# Levantamento de desempenho de um kernel a nível de aplicação na execução de códigos não confiáveis em containers

## *Performance assessment of an application layer kernel for executing untrusted code in containers*

Heloísa Gabriely Faria Preuss[*], Daniel Cavalcanti Jeronymo[†]

**RESUMO**

A gamificação tem sido empregada como forma de motivar iniciantes e promover a prática de programação. Para tal, é necessário dispor de ambientes capazes de compilar e executar códigos enviados por usuários, entretanto, a execução de código desconhecido e potencialmente malicioso forma um possível vetor de ataque ao servidor no qual os ambientes estão hospedados. Neste trabalho é explorado o uso de *containers Docker* para virtualização e isolamento de códigos não confiáveis. Uma camada adicional de segurança é explorada ao utilizar-se o *gVisor* como uma interface anterior ao *kernel*, conhecida como *kernel* a nível de aplicação. Um servidor web Eclipse *Jetty* foi utilizado para hospedar uma aplicação desenvolvida utilizando a *framework Wt Witty*, permitindo ao usuário enviar códigos e visualizar o resultado de seu programa. Foram realizadas 30 execuções de códigos não confiáveis diretamente no *kernel* do *container* e também com o *gVisor* como *kernel* intermediário. O tempo médio de execução foi próximo de seis segundos e sem diferença estatística de acordo com o teste de *Wilcoxon*, indicando que o *gVisor* não impacta significativamente o tempo de execução ao oferecer uma camada adicional de segurança no ambiente de *sandboxing*.

**Palavras-chave:** Aprendizado de Programação, Código Não Confiável, *Sandboxing*, gVisor.

**ABSTRACT**

Gamification has been used as a way to motivate beginners and promote programming practice. For this, it is necessary to have environments capable of compiling and executing codes sent by users, however, the execution of unknown and potentially malicious code forms a possible attack vector to the server in which the environments are hosted. This work explores the use of Docker *containers* for virtualization and isolation of untrusted code. An additional layer of security is exploited by using gVisor as a pre-kernel interface known as the application-level kernel. An Eclipse Jetty web server was used to host an application developed using the Wt Witty *framework*, allowing the user to send codes and visualize the results of their program. There were 30 executions of untrusted code directly in the *container* kernel and also with gVisor as intermediate kernel. The average execution time was close to six seconds and with no statistical difference according to the Wilcoxon test, indicating that gVisor does not significantly impact the execution time by providing an additional layer of security in the *sandboxing* environment.

**Keywords:** Programming Learning, Untrusted Code, *Sandboxing*, gVisor.

* Engenharia da Computação, Universidade Tecnológica Federal do Paraná, Toledo, Paraná, Brasil; heloisapreuss@hotmail.com
† Universidade Tecnológica Federal do Paraná, Campus Toledo; danielc@utfpr.edu.br

## 1 INTRODUÇÃO

A gamificação permite ao estudante controlar seu aprendizado enquanto fomenta a motivação de estudo já que o ensino de programação a alunos iniciantes é considerado difícil por abordar entidades abstratas e complexas tais como algoritmos e programas de computadores (OLSSON, MOZELIUS, COLLIN, 2015). O conceito trata de aplicar mecânicas e dinâmicas de jogos ao aprendizado, utilizando princípios adaptados da Teoria da Auto Determinação (SDT, *Self Determination Theory*), tais como as necessidades de interação e conexão com outros participantes, de obter competência em um determinado problema e de autonomia (**FOTARIS, MASTORAS, LEINFELLNER, ROSUNALLY, 2016**). A gamificação explora o controle de atenção do usuário e de acordo com ***Fotaris et al. (2016)*** as motivações intrínsecas, relacionadas ao desejo ou prazer na tarefa, e extrínsecas, relacionadas a fatores externos de recompensa pelo desempenho na atividade, afetam diretamente a motivação e compromisso dos estudantes.

De **Campos** e **Ferreira (2004)** introduziram no Brasil o primeiro sistema amplamente utilizado na correção automática de códigos e empregado pela SBC (Sociedade Brasileira de Informática) nas Maratonas de Programação. O BOCA utiliza internamente um módulo chamado *safeexec* desenvolvido pelos autores para rodar o código em um ambiente protegido, conhecido como *sandbox*, utilizando mecanismos do Linux como *jails* e *chroot*, para limitações de uso de disco, e *rlimits* (*Resource Limits*), para controlar tempo de execução e uso de memória. A *sandbox* é um mecanismo de segurança que permite executar códigos não confiáveis e impedir efeitos de códigos maliciosos, tais como danos ao servidor ou perda de seu controle.

O URI (Universidade Regional Integrada) Online Judge, ferramenta disponibilizada na web e utilizada amplamente por estudantes brasileiros no aprendizado de programação. A ferramenta foi criada com o propósito de automatizar a correção de atividades de alunos após a observação pelos autores das necessidades de novas ferramentas para complementar as aulas teóricos e disponibilizar problemas práticos onde a teoria pudesse ser aplicada (**BEZ, TONIN, RODEGHERI, 2014**). A gamificação foi implementada em versões posteriores na forma de um sistema de recompensas (*badges*) visando desafiar e encorajar os alunos a utilizarem a plataforma com maior frequência.

**Maguire, Maguire e Kelly (2018)** descrevem a ferramenta *HackerRank*, utilizada pela indústria para recrutar programadores e também em competições de programação, relatam os autores que o uso misto da plataforma combina as expectativas do aprendizado de programação com as expectativas da indústria. Os desafios apresentados nesta plataforma contemplam áreas de bancos de dados, estruturas de dados, matemática e algoritmos, para citar algumas, que podem ser resolvidos em Java, C, C++, Python, além de outras.

**Iancu (2019)** exemplifica as ferramentas *Codeacademy* e *CodeSignal* no contexto de gamificação como tendo efeitos positivos na retenção e compromisso de jogadores. O emprego de técnicas tipicamente utilizadas em jogos tem como resultado esperado promover um comportamento de aprendizado além de melhorar as habilidades pré-existentes e prender a atenção do estudante, precisando este de motivações extrínsecas e a capacidade de lidar com desafios **(IANCU, 2019).**

Neste trabalho é apresentado o protótipo de um sistema para correção automática de códigos sendo avaliado o uso do *gVisor* como *kernel* a nível de aplicação. O sistema foi construído utilizando a implementação em Java da linguagem Python, conhecida como *Jython*. O servidor Web foi implementado através do *Eclipse Jetty*, hospedando um *servlet* montado pela *framework Wt Witty*. O mecanismo de *sandbox* para executar código não confiável foi construído em Python e faz uso de virtualização a nível de sistema operacional através de *container Docker* configurado para utilizar o *gVisor* como *runtime*. O

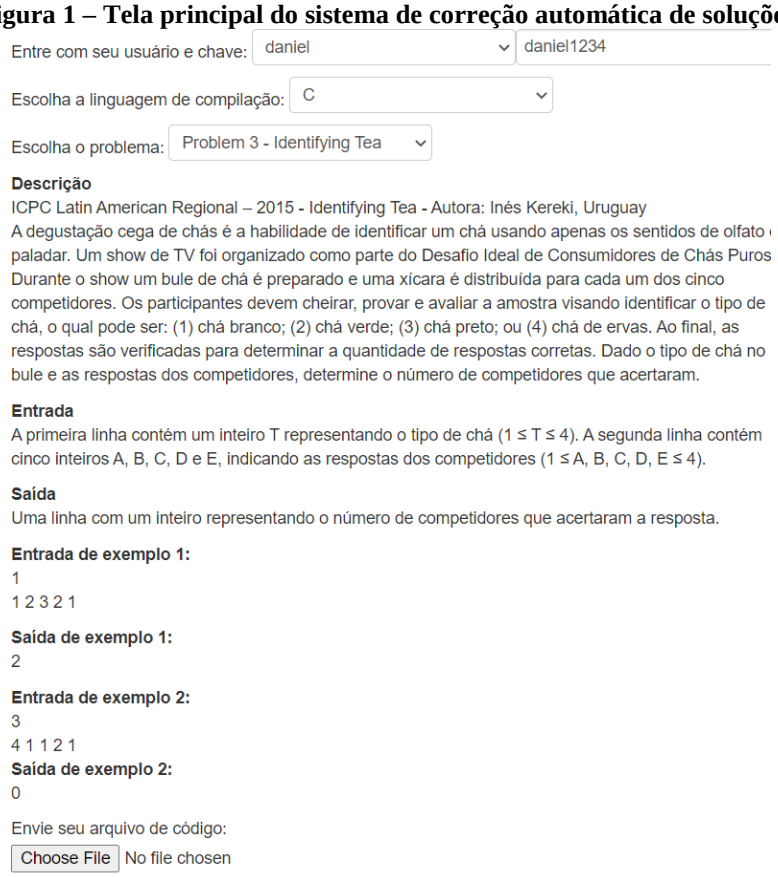
objetivo é avaliar se a segurança adicional fornecida pelo *gVisor* acarreta em perdas de desempenho da *sandbox*, medido em tempo de execução do código não confiável.

O restante deste trabalho está organizado da seguinte maneira. Na seção 2 é apresentada a metodologia e tecnologias utilizadas neste trabalho. Na seção 3 são apresentados os resultados e finalmente conclui-se com considerações finais apresentadas na seção 4.

## 2 METODOLOGIA

O sistema para correção automática de códigos foi construído em duas partes separadas as quais fazem uso de comunicação entre processos. A primeira parte consiste num servidor Web o qual faz uso da máquina virtual Java e do interpretador *Jython*, uma implementação da linguagem Python em Java, para hospedar um serviço Eclipse *Jetty*. As funcionalidades Web foram implementadas através da *framework Wt Witty* e seu módulo Java, o *jWt*. Nesta primeira versão o estudante pode enviar soluções nas linguagens C, C++ e Python. Na Figura 1 é apresentada a tela principal, exibindo a descrição de um problema com seus exemplos de entrada e saída.

**Figura 1 – Tela principal do sistema de correção automática de soluções.**




**Fonte: Autoria própria (2021)**

Após o envio do código pelo usuário, o sistema inicia a *sandbox* onde ocorre a execução protegida do código não confiável. A *sandbox* permite limitação de recursos de memória, processador, rede e arquivos.

Isso é realizado pela criação de um novo processo *shell* do sistema operacional onde um script em Python recebe o arquivo do usuário e linguagem escolhida como argumentos. É iniciado um *container* Docker preparado para a linguagem escolhida, o arquivo de código do usuário é transferido para este ambiente e são realizados os comandos de compilação e execução. Uma lista de arquivos padronizados com entradas e saídas do problema é utilizada para verificar os resultados do código. Tanto em caso de erro quanto em caso de sucesso as mensagens são transferidas em modo texto formatado no padrão JSON (*JavaScript Object Notation*) do processo *shell* para o processo do servidor Web.

Todas as tecnologias utilizadas foram escolhidas pelas suas características de código aberto, segurança e suporte a múltiplos sistemas operacionais. As máquinas virtuais do *Java* e *Python* fornecem seguranças adicionais ao criar novas camadas pelas quais um ataque deve passar.

O uso do interpretador *Jython* possibilita combinar *Java* e *Python*, permitindo o uso do servidor Eclipse *Jetty* com a biblioteca *Wt Witty*. A aplicação Web foi desenvolvida em *Python* fazendo uso da jWt, acelerando o processo de desenvolvimento e prototipação. A *Wt Witty* é uma biblioteca criada pela *Emweb* para desenvolvimento de aplicações Web, ou Web GUI (*Graphical User Interface*). Compartilha da filosofia Qt de desenvolvimento, onde todos os componentes são *widgets* que podem ser desenvolvidos individualmente e posteriormente combinados. Criada originalmente em C++, a biblioteca Wt disponibiliza uma variante compilada em Java, a jWt. A segurança dessa biblioteca vem de mecanismos internos construídos com o propósito de evitar ataques comuns na web, como sequestro de sessão, CSRF (*Cross Site Request Forgery*) e XSS (*Cross Site Scripting*). A biblioteca *Wt Witty* cria automaticamente os códigos HTML/XHTML, JavaScript, CGI, ou AJAX necessários, sem que o desenvolvedor precise dominar essas tecnologias, sendo necessário apenas desenvolver a aplicação como qualquer outra GUI, similar ao Qt.

O Docker foi escolhido por ser uma *Container Engine*, capaz de realizar virtualização a nível de sistema operacional, ao contrário da virtualização a nível de processador (**YOUNG *et al.*, 2019**). Um *container* neste contexto é um subprocesso com limitações impostas através de funcionalidades de isolação de processos modernas e nativas do Linux, como *cgroups* para limitar recursos de memória e processador, e *namespaces* para isolar a aplicação do sistema operacional, evitando acesso a lista de processos, sistemas de arquivos e dispositivos do sistema, como a rede. O uso de *Docker* é atrativo também pois há um sistema de arquivos temporário, logo subprodutos do processo de *sandboxing*, como o executável gerado pela compilação, são automaticamente removidos quando o *container* é finalizado.

O *gVisor* é o mecanismo de segurança utilizado na GCF (*Google Cloud Functions*) **(YOUNG *et al.*, 2019)** e reduz a superfície de ataque do *kernel* ao disponibilizar um *kernel* transparente a nível de aplicação. **Young et al. (2019)** descrevem a arquitetura do *gVisor* e explicam que é construída de maneira que não é possível comprometer o sistema inteiro através da exploração de falhas em um único subsistema. O *gVisor* implementa o padrão OCI (*Open Container Initiative*) e pode ser utilizado no *Docker* em substituição ao mecanismo padrão de criação e execução de *containers*, chamado de *runtime*.

Ao separar a *sandbox* da aplicação Web tem-se uma camada adicional de segurança em que o comprometimento da aplicação Web não afeta o sistema da *sandbox* e vice-versa. A comunicação entre processos através de mensagens formatadas em JSON garante que as mensagens devem ser propriamente formatadas e, portanto, mensagens corrompidas por uma violação na *sandbox* seriam descartadas e tratadas como erro.

## 3 RESULTADOS

Foram realizadas 30 execuções de códigos não confiáveis utilizando a *runtime* padrão do *Docker*, a *runc*, e 30 execuções com a *runtime* do gVisor, a *runsc*. Os resultados de tempo de execução, com valores mínimos, máximos, médias e medianas são apresentados na Tab. 1. Os resultados apresentados para a *runsc* são mais rápidos em todas as métricas, contrariando a expectativa de que o uso do *gVisor* tornaria o mecanismo de *sandbox* mais lento. Entretanto, por tratar-se de um processo estocástico de execução é necessário avaliar a significância estatística destes resultados antes que possa ser afirmado diferença entre os resultados.

Foi realizado o teste não-paramétrico de *Wilcoxon* no conjunto das populações de resultados para testar a hipótese de que as amostras são tiradas da mesma distribuição com significância estatística de 1%. O valor-p obtido pelo teste foi de 0,57, indicando que os resultados não são significativamente diferentes. O resultado de significância estatística indica que os resultados mais rápidos observados com o *gVisor* são fruto de variação estatística e não de ganho de desempenho já que não são estatisticamente diferentes.

**Tabela 1 – Tempos de execução.**

| ***Runtime*** | **Mínimo** | **Máximo** | **Média** | **Mediana** |
|---|---|---|---|---|
| *runc* | 3,49 | 8,56 | 6,02 | 6,12 |
| *runsc* | 3,46 | 7,87 | 5,91 | 6,11 |

**Fonte: Autoria própria (2021).**

## 4 CONCLUSÃO

Neste trabalho foi apresentado o protótipo de um ambiente de avaliação automática de códigos a ser futuramente utilizado nas aulas de Fundamentos de Programação do curso de Engenharia de Computação da UTFPR Campus Toledo. O ambiente consiste num servidor Web em que estudantes podem enviar códigos para resolver problemas e num mecanismo de *sandbox* para execução segura de código não confiável. Foi avaliado o uso da *runtime* padrão do *Docker* e uma *runtime* criada pela Google para maior segurança em *containers*, o gVisor.

Embora *containers* apresentem mecanismos de virtualização a nível de sistema operacional, isso não os torna inteiramente seguros já que o *kernel* do sistema operacional ainda está exposto. **Young et al. (2019)** explicam que recursos do sistema operacional que não podem ser completamente virtualizados são vulneráveis e falhas do *kernel* ainda podem ser exploradas já que a superfície de ataque é ampla e consiste em mais de 300 funções. O *gVisor* atua como um *kernel* a nível de aplicação, agindo como uma camada a mais e reduzindo a superfície de ataque ao indisponibilizar chamadas de sistema comumente exploradas **(YOUNG *et al.*, 2019)**, entretanto ao custo de maior sobrecarga computacional.

Foram realizados 30 testes de execução de código não confiável utilizando a *runtime* padrão do *Docker*, a *runc*, e mais 30 testes utilizando a *runtime* do *gVisor*, a *runsc*. Não foi observada diferença estatisticamente significativa de execução entre as duas *runtimes*. Os resultados indicam que o ganho de segurança e a sobrecarga criada pelo uso do gVisor não impactam significativamente a implementação da *sandbox*. **Young et al. (2019)** observaram diferenças em tempos de execução, todavia, foram executados testes muito maiores e mais complexos que o deste trabalho.

**AGRADECIMENTOS**

Agradeço ao professor Daniel por sua paciência e orientação e ao CNPq por seu auxílio financeiro.

## REFERÊNCIAS

BEZ, J.L., TONIN, N.A. AND RODEGHERI, P.R. **URI Online Judge Academic: A tool for algorithms and programming classes**, 2014 9th International Conference on Computer Science & Education, 2014, pp. 149-152, 2014.

DE CAMPOS, C. P., FERREIRA, C. E. **BOCA: um sistema de apoio a competições de programação (BOCA: A Support System for Programming Contests).** In: Workshop de Educacao em Computacao (Brazilian Workshop on Education in Computing), 2004, Salvador. Anais do Congresso da SBC, 2004.

FOTARIS, P., MASTORAS, T., LEINFELLNER, R. AND ROSUNALLY, Y. **Climbing up the leaderboard: An empirical study of applying gamification techniques to a computer programming class**, Electronic Journal of e-learning, v. 14, n. 2, pp. 94-110, 2016.

Iancu, B. **Gamification Applied in Computer Science Education: A Preliminary Approach.** Academy of Economic Studies. Economy Informatics, v. 19, n. 1, pp. 52-58, 2019.

MAGUIRE, P., MAGUIRE, R. AND KELLY, R., **Using automatic machine assessment to teach computer programming, Computer Science Education**, v. 27, n. 3-4, pp. 197-214, 2018.

OLSSON, M., MOZELIUS, P. AND COLLIN, J. **Visualisation and Gamification of e- Learning and Programming Education**, Electronic journal of e-learning, v. 13, n. 6, pp 452-465, 2015.

YOUNG, E.G., ZHU, P., CARAZA-HARTER, T., ARPACI-DUSSEAU, A.C. AND ARPACI-DUSSEAU, R.H. **The true cost of containing: A gVisor case study,** 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), 2019.