submódulos de BIOS/UEFI, permitindo também a geração de hashes dessas partes. Esta ferramenta tem a sua própria interface de análise, conforme imagem abaixo, o que foi utilizado neste projeto foi o UEFIExtract, ferramenta cli(command line interface), ferramenta de linha de comando do UEFITool responsável por extrair os módulos do binário.

Figura 2 - Interface da ferramente UEFITool



- **Binwalk**: Utilizada para a análise e extração de imagens de firmware, sendo amplamente empregada para identificar e desmontar diferentes componentes do firmware para um estudo detalhado. É uma ferramenta apenas de linha de comando, sem interface gráfica.
- **InnoExtract**: Empregada para extrair arquivos de instaladores executáveis, funcionalidade essencial para obter os binários de firmware quando estes estão empacotados em arquivos .exe. Também uma ferramenta de linha de comando sem interface gráfica.
- **Chipsec**: É uma ferramenta de linha de comando utilizada para extrair o dump da BIOS diretamente da máquina do usuário, e extrair módulos de binários, possibilitando uma auditoria do firmware em uso no momento da análise.

Quadro 2 - Ferramentas de Análise e Engenharia Reversa

Título	Principal Função	Tipo de Arquivo	Engenharia Reversa	Extração de Dados	Análise de Seções	Extração de HASHs
UEFITool	Visualizar e extrair módulos de BIOS/UEFI	BIOS/UEFI		x	x	x
Binwalk	Análise e extração de imagens de firmware	BIOS/UEFI	x	x	x	x
InnoExtract	Extrair arquivos de instaladores executáveis	Executáveis, ISOs		x		
Detect It Easy	Análise de seções de módulos/firmware	Drivers, Módulos			x	
Ghidra	Engenharia reversa de software e firmware	Software e Firmware	x	x	x	x

Fonte – Autoria própria

2.3 Ferramentas de Desenvolvimento

Arquitetura da plataforma foi projetada para ser modular, escalável e eficiente, separando a interface do usuário do processamento pesado de análise de firmware. A seguir, são apresentadas as ferramentas e tecnologias utilizadas no desenvolvimento:

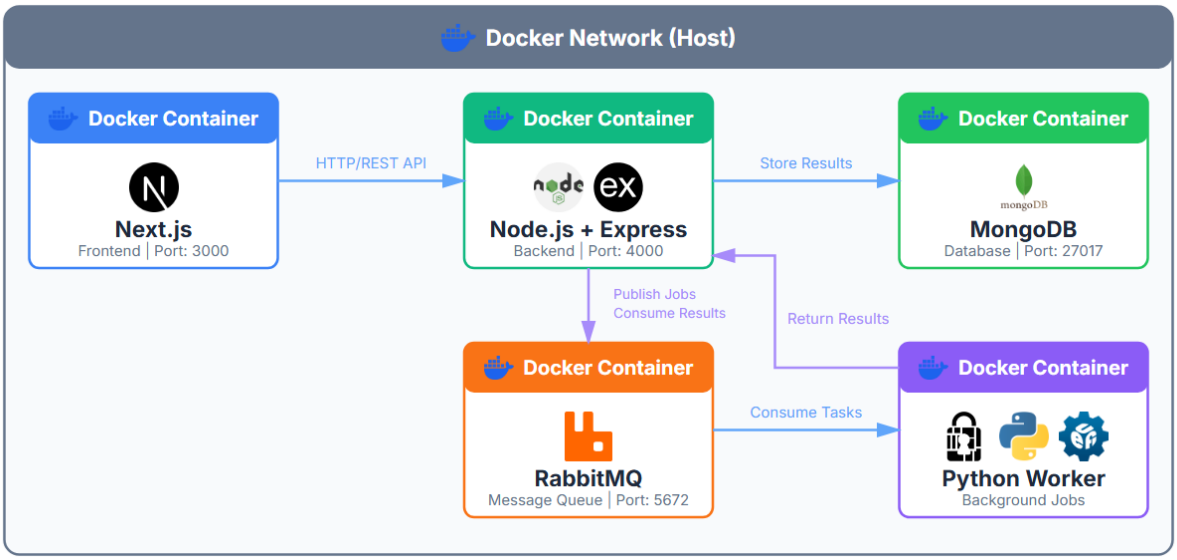
- **React e Next.js (Frontend):** A interface do usuário foi construída utilizando **React**, uma biblioteca JavaScript para o desenvolvimento de aplicações front-end baseadas em componentes, o que permite o reaproveitamento de código e facilita a manutenção. Sobre essa base, foi utilizado o framework **Next.js**, que oferece uma estrutura robusta com renderização no lado do servidor (Server-Side Rendering - SSR) para um carregamento inicial mais rápido, além de um sistema de roteamento automático que organiza as rotas com base na estrutura de arquivos do projeto, otimizando a experiência do desenvolvedor e do usuário final.

- **Express.js e Node.js (Backend):** O backend, responsável por gerenciar as requisições da API, a autenticação de usuários e a lógica de negócio, foi desenvolvido em JavaScript utilizando Node.js com o framework Express.js. Conhecido por sua abordagem minimalista e flexível, o Express.js foi utilizado para construir o servidor que orquestra a comunicação entre o frontend, o banco de dados e o sistema de processamento assíncrono, garantindo uma base leve e performática para a aplicação.
- **Docker, Python e RabbitMQ (Processamento de Análise):** Para executar a extração e análise dos firmwares, foi adotada uma arquitetura de microserviço. As ferramentas de análise (UEFITool, Binwalk etc.), escritas em Python, foram encapsuladas em um container Docker. Essa abordagem garantiu um ambiente de execução isolado e consistente, eliminando problemas de dependência. A comunicação entre o backend (Express.js) e este container foi gerenciado pelo RabbitMQ, um message broker. Quando um usuário realizava o upload de um arquivo, o backend publicava uma mensagem em uma fila do RabbitMQ. O serviço no container Docker, por sua vez, consumia essa mensagem, executava os scripts de análise de forma assíncrona e, ao final, reportava os resultados. Essa arquitetura desacoplada foi crucial para que tarefas pesadas e demoradas não travassem a aplicação principal.
- **MongoDB (Banco de Dados):** Foi utilizado o MongoDB, um banco de dados não relacional (NoSQL), para o armazenamento dos dados. Sua flexibilidade para lidar com dados, como os módulos e hashes extraídos, e sua escalabilidade horizontal, foram características decisivas para a escolha, já observando a expansão futura do sistema para um *data lake*.
- **Render (Hospedagem):** A aplicação completa foi hospedada na plataforma render, que simplificou o processo de implantação. O Render foi configurado para monitorar o repositório do projeto no Git, gerenciando automaticamente a compilação e a implantação do código a cada nova alteração, disponibilizando tanto o frontend quanto o backend em uma URL pública.

2.4 Diagrama de Arquitetura do Sistema

A arquitetura que foi utilizada no sistema está desenhada no diagrama abaixo, onde mostra que a porta de entrada para o sistema é o *frontend*, o usuário revisor acessará o sistema com seu usuário e senha e poderá fazer as operações de upload de BIOS via sistema e as comparações, após um upload de bios, o sistema enviará um payload para a fila do *rabbitmq*, o *Worker* python estará escutando esta fila, após identificar o payload ele realizará as extrações no binário enviado pelo usuário, após a extração, ele devolverá para o backend o json extraído do binário com todas as informações daquele binário de módulos e GUID para serem inseridas no banco, isto é feito para manter a segurança do banco, onde apenas o backend pode modificar o banco de dados.

Figura 3 - Diagrama de arquitetura do sistema



Fonte *– Autoria própria*

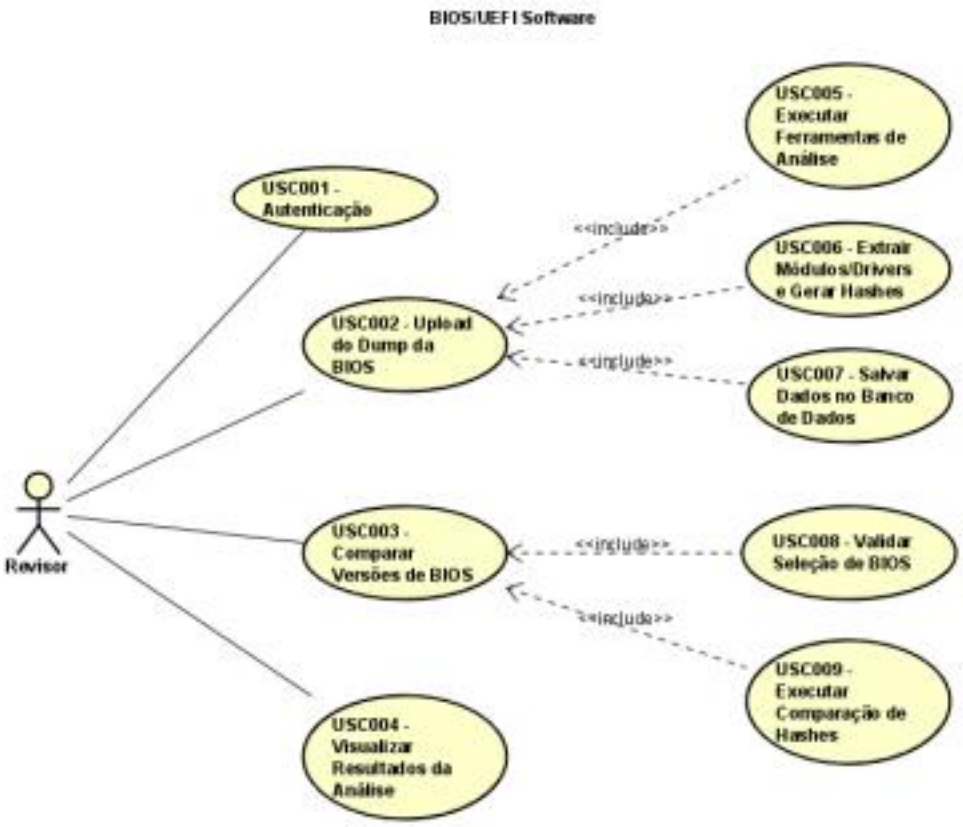
2.5 Casos de Uso

Os casos de uso detalharam as principais interações que o revisor pôde realizar com a plataforma finalizada:

- **USC001 - Autenticação:** O revisor precisou se autenticar no sistema para acessar suas funcionalidades, como o upload de dumps de BIOS e o uso das ferramentas de análise disponíveis.

- **USC002 - Upload dos Dumps da BIOS:** O revisor pôde realizar o upload de um ou mais arquivos de BIOS para análise. Este caso de uso incluiu a execução automática das ferramentas de análise (UEFITool, etc.), a extração de módulos/drivers com a geração de seus hashes e o salvamento de todos os dados extraídos no banco de dados MongoDB para consultas futuras.
- **USC003 - Comparar Versões da BIOS:** A plataforma permitiu que o revisor comparasse duas versões de BIOS previamente enviadas, visualizando as diferenças entre os módulos e seus respectivos hashes.
- **USC004 - Visualizar Resultados de Análise:** O revisor teve acesso aos resultados das análises, como a lista de módulos extraídos e os relatórios de comparação de hashes, em uma interface clara e organizada.
- **USC005: Executar Ferramentas de Análise:** Este caso de uso representa a ação do sistema de rodar as ferramentas específicas para analisar o conteúdo do dump da BIOS. Ele é mostrado como uma funcionalidade incluída (<<include>>) pelo USC002 - Upload de Dump da BIOS, indicando que a execução dessas ferramentas é uma parte obrigatória do processo iniciado pelo upload.
- **USC006: Extrair Módulos/Drivers e Gerar Hashes:** Este caso de uso detalha o processo de identificar, extrair os componentes individuais (módulos, drivers) dentro do dump da BIOS e calcular os hashes criptográficos de cada um. Assim como o USC005, ele é incluído (<<include>>) pelo USC002 - Upload de Dump da BIOS, sendo outra etapa essencial da análise.
- **USC007: Salvar Dados no Banco de Dados:** Representa a ação final do processo de análise iniciado pelo upload: persistir os resultados obtidos (informações do BIOS, módulos extraídos, hashes gerados) no banco de dados do sistema. Também é incluído (<<include>>) pelo USC002 - Upload de Dump da BIOS.
- **USC008: Validar Seleção de BIOS:** Este caso de uso descreve a validação que o sistema realiza quando o usuário tenta comparar duas versões de BIOS. Ele é incluído (<<include>>) pelo USC003 - Comparar Versões de BIOS. Isso sugere que, antes de prosseguir com a comparação, o sistema verifica se as duas versões selecionadas são válidas e adequadas para serem comparadas (por exemplo, se não são a mesma versão, se existem no banco etc.).

Figura 4 - Diagrama de casos de uso



Fonte *– Autoria própria*

2.6 Conceitos Gerais de Firmware e Análise

No que diz respeito à segurança de firmware, diversas palavras, siglas e termos técnicos, tanto em português quanto em inglês, são convencionadas para se referirem a características específicas da modalidade. A seguir, são listadas algumas das expressões mais comuns utilizadas ao longo deste trabalho:

- **BIOS (*Basic Input/Output System*)**: Refere-se ao firmware de sistema historicamente utilizado, servindo como a ponte fundamental entre o hardware de um computador e o software que ele executa.
- **UEFI (*Unified Extensible Firmware Interface*)**: É a substituição completa da arquitetura do BIOS. É considerado um "mini-sistema operacional" que, diferentemente do código monolítico do BIOS, possui um design modular.
- ***Dump* de BIOS/UEFI**: É o termo usado para o arquivo binário completo do firmware, que é submetido por um usuário (revisor) à plataforma para ser processado e auditado.

- **Módulos e GUIDs**: Como o firmware UEFI possui uma arquitetura modular, ele é composto por um ecossistema de drivers e aplicações. Estes componentes individuais são os módulos extraídos e analisados pela plataforma. Cada módulo é identificável por um GUID (*Globally Unique Identifier*), ou Identificador Único Global.
- ***Hash***: Refere-se a um valor criptográfico único (neste projeto, o SHA256) gerado para cada módulo extraído. O objetivo da plataforma é usar esses *hashes* para permitir a comparação eficiente entre diferentes versões de BIOS, identificando rapidamente quais módulos sofreram alterações.
- **Ferramentas de Análise (UEFITool, Binwalk, Chipsec)**: Conjunto de ferramentas de software especializadas, consolidadas no mercado, que a plataforma automatiza para realizar a análise.
    - **UEFITool**: É a ferramenta central para visualização e extração de módulos de BIOS/UEFI. O projeto utiliza especificamente o UEFIExtract, sua ferramenta de linha de comando, para extrair os módulos do binário.
    - **Binwalk**: Utilizada para a análise e extração de imagens de firmware, permitindo identificar e desmontar seus diferentes componentes.
    - **Chipsec**: Ferramenta de linha de comando empregada para extrair o *dump* da BIOS diretamente da máquina do usuário, permitindo a auditoria do firmware em uso no momento da análise.
- **Engenharia Reversa**: O processo de análise de *firmwares* para entender seu funcionamento interno. É uma tarefa tradicionalmente manual, demorada e que exige alto conhecimento técnico, sendo o principal processo que este trabalho mirou em automatizar.

2.7 Experimento de Pesquisa

A fim de fundamentar o desenvolvimento da plataforma em processos concretos e realistas, o experimento de pesquisa se baseou no fluxo de trabalho de revisores de segurança e especialistas na auditoria de firmwares BIOS/UEFI. A principal dificuldade identificada nesse processo é a verificação manual do que foi alterado entre uma versão de BIOS e outra, uma tarefa demorada, complexa e suscetível a erros, visto o tamanho dos arquivos e a falta de detalhes técnicos nas *releases notes* oficiais.

A proposta é que a aplicação seja utilizada por esses revisores para automatizar e agilizar essa auditoria. O revisor pode submeter um ou mais *dumps* de BIOS para a

plataforma. O sistema, então, executa de forma assíncrona a extração automatizada de todos os módulos e a geração de seus *hashes*, armazenando os resultados em um banco de dados. Com isso, o revisor pode utilizar a principal funcionalidade da plataforma: comparar duas versões de BIOS. O sistema exibe as diferenças, permitindo que o profissional foque sua análise apenas nas alterações, adições ou remoções de módulos, identificando potenciais vulnerabilidades de forma muito mais eficiente.

Assim, o desenvolvimento do projeto foi dividido em duas partes principais: a criação de uma arquitetura de *backend* e processamento assíncrono para a análise pesada dos firmwares, e a elaboração de uma aplicação web (*frontend*) para a interação com o usuário.

Na construção do *backend*, que atuou como o orquestrador do sistema, foi utilizada uma arquitetura de microserviços. O servidor principal, desenvolvido em Node.js com o framework Express.js, é responsável por gerenciar as requisições da API, a lógica de negócio e a autenticação. Para garantir que a análise de firmware, uma tarefa pesada e demorada, não travasse a aplicação, as ferramentas de análise (como UEFITool, Binwalk e Chipsec) foram encapsuladas em um *container* Docker separado. Este *worker* de processamento utiliza *scripts* em Python para executar as extrações.

A comunicação entre o *backend* (*Express*.js) e este *worker* (Python/Docker) foi gerenciada pelo *RabbitMQ*, um *message broker*. Quando um revisor realiza o *upload* de um arquivo, o *backend* publica uma mensagem na fila do *RabbitMQ*. O *worker* de análise, que escuta essa fila, consome a tarefa, executa todos os *scripts* de extração de forma assíncrona e, ao finalizar, devolve os resultados para o *backend*, que os insere no banco de dados.

Para o armazenamento, os dados extraídos são persistidos no *MongoDB*, um banco de dados não relacional (*NoSQL*). A escolha do MongoDB foi motivada por sua flexibilidade para lidar com dados não estruturados, como as listas de módulos e *hashes* extraídos, e sua alta escalabilidade, visando a expansão futura do sistema para um *data lake*.

A construção da aplicação web, responsável pela interação com o revisor, contou com interfaces desenvolvidas em React com o framework Next.js. O React foi

utilizado para a criação de componentes reutilizáveis, enquanto o *Nextjs* foi escolhido por sua estrutura robusta, que oferece renderização no lado do servidor (SSR) para um carregamento inicial mais rápido e um sistema de roteamento automático. A aplicação completa foi hospedada na plataforma render, que simplificou o processo de implantação, monitorando o repositório Git e gerenciando automaticamente a compilação e o *deploy* do código.

2.8 Prototipação

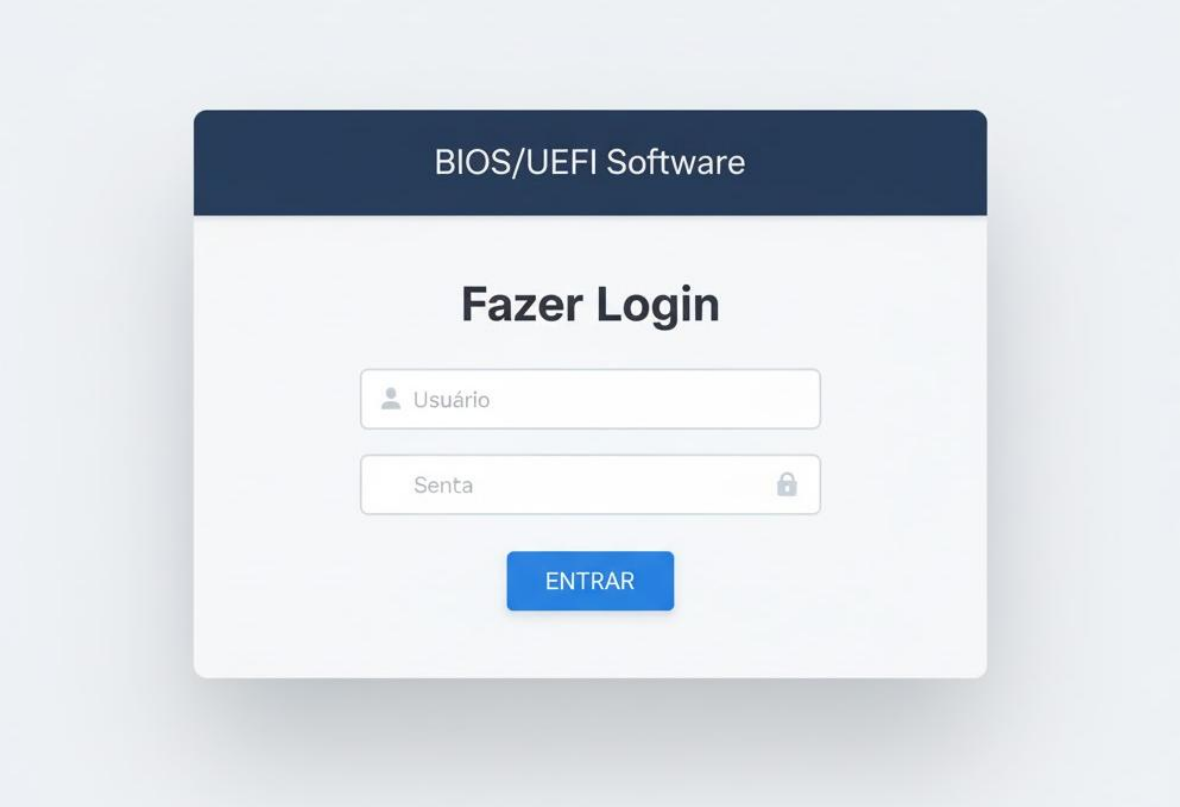
Protótipos foram desenvolvidos para observar e entender como se daria o fluxo desenvolvido previamente, com o objetivo de facilitar o desenvolvimento do sistema.

Para isso, foram pensadas nas seguintes telas:

- Tela de Login
- Tela de Dashboard
- Tela de Upload de BIOS
- Tela de Comparar versões
- Tela de resultados da Análise

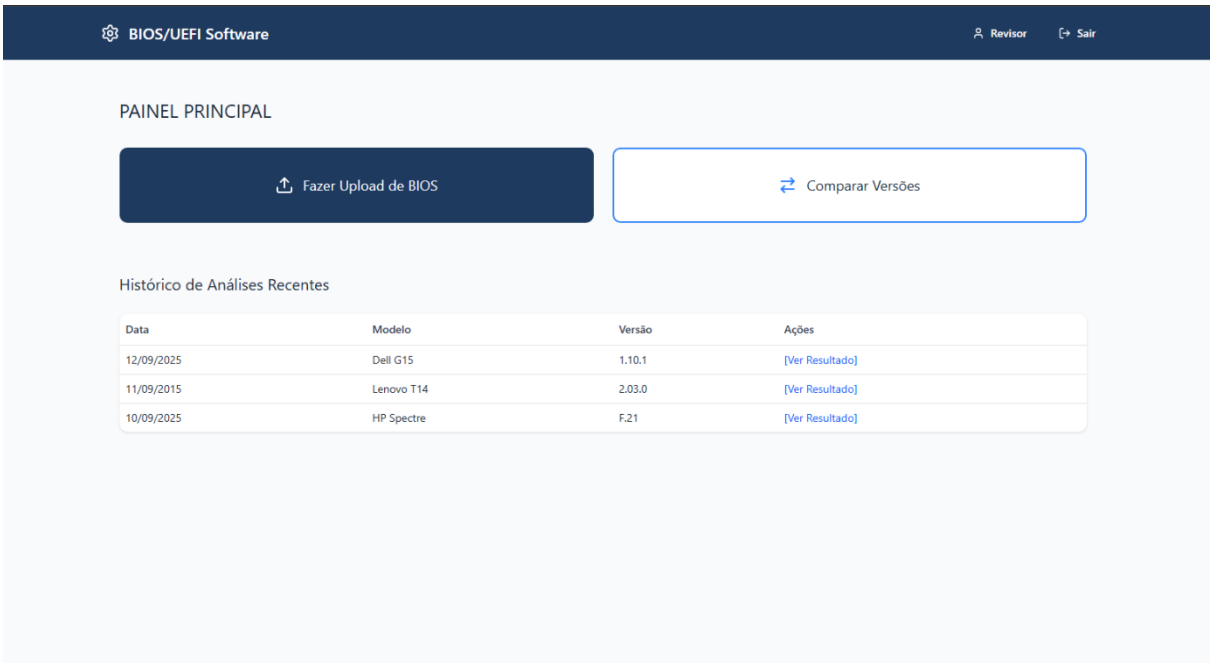
Após autenticação na tela inicial que é a de login, o Revisor é redirecionado para a tela inicial de Dashboard, onde ele terá acesso as funções principais do sistema.

Figura 5 - Protótipo Tela de Login



Fonte *– Autoria própria*

Figura 6 - Protótipo Tela principal (Dashboard)

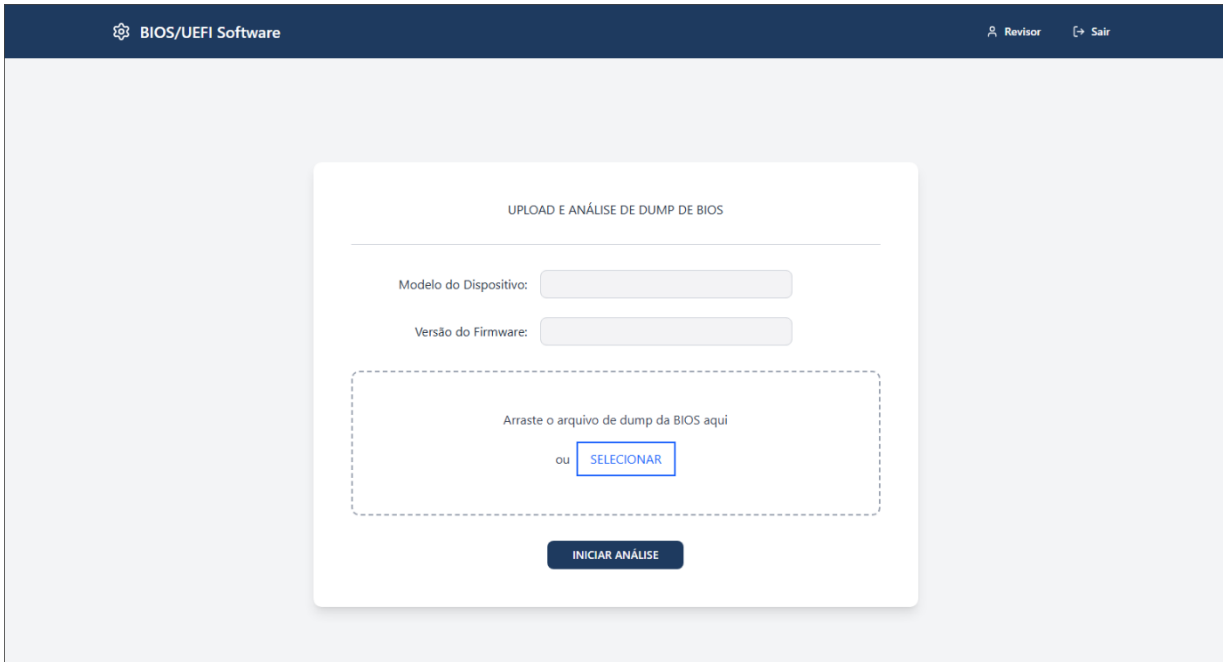


Fonte *– Autoria própria*

Na tela de *Dashboard*, o usuário poderá acessar os detalhes dos binários já extraídos, e observar o status da extração, tanto quanto acessar as funções de fazer upload da BIOS e comparar diferentes versões de BIOS que já constam no sistema.

Ao acessar a tela de *Upload* de BIOS, o usuário deverá inserir a versão do *firmware* que ele planeja extrair e qual o dispositivo em questão, após, inserir o binário ou pacote do binário para iniciar a extração.

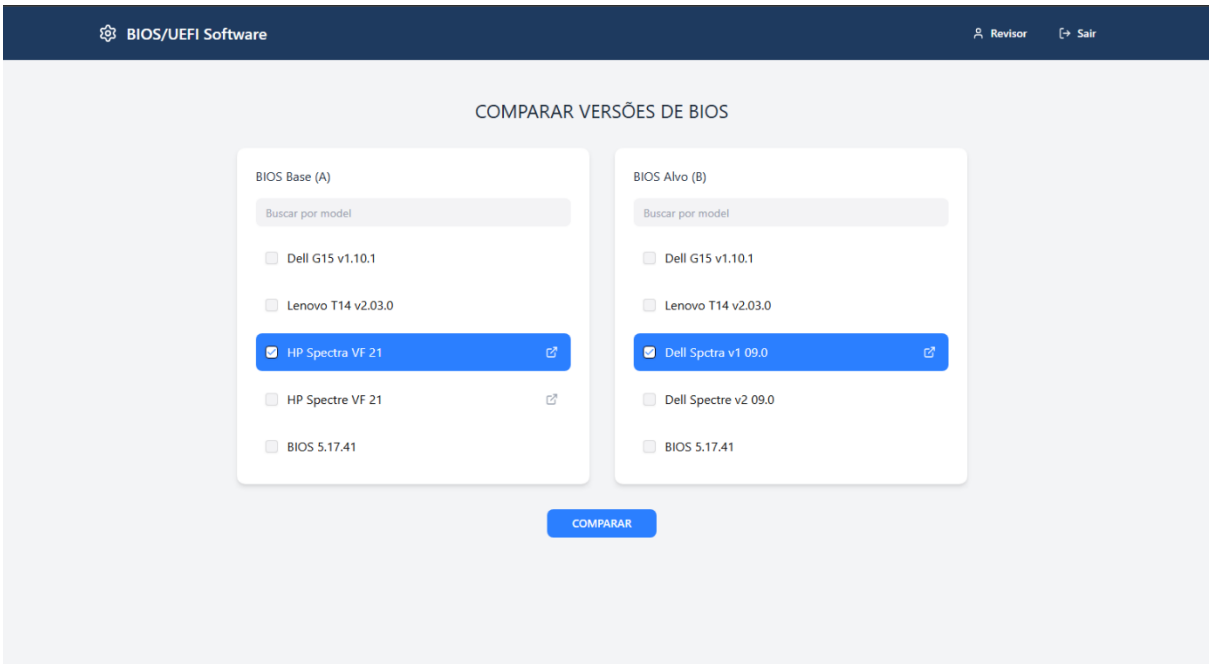
Figura 7 – Protótipo Tela de Upload (Envio do binário)



Fonte *– Autoria própria*

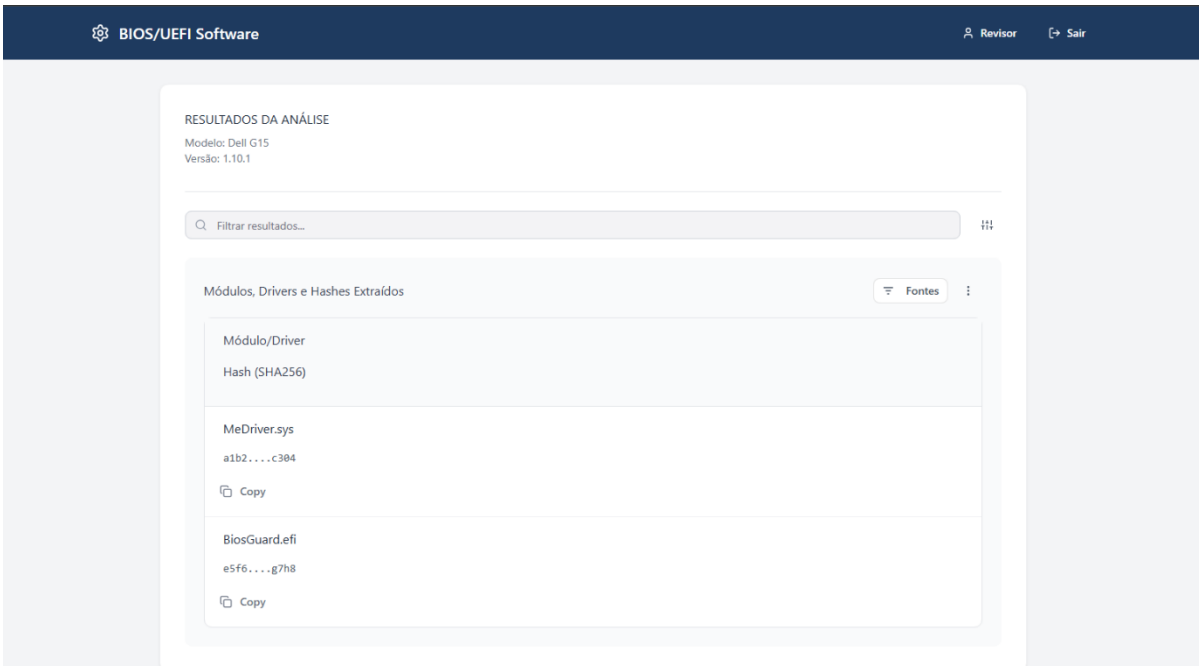
Com 2 ou mais binários já inseridos no sistema, o usuário poderá fazer a comparação de módulos entre ambos, para identificar quais módulos são iguais, quais são diferentes, quais foram modificados e quais constam em apenas um ou outro binário.

Figura 8 – Protótipo Tela de comparação de BIOS



Fonte *– Autoria própria*

Figura 9 - Protótipo tela de resultados



Fonte *– Autoria própria*

Após uma comparação realizada, o sistema redirecionará o usuário para o resultado daquela comparação, mostrando quais módulos foram encontrados iguais e diferentes dentre os binários comparados.

2.9 Critérios para avaliação da ferramenta

Com a finalização do desenvolvimento do sistema, ele foi submetido a análise de usuários para apreciar sua avaliação. Os usuários que foram escolhidos para testar a aplicação foram usuários com algum envolvimento com BIOS e Firmware, usuários como desenvolvedores de software, revisores e auditores de BIOS. Para coleta dos resultados foi disponibilizado um formulário do google para os usuários responderem com um retorno sobre a aplicação, que foi construído baseado no que a aplicação propôs como objetivos.

CAPÍTULO III

3 Apresentação, documentação e análise de dados

Este capítulo apresenta os resultados obtidos com o desenvolvimento da plataforma de análise de BIOS/UEFI. O foco é demonstrar como as ferramentas foram integradas, como os dados foram gerenciados e quais funcionalidades foram entregues na versão final do sistema, validando os objetivos propostos.

O nome da ferramenta foi determinado como *BIOS Analyzer* observando o que o sistema faz na prática, extrai binários e dispõe de análises para o usuário que pode ter *insights* baseado nisso.

As cores utilizadas no sistema foram azul e branco, utilizando de uma interface minimalista e direta a proposta final da ferramenta.

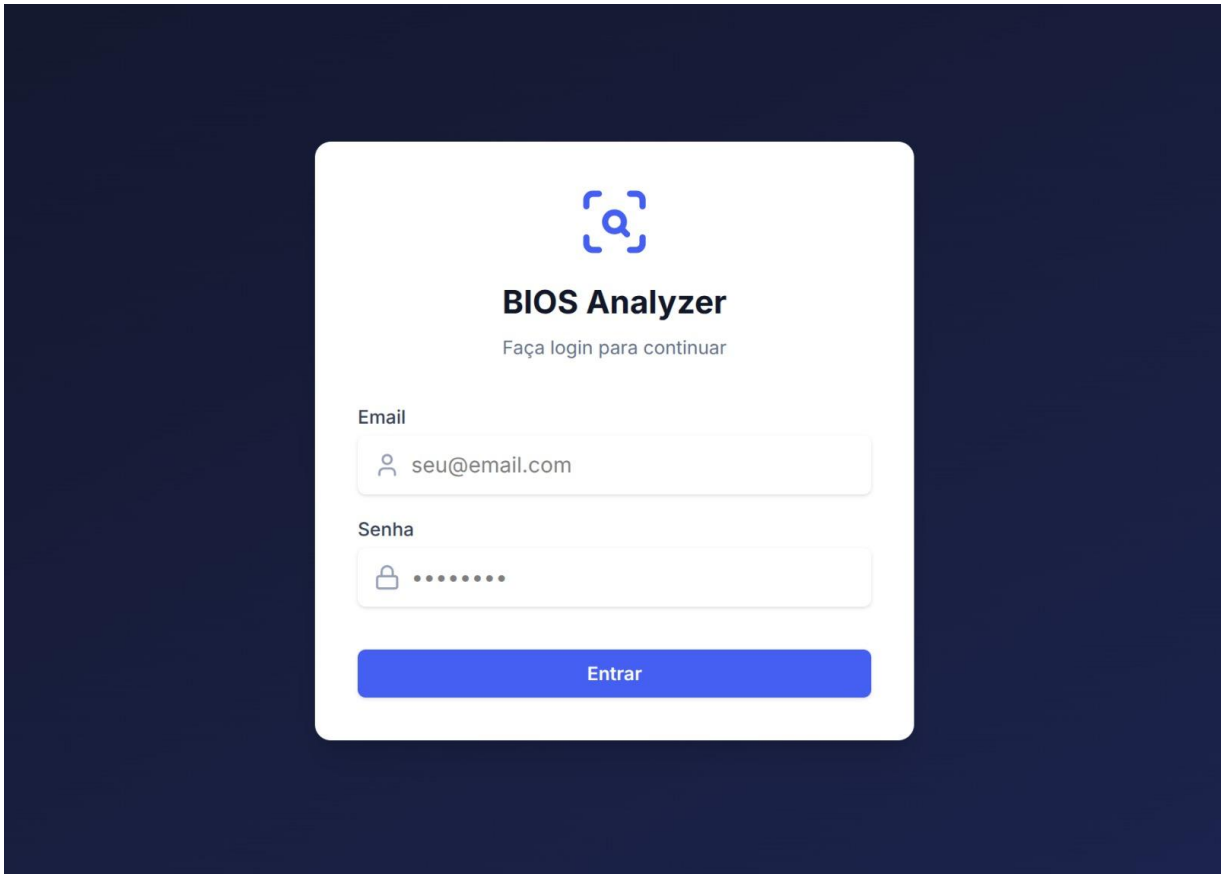
3.1 Desenvolvimento da interface

Baseado nos protótipos desenvolvido anteriormente, a aplicação frontend foi desenvolvida, as quais serão apresentadas na sequência.

3.1.1 Tela de *Login*

A primeira tela que o usuário tem acesso, ele utiliza de email e senha previamente cadastrados para acessar o sistema, com o login, ele recebe um token que espira depois de um tempo de uso, com o token espirado, o usuário deverá refazer o login para acessar as funcionalidades do sistema. Para conseguir um acesso, o usuário deverá contactar o administrador do sistema.

Figura 10 - Tela de Login

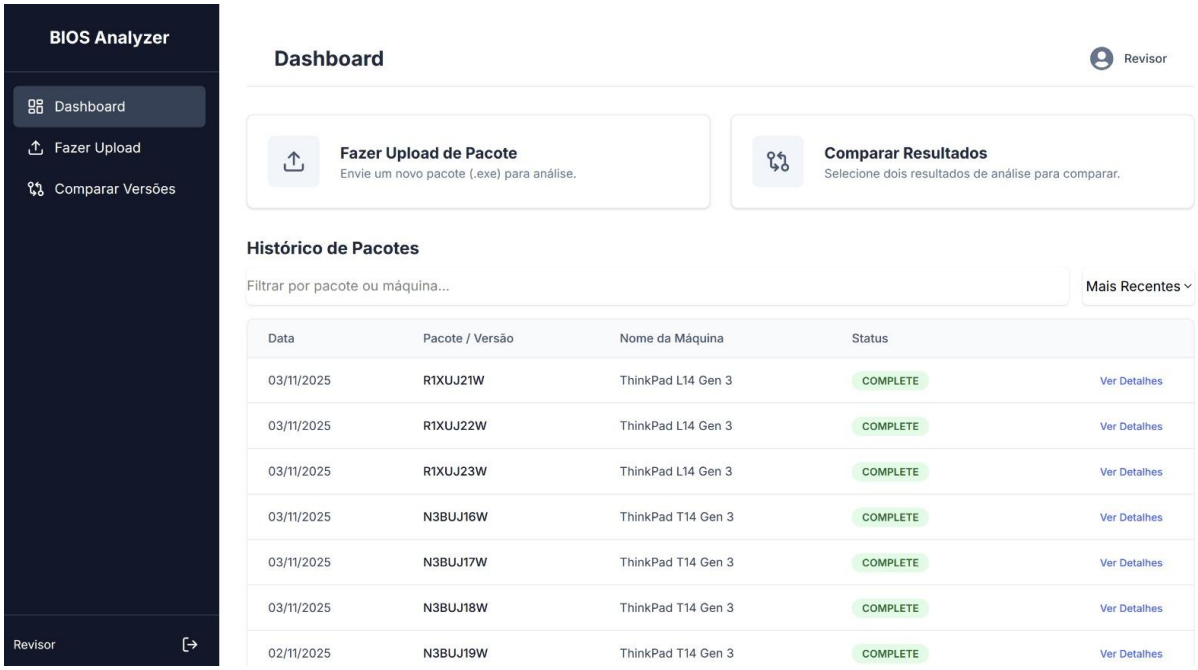


Fonte *– Autoria própria*

3.1.2 Tela do Menu principal (*Dashboard*)

Está é a tela principal do sistema, nela, o usuário visualizará todas os binários já analisados no sistema, acessar os detalhes da análise do binário, fazer o *upload* de novos binários e comparar binários já existentes.

Figura 11 - Tela Menu Principal

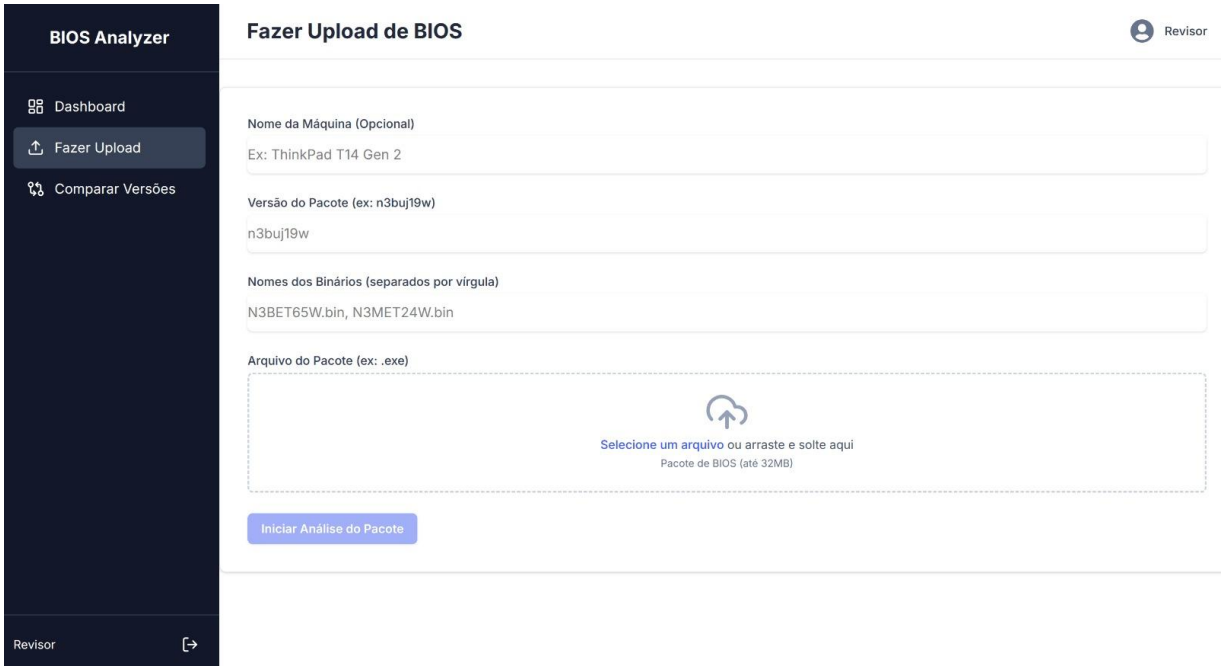


Fonte *– Autoria própria*

3.1.3 Tela de *Upload* de Binários

Esta tela é essencial, é onde o usuário poderá inserir novos componentes binários no sistema, o campo de dispositivo é opcional, mas o usuário deve preencher os campos de pacote e binário, com isso, o usuário seleciona o binário no explorador de arquivos para extração e análise.

Figura 12 - Tela de Upload de binários



Fonte *– Autoria própria*

3.1.4 Tela de comparar Binários

Com o binário no sistema, o usuário poderá realizar a função principal do sistema, a de comparação de binários, terão dois campos onde ele selecionará dentre os diversos binários no campo da direita para comparar com o binário selecionado no campo da esquerda, o usuário poderá selecionar o mesmo binário em versão diferente, ou binários totalmente diferentes para comparação.

Figura 13 - Tela de comparação de BIOS

Fonte *– Autoria própria*

3.1.5 Tela de resultados da análise (Comparação)

Esta tela é onde o usuário verá os resultados da comparação, é nela que ele poderá ter *insights* sobre uma comparação, o sistema mostrará 4 campos principais, o de módulos idênticos, módulos modificados que são módulos com o mesmo GUID mas com *hashes* diferentes, e os campos de módulos presentes apenas no binário 1, ou no binário 2.

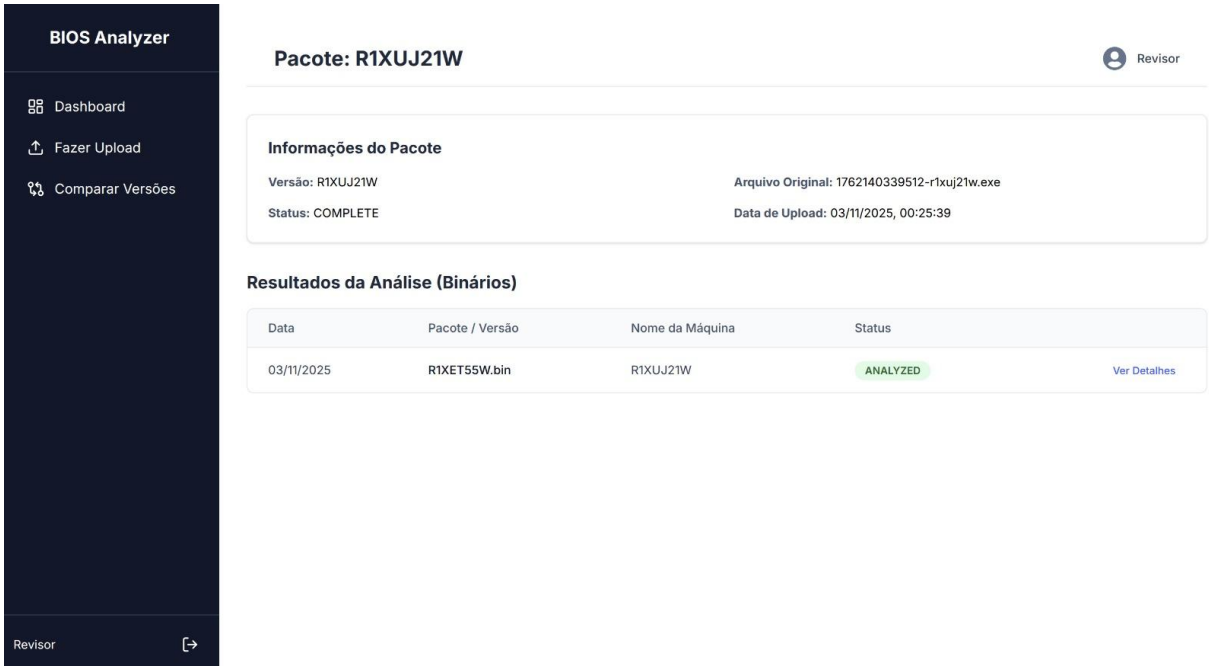
Figura 14 - Tela de resultados da Análise

Fonte *– Autoria própria*

3.1.6 Tela de detalhes do Pacote

Durante o desenvolvimento do projeto, foi identificado que um pacote poderia ter um ou mais binários dentro dele, por isso, optou-se por separar na tela inicial de *dashboard* os pacotes, e ao clicar em detalhes, o usuário seria levado para esta tela de detalhes do pacote, onde ele poderá ver os binários que estão vinculados a este pacote, e ver os detalhes deste binário.

Figura 15 - Tela de detalhes do pacote



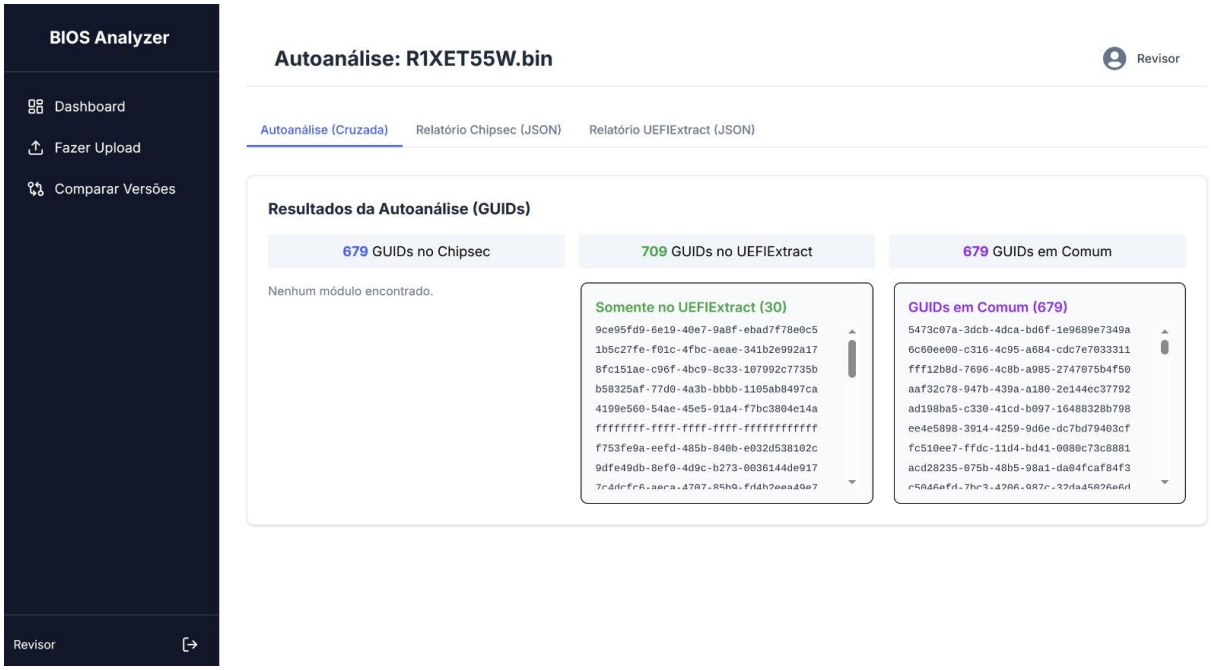
Fonte *– Autoria própria*

3.1.7 Tela de Autoanálise

Após o usuário observar os binários disponíveis dentro de um pacote na tela anterior de detalhes do pacote, e clicar em detalhes de um dos binários, ele será redirecionado para a tela de autoanálise, que nada mais é que uma tela que mostra os detalhes daquele binário, incluído uma comparação entre os GUIDs extraídos por ambas as ferramentas do *UEFIExtract* e Chipsec. Esta comparação identifica os módulos em comum de um mesmo binário extraído por ferramentas diferentes e os módulos presentes apenas em cada uma das ferramentas, respectivamente.

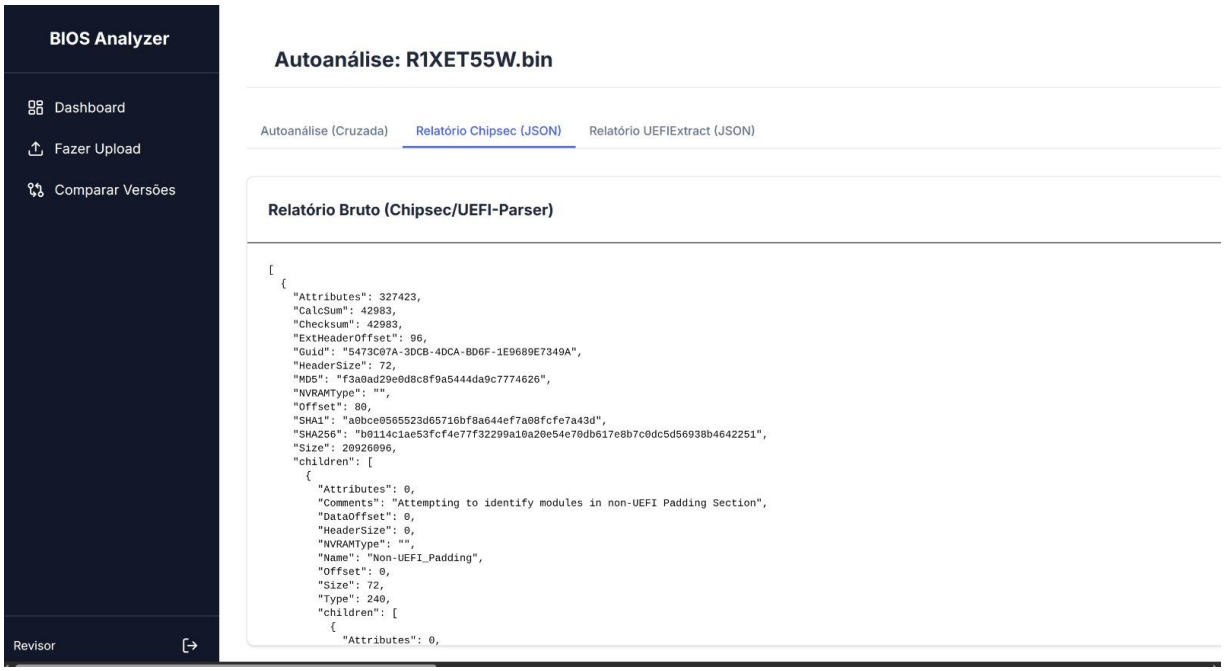
Também é possível observar nesta tela os dados do relatório que a ferramenta do Chipsec e do UEFIExtract.

Figura 16 - Tela de Autoanálise



Fonte *– Autoria própria*

Figura 17 - Tela de Autoanálise (Relatório Chipsec)



Fonte *– Autoria própria*

3.2 Avaliação do sistema

Conforme descrito na seção 2.9, com a finalização do desenvolvimento do protótipo funcional, o sistema foi submetido a uma avaliação de usuários para validar sua proposta de valor e eficácia na resolução do problema proposto. Foram coletadas 16 respostas no total. Um ponto fundamental desta avaliação é que a totalidade dos respondentes se enquadrou perfeitamente no público-alvo planejado para o sistema, sendo eles: 7 Desenvolvedores de Software (com foco em firmware ou segurança), 5 Revisores ou Auditores de BIOS/Firmware, 2 Pesquisadores de Segurança e 2 que se identificaram como "Outro". A participação exclusiva de especialistas da área tornou a avaliação altamente qualificada.

Os resultados quantitativos e qualitativos obtidos foram majoritariamente positivos. Houve unanimidade entre os 16 especialistas (100%) em considerar o objetivo da aplicação como "Muito importante". Isso valida de forma incontestável que o problema da análise manual de firmwares é uma "dor" real, complexa e relevante para os profissionais da área.

A usabilidade geral da plataforma foi bem recebida. 62,5% (10 de 16) dos usuários classificaram a navegação (acesso às funcionalidades de Dashboard, Upload e Comparação) como "Fácil", enquanto 37,5% (6 de 16) a classificaram como "Razoável". De forma similar, 68,75% (11 de 16) consideraram a interface "Agradável, limpa e fácil de entender", contra 31,25% (5 de 16) que a definiram como "Razoavelmente agradável". Nenhum usuário classificou a navegação ou a interface como "Difícil", indicando que o *design* minimalista e direto proposto foi bem-sucedido.

A funcionalidade central do sistema, a tela de "Resultado da Comparação", também foi validada: 62,5% (10 de 16) dos especialistas consideraram a organização das informações (Módulos Idênticos, modificados etc.) como "Muito clara, auxiliando na visualização". Os 37,5% (6 de 16) que a definiram como "Um pouco confusa" forneceram um feedback qualitativo crucial, sugerindo a implementação de "um filtro ou busca" para casos em que a "lista de módulos modificados for gigante".

O ponto crucial da avaliação foi a eficácia da ferramenta em seu propósito principal: 100% dos especialistas afirmaram que o *"BIOS Analyzer"* "pode contribuir" para o processo de auditoria e revisão de segurança. Deste total, 75% (12 de 16)

atestaram que ele "pode contribuir totalmente" e 25% (4 de 16) indicaram que "sim, mas precisa melhorar em alguns pontos". A nota média geral atribuída à ferramenta foi 9,31 em 10. O *feedback* qualitativo reforçou os dados, com um Pesquisador de Segurança destacando que a plataforma "agiliza um processo que levaria horas para dias" e um Revisor elogiando a clareza da tela de comparação como "o ponto forte".

As sugestões de melhoria mais citadas pelos 16 usuários foram: a capacidade de "exportar o resultado" (para PDF ou CSV), a implementação de "upload em lote" de múltiplos arquivos e a "integração com bancos de dados de vulnerabilidades (como CVE)". Em suma, a avaliação prática demonstrou que o *"BIOS Analyzer"* não apenas funciona tecnicamente, mas também entrega valor real aos seus usuários-alvo.

CONSIDERAÇÕES FINAIS

A segurança de *firmware* é um campo crítico e complexo, onde a análise manual de alterações entre versões de BIOS é ineficiente e propensa a erros. Este trabalho enfrentou esse desafio por meio do desenvolvimento de uma plataforma automatizada, o *"BIOS Analyzer"*, para extrair, analisar e comparar módulos de BIOS/UEFI. O objetivo foi simplificar o processo de auditoria de segurança e criar um repositório centralizado de dados, algo especialmente relevante no cenário brasileiro, que carece de ferramentas similares.

A hipótese inicial, de que a automatização da extração de módulos e da geração de *hashes* facilitaria o trabalho de revisão de segurança, foi validada pelos resultados práticos obtidos na avaliação do sistema. A plataforma foi testada por 16 usuários, sendo a vasta maioria composta por especialistas da área (revisores, desenvolvedores e pesquisadores), que foram unânimes (100%) em afirmar que a ferramenta contribui efetivamente para o processo de auditoria. Com 75% dos especialistas atestando que a ferramenta "pode contribuir totalmente" e uma nota média de 9,31 em 10, o projeto demonstrou ser uma solução funcional, eficaz e de alta aceitação. O feedback de que a plataforma "agiliza um processo que levaria horas para dias" confirma que o projeto atingiu seu objetivo central.

Com a plataforma estabelecida como uma base robusta, a avaliação dos usuários não apenas validou o trabalho, mas também abriu diversas possibilidades para pesquisas e evoluções futuras. O repositório de dados consolidado no *MongoDB* continua sendo um ativo para trabalhos futuros, como a aplicação de *Machine Learning* para identificar padrões de vulnerabilidades em novas versões de firmware.

Adicionalmente, com base no *feedback* direto dos especialistas, sugere-se como principal evolução a integração com bancos de dados de vulnerabilidades conhecidas, como o CVE. Isso permitiria que a plataforma cruzasse as informações dos módulos com ameaças já catalogadas, enriquecendo o relatório de segurança e alertando o revisor sobre riscos conhecidos.

Outras melhorias importantes de qualidade de vida (QoL) para o fluxo de trabalho do revisor seriam a implementação da exportação dos resultados (para PDF ou CSV) e a criação de um "upload em lote" de múltiplos arquivos, ambas sugestões

frequentes dos usuários. Para lidar com firmwares de alta complexidade, também foi sugerida a adição de filtros ou uma barra de busca na tela de resultados da comparação. Por fim, uma evolução natural seria a expansão do escopo da plataforma para incluir a análise de outros tipos de firmwares, como os de roteadores e dispositivos de Internet das Coisas (IoT), que enfrentam desafios de segurança semelhantes.

REFERÊNCIAS

- (NSA), NATIONAL SECURITY AGENCY (2024). Ghidra. https://github.com/NationalSecurityAgency/ghidra. Reverse engineering tool that supports Python and Java development for customization. (Acedido em 03/09/2024).
- Almeida Filho, Gelson José de (2024). “DevSecOps em Firmware baseado na especificação UEFI: Testes Dinâmicos em Pipeline CI”. Em: (acedido em 03/09/2024).
- Chen, Siyi et al. (2023). “Delving Deep into Reverse Engineering of UEFI Firmwares via Human Interface Infrastructure”. Em: Electronics 12.22. ISSN: 2079-9292. DOI: 10.3390/electronics12224601. URL: https://www.mdpi.com/2079-9292/12/22/4601.
- Evangelista, Francesco (2023). “Automatic Extraction of Exploitation Primitives in UEFI”. Master’s Thesis. Politecnico di Torino, Corso di laurea magistrale in Ingegneria Informatica (Computer Engineering).
- Hoel, Kathrine (2022). “Hiding in the Shadows - Towards Understanding Modern UEFI Bootkits”. URN:NBN:no-98166. Master’s Thesis. University of Oslo. URL: http://hdl.handle.net/10852/95659.
- HORSICQ (2024). Detect It Easy. https://github.com/horsicq/Detect-It-Easy. Ferramenta que analisa seções de drivers/módulos extraídos com UEFI Tool. (Acedido em 03/09/2024).
- Lebedev, Philip, Konstantin Kogos e Egor Vasilenko (2020). “On Way to Simplify the Reverse Engineering of UEFI Firmwares”. Em: Advances in Cyber Security. Ed. por Mohammed Anbar, Nibras Abdullah e Selvakumar Manickam. Singapore: Springer Singapore, pp. 220–231. ISBN: 978-981-15-2693-0.
- LONGSOFT (2024). UEFITool. https://github.com/LongSoft/UEFITool. Ferramenta para visualizar os módulos e submódulos de uma BIOS/UEFI, com opção de extrair estes módulos e seus HASHs. (Acedido em 03/09/2024).
- Machado, Rafael Rodrigues e Gustavo Maciel Dias Vieira (2017). “UEFI BIOS Accessibility for the Visually Impaired”. Em: 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 155–160. DOI: 10.1109/SBESC.2017.27.

- Pankov, I. D., A. S. Konoplev e A. Yu. Chernov (2019). “Analysis of the Security of UEFI BIOS Embedded Software in Modern Intel-Based Computers”. Em: Automatic Control and Computer Sciences 53.8, pp. 865–869. ISSN: 1558-108X. DOI: 10.3103/S0146411619080224. URL: https://doi.org/10.3103/S0146411619080224.
- REFIRMLABS (2024). Binwalk. https://github.com/ReFirmLabs/binwalk. Ferramenta que analisa e extrai imagens de firmware e faz engenharia reversa. (Acedido em 03/09/2024).
- Scharrer, Daniel (2024). InnoExtract. https://github.com/dscharrer/innoextract. Ferramenta para extração de arquivos de instaladores executáveis, incluindo ISOs de BIOS/UEFI. (Acedido em 03/09/2024).
- Schlej, Nikolaj (jun. de 2023). Building the Best UEFI GUID Database Using LVFS. https://github.com/issues/5869. Issue report on GitHub discussing the creation of a comprehensive UEFI GUID database using LVFS. URL: https://github.com/LongSoft/UEFITool/issues/5869 (acedido em 01/06/2023).
- Surve, Priyanka Prakash et al. (2023). SoK: Security Below the OS – A Security Analysis of UEFI. arXiv: 2311.03809 [cs.CR]. URL: https://arxiv.org/abs/2311.03809.
- Wu, Weiming et al. (2016/08). “UEFI Firmware Storage System Analysis”. Em: Proceedings of the 2016 5th International Conference on Environment, Materials, Chemistry and Power Electronics. Atlantis Press, pp. 187–191. ISBN: 978-94-6252-197-1. DOI: 10.2991/emcpe-16.2016.40. URL: https://doi.org/10.2991/emcpe-16.2016.40.
- Zimmer, Vincent (jun. de 2007). “Platform Trust Beyond BIOS Using the Unified Extensible Firmware Interface.” Em: pp. 400–405.
- BOUTIN, Jean-Ian; ŠTEFANKO, Filip. LoJax: First UEFI rootkit found in the wild. WeLiveSecurity, 27 set. 2018. Disponível em: https://www.welivesecurity.com/2018/09/27/lojax-first-uefi-rootkit-found-wild/. Acesso em: 3 nov. 2025.
- DUFLOT, Loïc; BROSSARD, D.; LEVILLAIN, O. Attacking SMM: System Management Mode from a Hypervisor's point of view. Vancouver: CanSecWest, 2009. Apresentação (slides). Disponível em: http://cansecwest.com/csw09/csw09-duflot.pdf. Acesso em: 3 nov. 2025.
- INTEL CORPORATION. Chipsec: A Platform Security Assessment Tool. [S.l.]: GitHub, 2014-2025. Repositório de software. Disponível em: https://github.com/chipsec/chipsec. Acesso em: 3 nov. 2025.

- UEFI FORUM. Unified Extensible Firmware Interface (UEFI) Specification: Version 2.10. [S.l.]: UEFI Forum, 2022. Disponível em: https://uefi.org/specifications. Acesso em: 3 nov. 2025.
- HOLTSCLAW, Brent; MITCHELL, Nathaniel; LOUCAIDIES, John. **CHIPSEC: A Trusted Analysis Suite is Evolving for New Firmware Threats and Modern Architectures**. Intel Tech, 28 set. 2022. Disponível em: https://medium.com/intel-tech/chipsec-a-trusted-analysis-suite-is-evolving-for-new-firmware-threats-and-modern-architectures-39a10474a57a. Acesso em: 7 nov. 2025.
- PAVO. **Ozmosis UEFI bios modding guide**. InsanelyMac, 19 jun. 2015. Disponível em: https://www.insanelymac.com/forum/topic/306744-ozmosis-uefi-bios-modding-guide. Acesso em: 7 nov. 2025.

Apêndice A

Questionário de avaliação do sistema “BIOS Analyzer”

Pesquisador: Anthony Santos de Oliveira
Proposta: Desenvolver uma plataforma automatizada para análise e auditoria de *dumps* de BIOS/UEFI, focando na extração e comparação de módulos e *hashes* de versões submetidas pelos usuários.

PERGUNTAS PESSOAIS
Nome:
Idade:
Gênero:
( ) Masculino
( ) Feminino
( ) Outro

Você se encaixa melhor em qual tipo de usuário?
( ) Revisor ou Auditor de BIOS/Firmware
( ) Desenvolvedor de Software (com foco em firmware ou segurança)
( ) Pesquisador de Segurança
( ) Outro: _____

PERGUNTAS SOBRE A FERRAMENTA

1. Sobre a ferramenta “BIOS Analyzer”, você considera que o objetivo da aplicação é:
( ) Muito importante
( ) Pouco importante
( ) Não acho importante

2. Com relação a navegação pela ferramenta, você considera que o acesso às funcionalidades (Dashboard, Upload, Comparação) é:
( ) Fácil

( ) Razoável

( ) Difícil

3. Você considera que a interface da ferramenta é:

( ) Agradável, limpa e fácil de entender

( ) Razoavelmente agradável, com alguns elementos confusos

( ) Pouco agradável e difícil de entender

4. Você considera que a organização das informações, na tela de "Resultado da Comparação" (Módulos Idênticos, Modificados, etc.), está:

( ) Muito clara, auxiliando na visualização das informações

( ) Um pouco confusa, gerando certa dificuldade na visualização das informações

( ) Muito confusa, dificultando a visualização das informações

5. Você considera que essa ferramenta pode contribuir para o processo de auditoria e revisão de segurança de firmwares?

( ) Sim, pode contribuir totalmente

( ) Sim, mas precisa melhorar em alguns pontos

( ) Não pode contribuir

6. Qual nota você dá para a ferramenta “BIOS Analyzer”?

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10

7. Deixe aqui qualquer outra sugestão, elogio ou crítica: