Vulnerability Detectors

Ana João Alves
pg57505

Gonçalo Brandão
pg57874

9 de março de 2025

Resumo

Este artigo tem como objetivo estudar ferramentas de detecção de vulnerabilidades em software. Para isso, exploramos as diferentes abordagens disponíveis, analisamos seu funcionamento e comparamos as vantagens e limitações de cada uma, além de avaliar as principais ferramentas disponíveis no mercado.

As abordagens analisadas — SAST (Static Application Security Testing), DAST (Dynamic Application Security Testing) e IAST (Interactive Application Security Testing) — apresentam diferentes níveis de eficácia, dependendo do contexto e dos desafios tecnológicos enfrentados. Ao longo do artigo, discutimos os conceitos essenciais para a compreensão desse tema, fornecendo uma visão abrangente sobre a segurança de aplicações.

Por fim, reforçamos a importância do desenvolvimento de software robusto e seguro, destacando boas práticas e o impacto da utilização dessas ferramentas na mitigação de riscos.

1 Introdução

Vulnerability Detectors são ferramentas ou sistemas que identificam falhas de segurança em software, hardware ou redes. Estas vulnerabilidades podem ser exploradas por hackers para comprometer sistemas, roubar dados ou causar danos. Os detetores de vulnerabilidades analisam código, configurações e comportamento dos sistemas para encontrar potenciais riscos de segurança antes que sejam explorados.

Ao identificar e corrigir proativamente as vulnerabilidades, as organizações podem reforçar a sua postura de segurança online e mitigar os riscos de violações de dados, perdas financeiras e danos à reputação. Além disso, a deteção de vulnerabilidades permite que as empresas cumpram regulamentações e normas do setor, demonstrando o seu compromisso com a proteção de informações sensíveis e a confiança dos seus clientes. Perante um cenário de ameaças em constante evolução e vetores de ataque cada vez mais sofisticados, investir em medidas robustas de deteção de vulnerabilidades é essencial para manter-se um passo à frente das ameaças cibernéticas e garantir a resiliência das plataformas e serviços baseados na web.

2 Ferramentas de Vulenrability Detectors

Estas ferramentas utilizam diferentes técnicas para analisar potenciais vulnerabilidades que podem ser exploradas por atacantes, sendo as principais técnicas:

- SAST - Static Application Security Testing
- DAST - Dynamic Application Security Testing
- IAST - Interactive Application Security Testing

A combinação destas várias abordagens aumenta a eficácia da deteção e ajuda as organizações a reforçar a sua segurança.

3 SAST - Static Application Security Testing

SAST é uma abordagem de detenção de vulnerabilidades de software focada na análise do código-fonte. Através de diferentes técnicas de análise como análise de dependências ou análise de fluxo de dados, esta abordagem permite identificar possíveis vulnerabilidades em fases anteriores à execução da aplicação.

3.1 Funcionamento

O SAST verifica sistematicamente o código tendo em conta um conjunto de regras ou condições predefinidas que dizem respeito à segurança das práticas de codificação. Na detecção de uma possível fraqueza, ele sinaliza a área no código onde encontrou o problema, dando aos desenvolvedores a oportunidade de remediá-lo antes da implantação. Os passos no processo SAST incluem:

- **Etapa 1: Código-fonte**

  As ferramentas SAST começam por analisar o código fonte, o código byte ou o código binário para criar uma Árvore de Sintaxe Abstrata (AST). A AST representa a estrutura do código e os seus vários componentes, tais como funções, loops, declarações condicionais e variáveis. Esta análise tem como objetivo de identificar vulnerabilidades, como problemas a nível da sintaxe do código, declaração de variáveis não inicializadas ou ainda o uso de livrarias *Deprecated*.

- **Etapa 2: Controlo de fluxo e fluxo de dados**

  A ferramenta SAST realiza a análise de controlo de fluxo e análise de fluxo de dados para entender o comportamento da aplicação. A análise do controlo de fluxo examina a estrutura do código-fonte do software, identificando possíveis vulnerabilidades relacionadas com estruturas de controlo de fluxo, como loops infinitos, operadores condicionais ou ainda vulnerabilidade de buffer overflow. A análise do fluxo de dados examina a forma como os dados circulam entre vários níveis do software. Isso ajuda na identificação de tratamento de dados inseguros, como injeções de SQL ou vulnerabilidades XSS.

- **Etapa 3: Arquitetura**

  Esta técnica de análise examina a estrutura geral do software, identificando possíveis vulnerabilidades relacionadas com a arquitetura do sistema, como problemas de autenticação, validação de entrada de dados e vulnerabilidades de configuração.

- **Etapa 4: Dependências**

  Esta técnica de análise examina as bibliotecas e dependências utilizadas pelo software, identificando possíveis vulnerabilidades relacionadas com falhas de segurança nessas bibliotecas (Deprecated).

- **Etapa 5: Regras e políticas de segurança**

  As ferramentas SAST contêm um conjunto de regras e políticas de segurança predefinidas que são usadas para analisar o código para possíveis vulnerabilidades. Essas regras são baseadas em padrões do setor, vulnerabilidades conhecidas e padrões de segurança, como OWASP Top Ten ou CWE / SANS Top 25.

- **Passo 6: Combinação de padrões e análise semântica**

  A ferramenta SAST aplica técnicas de correspondência de padrões e análise semântica para identificar componentes de código que correspondem às regras e políticas de segurança predefinidas. Esse processo ajuda a detectar práticas de codificação inseguras, como algoritmos de criptografia fracos ou senhas codificadas .

3.2 Vantagens

O uso de ferramentas SAST, permite identificar vulnerabilidades desde a fase inicial de desenvolvimento do software, permitindo corrigir e identificar pontos fracos do software antes do lançamento ou conclusão do mesmo. Com o uso de ferramentas de análise em fases iniciais de desenvolvimento, reduzimos o risco de aparecimento de falhas em pontos mais complexos do sistema, que requeriam mais tempo, custos e esforço para tratar.

3.3 Limitações

O uso de uma abordagem SAST para a deteção de vulnerabilidades pode se revelar bastante ineficiente. Como a análise é limitada ao nível do código-fonte do software, a identificação de problemas torna-se ineficiente. Listam-se algumas limitações:

- **Limitações na deteção de vulnerabilidades:** A análise estática de código apenas é eficaz na identificação de vulnerabilidades conhecidas. Além disso, a análise estática não permite analisar problemas de segurança em bibliotecas e scripts de terceiros.
- **Limitações na avaliação da segurança:** A abordagem SAST não permite analisar a execução do software, nunca avaliando o comportamento do mesmo em tempo-real. Esta abordagem não permite ainda analisar a segurança de camadas diferentes do software como, por exemplo, a base de dados.

3.4 Ferramentas SAST

Critério	**SonarQube**	**Checkmarx**	**Fortify**
Foco Principal	Qualidade de código e segurança básica	Segurança de código-fonte e vulnerabilidades críticas	Segurança avançada e conformidade regulatória
Resultados	Métricas de qualidade e segurança, gráficos interativos	Relatórios detalhados com risco e contexto	Relatórios detalhados com foco em segurança e conformidade
Integração com CI/CD	Integração fácil com Jenkins, GitLab, etc.	Integração com Jenkins, Azure DevOps, etc.	Integração com Jenkins, Azure DevOps, IDEs, etc.
Feedback	Feedback de qualidade e segurança em tempo real	Feedback imediato de vulnerabilidades com sugestões detalhadas	Feedback de segurança com foco em riscos e correções
Implementação	Fácil de usar, especialmente para pequenas equipa	Exige mais configuração, mas oferece poderosos recursos de segurança	Requer mais configuração e infraestrutura para grandes empresas
Preço	Versão gratuita (open-source), versão paga para recursos avançados	Licenciamento pago	Licenciamento pago

Tabela 1: Comparação entre SonarQube, Checkmarx e Fortify

SonarQube é ideal para pequenas equipas que desejam melhorar a qualidade do código e garantir a segurança básica sem custos elevados. Checkmarx é uma excelente escolha para empresas que priorizam segurança, oferecendo análises detalhadas de vulnerabilidades em código, com foco na detecção de falhas críticas. Fortify é adequado para grandes empresas que necessitam de uma análise profunda de segurança, com relatórios de conformidade e suporte para grandes volumes de código e exigências regulatórias.

4 DAST - Dynamic Application Security Testing

DAST é o processo de análise de uma aplicação web através do front-end para encontrar vulnerabilidades via ataques simulados. Este tipo de abordagem avalia a aplicação a partir da sua execução, atacando uma aplicação como um utilizador malicioso o faria. Após um scanner DAST realizar estes ataques, procura resultados que não fazem parte do conjunto de resultados esperados e identifica vulnerabilidades de segurança.

4.1 Funcionamento

O DAST realiza uma análise dinâmica da aplicação enquanto ela está em funcionamento, sem necessidade de acesso ao código-fonte. Ele simula ataques cibernéticos para identificar vulnerabilidades que podem ser exploradas enquanto a aplicação está a ser executada. O processo do DAST inclui os seguintes passos:

- **Etapa 1: Análise da Aplicação em Execução**

  As ferramentas DAST começam a análise monitorando a aplicação em execução, geralmente em um ambiente de teste. Elas interagem com a interface da aplicação (geralmente uma interface web ou API) e realizam um teste interativo para identificar falhas e vulnerabilidades enquanto a aplicação está em funcionamento. Esse processo inclui a simulação de ataques cibernéticos, como injeção de SQL, Cross-Site Scripting (XSS) e ataques de falsificação de solicitações entre sites (CSRF).

- **Etapa 2: Mapeamento e Testes de Entrada**

  Na segunda etapa, as ferramentas DAST mapeiam os pontos de entrada da aplicação, como formulários, URLs, cabeçalhos HTTP e campos de entrada do utilizador. Durante essa análise, são realizadas verificações para detectar falhas que podem ser exploradas, como entradas não validadas, falta de autenticação e controlo de acesso, ou permissões inadequadas para dados sensíveis.

- **Etapa 3: Exploração de Vulnerabilidades**

  Após identificar os pontos de entrada, as ferramentas DAST tentam explorar as vulnerabilidades, como a injeção de código malicioso ou ataques de negação de serviço (DoS). A exploração é realizada sem causar impacto nos dados reais, mas é projetada para identificar pontos críticos de falhas que poderiam ser explorados por atacantes.

- **Etapa 4: Validação de Respostas**

  O DAST valida a resposta do sistema aos ataques simulados. Se a aplicação falhar ou responder de maneira inesperada (por exemplo, exibindo dados sensíveis ou permitindo ações não autorizadas), a ferramenta DAST regista esses problemas como vulnerabilidades. Além disso, o DAST verifica a eficácia dos controles de segurança, como criptografia e proteção de dados.

- **Etapa 5: Análise de Configuração e Implementação**

  Nessa etapa, o DAST verifica as configurações de segurança e as práticas de implementação da aplicação, como a presença de headers de segurança (X-Content-Type-Options, X-XSS-Protection), a política de CORS, e outras configurações que possam impactar a segurança da aplicação em execução.

- **Etapa 6: Relatórios e Feedback em Tempo Real**

  Após concluir a análise, o DAST gera relatórios com as vulnerabilidades encontradas e fornece feedback em tempo real para os desenvolvedores. Os relatórios incluem detalhes sobre as falhas de segurança, o nível de gravidade, e recomendações de mitigação para cada vulnerabilidade detectada.

4.2 Vantagens

Entre as vantagens do DAST, destaca-se o facto de ser independente da aplicação, o que significa que pode ser utilizado em qualquer tecnologia ou linguagem. Além disso, essa metodologia identifica vulnerabilidades exploráveis imediatamente, ajudando a encontrar falhas reais que poderiam ser utilizadas por atacantes. Outro ponto positivo é que não requer acesso ao código-fonte, permitindo a análise de segurança sem precisar de modificar ou rever o código da aplicação. O DAST também simula ataques reais, ajudando a validar a segurança do sistema sob condições do mundo real, e pode ser aplicado em ambientes de produção, testando aplicativos já implementadas sem impacto direto no código.

4.3 Limitações

O DAST também possui limitações que devem ser levados em conta. Uma de suas limitações é que não identifica a localização exata da vulnerabilidade no código, exigindo análise adicional para corrigir as falhas encontradas. Além disso, o conhecimento de segurança é necessário para interpretar os relatórios, pois podem conter falsos positivos ou exigir uma avaliação mais detalhada. Outra desvantagem é que os testes podem ser demorados, especialmente em aplicações grandes e complexas. O DAST também é limitado na detecção de vulnerabilidades lógicas, pois pode não encontrar falhas relacionadas ao fluxo de negócios da aplicação. Por fim, diferente do SAST (Static Application Security Testing), não detecta problemas no código-fonte diretamente, o que significa que algumas vulnerabilidades podem passar despercebidas até a fase de execução.

4.4 Ferramentas DAST

Ferramenta	**Foco Principal**	**Resultados (Precisão)**	**Integração com CI/CD**	**Feedback (Usabilidade e Relatórios)**	**Implementação**
OWASP ZAP	Testes de segurança web manuais e automatizados	Boa, pode gerar falsos positivos	Sim, via scripts e API	Interface técnica, mas bem documentado	Open-source, instalação local
Burp Suite Pro	Testes manuais avançados e automação parcial	Muito precisa, ideal para testes manuais	Integração limitada (foco em testes interativos)	Relatórios detalhados e poderosos	Fácil de instalar, exige conhecimento técnico
Acunetix	Automação de testes de segurança para DevSecOps	Muito alta, detecção precisa de vulnerabilidades	Sim, integração nativa (Jenkins, GitLab, etc.)	Interface intuitiva e relatórios detalhados	Simples, com varredura automatizada
Netsparker (Invicti)	Automação com baixa taxa de falsos positivos	Extremamente precisa (verificação automática)	Sim, fácil integração com DevOps	Feedback claro e relatórios fáceis de entender	Baseado em nuvem e on-premises
HCL AppScan	Testes em aplicações corporativas (web e mobile)	Alta precisão, cobrindo aplicações complexas	Sim, integração com ferramentas corporativas	Relatórios detalhados, mas requer aprendizado	Instalação mais complexa, voltada para empresas
Detectify	Automação na nuvem, atualizada por especialistas	Boa, baseada em crowdsourcing	Sim, integração com CI/CD	Feedback rápido e amigável	Simples, totalmente baseada em nuvem

Tabela 2: Comparação das principais ferramentas DAST

5 IAST - Interactive Application Security Testing

IAST é uma abordagem de deteção de vulnerabilidades de software que se concentra na análise interativa da aplicação combinada com análise dinâmica durante a execução do programa. Por meio de diferentes técnicas de análise, como análise de comportamento e análise de entrada de dados, essa abordagem permite identificar possíveis vulnerabilidades em tempo real.

5.1 Tipos de IAST: Active IAST vs Passive IAST

Existem dois principais tipos de IAST: **Active IAST** e **Passive IAST**, cada um com características distintas:

5.1.1 Active IAST

O **Active IAST** interage ativamente com a aplicação durante a execução dos testes, injetando cargas maliciosas e simulando ataques para verificar possíveis vulnerabilidades. Ele combina elementos do *Dynamic Application Security Testing* (DAST) com análise aprofundada do código em tempo de execução.

Vantagens:

- Identifica vulnerabilidades em tempo real com alta precisão.
- Gera menos falsos positivos do que DAST.
- Pode ser integrado em pipelines CI/CD para automação.

Desvantagens:

- Pode afetar a performance da aplicação durante a execução.
- Exige interação ativa com o código e testes dinâmicos.

5.1.2 Passive IAST

O **Passive IAST** funciona de forma mais discreta, monitorando o tráfego e a execução da aplicação sem interagir diretamente com ela. Ele observa o comportamento da aplicação e analisa suas entradas e saídas para detectar vulnerabilidades.

Vantagens:

- Não interfere na performance da aplicação.
- Pode ser usado em ambientes de produção sem impacto.

Desvantagens:

- Pode não identificar algumas vulnerabilidades mais complexas.
- Depende do tráfego real da aplicação para análise, o que pode limitar a cobertura de testes.

Ambos os métodos possuem vantagens e desvantagens, sendo frequentemente usados em conjunto para maximizar a detecção de vulnerabilidades em aplicações modernas.

5.2 Funcionamento

- **Etapa 1: Instrumentação da Aplicação**

  O IAST começa com a instrumentação da aplicação em execução. Isso é feito por meio de agentes que são inseridos no ambiente de execução da aplicação, permitindo a análise do código-fonte, do tráfego e das interações do sistema em tempo real. Essa abordagem permite detectar vulnerabilidades enquanto a aplicação está sendo usada em um ambiente de teste ou até mesmo em produção, dependendo da configuração.

- **Etapa 2: Monitoramento de Entrada e Comportamento**

  Durante a execução da aplicação, o IAST monitora todas as entradas do utilizador e o fluxo de dados dentro do sistema. Diferente do DAST, que apenas observa a aplicação externamente, o IAST consegue acompanhar a trajetória dos dados internamente, identificando falhas como injeção de código, manipulação de parâmetros inseguros e vazamento de informações sensíveis.

- **Etapa 3: Análise em Tempo Real**

  As ferramentas de IAST realizam análises contínuas enquanto a aplicação está sendo utilizada. Isso significa que qualquer comportamento suspeito ou vulnerabilidade detectada pode ser imediatamente identificada e analisada. Essa abordagem permite uma resposta mais rápida às falhas de segurança, sem a necessidade de testes dedicados separados.

- **Etapa 4: Exploração e Validação de Vulnerabilidades**

  O IAST tenta validar as vulnerabilidades identificadas, verificando se elas podem realmente ser exploradas por atacantes. Diferente do DAST, que depende apenas de testes externos, o IAST pode correlacionar vulnerabilidades com o código-fonte, mostrando a linha exata onde o problema ocorre. Isso reduz falsos positivos e melhora a precisão na identificação dos riscos.

- **Etapa 5: Integração com CI/CD e Automação**

  Uma das vantagens do IAST é sua fácil integração com pipelines de CI/CD. Como ele funciona em tempo de execução e pode ser ativado durante os testes automatizados, as equipes de desenvolvimento recebem feedback imediato sobre vulnerabilidades antes da aplicação ser implantada em produção. Essa automação melhora a eficiência da segurança no desenvolvimento ágil.

- **Etapa 6: Relatórios e Feedback Contínuo**

  Após a execução dos testes, o IAST gera relatórios detalhados com as vulnerabilidades encontradas, indicando sua localização no código e fornecendo recomendações para mitigação. Como o IAST trabalha em tempo real, ele também pode fornecer alertas imediatos para os desenvolvedores sempre que uma nova vulnerabilidade for detectada.

5.3 Vantagens

Uma das principais vantagens do IAST é a identificação precoce de vulnerabilidades, o que possibilita a mitigação de riscos antes que possam ser explorados por hackers. Como a análise ocorre durante a execução da aplicação, possíveis falhas são detectadas rapidamente, reduzindo o tempo necessário para correções e minimizando impactos na segurança do sistema. Diferente de métodos puramente estáticos (SAST) ou dinâmicos externos (DAST), o IAST opera dentro do ambiente de execução da aplicação, fornecendo condições semelhantes às dos utilizadores reais. Isso permite a identificação de vulnerabilidades complexas que poderiam passar despercebidas em outros tipos de testes, como problemas relacionados à lógica da aplicação e permissões inadequadas.

Além disso, o IAST apresenta alta precisão na detecção de vulnerabilidades. Como ele combina elementos do SAST e do DAST, consegue fornecer informações mais detalhadas sobre as falhas encontradas, incluindo o trecho exato do código onde o problema ocorre. Essa abordagem reduz significativamente a ocorrência de falsos positivos, facilitando o trabalho das equipes de desenvolvimento ao priorizar e corrigir vulnerabilidades reais de forma eficiente.

5.4 Limitações

Apesar de suas vantagens, o IAST possui algumas limitações que devem ser consideradas antes da sua adoção. Uma delas é a dependência de ferramentas específicas. Para que o IAST funcione corretamente, é necessário o uso de soluções capazes de coletar informações em tempo real durante a execução dos testes. Isso pode restringir a escolha de ferramentas disponíveis e exigir compatibilidade com a linguagem de programação e o framework utilizados na aplicação.

Outra limitação importante é a necessidade de integração com o processo de desenvolvimento. Diferente de testes externos, que podem ser executados independentemente do fluxo de desenvolvimento, o IAST precisa de estar incorporado às práticas da equipa de engenharia. Isso exige um nível maior de

coordenação entre as equipas de desenvolvimento e segurança, garantindo que os testes sejam realizados de maneira contínua e que as vulnerabilidades sejam corrigidas antes do lançamento da aplicação.

Por fim, a adoção do IAST exige conhecimento técnico especializado, o que pode dificultar sua implementação em pequenas organizações ou equipes com pouca experiência em segurança de aplicações. A configuração das ferramentas, a interpretação dos resultados e a aplicação das correções requerem um nível avançado de entendimento sobre segurança cibernética, tornando essencial o investimento em treinamento ou a contratação de profissionais qualificados.

5.5 Ferramentas IAST

Ferramenta	**Foco Principal**	**Resultados (Precisão)**	**Integração com CI/CD**	**Feedback (Usabilidade e Relatórios)**	**Implementação**
Hdiv Security	Proteção em tempo real durante a execução da aplicação	Alta precisão, detecta vulnerabilidades em tempo real	Sim, integração fácil com Jenkins, GitLab, e outras ferramentas de CI/CD	Relatórios claros com informações detalhadas sobre as falhas no código	Baseado em nuvem e on-premises, fácil de integrar no fluxo de trabalho de desenvolvimento
Contrast Security	Testes em tempo real com foco em integração contínua	Muito alta precisão, detecta falhas com base no comportamento da aplicação	Sim, integração robusta com DevOps e ferramentas CI/CD (Jenkins, Bamboo, etc.)	Feedback em tempo real, relatórios detalhados e sugestões de correção	Baseado em nuvem e on-premises, fácil de integrar com outras ferramentas de segurança e desenvolvimento
Seeker by Synopsys	Análise de segurança de aplicativos em tempo real, integração contínua	Alta precisão, oferece visibilidade detalhada das falhas de segurança	Sim, integração total com Jenkins e outras ferramentas de CI/CD	Relatórios detalhados e fáceis de entender, com feedback contínuo	Instalação local ou nuvem, com suporte robusto a aplicações corporativas
Veracode IAST	Testes contínuos de segurança com foco em DevSecOps	Alta precisão, combina análise estática e dinâmica para melhor cobertura	Sim, integração fácil com pipelines de CI/CD	Relatórios claros, destacando vulnerabilidades críticas e sugerindo remediações	Baseado em nuvem, fácil de integrar com outras ferramentas de segurança
Micro Focus Fortify IAST	Solução IAST para análise contínua durante o desenvolvimento	Alta precisão, identifica vulnerabilidades em tempo real	Sim, integração com Jenkins, Bamboo e outras ferramentas de CI/CD	Relatórios detalhados com sugestões para mitigação de vulnerabilidades	Baseado em nuvem e on-premises, instalação e configuração mais complexas

Tabela 3: Comparação das principais ferramentas IAST

6 Conclusão

De forma a consolidar toda a informação reunida, a tabela apresentada compara as três metodologias de detecção de vulnerabilidades em aplicações: SAST (Static Application Security Testing), DAST (Dynamic Application Security Testing) e IAST (Interactive Application Security Testing). Cada

abordagem possui características distintas que atendem a diferentes necessidades de segurança ao longo do ciclo de desenvolvimento de software, como vemos na tabela seguinte.

Vulnerabilidade	**SAST**	**DAST**	**IAST**
Detecção precoce durante o desenvolvimento	✓		✓
Análise de código-fonte (estrutura, dependências)	✓		✓
Identificação de vulnerabilidades em ambiente de execução		✓	✓
Validação de controles de segurança em tempo real		✓	✓
Localização exata da falha no código	✓		✓
Análise do fluxo de dados e comportamento		✓	✓

Tabela 4: Comparação entre metodologias de detecção de vulnerabilidades: SAST vs DAST vs IAST.

Assim sendo, consideramos que uma combinação destas ferramentas assegura uma melhor cobertura e reduz o risco de vulnerabilidades no desenvolvimento de aplicações. Contudo, é necessário ter em mente que estas ferramentas não se destinam a substituir todas as outras práticas de codificação seguras, mas sim a fazer parte de um esforço maior de segurança aplicacional.