Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

VuMoS

Um sistema de monitoramento de vulnerabilidades de segurança para os sistemas independentes da USP

Daniel de Sousa Martinez

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. João Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da Superintendência de Tecnologia da Informação da USP

São Paulo
2021

Agradecimentos

Primeiramente, agradeço ao João Eduardo Ferreira pela oportunidade de participar do projeto Hackers do Bem e por sua orientação durante o desenvolvimento deste trabalho. Aos integrantes do IMEsec pelo conhecimento e experiências adquiridas durante minha participação. Aos integrantes do Hackers do Bem pela ajuda e apoio no desenvolvimento deste trabalho. E também a minha família por todo apoio que me dão.

Resumo

Daniel de Sousa Martinez. **VuMoS: *Um sistema de monitoramento de vulnerabilidades de segurança para os sistemas independentes da USP***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Vulnerabilidades de segurança nos sistemas hospedados pela USP podem ocasionar grandes problemas, pois podem expor dados pessoais dos alunos e funcionários, permitir acesso não autorizado a sistemas internos, prejudicar a imagem da universidade, dentre outros. E pela natureza descentralizada do funcionamento da rede da USP, torna-se difícil escanear todos os tipos de vulnerabilidades de segurança de forma organizada. Por conta disso, o objetivo deste trabalho foi coletar falhas de segurança em todos os sistemas publicamente expostos da USP, de maneira automatizada, concentrando todas as informações obtidas em um único local. Isso foi feito através do desenvolvimento de um sistema modular, implementado através de uma arquitetura de microsserviços, em que cada módulo pode utilizar outras ferramentas existentes para realizar sua função.

Palavras-chave: segurança. microsserviços. monitoramento. vulnerabilidades.

Abstract

Daniel de Sousa Martinez. **VuMoS:** ***Security vulnerability monitoring system for the independent systems at USP***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

Security vulnerabilities on systems hosted by USP may cause big problems, as they can expose student and staff personal data, allow unauthorized external access to internal systems, or even harm the university's public image. Due to the decentralized nature of the USP's network, it's difficult to scan for vulnerabilities in a coherent way. As such, the objective of this work is to collect in an automated way and concentrate relevant information about security issues in all publicly available USP systems. This was done through the development of a modular system, implemented using a micro services architecture, where each module may use other tools to do their corresponding job.

Keywords: security. microservices. monitoring. vulnerabilities.

Lista de Abreviaturas

SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
HDB	Hackers do Bem
VUMOS	Sistema de Monitoramento de Vulnerabilidades (*Vulnerability Monitoring System*)
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
TLS	Segurança de Camada de Transporte (*Transport Layer Security*)
API	Interface de Programação de Aplicações (*Application Programming Interface*)
CIRD	Roteamento entre domínios sem classes (*Classless Inter-Domain Routing*)
RCE	Execução de código remoto (*Remote Code Execution*)
CVE	Vulnerabilidades e Exposições Comuns (*Common Vulnerabilities and Exposures*)
CVSS	Sistema de Pontuação de Vulnerabilidades Comuns (*Common Vulnerability Scoring System*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de Figuras

2.1 Esquema de estabelecimendo de conexões TCP e UDP (figura retirada de Anbar *et al.*, 2012) 4

3.1 Diagrama inscrição/publicação no NATS. (Figura retirada de Aslam, 2016) 8
3.2 Diagrama de troca de mensagens entre um módulo e o gerenciador vumos-db-api. 9

4.1 Diagrama de comunicação dos microsserviços implementados. 11
4.2 Diagrama entidade relacional do banco de dados implementado 15
4.3 Diagrama entidade relacional em formato UML do banco de dados implementado 16
4.4 Captura de telas do *vumos-interface* mostrando a página "Dashboard". Mais capturas de telas podem ser vistas no apêndice A 18

A.1 Captura de tela do *vumos-interface* mostrando a página "Login" 23
A.2 Captura de tela do *vumos-interface* mostrando a página "Dashboard" . . . 24
A.3 Captura de tela do *vumos-interface* mostrando a página "Targets" 24
A.4 Captura de tela do *vumos-interface* mostrando os serviços encontrados de um "Target"na página "Targets" 25
A.5 Captura de tela do *vumos-interface* mostrando os dados "extra"de uma entrada na página "Targets" 25
A.6 Captura de tela do *vumos-interface* mostrando a página "Services" 26
A.7 Captura de tela do *vumos-interface* mostrando os dados "extra"de uma entrada na página "Services" 26
A.8 Captura de tela do *vumos-interface* mostrando a página "Issues". Alguns dados foram censurados para evitar expor os institutos afetados. 27

A.9 Captura de tela do *vumos-interface* mostrando os dados "extra"de uma entrada na página "Issues". Alguns dados foram censurados para evitar expor os institutos afetados. 27

Lista de Tabelas

4.1 Tabela gerada pela ferramenta cloc, que conta as linhas de código de um projeto. 20

Sumário

1 Introdução **1**
1.1 Motivação . . . 1
1.1.1 Hackers do Bem . . . 1
1.1.2 O desafio . . . 1
1.2 Objetivo . . . 2

2 Fundamentos **3**
2.1 Enumeração de DNS . . . 3
2.1.1 Força bruta . . . 3
2.1.2 Web Crawling . . . 3
2.1.3 Search enging scraping . . . 3
2.1.4 Extração de certificados . . . 3
2.1.5 Páginas arquivadas . . . 4
2.2 Port Scanning . . . 4
2.3 Web Scraping e Web Crawling . . . 5
2.4 SQL Injection . . . 5

3 Metodologia **7**
3.1 Microsserviços . . . 7
3.1.1 NATS . . . 7
3.2 Módulos . . . 8
3.2.1 Tópicos de mensagens . . . 9

4 Ferramenta desenvolvida **11**
4.1 Módulos Implementados . . . 11
4.1.1 Vumos Common . . . 11
4.1.2 Amass Scanner . . . 12
4.1.3 Nmap Scanner . . . 12
4.1.4 Crawler . . . 13

4.1.5 SQLMap Runner 13
4.1.6 vumos-db-api 14
4.1.7 vumos-interface 18
4.2 Organização do projeto 18
4.3 Sistema em produção 20

5 Conclusão **21**
5.1 Contribuição para a segurança da universidade 21
5.2 Contribuição para a formação acadêmica 21
5.3 Próximos passos 21

Apêndices

A Capturas de tela da aplicação **23**

Referências **29**

Capítulo 1

Introdução

1.1 Motivação

1.1.1 Hackers do Bem

Para melhor entender a motivação por trás desse projeto, é interessante conhecer o grupo responsável pelo seu desenvolvimento. O grupo *Hackers do Bem* foi formado em abril de 2020, motivado, em parte, pelo vazamento de dados da Faculdade de Medicina da Universidade de São Paulo. Composto por alunos do Bacharelado em Ciência da Computação do Instituto de Matemática e Estatística da USP, e supervisionado pelo Superintendente de Tecnologia da Informação, ele conta atualmente com 4 integrantes: Daniel de Sousa Martinez, o autor deste trabalho; Cainã Setti Galante; Victor Aristóteles Rocha Campos e Victor Seiji Hariki, contando também com a supervisão do professor João Eduardo Ferreira.

A principal atividade do grupo consiste em acessar diversos sistemas hospedados pela universidade e buscar em cada um deles problemas que podem causar brechas de segurança. Mas, além disso, também é importante verificar se os sistemas deveriam estar sendo hospedados publicamente acessíveis. Logo, outra atividade do grupo é vasculhar as subredes dos institutos e descobrir que tipos de serviços estão sendo expostos indevidamente, como impressoras ou servidores de arquivos que permitem acesso sem autenticação, por exemplo.

1.1.2 O desafio

Por conta do tamanho da rede da USP [1], que conta atualmente com mais de 180 mil possíveis endereços de IP, e por ela ser composta por diversas subredes independentes (para cada instituto, por exemplo), sem um controle central, uma abordagem manual para vasculhá-la é inviável. Portanto, inicialmente, o grupo operava por meio de amostragem dos endereços de IP encontrados por ferramentas como NMAP.

Embora existam ferramentas automatizadas para busca de vulnerabilidades, nenhuma

[1] Mais detalhes em https://ipinfo.io/AS28571

delas é compreensiva e escalável o suficiente para uma rede desse tamanho. O serviço que mais se aproxima é o oferecido pelo shodan.io, porém ele não traz informações detalhadas o suficiente sobre alguns tipos de vulnerabilidade mais graves, como SQL Injection e Cross-Site Scripting.

Por conta disso, em Janeiro de 2021 iniciou-se o desenvolvimento do VuMoS, cujo nome é um acrônimo para *Vulnerability Monitoring System*, ou em português: Sistema de monitoramento de vulnerabilidades.

1.2 Objetivo

O objetivo desse projeto, portanto, é concentrar em uma única plataforma as informações obtidas através de diversas ferramentas e serviços, e através disso, construir um inventário dos sistemas rodando na USP e suas possíveis vulnerabilidades. Adicionalmente, essa plataforma deve ser escalável para centenas de milhares de entradas, possuir uma arquitetura modular e possibilitar a adição e substituição de módulos sem interromper o funcionamento dos demais, pois é possível que um escaneamento total demore diversas semanas por conta do tamanho da rede.

Capítulo 2

Fundamentos

Para melhor exemplificar o funcionamento do sistema implementado, é pertinente introduzir alguns fundamentos utilizados.

2.1 Enumeração de DNS

Enumeração de DNS é uma técnica que consiste em obter registros de DNS e seus respectivos valores (geralmente endereços de IP) sem conhecimento prévio. Existem diversas maneiras pelas quais um registro pode ser descoberto, e vale ressaltar que todos esses métodos podem ser utilizados recursivamente nos subdomínios encontrados para uma busca ainda mais completa.

2.1.1 Força bruta

São testados diversos nomes comuns de subdomínios e de serviços conhecidos em um servidor de DNS para verificar se algum deles está disponível no domínio em questão.

2.1.2 Web Crawling

Se o domínio principal estiver rodando um servidor HTTP, é possível rodar nele um *crawler* que busca por subdomínios deste mesmo domínio.

2.1.3 Search enging scraping

Este método consiste em buscar o domínio principal em um motor de busca existente (como Google ou DuckDuckGo), filtrando através de comandos desses motores de busca para páginas daquele domínio (usando "inurl:example.com", por exemplo).

2.1.4 Extração de certificados

Através deste método, são verificados os valores válidos para subdomínios encontrados nos certificados TLS fornecidos pelos servidores HTTP rodando no domínio principal.

Além disso, em alguns casos, também é possível pesquisar diretamente em algumas das próprias autoridades certificatórias, por possíveis subdomínios que são certificados por elas.

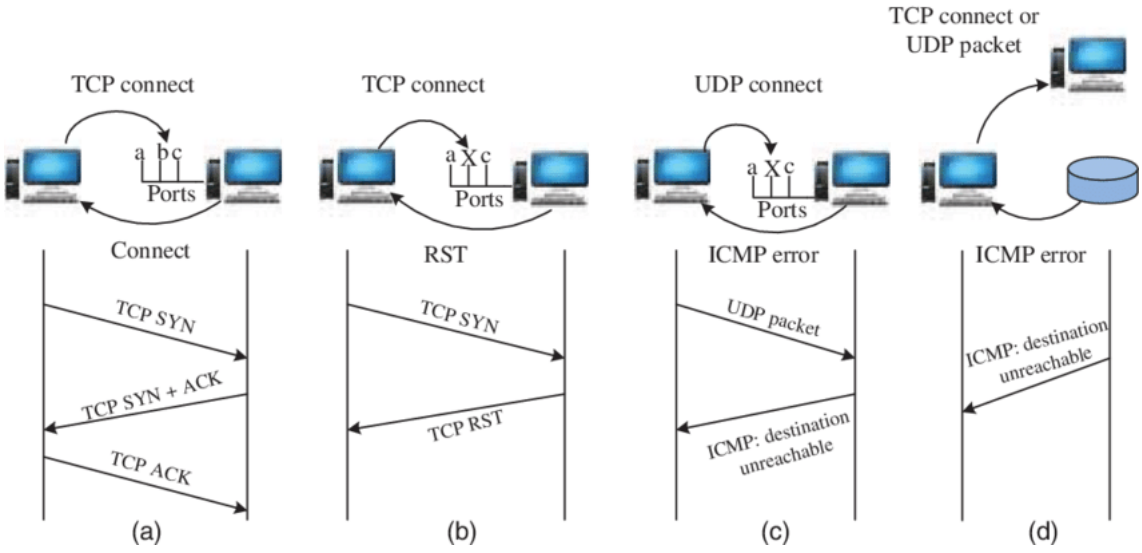
2.1.5 Páginas arquivadas

São buscados subdomínios em páginas arquivadas por serviços como ArchiveIt[1] ou WaybackMachine[2], tanto para encontrar subdomínios que não estão mais publicamente acessíveis, mas que ainda podem ser válidos, quanto para buscar subdomínios que lá estão indexados, mas que não estão listados nos motores de busca tradicionais (devido a configurações do robots.txt, por exemplo).

2.2 Port Scanning

O mapeamento de portas abertas, do inglês *port scanning* ou *port mapping*, é fundamental na descoberta de quais serviços estão rodando em uma determinada máquina.

Existe uma multitude de técnicas empregadas por diversas ferramentas para realizar mapeamentos de rede no geral (De Vivo *et al.*, 1999), mas de forma geral, este mapeamento consiste no envio de um pacote específico e na análise da resposta da máquina em questão, como exemplificado no esquema mostrado na figura 2.1.



(a) *Conexão TCP bem sucedida*

(b) *Conexão TCP mal sucedida: porta de destino fechada*

(c) *Porta UDP de destino fechada*

(d) *Endereço de IP de destino não existe*

Figura 2.1: *Esquema de estabelecimendo de conexões TCP e UDP (figura retirada de Anbar et al., 2012)*

[1] https://archiveit.com/

[2] https://web.archive.org/

2.3 Web Scraping e Web Crawling

Web Scraping é uma técnica de extração de dados de páginas *web*. Ela geralmente é utilizada para extração de dados de sites de terceiros, onde não é viável a consulta de dados diretamente. *Web Crawling* consiste em, recursiva e programaticamente, navegar através de páginas *web* utilizando um algoritmo de busca em largura, extraindo determinadas informações obtidas com *web scraping*. Dentre essas informações obtidas, está uma lista de links para outras páginas, que serão adicionadas na fila de páginas a serem escaneadas caso se enquadrem em determinados critérios, geralmente definidos pelo tipo de aplicação dos outros dados das páginas.

O principal uso conhecido dos *web crawlers* é nos motores de busca (como DuckDuckGo ou Google), onde são extraidos *links*, imagens e outras informações relevantes das páginas para consulta do público geral. No entanto, existem diversos outros casos de uso para tais ferramentas, já que é possível programá-las para extrair quaisquer informações disponíveis nas páginas.

2.4 SQL Injection

A linguagem SQL (Structured Query Language ou Linguagem de Consulta Estruturada) é o principal método de interação com sistemas gerenciadores de bancos de dados relacionais (SGBDs), e por conta disso é utilizada em uma grande porcentagem dos serviços em produção. No entanto, quando os dados de entrada dos usuário não são propriamente sanitizados, é possível que eles contenham caracteres que influenciem a comunicação da aplicação com o SGBD. Caso o usuário seja mal intencionado, ele pode ser capaz de executar seus próprios comandos SQL, o que determina se um sistema está vulnerável a *SQL Injection*.

Logo, *SQL Injection* se trata de uma vulnerabilidade de execução de código remoto (RCE), que pode comprometer todos os dados da aplicação, ou até mesmo todo o sistema que a roda.

Existem diversas maneiras de se evitar este tipo de vulnerabilidade, e geralmente não são difíceis de ser implementadas. Por exemplo, sanitizar as entradas dos usuários já torna essa vulnerabilidade muito mais difícil de acontecer (embora ainda possível, caso haja algum problema no método de sanitização). Da mesma forma, existem diversas maneiras com que uma vulnerabilidade de *SQL Injection* pode ser explorada, como pode ser visto em mais detalhes na implementação do módulo de SQLMap do VuMoS 4.1.5, uma ferramenta que as detecta e explora.

Capítulo 3

Metodologia

3.1 Microsserviços

O VuMoS foi implementado utilizando uma arquitetura modular de microsserviços, para que ele possa ser facilmente estendido e modificado no futuro, caso necessário, e também para proporcionar uma maior facilidade de desenvolvimento de cada um dos módulos.

Estes microsserviços usam um sistema de troca de mensagems como base para comunicação entre si, facilitando a comunicação assíncrona entre eles. Tal sistema também facilita o envio de mensagens de *broadcast*, para que informações de um dos módulos chegue em diversos outros módulos enviando-as apenas uma vez.

3.1.1 NATS

Foi escolhido o NATS[1] como o sistema de mensageria da aplicação, principalmente devido a sua escalabilidade, simplicidade e performance. Ele envia mensagens do tipo JSON através de um socket TCP, e portanto pode ser facilmente integrado em diversas arquiteturas e linguagens de programação. Além disso, as principais delas já contam com bibliotecas que implementam o protocolo NATS em seus repositórios (como Pipy[2] e NPM[3], por exemplo).

Este sistema segue o modelo de "inscrição e publicação", onde cada serviço interessado pode se inscrever em um tópico específico e receber somente as mensagens publicadas a tal tópico por quaisquer outros serviços.

[1] https://nats.io

[2] https://pypi.org/project/asyncio-nats-client/

[3] https://www.npmjs.com/package/nats

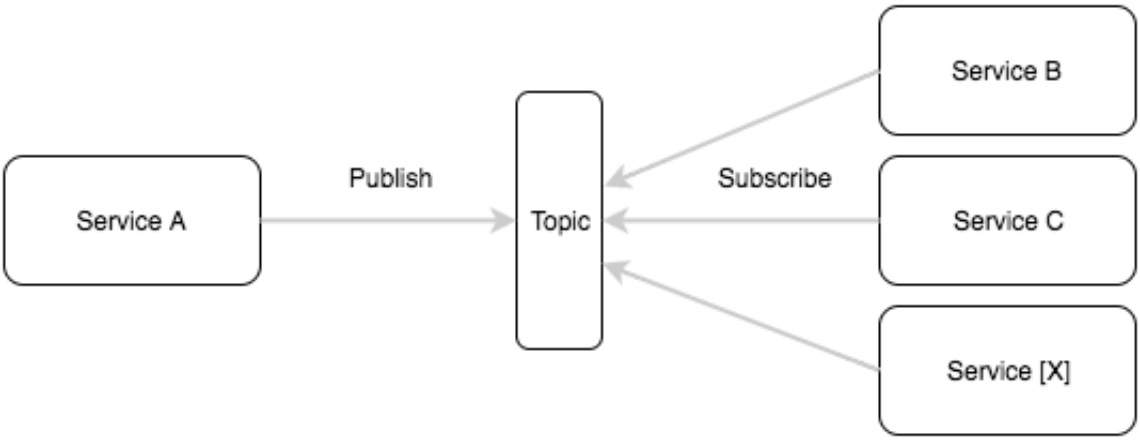


Figura 3.1: *Diagrama inscrição/publicação no NATS. (Figura retirada de Aslam, 2016)*

3.2 Módulos

O sistema foi projetado para que cada módulo possa ser implementado em qualquer linguagem de programação, e usando qualquer paradigma, desde que ele seja capaz de se comunicar com o sistema de mensageria NATS e siga o protocolo de mensagens especificado. Este protocolo foi definido no repositório *common* usando *json-schema*[4], já que uma mensagem no NATS é enviada no formato JSON.

[4] https://json-schema.org/

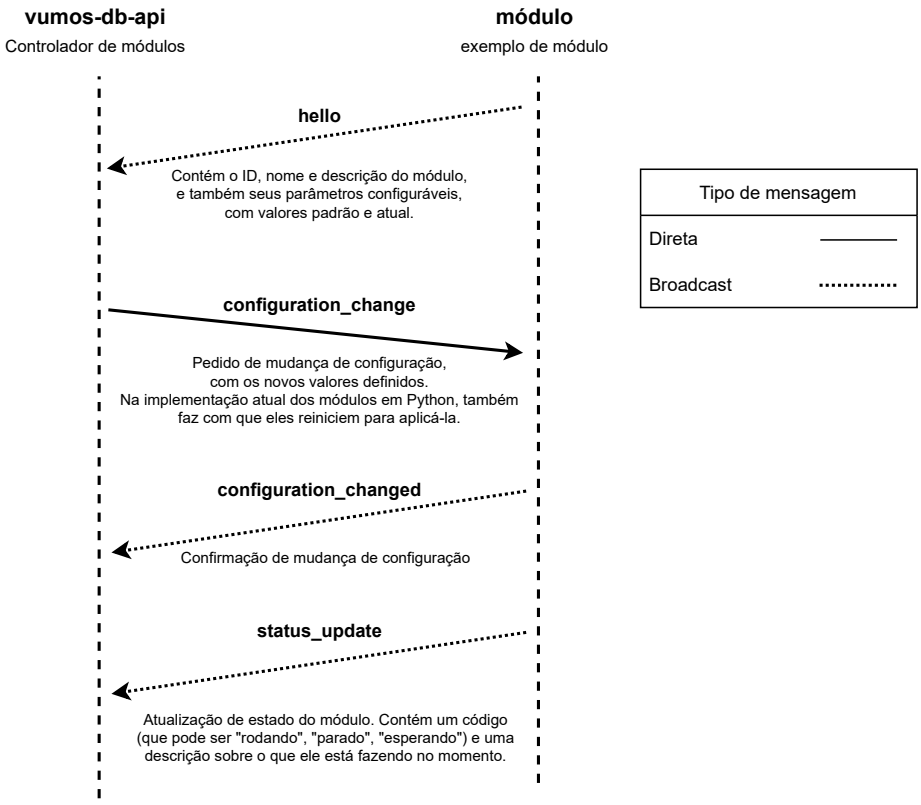


Figura 3.2: *Diagrama de troca de mensagens entre um módulo e o gerenciador vumos-db-api.*

Como exemplificado em 3.2, quando um módulo é inicializado ele envia uma mensagem do tipo *hello*, que anuncia sua presença aos outros módulos do sistema. Essa mensagem contém o *VUMOS_ID* (que corresponde a um *uuid* que representa o módulo), nome e descrição do módulo, e também quais são seus parâmetros configuráveis. Nessa mensagem também são enviados os valores padrão de tais configurações.

Além disso, módulos podem interagir entre si de maneiras distintas, quando necessário, como é o caso do *Crawler* e do SQLMap Runner, por exemplo.

3.2.1 Tópicos de mensagens

Para padronização do sistema, cada módulo sempre se inscreve em um tópico de *broadcast* e em um tópico com seu próprio *VUMOS_ID*. Isso permite que sempre seja possível enviar uma mensagem que todos os módulos vão receber, e também torna possível enviar uma mensagem diretamente a um módulo específico (como um pedido de mudança de configuração, por exemplo, que não tem valor a outros módulos senão aquele que precisa alterar sua configuração).

Capítulo 4

Ferramenta desenvolvida

4.1 Módulos Implementados

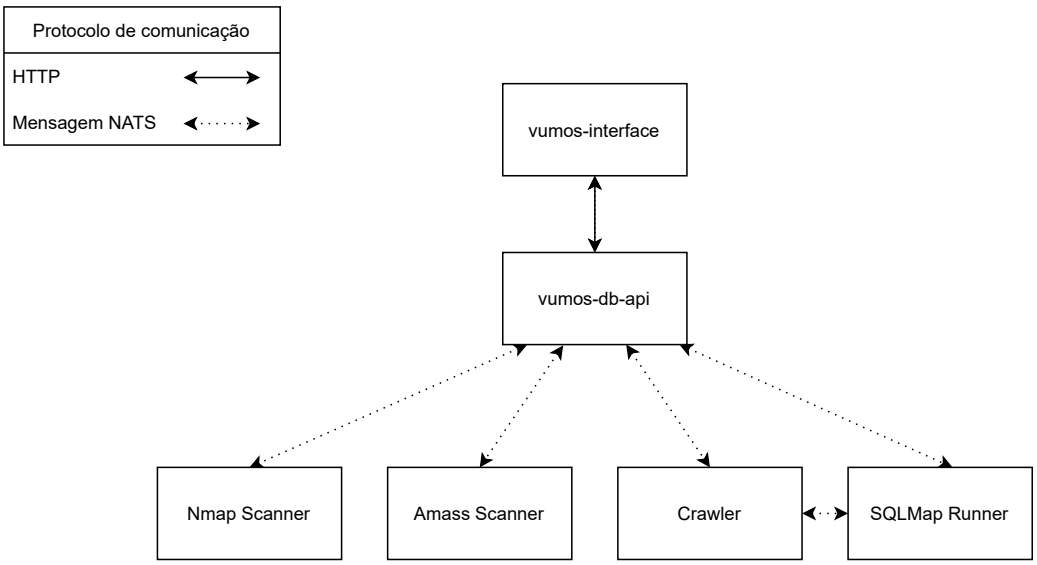


Figura 4.1: *Diagrama de comunicação dos microsserviços implementados.*

4.1.1 Vumos Common

Seguindo a arquitetura de microsserviços, cada módulo é responsável pelo armazenamento das informações relevantes a seu funcionamento, podendo ser elas obtidas através do prório módulo ou através de mensagens recebidas por outros módulos.

Para facilitar o processo de desenvolvimento de um novo módulo, foi implementada uma biblioteca em *Python* que funciona como arcabouço para os outros módulos, e também contém o *schema* das mensagens da aplicação.

Ele conta com um banco de dados local implementado em SQLite, usado principalmente para o armazenamento das configurações do módulo. Para facilitar o seu acesso do ponto

de vista do desenvolvedor, foi implementado um conjunto de funções que tornam o seu uso transparente.

Para que seja possível enviar e receber mensagens do módulo enquanto ele executa outras tarefas, como escrita no banco de dados ou requisições HTTP, foi utilizada uma arquitetura de computação assíncrona, utilizando a biblioteca *asyncio* do Python. Isso nos proporciona métodos interessantes para a execução de subprocessos, o que é necessário para uma grande parte dos módulos a serem implementados, além de proporcionar um maior desempenho no que se diz respeito a esse tipo de operação.

4.1.2 Amass Scanner

Este módulo tem como objetivo encontrar subdomínios existentes no domínio principal usp.br, usando para isso a ferramenta de código aberto AMASS [1]. Esta ferramenta utiliza diversas técnicas de enumeração de DNS para tentar encontrar subdomínios válidos, como a força bruta, *search engine scraping*, busca em certificados e páginas arquivadas.

Como variáveis configuráveis deste módulo, temos somente:

- *Initial Domain*: Domínio inicial que será escaneado recursivamente

Uma vez encontrados os subdomínios, é enviada uma mensagem do tipo *data_target* como *broadcast*, com tal informação em seus respectivos hosts/alvos.

4.1.3 Nmap Scanner

Este módulo tem como objetivo encontrar máquinas na rede e seus serviços, obtendo também informações sobre eles, como: em quais portas estão rodando, quais versões, e, possivelmente, se possuem alguma vulnerabilidade conhecida. Para isso, é utilizada a ferramenta de código aberto de mapeamento de rede NMAP [2], que realiza todas essas funções para cada um dos IPs que ela encontra.

Como variáveis configuráveis deste módulo, temos:

- *Flags*: São os argumentos de linha de comando que serão enviados ao NMAP, ativando ou desativando cada uma de suas features. Por exemplo, aqui podemos ativar ou desativar a busca automatizada por vulnerabilidades conhecidas.
- *Redo Days*: Quantidade de dias até que o escaneamento seja realizado novamente
- *IP Ranges*: Quais IPs serão escaneados. São aceitos IPs no formato *CIDR Standard* separados por vírgula, ou seja, é possível definí-lo para escanear todas as subredes da USP de uma vez.

Por fim, para cada serviço encontrado é enviada uma mensagem do tipo *data_service* como *broadcast* e também é enviada uma mensagem do tipo *data_target* com o seu respectivo alvo.

---

[1] https://github.com/OWASP/Amass

[2] https://github.com/nmap/nmap

4.1.4 Crawler

Este módulo tem o objetivo de, a partir de uma página HTML inicial, encontrar novas páginas que possuem links dinâmicos. Isso é feito usando um *crawler* customizado que utiliza o módulo de Python Scrapy [3]. Como não é possível saber se um link é processado dinamicamente ou não pelo servidor sem uma análise extensa em cada um deles, é utilizada uma heurística, que nos permite conseguir uma grande quantidade de links dinâmicos sem processamento extra. Portanto, uma URL é considerada dinâmica se contiver uma interrogação (indicando que contém parâmetros, sendo eles analisados para uso futuro), ou se for o resultado de um envio de formulário, onde os parâmetros são seus campos.

Como variáveis configuráveis deste módulo, temos:

- *Allowed domains*: Domínios em que o *crawler* tem permissão para rodar. Isso impede que domínios fora do escopo sejam escaneados quando um link externo (como do Youtube, por exemplo) for encontrado.
- *Initial URLs*: Lista separada por vírgulas com as URLs por onde o *crawler* começará a procurar links.

Para cada URL dinâmica encontrada, é enviada uma mensagem do tipo *data_path* como *broadcast*, e também é enviada uma mensagem do tipo *data_target* com seu alvo e uma *data_service* com seu serviço, sendo tais informações extraidas tanto da URL em si quanto da página que a contém.

4.1.5 SQLMap Runner

Este módulo tem como objetivo procurar vulnerabilidades de SQL Injetion em uma determinada URL dinâmica através da ferramenta de código aberto SQLMap [4]. Para isso, este módulo escuta pelas mensagens de *broadcast* do tipo *data_path*, que são enviadas pelo módulo do *Crawler* e, para cada uma delas, coloca numa fila persistente em disco. Isso é feito pois o SQLMap, devido à sua complexidade, demora diversas vezes mais para ser executado em relação ao *Crawler*, portanto as URLs são acumuladas nesta fila e processadas assim que possível.

Para tornar o processo mais rápido, este módulo implementa também um sistema de *multithreading*, no qual diversas instâncias do SQLMap são executadas ao mesmo tempo em URLs diferentes para paralelizar o processo. Para tal, foi utilizado o gerenciamento de *threads* de execução do *asyncio*, onde cada instância do SQLMap é iniciado em uma co-rotina assíncrona com um executor próprio.

Para buscar tais vulnerabilidades, o SQLMap executa diversas requisições HTTP com alterações em seus parâmetros, e verifica por diferenças nos resultados, de acordo com as seguintes técnicas:

- *Boolean-based blind*: Em português "ataque cego baseado em booleanos", esta técnica adiciona parâmetros de SQL na requisição HTTP que retornem ou verdadeiro ou

[3] https://scrapy.org/

[4] https://github.com/sqlmapproject/sqlmap

falso, e verifica a página de resposta por diferenças entre os dois valores. Dessa forma, esta técnica é capaz de extrair dados, caractere a caractere, utilizando o método da bisseção para diminuir a quantidade de requisições necessárias para as comparações.

- *Time-based blind*: Em português "ataque cego baseado em tempo", esta técnica funciona de forma similar à anterior, porém é capaz de extrair dados até mesmo quando a aplicação não retorna valores diferentes para verdadeiro e falso em sua página de resposta. Isso é feito através da inserção de um comando *SLEEP*, que faz a aplicação demorar mais para responder em caso positivo. Em seguida, é medido o tempo de resposta de cada requisição, e com isso é possível inferir se o resutado foi positivo ou negativo, e é aplicado o método da biseção novamente.
- *Error-based*: Em português "ataque baseado em erros", esta técnica só funciona quando a aplicação retorna o erro do sistema gerenciador de banco de dados (SGBD) para o usuário. Ela envia uma requisição que contém um comando que provoque um erro no SGBD após a execução da query de extração de dados, dessa forma os resultados são exibidos na página de erro e podem ser facilmente extraídos.
- *UNION query-based*: Em português "ataque baseado em requisições UNION", esta técnica envia requisições SQL que contém um UNION, de forma que quando os resultados da aplicação forem exibidos, os valores que se deseja extrair também estarão presentes.
- *Stacked Queries*: Em português "requisições empilhadas", esta técnica é mais focada na inserção de dados e execução de comandos arbitrários no SGBD. Ela verifica se a aplicação suporta múltiplas requisições SQL por requisição HTTP com o uso de um ";", e em caso positivo, é possível executar qualquer tipo de comando SQL sem restrição da aplicação HTTP.

Como variáveis configuráveis deste módulo, temos:

- *Number of instances*: Quantidade de instâncias do SQLMap que serão executadas ao mesmo tempo.
- *SQLMap Techniques*: Cadeia de caracteres que representa quais técnicas de injeção serão testadas em cada um dos sites.
- *SQLMap Level*: Número inteiro de 1 a 5 que representa o nível de execução do SQLMap. Tal valor representa o quão minucioso o SQLMap será em sua busca por vulnerabilidades, sendo um nível maior mais exaustivo, mas também mais demorado.

Uma vez encontrada uma página vulnerável, é enviada uma mensagem do tipo *data_issue* como *broadcast*, anunciando-a com todas as informações pertinentes.

4.1.6 vumos-db-api

Este é o módulo central da aplicação, que é responsável por controlar os outros módulos e receber seus dados para armazenamento e exibição ao usuário. Ao contrário dos módulos acima, ele foi feito usando NodeJS, Express e Typescript , e utiliza um banco de dados PostgreSQL, que permite o uso de algumas de suas funcionalidades mais avançadas

pela biblioteca de ORM (Object-Relational Mapping, ou Mapeamento Objeto-Relacional) TypeORM [5].

Banco de dados

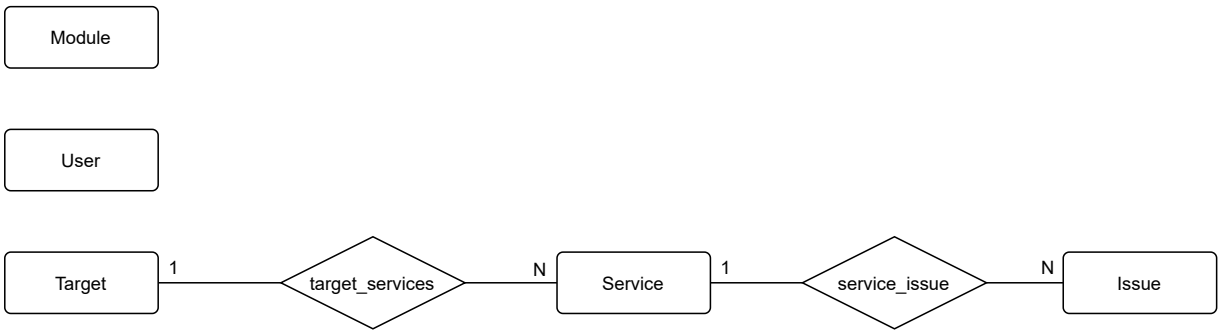


Figura 4.2: *Diagrama entidade relacional do banco de dados implementado*

[5] https://typeorm.io/

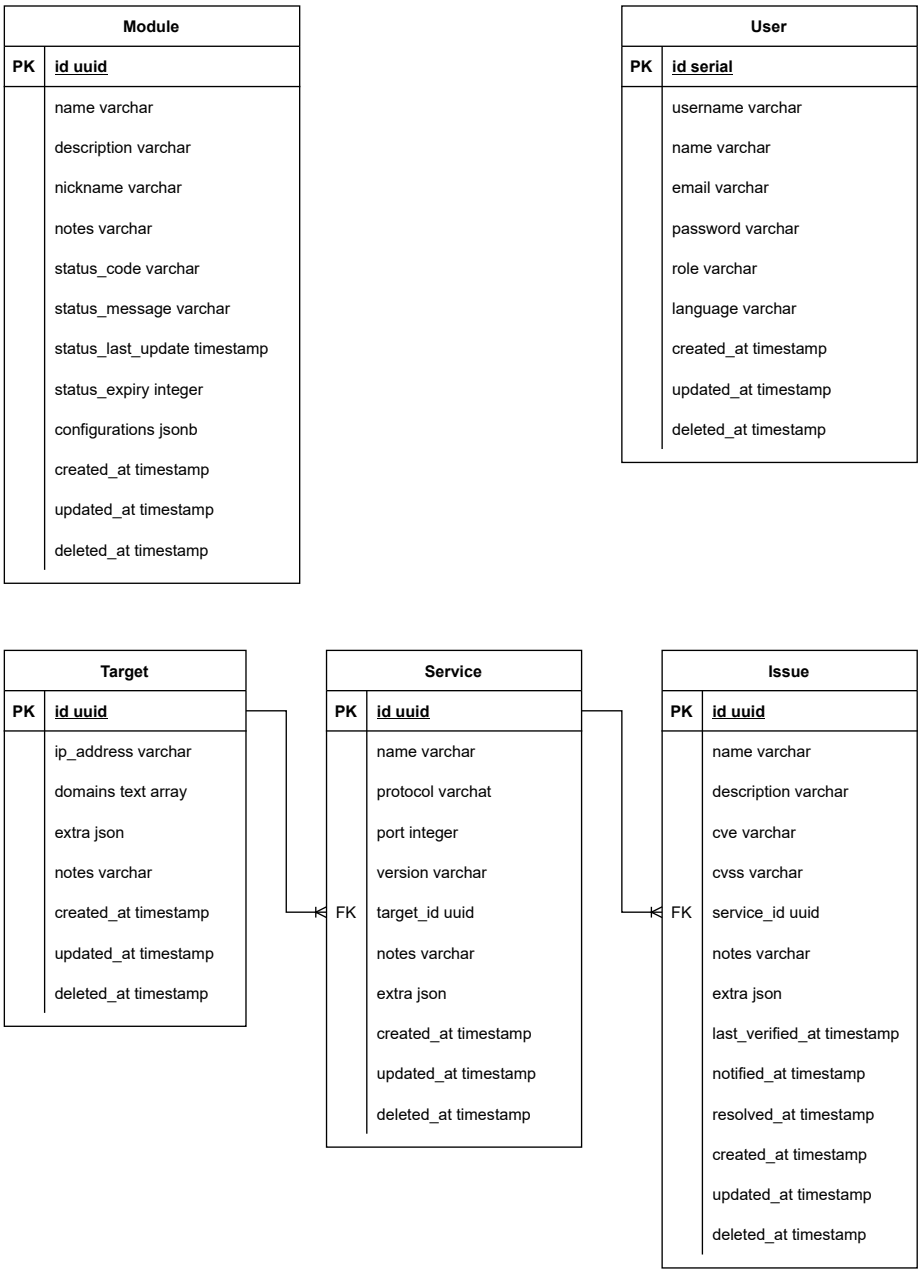


Figura 4.3: *Diagrama entidade relacional em formato UML do banco de dados implementado*

As seguintes tabelas foram implementadas utilizando o TypeORM:

- *User*: Esta tabela contém os dados para identificação e autenticação dos usuários. Ela contém também o campo *role*, que identifica o cargo do usuário para uso futuro, para implementação de um usuário que possui somente acesso a operações de leitura, por exemplo.
- *Module*: Esta tabela contém as informações sobre os módulos conhecidos pelo vumos-db-api, contendo suas configurações e status atual de seu funcionamento.
- *Target*: Esta tabela contém os dados de cada "alvo"da aplicação, que é composto principalmente por um endereço de IP e uma lista de domínios.

- *Service*: Esta tabela contém os dados de cada serviço encontrado pelo sistema, sendo eles identificados principalmente por um nome e uma porta, e relacionados com um Target.
- *Issue*: Esta tabela contém os dados de cada problema, ou vulnerabilidade, encontrado pela aplicação, sendo ele composto principalmente por um nome e uma descrição, podendo contar, opcionalmente, com um código de CVE (Common Vulnerabilities and Exposures, Mell e Grance, 2002) ou com a pontuação CVSS (Common Vulnerability Scoring System, Scarfone e Mell, 2009).

Todas as entradas em Target, Service e Issue podem contar também com carimbos de data e hora para os horários de criação, atualização e remoção (sendo os dados não removidos efetivamente do banco, e somente marcados como removidos), e também os seguintes campos:

- *extra*: contém informações adicionais disponibilizadas pelo módulo que a adicionou, como a fonte de tal informação, por exemplo.
- *notes*: campo opcional no qual o usuário pode adicionar notas sobre aquela entrada, especificando o institudo a qual um IP pertence, por exemplo.

Autenticação

Para autenticação, um usuário pode fazer *login* através de seu nome de usuário e senha (que é guardada no banco de dados como um *hash* com 8 dígitos de sal, gerada com o método *bcrypt*), e receberá como resultado um *Bearer token*, que deverá ser enviado em todas as requisições futuras que requeiram autenticação, seguindo o modelo *OAuth2*, como explicado em Jones e Hardt, 2012.

Para ativar o registro de usuários, uma variável de ambiente pode ser definida, podendo ser desativada num momento futuro.

Rotas

Para cada uma das tabelas do banco de dados, foi implementado um conjunto de rotas na API HTTP do tipo GET, POST, PATCH, DELETE, que buscam, inserem, atualizam ou deletam entradas nelas, respectivamente.

Mensagens

Além de ser possível realizar operações nos dados do banco de dados através de requisições HTTP, este módulo também escuta por mensagens dos tipos *data_target*, *data_service* e *data_issue*, que armazenam também os dados enviados através do sistema de mensageria, atualizando-os quando necessário.

Além disso, esse módulo escuta por mensagens do tipo *hello*, *configuration_changed* e *status_update* as utilizam para atualizar as informações referentes a cada um dos módulos presentes no sistema no banco de dados.

Vale ressaltar que as mensagens do tipo *data_path* são ignoradas por esse módulo, já que armazenar cada URL encontrada pelo *crawler* não é muito relevante para o funcionamento

da aplicação.

4.1.7 vumos-interface

Para gerenciar todas as informações recebidas pelo módulo *vumos-db-api* de forma intuitiva ao usuário, foi implementada também uma interface gráfica, usando para isso a biblioteca de *front-end* Vue.js [6].

Este é o único módulo da aplicação que não se comunica através do sistema de mensageria, utilizando apenas as APIs HTTP do *vumos-db-api*. Além disso, o Vue.js gera arquivos HTML, CSS e Javascript estáticos e independentes, que podem ser servidos a partir de qualquer servidor de arquivos, como Apache ou Nginx, por exemplo.

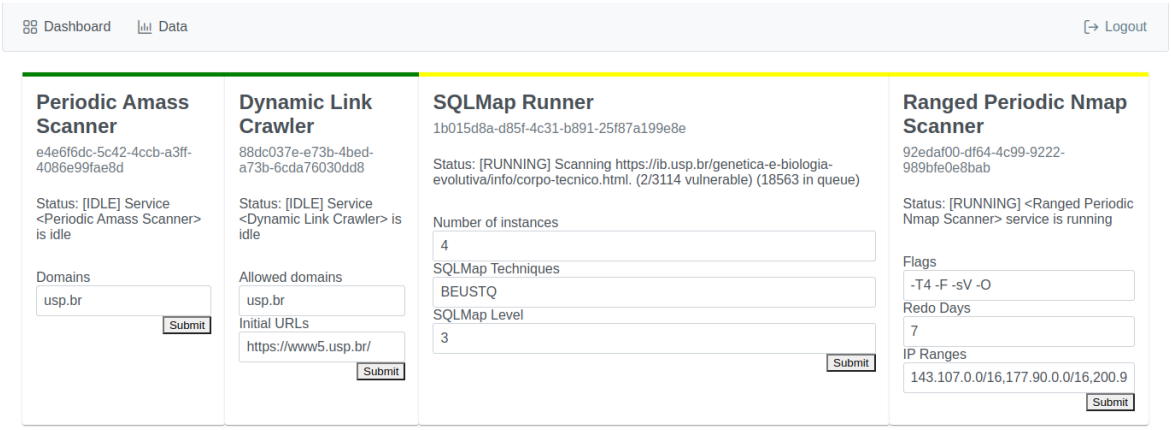


Figura 4.4: *Captura de telas do vumos-interface mostrando a página "Dashboard". Mais capturas de telas podem ser vistas no apêndice A*

As páginas contém informações atualizadas dinamicamente, que podem ser obtidas através de mensagens em um *websocket*. Caso este falhe ou se desconecte por algum motivo, dentro de 15 segundos é efetuada uma nova requisição HTTP para a o vumos-db-api a fim de atualizar os dados e tentar reconectar o *websocket*.

4.2 Organização do projeto

Cada um dos módulos do projeto está contido em seu próprio repositório, hospedado no projeto do *hackersdobem-usp* no Gitlab [7], sendo o *vumos-common* adicionado como submódulo em todos os outros módulos feitos em Python. Além disso, todos os módulos contém um *Dockerfile*, que gera um container independente para cada um. Isso torna mais

[6] https://br.vuejs.org/

[7] https://gitlab.com/hackersdobem-usp

fácil a orquestração e a implantação de cada um deles em produção, através da ferramenta *docker-compose* [8].

[8] https://docs.docker.com/compose/

Language	files	blank	comment	code
TypeScript	103	583	128	2760
JSON	16	0	0	1172
Vuejs Component	16	152	92	1131
Python	18	297	261	982
Bourne Shell	2	14	4	171
YAML	6	16	17	127
Dockerfile	6	62	45	95
JavaScript	5	6	2	84
Markdown	3	22	0	66
GraphQL	2	0	0	60
HTML	1	0	1	16
SVG	1	0	0	3
SUM:	179	1152	550	6667

Tabela 4.1: *Tabela gerada pela ferramenta cloc*[9]*, que conta as linhas de código de um projeto.*

Como pode ser visto na tabela 4.1, o projeto conta com aproximadamente 6000 linhas de código, sendo a maioria delas provenientes do *vumos-db-api*, que é o único módulo feito em TypeScript.

4.3 Sistema em produção

O VuMoS foi colocado em produção em http://vumos.hackersdobem.sti.usp.br/ e já está atualmente escaneando todos os subdomínios de usp.br e também todas as subredes da universidade.

Até o momento, já foram encontrados na rede pública da usp: 28398 *Targets*, cada um com seu respectivo endereço de IP e 89195 serviços rodando neles. O *crawler* já encontrou 21677 URLs dinâmicas, das quais 3128 já foram escaneadas pelo SQLMap. E destas 3128, foram encontradas duas vulneráveis a SQL Injection (os responsáveis já foram avisados).

[9] https://github.com/AlDanial/cloc

Capítulo 5

Conclusão

5.1 Contribuição para a segurança da universidade

Pode-se dizer que o objetivo inicial do projeto foi atingido, pois agora existe uma plataforma capaz de unificar todos os dados de vulnerabilidades encontradas na universidade.

Além disso, os módulos em execução já encontraram vulnerabilidades de *SQL Injection* que precisam ser corrigidas, e, portanto, já beneficiaram a comunidade como um todo.

5.2 Contribuição para a formação acadêmica

Durante o desenvolvimento deste projeto, aprendi muito sobre a arquitetura de microsserviços e como ela pode tanto facilitar quanto dificultar o desenvolvimento de um projeto. Também pude me especializar na área de segurança da informação, aprendendo a fundo detalhes sobre diversos sistemas, quais podem ser suas possíveis vulnerabilidades, e também como explorá-las. Isso me traz um conhecimento de como desenvolver aplicações mais seguras, evitando erros que são comumente cometidos.

5.3 Próximos passos

Para os próximos passos do projeto, poderão ser implementados novos módulos, como:

- *Cameradar*: Um módulo que escaneia por câmeras com pouca ou nenhuma autenticação.
- *Metasploit*: Diversos módulos do metasploit podem ser implementados também, para buscar vulnerabilidades em outros tipos de sistema, como RDP, OpenSSL ou SSH, por exemplo.
- *Nmap NSE*: Diversos módulos que utilizam *scripts* do *Nmap Scripting Engine* também podem ser implementados, para que os serviços encontrados sejam também

associados a uma CVE, por exemplo.

- *Shodan.io*: Um módulo que extrai as informações existentes em shodan.io também pode ser facilmente implementado, e disponibilizaria uma grande quantidade de informações pré escaneadas. Porém, ele necessita de uma licença para fazer uso completo de sua API.

Além disso, podem ser feitas melhorias na interface do usuário, permitindo que *Services*, *Targets* e *Issues* sejam adicionados manualmente para incluir vulnerabilidas encontradas fora da plataforma, por exemplo.

Isso pode ser feito por novas gerações de integrantes do projeto *Hackers do Bem*, pois grande parte do trabalho, que antes era feito manualmente pelo grupo, pode ser automatizado de forma relativamente simples utilizando o sistema VuMoS.

Apêndice A

Capturas de tela da aplicação

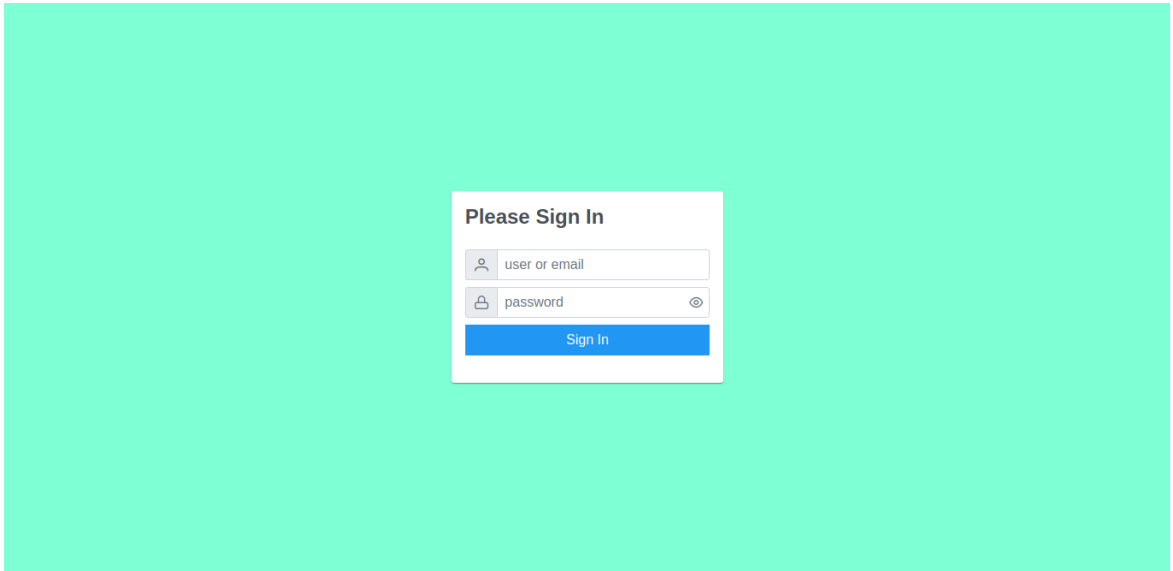


Figura A.1: *Captura de tela do vumos-interface mostrando a página "Login"*

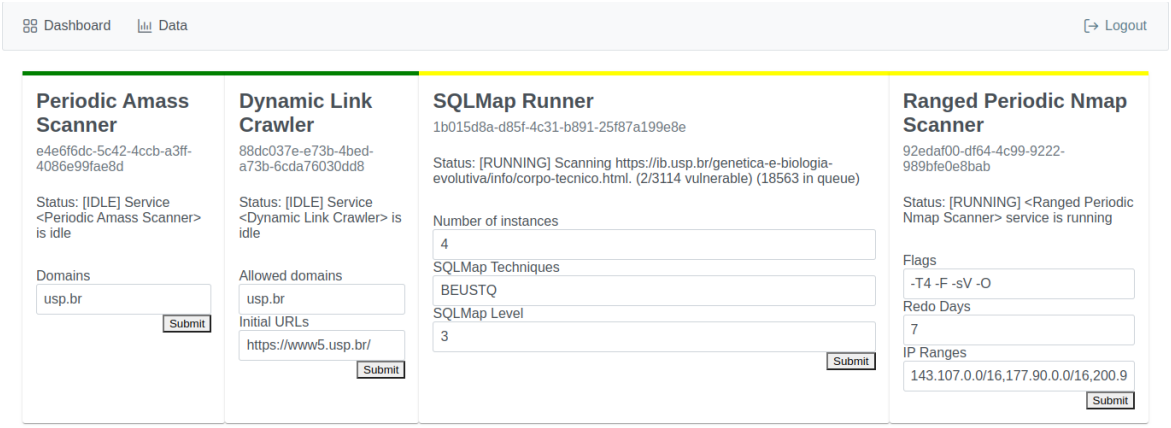


Figura A.2: *Captura de tela do vumos-interface mostrando a página "Dashboard"*

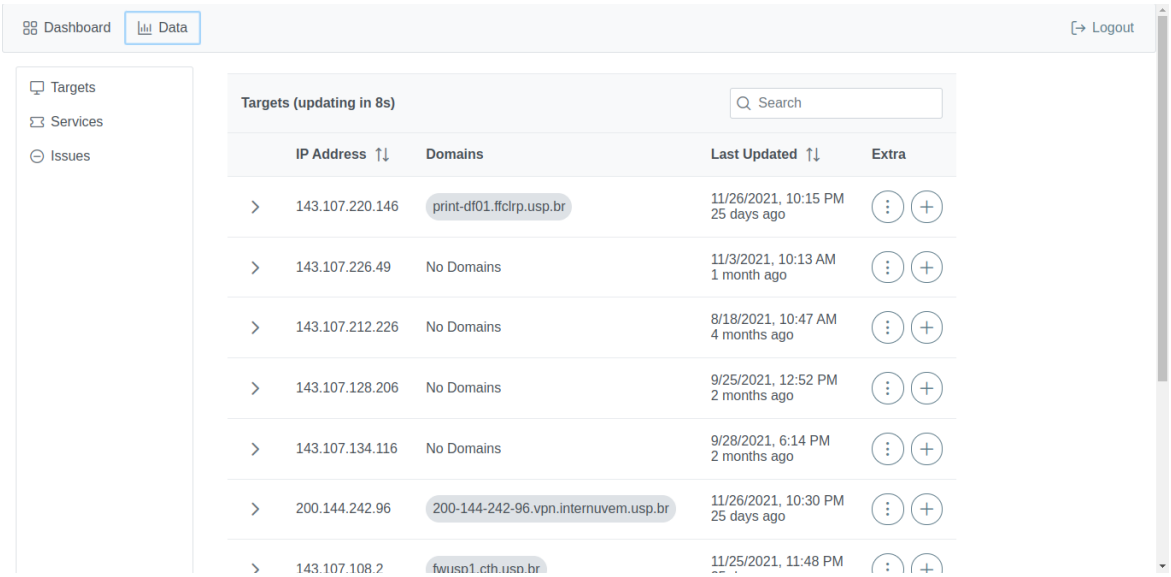


Figura A.3: *Captura de tela do vumos-interface mostrando a página "Targets"*

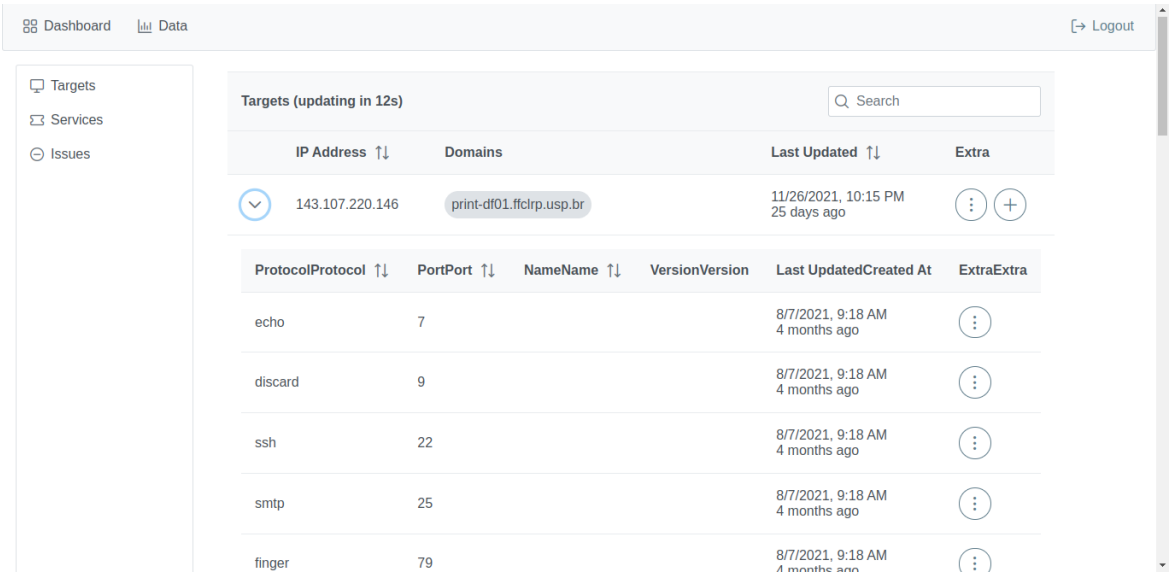


Figura A.4: *Captura de tela do vumos-interface mostrando os serviços encontrados de um "Target"na página "Targets"*

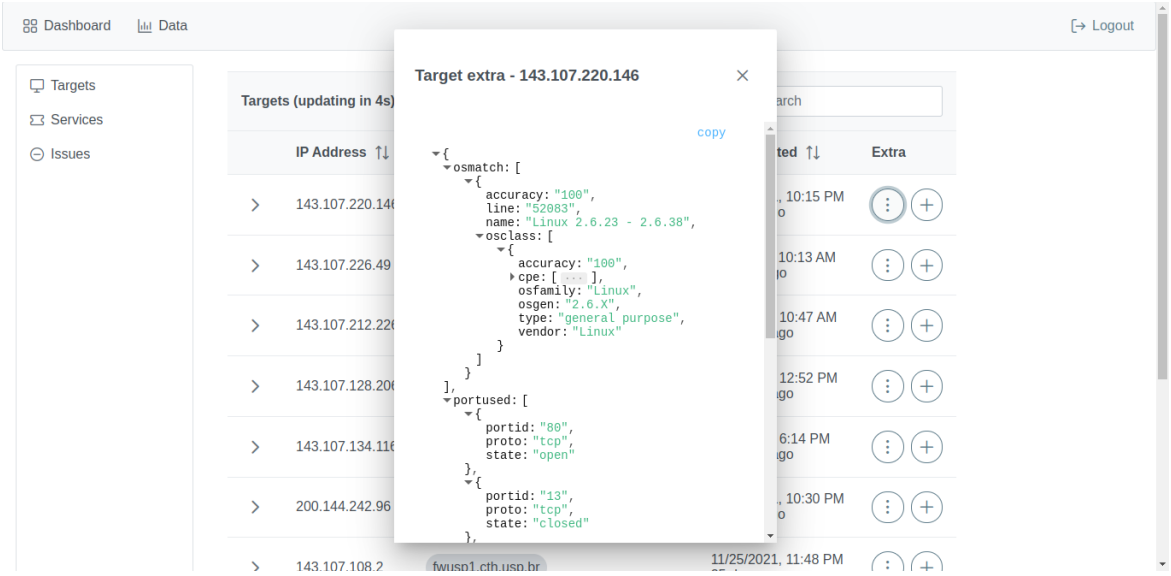


Figura A.5: *Captura de tela do vumos-interface mostrando os dados "extra"de uma entrada na página "Targets"*

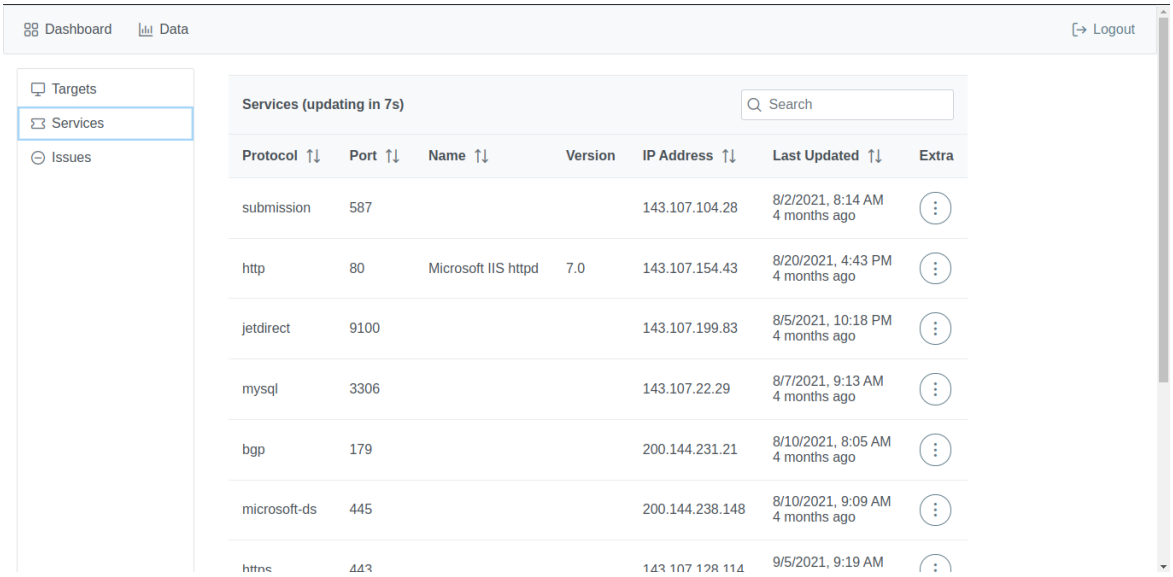


Figura A.6: *Captura de tela do vumos-interface mostrando a página "Services"*

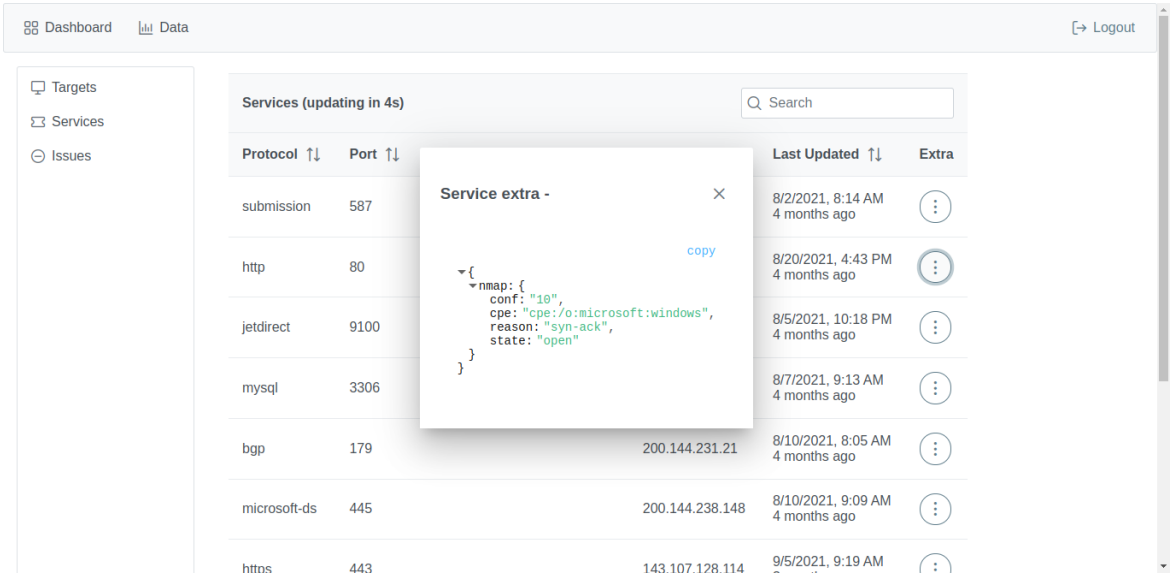


Figura A.7: *Captura de tela do vumos-interface mostrando os dados "extra"de uma entrada na página "Services"*

Figura A.8: *Captura de tela do vumos-interface mostrando a página "Issues". Alguns dados foram censurados para evitar expor os institutos afetados.*

Figura A.9: *Captura de tela do vumos-interface mostrando os dados "extra"de uma entrada na página "Issues". Alguns dados foram censurados para evitar expor os institutos afetados.*

Referências

[Anbar *et al.* 2012] Mohammed Anbar, Ahmed Manasrah e Selvakumar Manickam. "Statistical cross-relation approach for detecting tcp and udp random and sequential network scanning (scans)". Em: *International Journal of Computer Mathematics* 89 (out. de 2012), pgs. 1952–1969. doi: 10.1080/00207160.2012.696621 (citado na pg. 4).

[Aslam 2016] Asim Aslam. *Micro on NATS - Microservices with Messaging*. 2016. url: https://nats.io/blog/microonnats/ (acesso em 11/12/2021) (citado na pg. 8).

[De Vivo *et al.* 1999] Marco De Vivo, Eddy Carrasco, Germinal Isern e Gabriela O De Vivo. "A review of port scanning techniques". Em: *ACM SIGCOMM Computer Communication Review* 29.2 (1999), pgs. 41–48 (citado na pg. 4).

[Jones e Hardt 2012] Michael Jones e Dick Hardt. *The oauth 2.0 authorization framework: Bearer token usage*. Rel. técn. RFC 6750, October, 2012 (citado na pg. 17).

[Mell e Grance 2002] Peter Mell e Tim Grance. *Use of the common vulnerabilities and exposures (cve) vulnerability naming scheme*. Rel. técn. NATIONAL INST OF STANDARDS e TECHNOLOGY GAITHERSBURG MD COMPUTER SECURITY DIV, 2002 (citado na pg. 17).

[Scarfone e Mell 2009] Karen Scarfone e Peter Mell. "An analysis of cvss version 2 vulnerability scoring". Em: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2009, pgs. 516–525 (citado na pg. 17).