Security Smells em Infraestrutura como Código utilizando Docker

Daniel David Fernandes[1], **Lucas Dantas Gama Ayres**[1], **Cláudio Nogueira Sant'Anna**[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Av. Adhemar de Barros, s/nº - Ondina, Salvador - BA, 40170-110 – Salvador – BA – Brasil

{daniel.david,lucas.ayres}@ufba.br, santanna@dcc.ufba.br

Abstract. *Infrastructure as code (IaC) is an approach that allows you to create an infrastructure automatically. Docker is a popular tool to provide environments automatically, through source code. By developing configuration code, IT practitioners can introduce Security Smells, which can lead to a security hole. This work aims to propose Security Smells to the Docker ecosystem and assess its security impacts in order to avoid them. An analysis was made in 1500 GitHub repository Dockerfiles, where we verified that the Security Smells proposed to Puppet also apply to the context of the Docker, and we also propose two new Security Smells for IaC scripts.*

Resumo. *Infraestrutura como código (IaC) é uma abordagem que permite criar uma infraestrutura automaticamente. Docker é uma ferramenta popular para prover ambientes de forma automática, por meio de código-fonte. Ao desenvolver esse código de configuração, profissionais de TI podem introduzir Security Smells, que podem levar a uma falha de segurança. Este trabalho tem como objetivo propor Security Smells ao ecossistema do Docker e avaliar seus impactos de segurança, com o intuito de evitá-los. Foi feita uma análise em 1500 Dockerfiles de repositórios do GitHub, onde verificamos que os Security Smells propostos ao Puppet também se aplicam ao contexto do Docker, além disso propomos dois novos Security Smells para scripts de IaC.*

1. Introdução

Infraestrutura como Código (em inglês: *Infrastructure as Code - IaC*) é uma abordagem utilizada para definir uma infraestrutura de rede e computadores por meio de código-fonte, portanto pode ser tratada como uma aplicação de *software* [Fowler 2016]. Os *scripts* IaC ajudam os profissionais de TI a provisionar e gerenciar uma infraestrutura baseada em nuvem [Pahl et al. 2017]. Uma das ferramentas de IaC mais populares na atualidade é o *Docker*, que é uma plataforma que automatiza a implantação de aplicações dentro de ambientes isolados denominados *containers* [Pahl et al. 2017].

Ao desenvolver esse código de configuração, profissionais de TI podem introduzir *Security Smells*, que são padrões de codificação que podem indicar a uma falha de segurança [Rahman et al. 2019]. Atualmente existem poucas pesquisas relacionadas a segurança deste código de configuração e quais impactos esses *Security Smells* podem ocasionar na infraestrutura.

Em 2016, uma configuração incorreta no banco de dados, expôs mais de 93 milhões de registros confidenciais de eleitores mexicanos [Ragan 2016]. Em 2017, a empresa IBM utilizou credenciais incorporadas na imagem *Docker* de um dos seus produtos,

ocasionando o vazamento da mesma [Chang 2017]. Esses casos mostram a importância da segurança no desenvolvimento de *scripts* de IaC e o quanto isso pode impactar em empresas e usuários.

Este trabalho tem como objetivo propor *Security Smells* em *scripts* de IaC ao *Docker* e avaliar os impactos de segurança que podem ser causados na infraestrutura, com o intuito de ajudar os profissionais de TI a evitarem falhas de segurança em *scripts* IaC. Neste artigo, respondemos às seguintes questões de pesquisa:

- **RQ1:** Os sete *Security Smells* propostos por Rahman et al. [Rahman et al. 2019] para o *Puppet* são aplicáveis também ao contexto do *Docker*?
- **RQ2:** É possível propor novos *Security Smells* para o *Docker*?
- **RQ3:** Com que frequência os *Security Smells* ocorrem em *scripts Docker*?

Para responder essas perguntas de pesquisa, foi realizada uma análise manual de 1500 *Dockerfiles*. Através dessa análise foi possível constatar que todos os *Security Smells* propostos para o *Puppet*, por Rahman et al. [Rahman et al. 2019], também se aplicam para o contexto do *Docker*, mostrando que as definições desses *Security Smells* são gerais e podem ser aplicadas a diferentes tecnologias e ferramentas.

Para propor novos *Security Smells*, nos baseamos no *Common Weakness Enumeration* (CWE) [MITRE 2008]. CWE é uma lista utilizada como base para ferramentas que medem a segurança em softwares e também é um padrão adotado para identificação, mitigação e prevenção de vulnerabilidades. Assim, conseguimos propor dois novos *Security Smells* para o *Docker* e, por fim, verificamos a frequência desses *Security Smells* em arquivos *Dockerfile*.

As principais contribuições deste trabalho são: (i) Análise de presença de *Security Smells* em *Dockerfiles*; (ii) Novos *Security Smells* e suas definições; (iii) Frequência que *Security Smells* ocorrem em *Dockerfiles*; (iv) Discussão sobre os impactos de segurança que cada *Security Smell* pode causar na infraestrutura.

2. Trabalhos Relacionados

Rahman *et al.* [Rahman et al. 2019] propõem 7 *Security Smells* para o *Puppet* de acordo com tipos de vulnerabilidades encontrados no CWE. Foi realizada uma análise através da execução de uma ferramenta para automatizar a detecção em 15.232 *scripts* de IaC.

Ghafari *et al.* [Ghafari et al. 2017] discutem *Security Smells* em Android, buscando mitigar efeitos dessas vulnerabilidades. Com o apoio de uma ferramenta desenvolvida, identificou dez *smells* em aplicações Android, obtendo em média 3 *Security Smells* por aplicativo.

Cito *et al.* [Cito et al. 2017] analisam problemas de qualidade e comportamento da evolução dos *Dockerfiles*. Analisaram 38079 projetos, onde observaram que *Dockerfiles* são atualizados, em média, apenas 4,7 vezes por ano.

Duarte e Antunes [Duarte and Antunes 2018] investigam vulnerabilidades de segurança do *Docker* e o que pode ser feito para evitá-las, sistematizando-os causas, efeitos e consequências, baseando-se em conceitos do *Common Vulnerabilities and Exposure* (CVE).

No estado da arte há análises de vulnerabilidades no contexto Docker, mas não apresenta análises de *Security Smells* aplicados ao mesmo.

3. Metodologia e Resultados

Nesta seção apresentamos a metodologia proposta para responder nossas questões de pesquisa e os resultados obtidos através dessas análises. Para construir o *dataset*, foi desenvolvido um *script* em *Python* com o intuito de selecionar e coletar *Dockerfiles* de repositórios do *GitHub*, utilizando como critério de busca aplicações recentemente atualizadas.

3.1. Verificando se os *Security Smells* propostos ao *Puppet* se aplicam ao *Docker*

Para verificar se os *Security Smells* propostos para o *Puppet* [Rahman et al. 2019] também se aplicam ao *Docker*, realizamos uma análise manual nos 1500 *Dockerfiles* coletados, para tentar identificar os *Security Smells*, já que *Docker* e *Puppet* possuem sintaxes e abordagens distintas.

Tabela 1. Security Smells propostos ao Puppet [Rahman et al. 2019].

Security Smell	Código CWE	Descrição
Admin por padrão	CWE-250	Execução com privilégios desnecessários
Senha vazia	CWE-258	Senha vazia no arquivo de configuração
Credenciais definidas em texto claro	CWE-789 CWE-259	Uso de credenciais em texto claro Uso de senha em texto claro
Vinculação com endereço IP impróprio	CWE-284	Controle de acesso inadequado
Comentário suspeito	CWE-546	Comentário suspeito
Uso de HTTP sem TLS	CWE-319	Transmissão de informação sensível em texto não criptografado
Uso de Algoritmos de criptografia fraca	CWE-327 CWE-326	Uso de algoritmo criptográfico quebrado ou inseguro Intensidade de criptografia inadequada

Identificamos todos os sete *Security Smells* propostos para o Puppet em arquivos *Dockerfile*. Os resultados estão a seguir através de exemplos de *Dockerfiles* contendo esses *Security Smells* encontrados, com discussões sobre seus impactos de segurança e alternativas para mitigá-los.

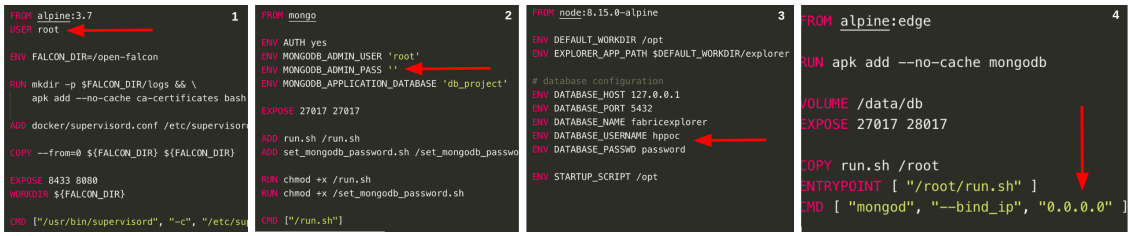

Figura 1. Exemplos de *Dockerfiles* com admin por padrão, senha vazia, credenciais definidas em texto claro e vinculação com endereço IP impróprio.

***Admin* por padrão**: Na imagem 1 da Figura 1, a instrução *USER* define o usuário que terá acesso ao *container*, caso não seja especificado, por padrão o usuário será o *root*. Um invasor que tem acesso a um *container* como *root*, terá acesso total ao *host*, que é a máquina onde está instalado o *Docker*. Para evitar este caso recomenda-se definir um usuário específico, concedendo apenas os privilégios necessários para realizar as determinadas ações.

Senha vazia: Na imagem 2 da Figura 1, mostra uma senha vazia para um usuário de banco de dados, facilitando a adivinhação pelos atacantes. É recomendado que se utilize sempre padrões de senhas fortes.

Credenciais definidas em texto claro: Na imagem 3 da Figura 1, usuário e senha de um banco de dados são informados diretamente. Se um invasor tiver acesso ao *container* terá acesso a estas informações. Para evitar esse tipo de vulnerabilidade é importante que estas credenciais sejam definidas em variáveis de ambiente ou que estejam criptografadas.

Vinculação com endereço IP impróprio: Na imagem 4 da Figura 1, realiza-se uma conexão através do endereço IP "0.0.0.0", podendo expor um servidor, serviço ou instância a conectar-se em todas as redes possíveis [Mutaf 1999]. Para evitar que isso aconteça é importante utilizar hosts confiáveis ou *localhost*, pois permite apenas conexões da própria máquina, tornando a conexão mais segura.

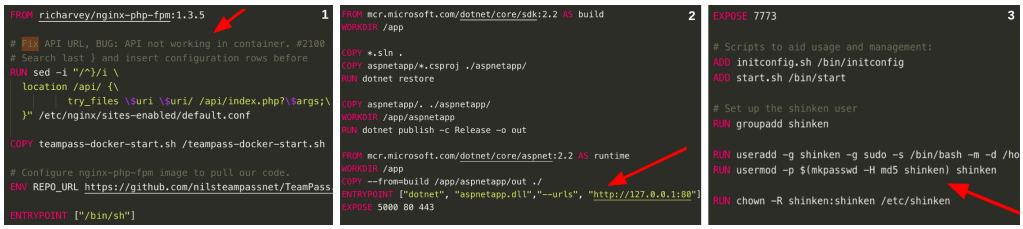

Figura 2. Exemplos de *Dockerfiles* com comentário suspeito, uso de *HTTP* sem *TLS* e uso de algoritmos com criptografia fraca.

Comentário suspeito: Na imagem 1 da Figura 2 há um comentário com informações sobre a presença de defeitos, funcionalidade ausente ou fraqueza do sistema. Como os *scripts* IaC guardam informações e executam dados sensíveis, esta exposição pode indicar uma falha de segurança no sistema, estes comentários em *scripts* IaC devem ser evitados.

Uso de *HTTP* sem *TLS*: Na imagem 2 da Figura 2, executa-se uma aplicação de banco de dados utilizando o protocolo *HTTP* sem uso de *TLS*, tornando a comunicação entre duas entidades suscetível a ataques. O *HTTPS* protege a privacidade e a integridade dos dados trocados.

Uso de algoritmos de criptografia fraca: Na imagem 3 da Figura 2, cria-se uma senha aleatória para um usuário, onde é criptografada com o algoritmo de *hash MD5*. O uso de algoritmos de criptografia fracos torna a aplicação suscetível a ataques, pois em alguns casos é possível descriptá-los. Para maior segurança recomenda-se utilizar senhas difíceis com uma criptografia forte em variáveis de ambiente.

Através dessas análises respondemos **RQ1**, constatando que todos os *Security Smells* propostos para o *Puppet*, por Rahman et al. [Rahman et al. 2019], também se aplicam ao *Docker*, mostrando que as definições desses *Security Smells* são gerais e podem ser aplicadas a diferentes tecnologias e ferramentas.

3.2. Identificar e propor novos *Security Smells* para o *Docker*

Durante a análise dos *Dockerfiles* coletados foi investigado fatores que podem levar a vulnerabilidades, utilizando como base as definições do CWE, documentações oficiais do *Docker* e artigos de segurança escritos pela comunidade. Com isso, conseguimos propor dois novos *Security Smells*, vistos a seguir.

Figura 3. Exemplos de *Dockerfiles* usando imagens não oficiais e fornecendo permissão total ao sistema de arquivos.

Uso de imagens *Docker* não oficiais: As imagens oficiais são mantidas pela empresa *Docker* e disponibilizadas no *dockerhub*. As imagens não oficiais são mantidas pelos usuários que as criaram [Gomes 2019]. O uso de imagens *Docker* não oficiais ou desconhecidas podem levar a uma vulnerabilidade de segurança, pois não se sabe a autenticidade ou se foram adulteradas, estes arquivos podem conter informações sensíveis ou comandos que podem ocasionar falhas. A imagem 1 da Figura 3 mostra um exemplo desta aplicação.

Permissão total ao sistema de arquivos: De acordo com o *CWE-732*: "Atribuição de Permissão Incorreta para Recurso Crítico", configuração de permissões que fornece acesso total, podendo levar à exposição de informações confidenciais ou à modificações indesejadas. A imagem 2 da Figura 3 mostra um *Dockerfile*, no qual é utilizado o comando *chmod 777* que dá permissão total (leitura, escrita e execução) ao diretório da aplicação. O ideal é que seja dada apenas permissão do que for utilizado.

Com esses resultados, conseguimos responder nossa pergunta de pesquisa **RQ2**, mostrando ser possível propor novos *Security Smells* para *scripts* IaC, no contexto *Docker*.

3.3. Ocorrência de *Security Smells* em *Dockerfiles*

O gráfico da Figura 4 mostra os resultados das ocorrências de cada *Security Smell* discutido neste trabalho.

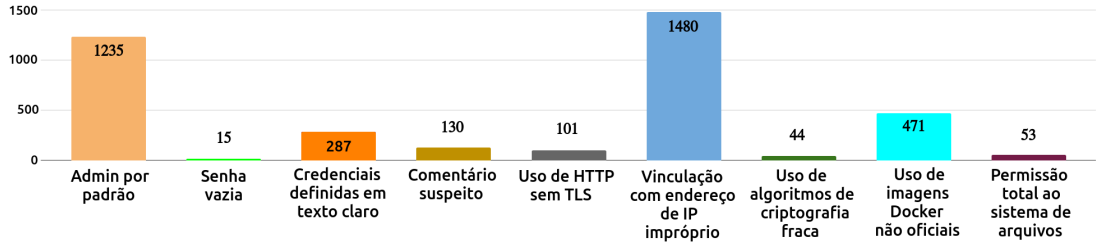

Figura 4. Ocorrências de *Security Smells* em *Dockerfile*

Detectamos 3.816 ocorrências de *Security Smells* encontradas em 1500 *Dockerfiles* analisados. Os três Security Smells mais frequentes foram "Vinculação com endereço de IP impróprio" (1480), "Admin por padrão" (1235) e "Uso de imagens *Docker* não oficiais" (471). Esses resultados respondem nossa questão de pesquisa **RQ3**.

4. Conclusão e Trabalhos Futuros

Os *Security Smells* são padrões de codificação recorrentes em *scripts* IaC, são indicativos de falhas de segurança que podem potencialmente levar a violações no sistema. Os

Security Smells propostos e suas definições irão ajudar profissionais de TI a entenderem riscos e a evitarem práticas de programação em scripts IaC que podem levar a falhas de segurança.

Dos 1500 *Dockerfiles* analisados, foi observado que todos os *Security Smells* propostos para o *Puppet* também se aplicam ao contexto do *Docker*. Com essa análise também foi possível identificar e propor 2 novos *Security Smells* para o *Docker*, além dos 7 já propostos por Rahman et al. [Rahman et al. 2019]. Além disso, foram identificadas 3816 ocorrências de *Security Smells* dentre os 9 discutidos neste artigo.

Em trabalhos futuros pretende-se criar uma ferramenta para detecção automática de *Security Smells* em *Docker*, permitindo analisar uma amostra maior; além de identificar e propor novos *Security Smells*.

Referências

Chang, W. (2017). Ibm data science experience: Whole-cluster privilege escalation disclosure. Acessado: 06/06/2019.

Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., and Gall, H. C. (2017). An empirical analysis of the docker container ecosystem on github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 323–333, Buenos Aires, Argentina. IEEE.

Duarte, A. and Antunes, N. (2018). An empirical study of docker vulnerabilities and of static code analysis applicability. In *2018 8th Latin-American Symposium on Dependable Computing (LADC)*, pages 27–36, Foz do Iguaçu - PR. LADC.

Fowler, M. (2016). Infrastructure as code. `https://martinfowler.com/bliki/InfrastructureAsCode.html`. Acessado: 25/05/2019.

Ghafari, M., Gadient, P., and Nierstrasz, O. (2017). Security smells in android. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 121–130, Shanghai, China. IEEE.

Gomes, R. (2019). *Docker para Desenvolvedores*. Leanpub, Salvador, Bahia, 1st edition.

MITRE (2008). Cwe-common weakness enumeration. `https://cwe.mitre.org/index.html`. Online: acessado 10-06-2019.

Mutaf, P. (1999). Defending against a denial-of-service attack on tcp. In *International Symposium on Recent Advances in Intrusion Detection (RAID)*.

Pahl, C., Brogi, A., Soldani, J., and Jamshidi, P. (2017). Cloud container technologies: a state-of-the-art review.

Ragan, S. (2016). Mongodb configuration error exposed 93 million mexican voter records. Acessado: 06/06/2019.

Rahman, A., Parnin, C., and Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. In *Proceedings of the 41st International Conference on Software Engineering*, Montreal, QC, Canada. ACM.