Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Sistemas de Informação
Trabalho de Conclusão de Curso – Artigo

Uma Ferramenta de Análise Estática de Código Fonte para Aplicações Web PHP

Jonas Antônio Gomes Vicente[1]
Centro Universitário Academia, Juiz de Fora, MG

Tassio Ferenzini Martins Sirqueira[2]
Centro Universitário Academia, Juiz de Fora, MG

Linha de pesquisa: Engenharia de Software

Resumo

As linguagens de programação, bem como o software, evoluem e, consequentemente, tais evoluções podem ter impacto nos sistemas desenvolvidos. Por isso, neste trabalho foi desenvolvida uma ferramenta para auxiliar no processo de manutenção e evolução do software, identificando funções marcadas como vulneráveis e funções depreciadas com a evolução da linguagem de programação, que impacta em aplicações web desenvolvidas com a linguagem PHP, através da análise estática do código fonte. Chamada de "PHP Analyzer", a ferramenta web procura termos específicos, verificando o código fonte de duas formas, a primeira através do upload de ficheiros através da máquina do utilizador e a segunda através do link no repositório GIT, onde se encontra o código fonte. Após a análise, a ferramenta PHP Analyzer fornece um relatório com todas as verificações, indicando os pontos críticos e fornecendo um catálogo para ajudar na correção, além de explicar os pontos mostrados.

Palavras-chave: Manutenção software. Funções depreciadas. Evolução de Software.

ABSTRACT

Programming languages, as well as software, evolve, and consequently, such evolutions can impact developed systems. Because of that, in this work a tool was developed to assist in the process of maintenance and evolution of software,

[1] Discente do Curso de Sistemas de Informação do Centro Universitário Academia - UniAcademia. E-mail: jayv.jonas@gmail.com
[2] Docente do Curso de Sistemas de Informação do Centro Universitário Academia - UniAcademia. Orientador.

identifying functions marked as vulnerable and functions depreciated with the evolution of the programming language, which impacts on web applications developed with the PHP language, through the static analysis of the source code. Called "PHP Analyzer", the web tool searches for specific terms, checking the source code in two ways, the first through the upload of files via the user's machine and the second through the link in the GIT repository, where the source code is found. After the analysis, the PHP Analyzer tool provides a report with all the checks, indicating the critical points and providing a catalog to help in the correction, besides explaining the points indicated.

1 INTRODUÇÃO

Segundo Mendes (2020), desde a sua liberação para fins comerciais, em 1995 no Brasil, a internet vem se popularizando cada dia mais, onde hoje pode ser acessada através de diversos dispositivos a qualquer hora e em qualquer lugar que se tenha conectividade. Com isso, diversos serviços, *e-commerce*, consumo de mídias via *streaming,* jogos on-line, transações bancárias, fóruns de discussão dentre outras podem ser utilizadas. Para dar suporte à criação desses serviços, surgiram diversas linguagens de programação, tais como o PHP (à princípio, *Personal Home Page Tools* e que posteriormente mudou de nome para *PHP: Hypertext Preprocessor*), o *Java*, o *Javascript*, *Python*, dentre outras.

Conforme o site oficial do PHP[3], a linguagem surgiu em 1994, criada por 'Rasmus Lernof' e escrita na linguagem C, tinha como finalidade registrar a quantidade de visitas a uma de suas páginas web. Com o tempo, evoluções na linguagem proporcionaram novas funcionalidades, tais como, a interação com bancos de dados. Em 1995, o código fonte do PHP foi liberado, possibilitando a melhoria da linguagem pela comunidade, que ao longo dos anos aperfeiçoaram suas funcionalidades, tornando-se uma das mais populares para o desenvolvimento de aplicações web. Atualmente o PHP está na versão 7.4.

Segundo a pesquisa do *Stack Overflow* de 2019[4], o PHP aparece em oitavo lugar dentre as linguagens mais populares utilizadas no mundo. De acordo com site W3Techs[5], a linguagem é utilizada em 79% das aplicações web que

[3] PHP. Disponível em: <https://www.php.net>. Acesso em: 18 de maio de 2020.

[4] Pesquisa Stack Overflow. Disponível em: <https://insights.stackoverflow.com/survey/2019#technology> Acesso em: 06 de abril de 2020.

[5] W3Techs. Disponível em: <https://w3techs.com/technologies/overview/programming_language>. Acesso em: 21 de Junho de 2020.

conhecemos. Sua popularidade se deve a curva de aprendizagem pequena e fácil manutenção para entendimento do código fonte, além das diversas bibliotecas e *frameworks* disponíveis para o PHP, que facilitam o desenvolvimento de aplicações, tais como o *Wordpress*[6], *Laravel*[7], *CodeIgniter*[8] e *Zend*[9]. Grandes portais como o Facebook e o Wikipédia utilizam o PHP como linguagem *back-end,* conforme a W3Techs.

Softwares, eventualmente, precisaram de manutenção ou de mudanças para se manterem ativos e atendendo as necessidades dos usuários. Os processos de manutenção e evolução de software são extremamente importantes durante a vida útil de um sistema. De acordo com Bhatt *et al.* (2004), a construção de um software consome apenas 25% a 33% dos recursos para desenvolvimentos do software. Grande parte dos recursos são destinados à manutenção e evolução do software.

O objetivo deste trabalho é apresentar uma ferramenta capaz de realizar análise estática de código fonte, em aplicações web desenvolvidas com o uso da linguagem PHP, para auxiliar no processo de manutenção e evolução de software, identificando pontos de vulnerabilidades conhecidos e marcados pelo próprio site PHP e funções depreciadas pela evolução da linguagem.

Este artigo está organizado da seguinte forma. A seção 2 apresenta os trabalhos relacionados a esta proposta. A seção 3 detalha a ferramenta desenvolvida. Uma diretriz de utilização é apresentada na seção 4. A seção 5 aborda as considerações sobre o trabalho, as ameaças à validade e os trabalhos futuros.

2 ARCABOUÇO

Segundo Bellairs (2020), a análise estática de código fonte é um método de depuração que ocorre antes que o código seja executado. Neste trabalho desenvolvemos uma ferramenta para análise de códigos fonte voltada para aplicações web desenvolvidas na linguagem PHP, que seja independente das IDEs (*Integrated Development Environment* - Ambiente Integrado de Desenvolvimento) e que possa ser acessado via internet.

No mercado, existem diversas dessas ferramentas para análise estática de código fonte, tais como *CPPcheck (*Marjamäki, 2020*)*, *RIPS (RIPS Technologies)*, *Jtest (Parasoft)*, *Pylint (Pylint)* e *Code Smell Analyzer (*Sirqueira *et al.,* 2016*)*.

---

[6] *Wordpress.* Disponível em: <https://br.wordpress.com/>. Acesso em: 02 de junho de 2020.
[7] Laravel. Disponível em: <https://laravel.com/>. Acesso em: 08 de abril de 2020.
[8] *CodeIgniter.* Disponível em: <https://codeigniter.com/>. Acesso em: 02 de junho de 2020.
[9] Zend. Disponível em: <https://framework.zend.com/>. Acesso em: 02 de junho de 2020.

Segundo Marjamäki (2020), a ferramenta *CPPcheck* realiza a análise estática de código fonte escritos nas linguagens *C* e *C++* e possui foco na detecção de comportamentos indefinidos e perigosos no código fonte. Trata-se de um projeto *open-source* que é capaz de analisar código fonte mesmo que não estejam utilizando uma sintaxe padrão. Os desenvolvedores abordam que uma das metas do *CPPcheck* é gerar poucas ocorrências de falsos positivos.

A ferramenta *RIPS*, conforme a RIPS Technologies GmbH, automatiza os testes de segurança e gerencia os riscos de uma aplicação e possibilita o CI/CD (*continuous integration and continuous delivery* - integração contínua e entrega contínua). O RIPS é capaz de analisar códigos fonte escritos nas linguagens PHP, *Java* e *Javascript* (voltado para o *NodeJS*). A ferramenta teve início em 2010 como um software open-source de análise de código fonte voltada para a linguagem PHP. Em 2015 a RIPS Technologies foi fundada e a ferramenta foi aperfeiçoada para atender as linguagens *Java* e *Javascript*, como um software proprietário.

O *Jtest (Parasoft)* é um *software* para testes voltado para a linguagem de programação *Java* desenvolvido pela *Parasoft*. A ferramenta é capaz de realizar a análise estática do código fonte e testes unitários com o *JUnit*. Já o *Pylint (Pylint)* é uma ferramenta de análise estática de código fonte desenvolvida para a linguagem Python. Sua funcionalidade é voltada para a validação de sintaxe e detecção de erros, podendo ser integrado com editores de texto e IDEs.

De acordo com Sirqueira *et al.* (2016), a ferramenta *Code Smell Analyzer* trabalha integrada a IDE do Eclipse e visa auxiliar na identificação de maus cheiros de código fonte e refatoração. A Tabela 1 demonstra um comparativo entre as ferramentas citadas e a que desenvolvemos:

Tabela 1. Comparativo das ferramentas de análise estática.

	Linguagens Suportadas	Licença	Ocorrência de falsos positivos	Possibilidade de CI/CD	Disponível Online
PHP Analyzer	PHP	Open-source	Sim	Sim	Sim
CPPcheck	C/C++	Open-source	Sim	Não	Não
RIPS	PHP, Java, Node.js	Open-source (PHP) e proprietária	Sim	Sim	Sim
Jtest	Java	Proprietária	Sim	Sim	Não
Pylint	Python	Open-source	Sim	Sim	Não
Code Smell Analyzer	Java	Open-source	Sim	Não	Não

Fonte: Elaboração própria.

De acordo com a Tabela 1, podemos concluir que no mercado existem ferramentas robustas, oferecendo a possibilidade de CI/CD para automatizar os processos de testes e implantação em produção. Apesar disso, todas as ferramentas citadas, incluindo a que desenvolvemos, podem gerar falsos positivos em suas análises, o que tratamos como uma ameaça a validade, entretanto, é um problema comum em ferramentas de análise estática.

Acusar um problema onde não há ou, ainda, não acusar um onde há, são ameaças a validade, mas conforme a Tabela 1, é comum a todas as ferramentas. Segundo Bellairs (2018), ferramentas não são perfeitas e é inevitável que erros ocorram. Portanto, mesmo utilizando uma ferramenta de análise estática, é necessário que uma pessoa leia o código e os resultados das análises realizadas pelas ferramentas, para julgar as ocorrências de problemas a fim de identificar a necessidade de ajustes.

Aplicações web possuem algumas vulnerabilidades que podem proporcionar alguns ataques aos sistemas, tais como *SQL Injection e Command Injection.* Segundo Chandrashekhar et al. (2011, p. 524) SQL Injection é “um método de invasão no qual consultas SQL malformadas são produzidas por meio de entrada não autorizada do usuário.”. Este tipo de ataque tem a capacidade de expor ou corromper dados dos usuários da aplicação. De acordo com *The Mitre Corporation* o *Command Injection* (ou *Shell Injection*) permite que comandos inesperados e/ou perigosos sejam executados diretamente no sistema operacional do servidor.

O *SQL Injection* não pode ser testado com a ferramenta *PHP Analyzer*, pois depende da execução do código para ser validado. Já o *Command Injection* pode ser prevenido ao utilizar o *PHP Analyzer*, pois a ferramenta é capaz de localizar funções que podem executar comandos no servidor.

As vulnerabilidades analisadas com a ferramenta *PHP Analyzer,* são as definidas no catálogo do *PHP* como perigosas e recomendadas para serem desabilitadas, onde a ferramenta indica a existência das mesmas e os detalhes de implementação, ou seja, vulnerabilidades que podem ser identificadas por análise estática.

Para este trabalho, também é necessário falar dos repositórios de código fonte. Em 2005, Linus Torvalds, desenvolvedor do *kernel Linux,* desenvolveu o *Git*[10], uma ferramenta de versionamento de código fonte que opera de forma

[10] Git. Disponível em: <https://git-scm.com/>. Acessado em: 23 de maio de 2020

descentralizada, o que possibilita o desenvolvimento simultâneo de sistemas de software. O *Git* é base das plataformas como o Gitlab[11], Bitbucket[12] e Github[13].

Em Junho de 2018 o Github foi comprado pela Microsoft. A plataforma é utilizada por grandes empresas, tais como Google, Facebook e a própria Microsoft, além de ser um repositório de projetos *open-source* amplamente conhecidos como, por exemplo, o *kernel* do sistema operacional GNU/Linux.

A ferramenta *PHP Analyzer* também é capaz de se conectar ao Github e baixar o código fonte versionado de um repositório com o intuito de analisá-lo. Tal recurso será apresentado nas diretrizes de utilização do *PHP Analyzer*. O código fonte da ferramenta encontra-se disponível para *download* em https://github.com/ces-jf/PHP-Analyzer.

Na próxima seção discutiremos o desenvolvimento da ferramenta, apresentando detalhes tecnológicos, *frameworks*, bibliotecas e do sistema gerenciador de banco de dados que foram utilizadas em sua construção, além de detalhar o versionamento de código em repositórios.

3 *PHP ANALYZER*

A ferramenta *PHP Analyzer* foi desenvolvida na linguagem *PHP* em sua versão atual, até no momento é 7.4. Para acelerar o desenvolvimento, utilizou-se o *framework Laravel*. O *Laravel*[14] é um *framework* para aplicações web *PHP* que facilita o desenvolvimento de tarefas comuns na maioria dos projetos. Optou-se pelo *framework* por oferecer um ecossistema completo e rápido para o desenvolvimento das funcionalidades principais da ferramenta. Questões como conexão com banco de dados, criação de esquemas de usuários, rotas, dentre outras, podem ser resolvidas com o auxílio do *Laravel*.

Como base de dados utilizou-se o *PostgreSQL*[15], um sistema gerenciador de banco de dados (SGBD) *open-source,* com mais de 30 anos no mercado e mais de 500 contribuidores. A escolha desse SGBD deu-se pelo baixo custo de manutenção, plataforma com disponibilidade em sistemas operacionais e serviços na nuvem, compatibilidade com a linguagem de programação, além da possibilidade de criar tipos de dados próprios para a aplicação, se necessário.

---

[11] Gitlab. Disponível em: <https://about.gitlab.com/>. Acessado em: 23 de maio de 2020
[12] Bitbucket. Disponível em: <https://bitbucket.org/>. Acessado em: 23 de maio de 2020
[13] Github. Disponível em: <https://github.com/>. Acessado em: 23 de maio de 2020
[14] Laravel. Disponível em: <https://laravel.com/>. Acessado em: 08 de abril de 2020
[15] PostgreSQL. Disponível em: <https://www.postgresql.org/>. Acessado em: 23 de maio de 2020

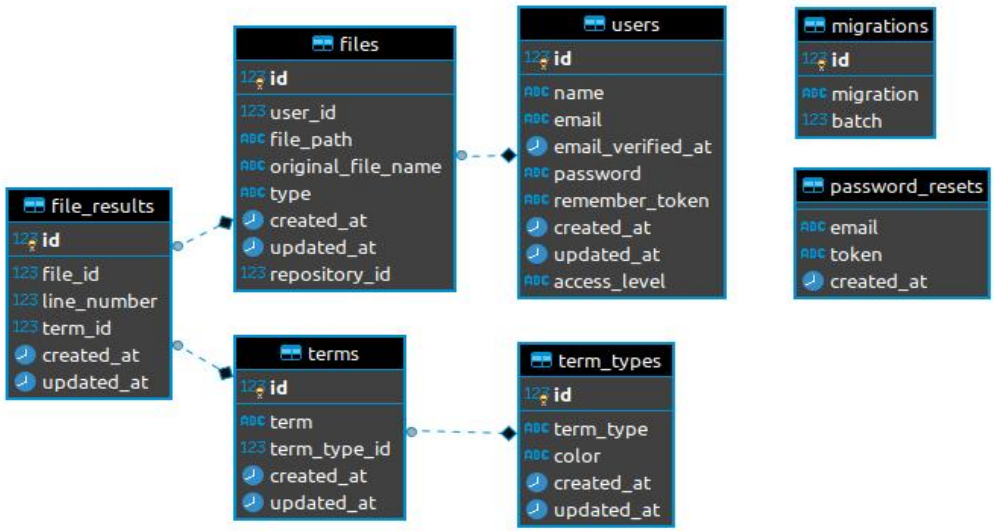
Na Figura 1, podemos ver o diagrama entidade relacionamento da aplicação. Já conforme a Figura 2, o sistema possui cinco classes de modelo principais, as quais possuem suas respectivas classes de controle que são responsáveis por realizar as operações de inserção, atualização, exclusão e consulta no banco de dados, nas respectivas tabelas no banco de dados da Figura 1.

A classe "*User*" é o modelo da tabela "*users*". A classe "*File*" é o modelo da tabela "*files*" e possui um auto relacionamento para gravar o local da pasta raiz com o atributo "*repository_id*".

A classe "*FileResults*" é o modelo da tabela "*file_results*" e é responsável por armazenar os resultados encontrados em um arquivo, gravando dados importantes para gerar o relatório da análise, sendo eles qual o arquivo, qual linha do arquivo foi identificado o problema e qual problema foi identificado.

A classe "*Terms*" é o modelo da tabela "*terms*" e é responsável por gravar os termos que serão comparados para realizar a análise. Por fim, a classe "*TermTypes*" é o modelo da tabela "*term_types*" e é responsável por categorizar os termos que serão utilizados na análise.

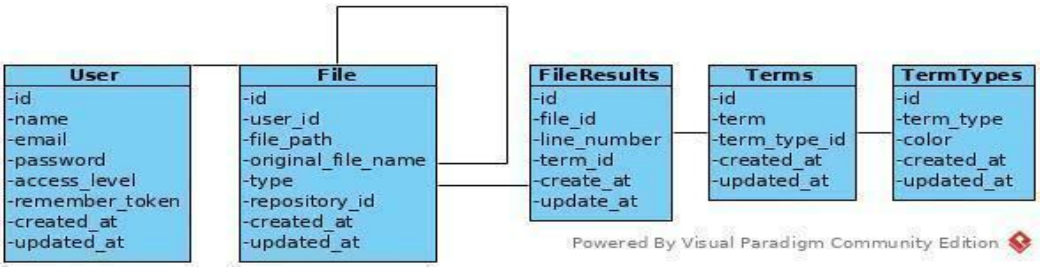
Figura 1. Diagrama entidade-relacionamento da aplicação.



Fonte: Elaboração Própria.

A seguir, na Figura 2, podemos ver o diagrama de classes do sistema, que se correlaciona com o modelo de entidade relacionamento da Figura 1.

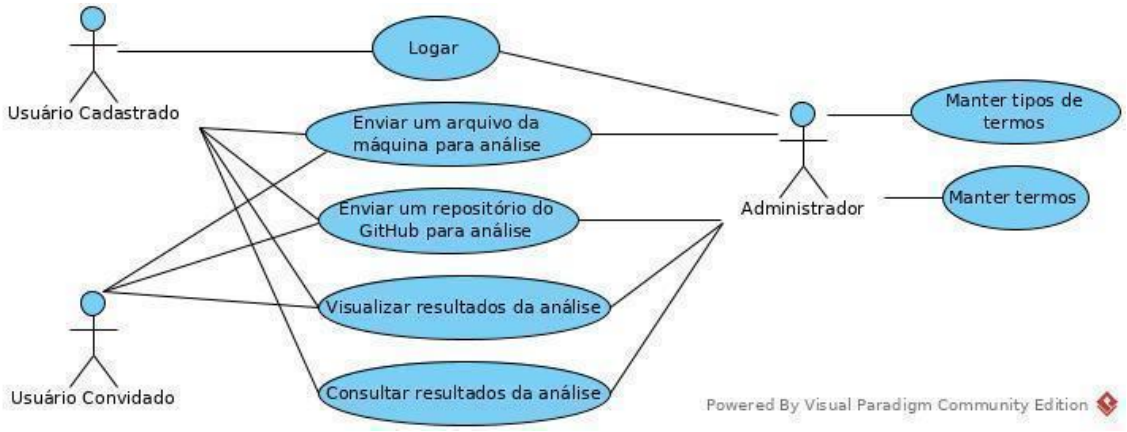
Figura 2. Diagrama de classes da aplicação.



Fonte: Elaboração Própria.

Na Figura 3, podemos ver o diagrama de caso de uso da aplicação, representando a interação dos usuários com as funções que a ferramenta apresenta. No diagrama de caso de uso (Figura 3), podemos identificar três atores no sistema. O “Usuário cadastrado”, o “Usuário convidado” e o “Administrador” do sistema. Os atores do sistema podem realizar, basicamente, todas as operações relacionadas à análise do código fonte. Todos os atores podem realizar o *upload* através da sua máquina, submeter algum repositório de código fonte e visualizar os resultados da análise. Apenas o usuário convidado não pode realizar uma consulta posterior dos seus arquivos analisados. O administrador tem o poder de gerenciar os termos que serão utilizados na análise e seus tipos.

Figura 3. Diagrama de caso de uso.



Fonte: Elaboração Própria.

4 DIRETRIZES DE UTILIZAÇÃO

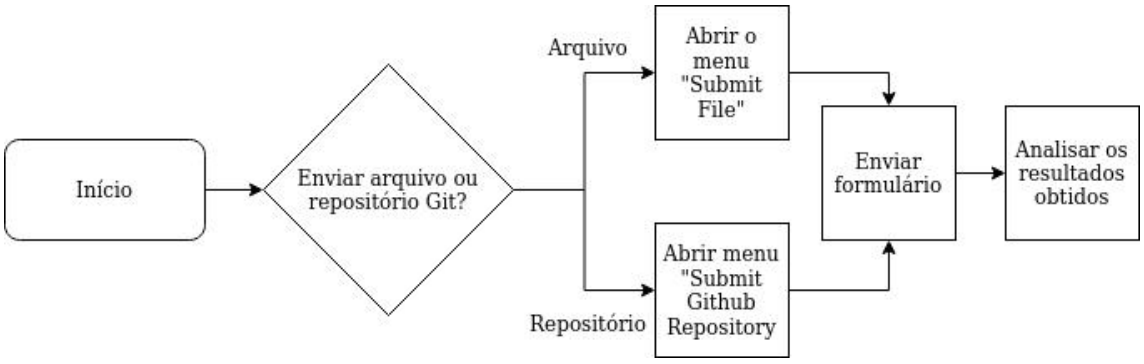
A ferramenta *PHP Analyzer* busca por termos pré-definidos no código fonte da aplicação *PHP*, verificando sua incidência linha a linha. Foi definido como critério para este trabalho, as funções depreciadas, desabilitadas por padrão nas configurações do *PHP* e as funções capazes de executar comandos dentro do sistema operacional, a fim de exemplificar seu funcionamento.

Funções obsoletas são aquelas que com a evolução da linguagem deixaram de ser suportadas. Já as funções desabilitadas por padrão, podem comprometer a segurança do sistema e até mesmo do servidor da aplicação devido às suas propriedades, tais como *pcntl_setpriority,* capaz de alterar a prioridade de qualquer processo do servidor.

Funções de execução de comandos são perigosas se utilizadas de maneira incorreta. Para Atkinson (2000), É muito perigoso se utilizar qualquer informação enviada pelo usuário como parâmetro para essas funções, pois algum script malicioso pode ser executado no servidor através delas, o que pode abrir brechas na segurança para alguma invasão mais completa, com roubo de dados dos usuários, do código fonte da aplicação ou até mesmo negação de serviço. Portanto, funções como *exec, passthru, shell_exec e system* devem ser evitadas para mitigar possíveis problemas de segurança. Caso seja necessário utilizar essas funções com algum parâmetro enviado pelo usuário, Atkinson (2000) afirma ser necessário tratar a entrada com a função *escape-shellcmd*.

A fim de demonstrar a execução da ferramenta, o fluxograma descrito na Figura 4 será adotado.

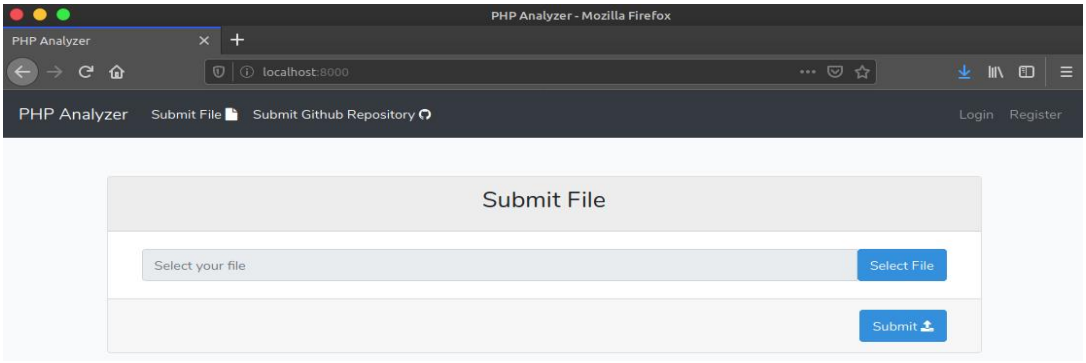
Figura 4. Fluxograma de uso da ferramenta.



Fonte: Elaboração Própria.

Ao iniciar a ferramenta *PHP Analyzer*, a primeira tela exibida possibilita realizar o *upload* de um arquivo por meio da máquina do usuário (Figura 5). Esta tela também pode ser acessada através da barra de navegação no botão "*Submit File*" e clicando sobre o nome da aplicação. Através desta tela (Figura 5), é possível efetuar o *login* no sistema ou se registrar caso ainda não tenha uma conta. Além disso, é possível acessar a tela para análise de arquivos a partir de um repositório do GitHub.

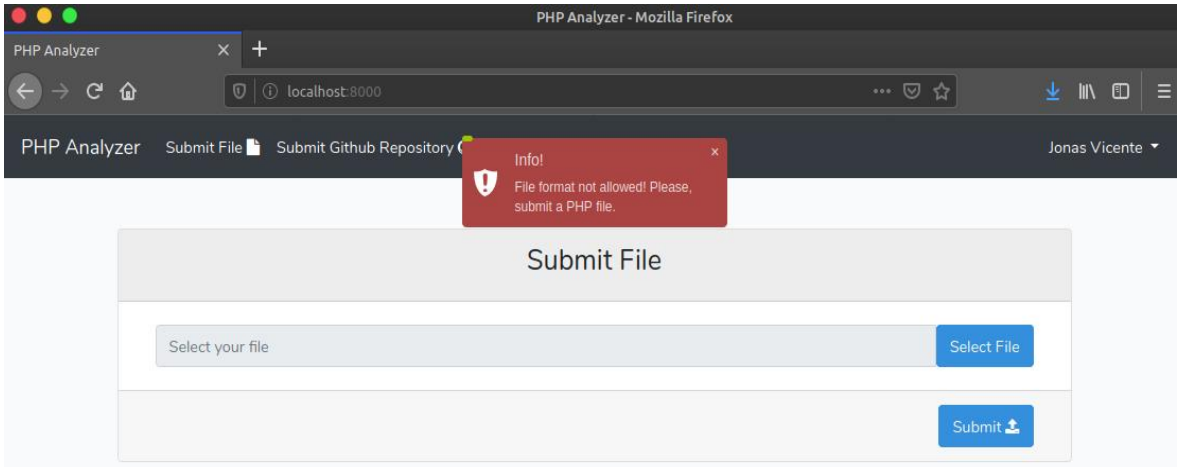
Figura 5. Formulário para realizar o upload de um arquivo.



Fonte: Elaboração Própria.

Na imagem acima (Figura 5), podemos ver apenas um campo. Este campo é utilizado para enviar um arquivo *PHP* para a análise, caso seja selecionado um arquivo de outro formato, o sistema exibe uma mensagem de erro, indicando que o formato não é condizente com o esperado (Figura 6). Ao enviar o formulário (Figura 5), o sistema irá realizar o *upload* do arquivo e, ao seu término, irá analisá-lo. Com a análise concluída os problemas encontrados são exibidos na tela, que apresentaremos a frente.

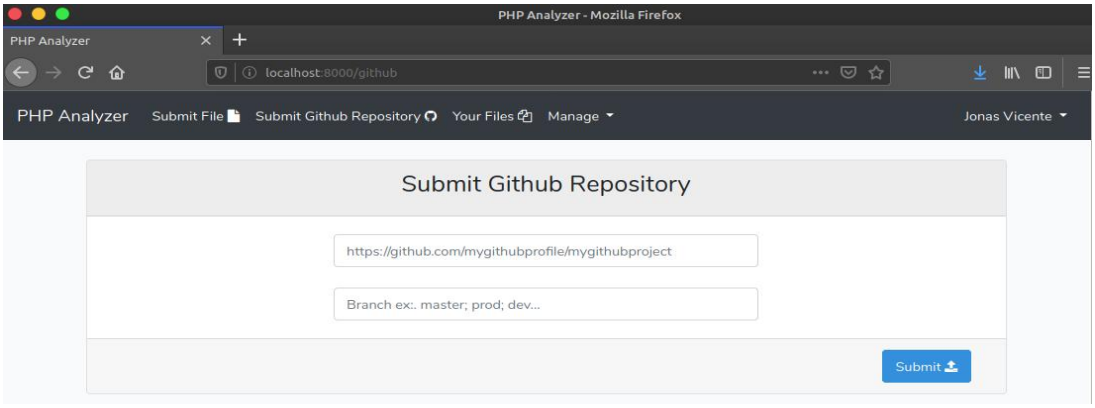
Figura 6. Mensagem de erro devido a formato não condizente.



Fonte: Elaboração Própria.

Na Figura 7, observa-se a tela responsável por realizar o envio de um repositório para a análise. No primeiro campo deve-se preencher a URL (*Uniform Resource Locator* - Localizador Uniforme de Recursos) onde se encontra o código fonte a ser analisado. No segundo campo, deve-se preencher o *Branch* (ramo) do repositório a ser analisado. Ao enviar o formulário, o sistema irá realizar o *download* do repositório compactado no formato *ZIP*.

Figura 7. Formulário para enviar um repositório para análise.



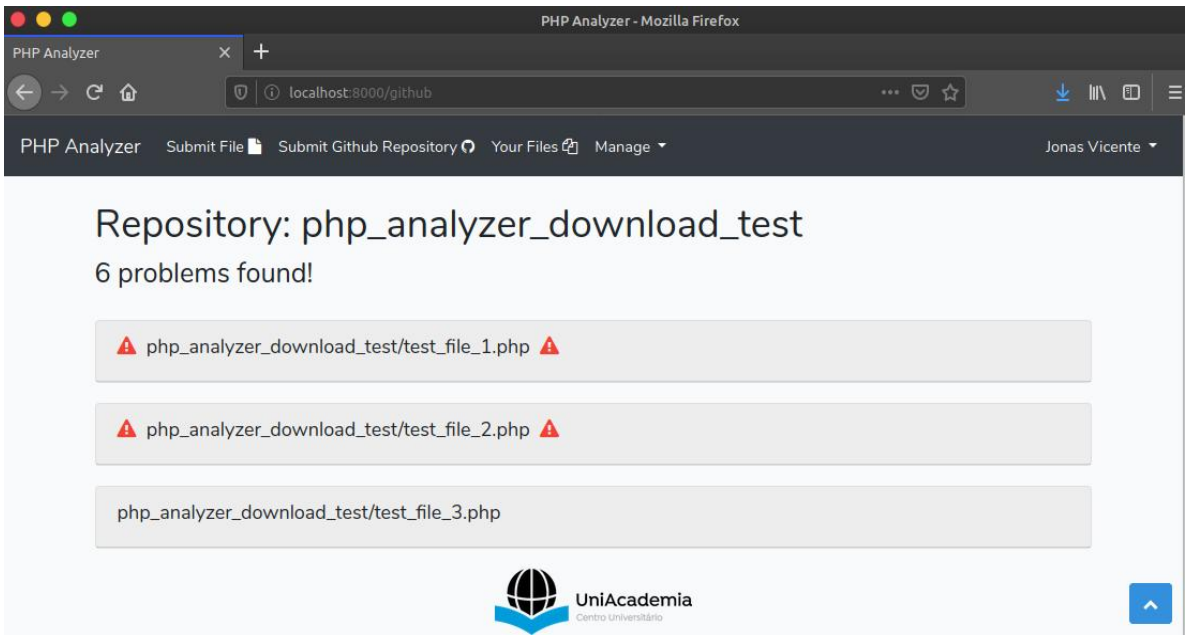
Fonte: Elaboração Própria.

Com o *download* concluído, o sistema descompacta o arquivo *ZIP* do repositório e inicia a análise dos arquivos PHP encontrados no repositório. Arquivos de outros formatos serão ignorados durante a análise e não serão exibidos no resultado.

Ao término da análise de um repositório é exibido a tela da Figura 8. Nesta tela (Figura 8) serão listados todos os arquivos PHP encontrados, mesmo que não contenham problemas detectados.

Na Figura 8, exibe-se os arquivos *PHP* de uma análise. Para facilitar a visualização, os arquivos vêm fechados e exibe-se apenas seu nome, onde caso algum problema seja encontrado no código fonte, o nome do arquivo é cercado pelos ícones de atenção (triângulos vermelhos com exclamação).

Figura 8. Tela com a exibição dos resultados da análise de um repositório.

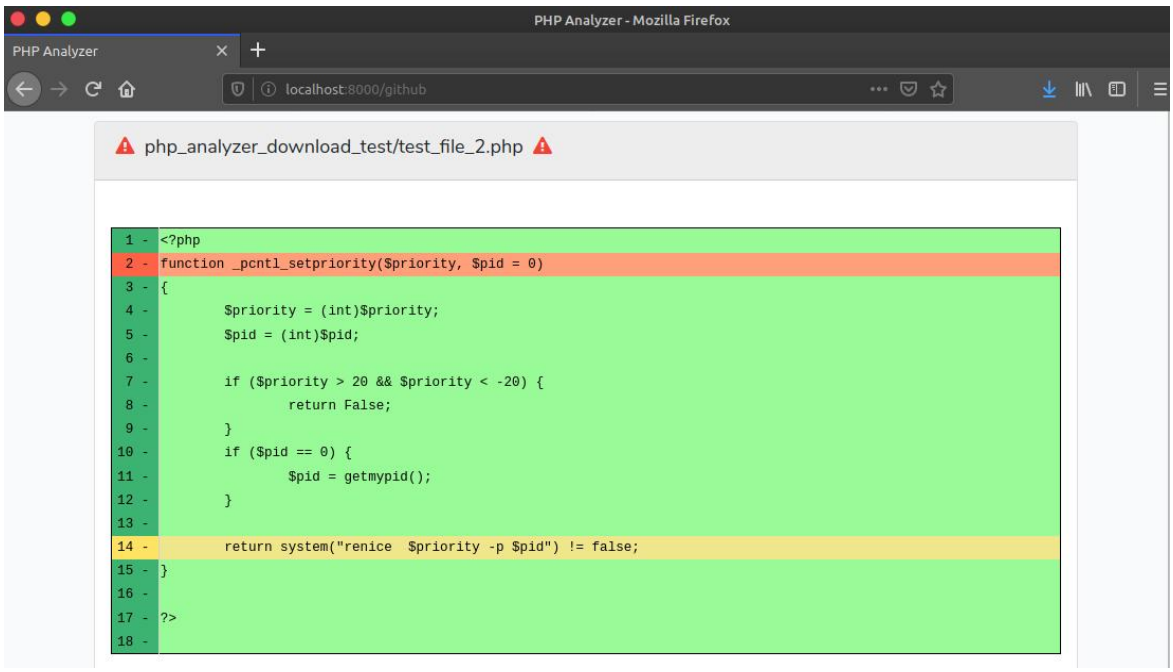


Fonte: Elaboração Própria.

Na Figura 9 podemos ver a exibição dos resultados da análise de um dos arquivos encontrados no repositório enviado. Pode-se observar na Figura 9, que o *PHP Analyzer* considerou o nome da função "*pcntl_setpriority"* (linha dois do código fonte) como um problema.

A Figura 10 apresenta um relatório com os dois problemas encontrados no arquivo de exemplo analisado, acusando o uso da função "*pcntl_setpriority*", sendo uma das funções desabilitadas no arquivo de configurações do PHP. A ferramenta também acusa o uso da função "*system*", a qual se enquadra na categoria de funções de execução de programas no servidor.

Figura 9. Código fonte com problemas detectados.

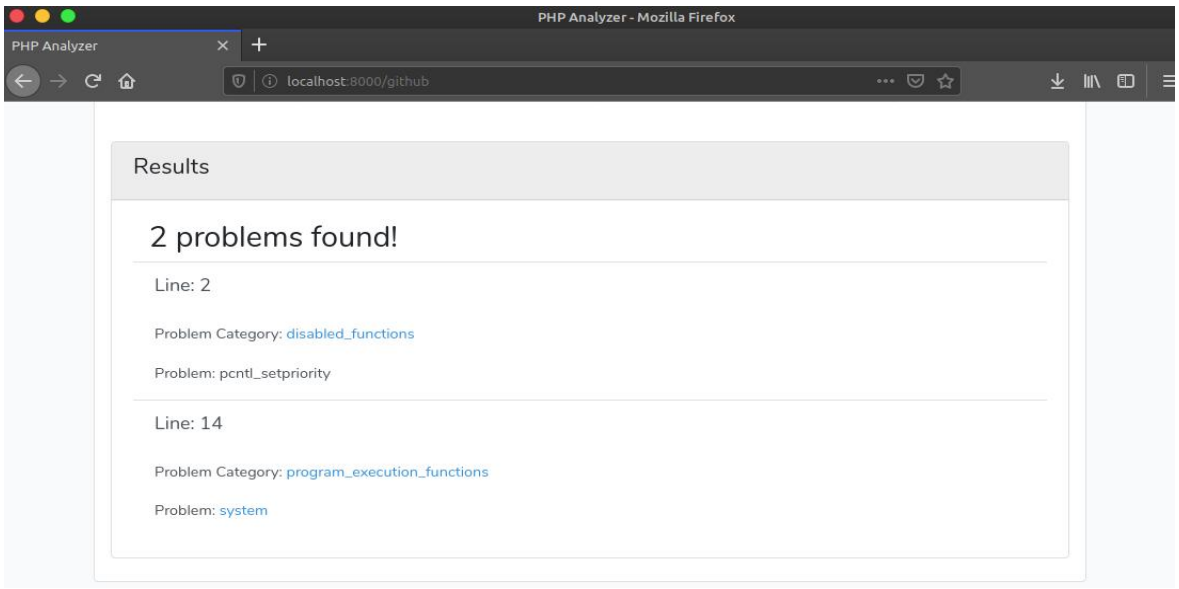


Fonte: Elaboração Própria.

Para os problemas de uso de funções de execução de programas, é possível clicar sobre o nome da função no relatório para visualizar a definição da mesma no site oficial do PHP. Algumas das funções podem não possuir documentação oficial, portanto o link pode retornar uma mensagem dizendo que a página não existe.

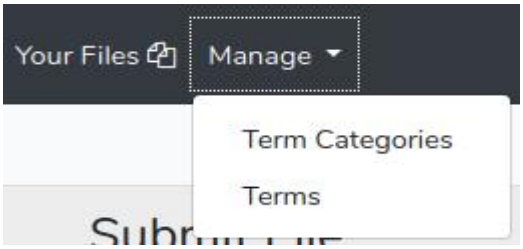
Usuários administradores podem gerenciar as cores e os termos a serem buscados durante a análise. Na Figura 11 podemos ver dois menus que sãos disponíveis apenas para usuários logados no sistema. O menu "*Your Files*" exibe todos os arquivos e repositórios analisados referentes ao usuário logado (Figura 12).

Figura 10. Lista dos problemas encontrados na análise do código fonte.



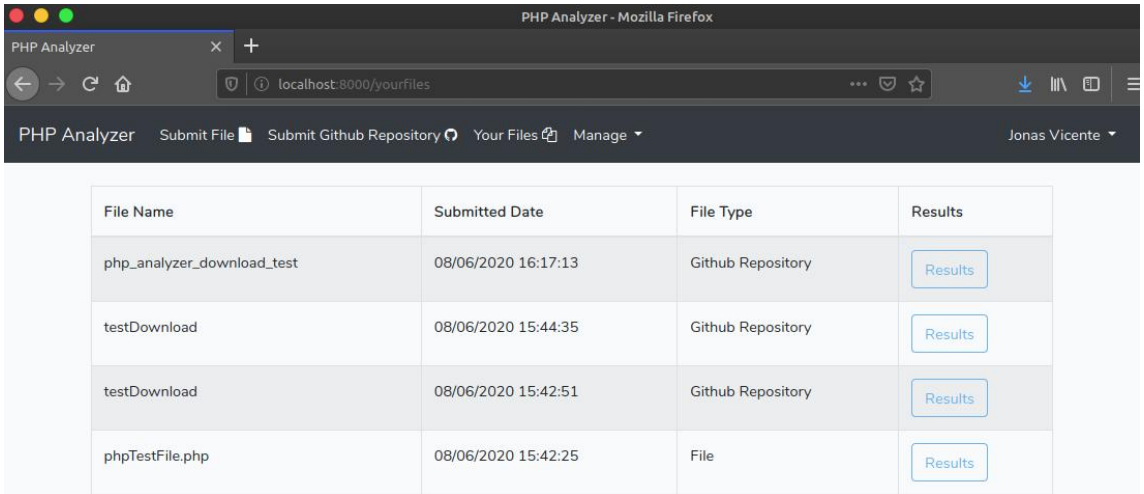
Fonte: Elaboração Própria.

Figura 11. Menus com usuários logados no sistema.



Fonte: Elaboração Própria.

Figura 12. Tela com a lista dos arquivos e repositórios analisados.



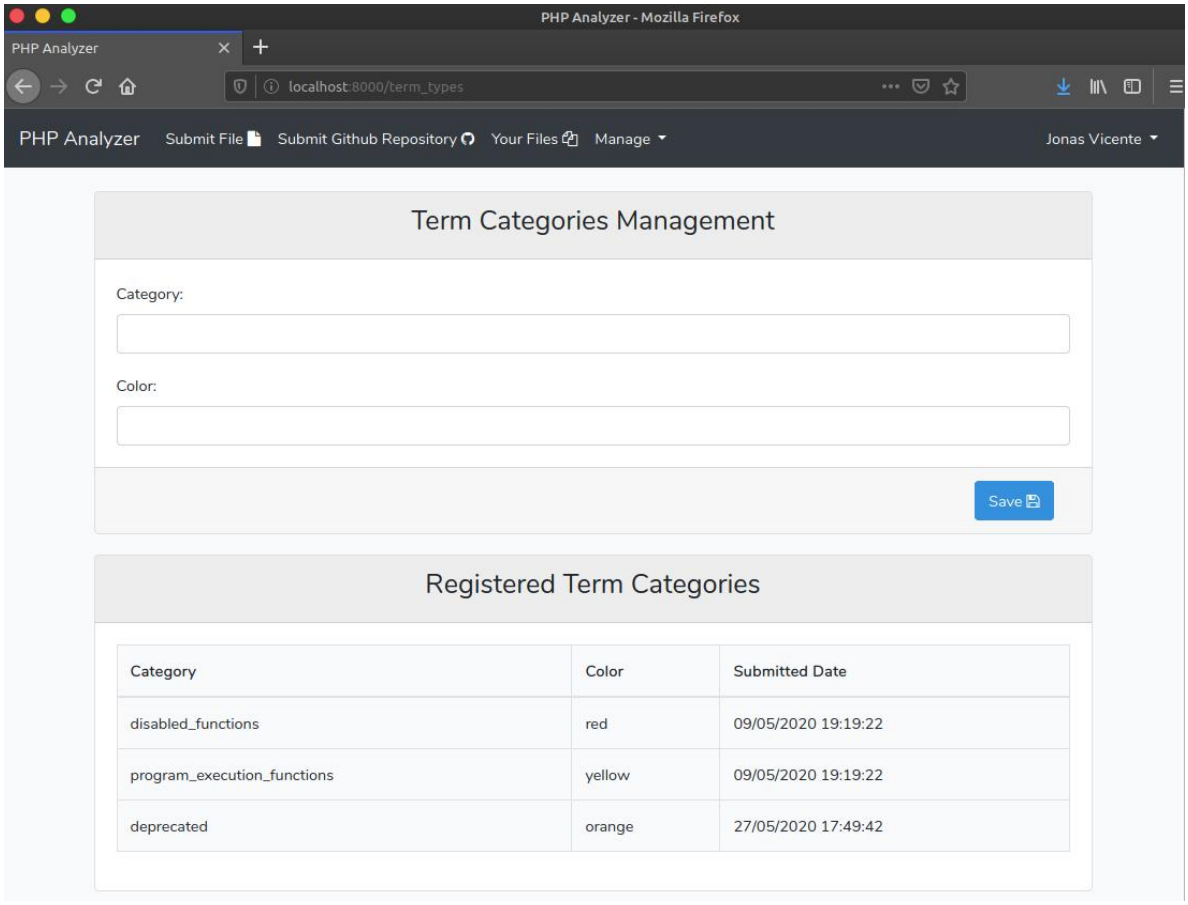
Fonte: Elaboração Própria.

Na Figura 12, podemos observar a tela com a listagem dos arquivos e repositórios enviados para análise de um usuário. A tabela possui quatro colunas, sendo elas o nome do arquivo analisado, a data em que foi analisado, o método utilizado para seu envio e um botão para exibir os problemas encontrados.

O menu "*Manage*" (Figura 11) e seus submenus "*Term Categories*" e "*Terms*" são exclusivos para usuários com acesso administrativo ao sistema. Neste menu é possível gerenciar as categorias dos termos, onde define-se nome e cor (Figura 13) e também os termos que serão utilizados na análise, onde define-se nome e categoria a qual se enquadra (Figura 14).

Na Figura 13, temos o formulário para realizar o cadastro das categorias dos termos, onde podemos definir um nome e uma cor. Logo abaixo do formulário (Figura 13), temos uma tabela com todas as categorias cadastradas.

Figura 13. Tela para o gerenciamento de categorias de termos.



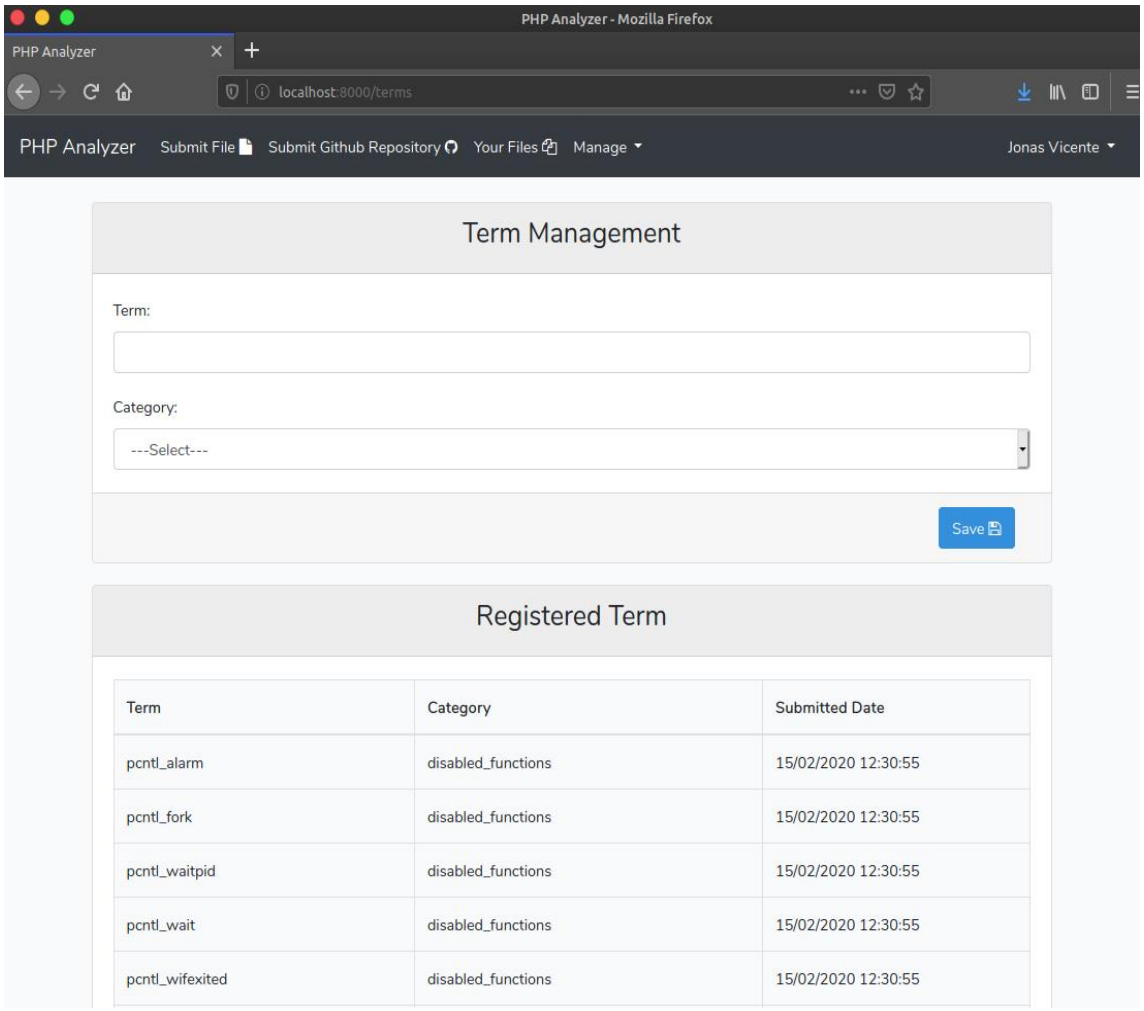
Fonte: Elaboração Própria.

Ao clicar sobre uma das linhas da Figura 13, o sistema irá redirecioná-lo à tela de edição da categoria. A tela de edição é exatamente a mesma que da

Figura 13, porém, os campos do formulário preenchidos com os dados da categoria a ser editada.

Na Figura 14 observa-se a tela para gerenciar os termos que serão utilizados para a análise do código fonte. Seu funcionamento é semelhante ao formulário de cadastro de categorias de termos. Neste precisa-se informar o nome do termo e sua categoria. Logo abaixo do formulário (Figura 14), temos uma tabela listando todos os termos cadastrados. Ao clicar sobre uma das linhas, o usuário é redirecionado para a tela de edição.

Figura 14. Tela para o gerenciamento de termos.



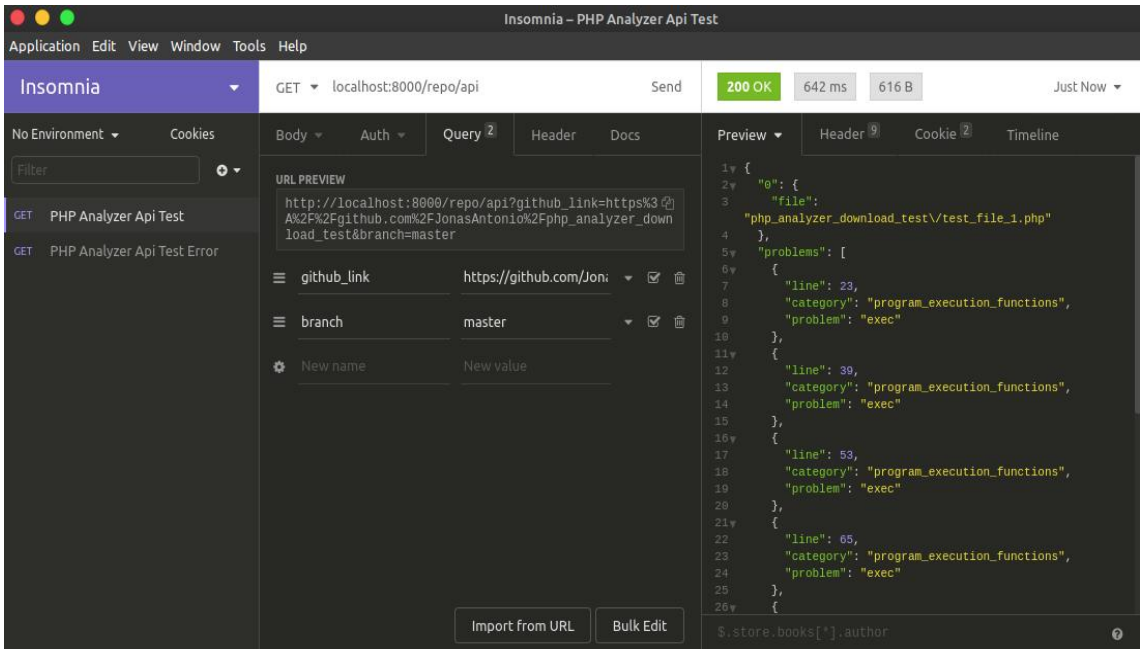
Fonte: Elaboração Própria.

Além dessas formas de análise, também é possível realizar a análise do código fonte via API, com o método GET através da URL com final /repo/api, passando o parâmetro "*github_link*" com o link para o repositório e o "*branch*" a

ser analisado. Na Figura 15 podemos ver a simulação de uma requisição à API utilizando o *software* Insomnia[16].

Com a entrada correta dos parâmetros, a API retorna um *JSON* (*JavaScript Object Notation*) com os problemas encontrados na análise. O processo é similar ao realizado para o resultado apresentado na Figura 10. Em caso de entrada incorreta, a ferramenta retorna uma mensagem de erro (Figura 16).

Figura 15. Simulação de requisição à API da ferramenta PHP Analyzer.

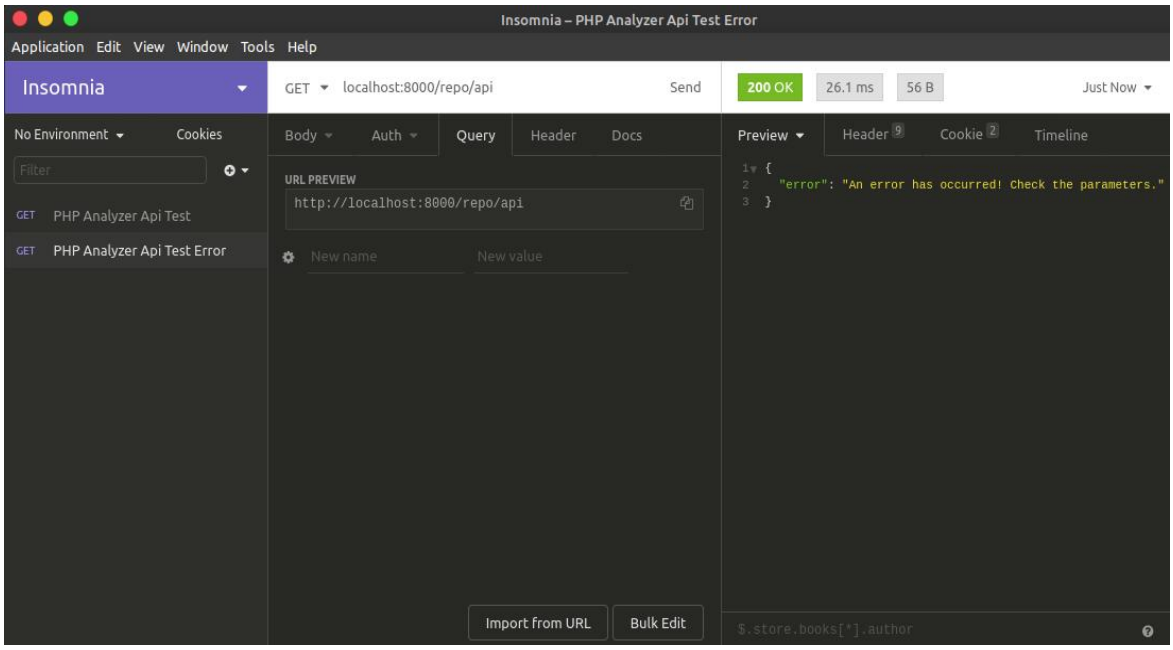


Fonte: Elaboração Própria.

Uma demonstração de uso da ferramenta está disponível em https://youtu.be/X0VC8wGDCbc. Na próxima seção abordaremos algumas considerações sobre este trabalho, as ameaças a validade e os trabalhos futuros.

[16] Imsomnia. Disponível em: <https://insomnia.rest/>. Acessado em: 26 de maio de 2020

Figura 16. Requisição à API com erro nos parâmetros.



Fonte: Elaboração Própria.

5 CONSIDERAÇÕES FINAIS

As linguagens de programação, assim como os sistemas de software evoluem e essa evolução necessita ser acompanhada em ambos. Podemos observar que com a evolução da linguagem, funções se tornam obsoletas, são marcadas como perigosas e que podem afetar a segurança de sistemas. Problemas de segurança por conta de funções obsoletas e que permitem a execução de código, podem ser identificados com uma ferramenta de análise estática do código fonte, auxiliando no processo de manutenção e evolução dos sistemas.

Embora ferramentas de análise estática de código fonte sejam de grande utilidade, não podemos abandonar o fator humano nas análises. Os resultados da ferramenta devem ser analisados por um humano, pois falsos positivos e falsos negativos podem ocorrer, onde segundo Bellairs (2018), ferramentas não são perfeitas e é inevitável que erros ocorram.

Algumas limitações da ferramenta são:

- Integração com apenas um sistema Git;
- Gerenciamento manual dos termos para análise;
- Não identificação de código fonte duplicado.

Como objetivos futuros para a ferramenta *PHP Analyzer*, pretende-se melhorar as seguintes funcionalidades:

- Exportação dos resultados em PDF;
- Automação dos termos para análise;
- Envio de um diretório completo via upload da máquina do usuário;
- Suporte para a análise de código fonte de outras linguagens de programação tais como o *Java, Javascript e Python*;
- Melhorias na análise, visando a redução de falsos positivos nos resultados;
- Ignorar linhas comentadas nas análises;
- Desenvolvimento de um *middleware* para conexão com repositórios;
- Conexão com as APIs do Github, do Gitlab e do Bitbucket;
- Desenvolver uma profundidade de busca por meio de filtros de categorias que devem ser analisadas;
- Limitar o acesso ao sistema apenas para usuários cadastrados;
- Testar a ferramenta em alguma empresa, a fim de avaliar suas funcionalidades em um ambiente profissional.

Por fim, com esta ferramenta os desenvolvedores de software podem rastrear funções vulneráveis, que foram marcadas como perigosas na linguagem PHP ou que se tornaram obsoletas e que forçam o desenvolvedor a atualizar os sistemas para manterem-se compatíveis com a nova versão da linguagem. Com uma ferramenta automatizada, o processo de busca de funções dentro de um sistema é mais rápido e, apesar dos falsos positivos, menos falho do que depende de um humano para ler linha por linha do código fonte.

Referências

ATKINSON, L. **Core PHP Programming:** 2. ed. Prentice Hall PTR. 2000.

BELLAIRS, R. **What Are False Positives and False Negatives?**. Disponível em: <https://www.perforce.com/blog/qac/what-are-false-positives-and-false-negatives>. Acesso em: 05 de maio de 2020.

BELLAIRS, R. **What is Static Analysis? And What is Code Static Analysis**. Disponível em: <https://www.perforce.com/blog/sca/what-static-analysis>. Acesso em: 22 de junho de 2020.

BHATT P; SHROFF G; MISRA A. **Dynamics of Software Maintenance**. ACM SIGSOFT Software Engineering Notes. 2004. v. 29, n. 5, p. 1-5.

CANTO, F; WEBER, R. **Vulnerabilidades da linguagem PHP**. Disponível em: <https://www.lume.ufrgs.br/handle/10183/31030>. Acesso em: 06 de abril 2020.

CHANDRASHEKHAR, R et al. **SQL Injection Attack Mechanisms and Prevention Techniques**. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-642-29280-4_61>. Acesso em: 07 de maio de 2020.

THE MITRE CORPORATION. **CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection').** Disponível em: <https://cwe.mitre.org/data/definitions/78.html>. Acesso em: 21 de junho de 2020.

G1. **Microsoft compra GitHub por US$ 7,5 bilhões e anuncia mudanças**. Disponível em: <https://g1.globo.com/economia/tecnologia/noticia/microsoft-compra-github-por-us-75-bilhoes.ghtml>. Acesso em: 09 de abril de 2020.

GITHUB. **Github, Página inicial**. Disponível em: <https://github.com>. Acesso em: 08 de abril 2020.

LARAVEL. **Laravel: The PHP Framework For Web Artisans**, 2020. Página inicial. Disponível em: <https://laravel.com>. Acesso em: 08 de abril 2020.

MARJAMÄKI, D. **Cppcheck - A tool for static C/C++ code analysis**, 2020. Disponível em: <https://sourceforge.net/p/cppcheck/wiki/Home/>. Acesso em: 02 de junho. 2020.

MENDES, Carolina de Aguiar Teixeira. **Como Surgiu a Internet?. Brasil Escola**. Disponível em: <https://brasilescola.uol.com.br/curiosidades/como-surgiu-a-internet.htm>. Acesso em 06 de abril 2020.

PARASOFT. **Parasoft: Automated Software Testing**, 2020. Página inicial. Disponível em: <https://www.parasoft.com>. Acesso em: 08 de abril 2020.

PHP GROUP. **História do PHP**. Disponível em: <https://www.php.net/manual/pt_BR/history.php.php>. Acesso em: 06 de abr. 2020.

POSTGRESQL. **PostgreSQL: About**, 2020. Sobre. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 09 de abril 2020.

PYLINT. **Pylint: code analysis for Python**, 2020. Página inicial. Disponível em: <https://www.pylint.org>. Acesso em: 08 de abril 2020.

RIPSTECH. **RIPS: The Technology Leader In Static Application Security Testing**, 2020. Página inicial. Disponível em: <https://www.ripstech.com/>. Acesso em: 08 de abril 2020.

SIRQUEIRA, Tassio Ferenzini Martins et al. **Code smell analyzer: a tool to teaching support of refactoring techniques source code**. IEEE Latin America Transactions, v. 14, n. 2, p. 877-884, 2016.

STACK Overflow. **Developer Survey Results** 2019. Disponível em: <https://insights.stackoverflow.com/survey/2019#technology>. Acesso em: 06 de abril 2020.

TERRA, R. **Análise estática de código**. Disponível em: <http://professores.dcc.ufla.br/~terra/public_files/2008_palestra_aec.pdf>. Acesso em: 08 de abril 2020.

W³Techs. **Usage statistics of PHP for websites**. Disponível em: <https://w3techs.com/technologies/details/pl-php>. Acesso em: 06 de abril 2020.

W³Techs. **Usage statistics of server-side programming languages for websites**. Disponível em: <https://w3techs.com/technologies/overview/programming_language>. Acesso em: 06 de abril 2020