Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aprimorando a detecção de vulnerabilidades em APIs criptográficas Java: uma abordagem qualitativa integrando CogniCrypt, CryptoGuard e LibScout

Guilherme Andreúce S. Monteiro

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Rodrigo Bonifacio de Almeida

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Aprimorando a detecção de vulnerabilidades em APIs criptográficas Java: uma abordagem qualitativa integrando CogniCrypt, CryptoGuard e LibScout

Guilherme Andreúce S. Monteiro

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Rodrigo Bonifacio de Almeida (Orientador)
CIC/UnB

Prof. Dr. Donald Knuth
Stanford University

Dr. Leslie Lamport
Microsoft Research

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 19 de setembro de 2023

Dedicatória

Eu dedico este trabalho a minha esposa, Nicole Borba Monteiro, que me apoiou e incentivou durante todo o processo de desenvolvimento deste trabalho. Dedico também aos meus pais, Karla e Marlos Monteiro, que sempre me apoiaram e me incentivaram a estudar mesmo sem entender muito bem o que eu estava fazendo. Também aos meus amigos que me ajudaram a manter a sanidade durante o processo de desenvolvimento deste trabalho e que sempre me incentivaram a nunca desistir.

Agradecimentos

Agradeço ao Prof. Dr. Rodrigo Bonifacio de Almeida pela persistencia e paciência em me orientar durante o desenvolvimento deste trabalho. Agradeço também em especial ao Luis Amaral por não só ter me ajudado com tudo o que foi necessário como também por ter me incentivado a continuar quando eu estava prestes a desistir. Agradeço ao Rafael Bressan por ter me ajudado com a união dos resultados dos csvs gerados. Agradeço também ao Bruno Chaves pelas dicas e sugestões que me ajudaram a melhorar o trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Este estudo apresenta uma abordagem inovadora para aprimorar a detecção de vulnerabilidades em APIs criptográficas Java, visando fortalecer a segurança de aplicações baseadas nessa tecnologia. Para isso, integramos as ferramentas CogniCrypt e CryptoGuard com o LibScout, permitindo a identificação precisa da origem de warnings relacionados a bibliotecas externas. Essa abordagem qualitativa representa um avanço significativo na promoção da segurança em aplicações Java, contribuindo para um ecossistema digital mais resiliente e protegido contra potenciais ameaças cibernéticas. Ao incorporar a identificação da origem dos warnings, também possibilitamos sugestões diretas aos desenvolvedores das bibliotecas, otimizando o processo de correção de vulnerabilidades. No entanto, enfrentamos desafios ao analisar código obfuscado e ao utilizar clusters e datasets no LibScout, evidenciando a necessidade de aprimoramentos nessa ferramenta. A integração proposta neste trabalho representa um passo significativo em direção à segurança abrangente de dados sensíveis e sistemas críticos em aplicações Java.

Palavras-chave: CogniCrypt, Eclipse, Segurança do código, Análise

Abstract

This study introduces a innovative approach to enhance the detection of vulnerabilities in Java cryptographic APIs, aiming to strengthen the security of applications built on this technology. By integrating the tools CogniCrypt and CryptoGuard with LibScout, we enable the precise identification of the source of warnings related to external libraries. This qualitative approach represents a significant advancement in promoting security in Java applications, contributing to a more resilient digital ecosystem protected against potential cyber threats. The incorporation of warning source identification also allows for direct suggestions to library developers, streamlining the vulnerability correction process. However, we encountered challenges when analyzing obfuscated code and utilizing clusters and datasets in LibScout, highlighting the need for improvements in this tool. The integration proposed in this work represents a significant step towards comprehensive security for sensitive data and critical systems in Java applications.

Keywords: CogniCrypt, Eclipse, Code Security, Analisys

Sumário

1 Introdução **1**
1.1 Introdução 1
1.2 Objetivos 2
1.3 Justificativa 3

2 Trabalhos Correlatos e Revisão de Literatura **4**
2.1 Trabalhos Correlatos e Revisão de Literatura 4
2.1.1 Criptografia 4
2.1.2 Análise dinâmica de APIs criptográficas 5
2.1.3 Detecção de vulnerabilidades em APIs criptográficas Java 6
2.1.4 Detecção de bibliotecas externas em aplicações Android 8
2.1.5 LibScout 11
2.1.6 Aplicativos obfuscados 12
2.1.7 O que é a ferramenta CogniCrypt? 13
2.1.8 O que é a linguagem CrySL? 15
2.1.9 O que é a ferramenta CryptoGuard? 16
2.1.10 CryptoGuard vs CrySL 18

3 Metodologia **19**
3.1 Hipótese de Trabalho 19
3.2 Metodologia 19

4 Resultados **23**
4.1 Percepção dos desenvolvedores em relação as vulnerabilidade em aplicativos open source 23
4.2 Análise quantitativa em aplicativos android 24
4.2.1 Análise qualitativa da integração do Cryptoguard e do Cognicrypt com o LibScout 25

5 Conclusão **33**
5.1 Conclusão 33

Referências **35**

Lista de Figuras

3.1 Metodologia adotada no artigo 'Perceptions of Software Practitioners Regarding Crypto-API Misuses and Vulnerabilities' 20
3.2 Metodologia adotada neste trabalho' 20
3.3 'Exemplo de uma Gist' 22

4.1 Quantidade média de alertas por aplicativo 25
4.2 Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria Connectivity 29
4.3 Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria Finances 30
4.4 Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria SMS 31
4.5 Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria System 32

Lista de Tabelas

4.1 Aplicativos por categoria sem warning das ferramentas CogniCrypt e CryptoGuard 24
4.2 Warnings encontrados nas ferramentas CogniCrypt e CryptoGuard 24
4.3 Resultados da integração do CogniCrypt com o LibScout na categoria Connectivity 26
4.4 Resultados da integração do CogniCrypt com o LibScout na categoria Finances 26
4.5 Resultados da integração do CogniCrypt com o LibScout na categoria SMS 26
4.6 Resultados da integração do CogniCrypt com o LibScout na categoria System 26
4.7 Resultados da integração do CryptoGuard com o LibScout na categoria Connectivity 27
4.8 Resultados da integração do CryptoGuard com o LibScout na categoria Finances 27
4.9 Resultados da integração do CryptoGuard com o LibScout na categoria SMS 27
4.10 Resultados da integração do CryptoGuard com o LibScout na categoria System 27

Capítulo 1

Introdução

1.1 Introdução

A criptografia, uma disciplina essencial da segurança da informação, é fundamental para proteger sistemas digitais e dados sensíveis de ameaças cibernéticas. [1] Com a complexidade das aplicações aumentando e a variedade de bibliotecas e frameworks disponíveis, o desenvolvimento de ferramentas automatizadas capazes de detectar possíveis falhas e vulnerabilidades nas interfaces de programação de aplicações (APIs) criptográficas torna-se crucial. [2]

O surgimento da linguagem CrySL permitiu a definição precisa de regras para o uso seguro de APIs criptográficas em código Java. [3] A linguagem permitiu a criação de padrões mais rigorosos para a implementação de métodos de criptografia seguros. No entanto, há um grande número de investigações sobre as dúvidas sobre a eficácia das ferramentas atuais e a precisão de seus alertas.

Neste contexto, o presente estudo empreende uma análise qualitativa abrangente da detecção de vulnerabilidades em APIs criptográficas, valendo-se das ferramentas CogniCrypt [3] e CryptoGuard [4].

Deparamo-nos com a complexidade inerente à análise de código obfuscado durante a realização deste estudo, o que reforçou o valor de considerar uma variedade de contextos de implementação ao avaliar a eficácia das ferramentas de detecção de vulnerabilidades. [5]

Adicionalmente, foi observado que as ferramentas CogniCrypt e CryptoGuard, embora extremamente importantes em termos de sua capacidade de detectar possíveis vulnerabilidades, não são capazes de identificar de onde surgem os alertas, sejam eles originários de bibliotecas nativas ou externas. [6] Tal limitação poderia potencialmente acarretar em falsos positivos ou negligenciar alertas de importância vital advindos de bibliotecas de fundamento.

Para superar esse desafio, lançamos mão do estudo intitulado "Automated Third-Party Library Detection for Android Applications: Are We There Yet?". [6] A partir dessa fonte, propomos uma solução inovadora ao integrar o resultado do LibScout ao contexto do CryptoGuard e CogniCrypt. Esta abordagem possibilitou não apenas a detecção de potenciais vulnerabilidades, mas também a identificação precisa de correspondências associadas a bibliotecas externas. Desse modo, concebeu-se uma flag adicional, denominada "external_library", destinada a sinalizar a presença de uma biblioteca externa quando uma correspondência era identificada.

Também foi considerado o mapeamento geral das bibliotecas encontradas nos resultados da ferramenta LibScout [7] o que nos possibilitou identificar não só as bibliotecas que definitivamente eram externas como também fazer o casamento das classes apresentadas pelos analisadores estáticos surgindo assim outra flag denominada "possible_external". Esta destinada a sinalizar se a biblioteca continha classes que poderiam ser externas.

No entanto, é essencial mencionar os desafios enfrentados ao usar o LibScout. Por vezes, a ferramenta apresentou limitações ao definir clusters com diferentes graus de granularidade. Como resultado, os resultados podem não incluir bibliotecas conhecidas como não-nativas. Além disso, o conjunto de dados mais recente que está disponível para uso data de julho de 2019, o que pode alterar a extensão das correspondências identificadas.

A inclusão deste recurso não apenas aumentou a precisão da detecção de falhas, mas também abriu novas perspectivas. Agora somos capazes de fornecer diretamente recomendações aos desenvolvedores das bibliotecas em questão, o que permite uma intervenção mais direta e eficaz na resolução de possíveis vulnerabilidades. Antes, os desenvolvedores precisavam se encarregar da tarefa.

Este trabalho representa um avanço significativo na promoção da segurança de aplicações baseadas em Java, com o objetivo de proteger sistemas vitais e dados sensíveis de ameaças cibernéticas. Para contribuir para um ecossistema digital mais resiliente e protegido, as práticas de segurança na implementação de APIs criptográficas serão fortalecidas por meio dessa abordagem qualitativa e da integração de ferramentas de detecção.

1.2 Objetivos

O objetivo inicial deste estudo era fornecer aos desenvolvedores uma forma de identificar vulnerabilidades em APIs criptográficas. No entanto, ao longo do estudo, percebeu-se que a detecção de vulnerabilidades em APIs criptográficas, valendo-se das ferramentas CogniCrypt e CryptoGuard, não era suficiente para identificar a origem dos alertas.

Dessa forma, o objetivo deste estudo foi ampliado para incluir a identificação da origem dos alertas. Esta expansão se revelou crucial, uma vez que a capacidade de precisamente

determinar a origem de um alerta é de extrema importância para os desenvolvedores. Isso possibilita ações direcionadas e específicas para corrigir possíveis vulnerabilidades, economizando tempo e recursos valiosos no processo de desenvolvimento e garantindo a segurança efetiva das aplicações.

Para isso, foi necessário integrar o resultado do LibScout ao contexto do CryptoGuard e CogniCrypt. Esta abordagem possibilitou não apenas a detecção de potenciais vulnerabilidades, mas também a identificação precisa de correspondências associadas a bibliotecas externas, fornecendo uma visão clara da origem dos alertas e permitindo a implementação de soluções de segurança de forma eficiente e focalizada.

1.3 Justificativa

A crescente complexidade das aplicações Java, aliada à importância crítica da segurança da informação, torna imperativo o desenvolvimento de técnicas e ferramentas que auxiliem os desenvolvedores na identificação e correção de potenciais vulnerabilidades em APIs criptográficas. Diversos estudos demonstraram que o uso inadequado dessas APIs é uma das principais fontes de vulnerabilidades em software.

Diante desse cenário, a presente pesquisa se propõe a aprimorar a detecção de vulnerabilidades em APIs criptográficas, proporcionando aos desenvolvedores uma solução mais abrangente e eficaz para garantir a segurança das aplicações Java. A integração dos resultados do LibScout às ferramentas CryptoGuard e CogniCrypt representa um avanço significativo, pois não apenas identifica potenciais vulnerabilidades, mas também localiza a origem desses alertas, permitindo uma intervenção mais precisa e efetiva por parte dos desenvolvedores.

Portanto, este estudo se justifica pela necessidade premente de fortalecer a segurança das aplicações Java e pela contribuição inovadora que a abordagem proposta representa para esse fim.

Capítulo 2

Trabalhos Correlatos e Revisão de Literatura

2.1 Trabalhos Correlatos e Revisão de Literatura

2.1.1 Criptografia

A criptografia é um método de proteger informações e comunicações por meio do uso de códigos, para que apenas aqueles para quem as informações são destinadas possam lê-las e processá-las [1].

Em ciência da computação, a criptografia se refere a técnicas seguras de informações e comunicações derivadas de conceitos matemáticos e de um conjunto de cálculos baseados em regras chamados algoritmos, para transformar mensagens de maneiras difíceis de decifrar [1].

A criptografia moderna preocupa-se com quatro objetivos principais: confidencialidade, integridade, não repúdio e autenticação [1].

Algoritmos de criptografia de chave única ou simétrica criam um bloco de tamanho fixo de bits conhecido como cifra com uma chave secreta que o criador/remetente usa para cifrar dados. Um exemplo disso é o Advanced Encryption Standard (AES) [1].

Exemplos de criptografia de chave pública incluem RSA, Elliptic Curve Digital Signature Algorithm (ECDSA) e Digital Signature Algorithm (DSA) [1].

A palavra "criptografia"é derivada do grego "kryptos", que significa escondido. A origem da criptografia geralmente remonta a cerca de 2000 a.C., com a prática egípcia de hieróglifos. No entanto, a criptografia moderna teve início com Júlio César, que criou um sistema de substituição de caracteres em mensagens [1].

Apesar da importância da criptografia para a segurança dos sistemas, muitos desenvolvedores se deparam com desafios significativos ao tentar implementá-la corretamente.

Sem o conhecimento especializado em criptografia, é possível utilizar erroneamente algoritmos e técnicas criptográficas inadequadas. Isso pode resultar em vulnerabilidades que comprometem a segurança e a privacidade dos dados dos usuários. Portanto, é essencial contar com ferramentas que possam orientar os desenvolvedores na aplicação correta das práticas criptográficas, reduzindo assim os riscos associados à implementação inadequada de medidas de segurança em software.Apesar da criptografia ser um componente fundamental para garantir a segurança dos sistemas, muitos desenvolvedores encontram grandes desafios ao tentar fazê-la funcionar corretamente. Sem a expertise em criptografia, alguém pode erroneamente utilizar algoritmos e técnicas criptográficas inadequadas. Dessa forma, as aplicações podem apresentar vulnerabilidades que prejudicam a segurança e a privacidade dos dados dos usuários. É crucial contar com ferramentas que possam orientar os desenvolvedores na aplicação correta das práticas criptográficas, a fim de reduzir os riscos envolvidos quando as medidas de segurança em software são implementadas incorretamente.[2]

2.1.2 Análise dinâmica de APIs criptográficas

Um estudo empírico importante realizado por Torres et al. [8] fornece uma meticulosa investigação comparativa de métodos de detecção de uso inadequado de APIs criptográficas em projetos Java. A pesquisa conduzida oferece uma análise detalhada das técnicas empregadas, incluindo a abordagem de Verificação em Tempo de Execução (Runtime Verification, ou RVSec), juntamente com notáveis analisadores estáticos como CogniCrypt e CryptoGuard, além da ferramenta CryLogger.

Ao longo da análise, o estudo destaca tanto as virtudes como as limitações inerentes a cada uma dessas abordagens. Detalhes sobre os cenários em que cada técnica demonstra maior eficácia, bem como os casos em que pode resultar em falsos positivos e negativos, são minuciosamente delineados. Adicionalmente, são propostas recomendações para otimizar a precisão e efetividade na detecção de discrepâncias no uso de APIs criptográficas.

Este estudo desempenha um papel fundamental ao estabelecer um sólido alicerce para nossa própria investigação. Ele não apenas fornece insights valiosos sobre as técnicas de detecção de vulnerabilidades em APIs criptográficas, mas também nos motivou a explorar uma perspectiva complementar. Nossa pesquisa se concentra em avaliar a percepção dos desenvolvedores sobre as vulnerabilidades identificadas pelos métodos analisados, preenchendo assim uma lacuna crucial no entendimento do impacto dessas detecções no processo de desenvolvimento de software. Ao considerar tanto a eficácia técnica quanto a perspectiva dos desenvolvedores, buscamos enriquecer o panorama das práticas de segurança na utilização de APIs criptográficas em projetos Java.

2.1.3 Detecção de vulnerabilidades em APIs criptográficas Java

O estudo conduzido por Zhang et al. no artigo intitulado "Automatic Detection of Java Cryptographic API Misuses: Are We There Yet?"[2] aborda os desafios enfrentados pelos desenvolvedores ao trabalhar com APIs criptográficas Java. O artigo fornece uma análise detalhada desses desafios, destacando questões como a complexidade das APIs, a falta de documentação adequada e a falta de treinamento em segurança cibernética por parte dos desenvolvedores [2].

Além disso, muitos desenvolvedores podem não possuir o treinamento em cibersegurança necessário para compreender plenamente as implicações de segurança ao utilizar as opções de codificação dentro das APIs criptográficas [2]. Eles podem não estar completamente cientes dos valores de parâmetros apropriados, das sequências de chamadas corretas ou da lógica de substituição, o que pode resultar na implementação insegura de funcionalidades de segurança [2].

Outro desafio comum decorre da falta de compreensão e treinamento, levando os desenvolvedores a fazerem uso inadequado das APIs criptográficas [2]. Isso pode se manifestar na escolha de métodos incorretos, na passagem de parâmetros inadequados ou em outros erros que introduzem vulnerabilidades ou fragilidades no software desenvolvido [2].

Adicionalmente, é comum que os desenvolvedores recorram à prática de copiar e colar trechos de código de fontes online, como o StackOverflow [2]. No entanto, essa abordagem nem sempre é acompanhada de uma compreensão plena do uso da API criptográfica em questão. Tal prática pode resultar na disseminação de usos inadequados da API em diversas aplicações [2].

Esses desafios podem, por sua vez, culminar na existência de vulnerabilidades exploráveis. Hackers têm a capacidade de aproveitar-se dessas vulnerabilidades relacionadas às APIs, potencialmente comprometendo a segurança de dados sensíveis, como credenciais de usuários [2]. Portanto, a importância de uma utilização adequada das APIs criptográficas torna-se evidente [2].

Em síntese, os obstáculos enfrentados pelos desenvolvedores nesse contexto emergem da complexidade das APIs, da falta de treinamento em cibersegurança e da possibilidade de utilização inadequada das mesmas, o que pode resultar na introdução de vulnerabilidades no software desenvolvido [2].

Também foi identificado pelo estudo que diversas vulnerabilidades relacionadas ao uso inadequado de APIs criptográficas Java [2]. Entre elas, destacam-se:

Os desenvolvedores frequentemente utilizavam parâmetros inadequados em métodos como Cipher.getInstance(), MessageDigest.getInstance() e SecretKeyFactory.getInstance(), indicando a utilização de algoritmos criptográficos quebrados ou arriscados, potencialmente expondo informações sensíveis [2].

Houve casos em que os desenvolvedores empregaram métodos inseguros para gerar chaves criptográficas, comprometendo a segurança dos processos de criptografia e descriptografia e facilitando possíveis ataques [2].

Foi observado que o gerenciamento de chaves criptográficas muitas vezes deixou a desejar. Isso incluiu práticas como a codificação rígida de chaves no código-fonte, o armazenamento em locais inseguros e o uso de mecanismos de armazenamento de chaves frágeis, o que poderia resultar em acessos não autorizados a dados sensíveis [2].

Em algumas situações, os desenvolvedores falharam na inicialização correta de objetos criptográficos, como definir o modo ou preenchimento adequado para algoritmos de criptografia. Essas falhas poderiam levar a processos de criptografia ou descriptografia vulneráveis, tornando o sistema suscetível a ataques [2].

Foram encontrados casos em que os desenvolvedores utilizaram métodos inseguros para gerar números aleatórios, como a classe java.util.Random em vez da mais segura java.security.SecureRandom. Essa prática poderia comprometer a segurança das operações criptográficas [2].

Por fim, o estudo identificou situações em que os desenvolvedores não autenticavam ou validavam adequadamente operações criptográficas, como a falta de verificação de assinaturas digitais ou a validação de certificados. Isso poderia resultar na aceitação de dados forjados ou adulterados, prejudicando a integridade e autenticidade do sistema [2].

Em relação às percepções apresentadas pelos desenvolvedores, tivemos visões divergentes em relação às ferramentas existentes para detectar usos incorretos de APIs. Dos 47 feedbacks recebidos, a maioria dos desenvolvedores (30) rejeitou as vulnerabilidades relatadas, indicando que não as consideravam válidas ou relevantes. Menos desenvolvedores (17) demonstraram disposição para abordar os problemas reportados e fazer as correções necessárias. Ainda menos desenvolvedores (9) efetivamente substituíram os usos incorretos de APIs com base nas orientações fornecidas pelas ferramentas [2].

O estudo identificou três fatores que contribuíram para a relutância dos desenvolvedores em lidar com os problemas reportados. Em primeiro lugar, as sugestões de correção fornecidas pelas ferramentas muitas vezes eram vagas e incompletas, tornando difícil para os desenvolvedores compreender como corrigir os usos incorretos de forma eficaz. Em segundo lugar, os desenvolvedores expressaram a necessidade de evidências de exploração de segurança que pudessem ser habilitadas pelas vulnerabilidades relatadas. Eles queriam compreender o impacto potencial e a gravidade dos problemas antes de investir tempo em corrigi-los. Por fim, alguns dos usos incorretos detectados foram encontrados em código de teste ou em contextos de programa que não eram considerados relevantes para a segurança. Os desenvolvedores acreditavam que esses problemas não teriam consequências de segurança, levando-os a ignorá-los ou descartá-los [2].

No geral, o estudo revelou uma lacuna significativa entre as ferramentas existentes e as expectativas dos desenvolvedores. Os relatórios gerados pelas ferramentas não alteraram efetivamente as práticas de codificação dos desenvolvedores, e estes tinham preocupações sobre as capacidades das ferramentas, a correção das correções sugeridas e a exploração dos problemas relatados [2].

Diante dos argumentos expostos, o presente estudo se fundamenta na análise crítica do trabalho suplementar, visando compreender se as vulnerabilidades identificadas pelas ferramentas CryptoGuard e CogniCrypt têm sua origem associada a bibliotecas de caráter nativo ou externo. Este enfoque pode se revelar essencial para uma apreciação abrangente das fragilidades apontadas, proporcionando um discernimento mais aprofundado acerca das nuances envolvidas na integridade e segurança do sistema em questão.

2.1.4 Detecção de bibliotecas externas em aplicações Android

A incorporação de bibliotecas de terceiros em aplicações Android suscita diversas preocupações relevantes. Uma delas refere-se à possibilidade de presença de código malicioso nessas bibliotecas, representando uma ameaça à segurança e à privacidade dos dispositivos dos usuários. Esse código tem o potencial de ser utilizado em atividades como a apropriação indevida de dados, acessos não autorizados e até mesmo o controle remoto do dispositivo. [5]

Além disso, a existência de vulnerabilidades nas bibliotecas de terceiros é outra preocupação significativa. Estas vulnerabilidades podem ser exploradas por atacantes, possibilitando o acesso não autorizado ao dispositivo, a execução de código arbitrário ou a realização de atividades maliciosas [5].

Outro ponto de preocupação refere-se à possível incompatibilidade dessas bibliotecas com a aplicação em si ou com outras bibliotecas utilizadas no desenvolvimento. Tal cenário pode resultar em falhas, dificuldades de desempenho ou em outros comportamentos inesperados no âmbito da aplicação [5].

A falta de atualizações regulares ou suporte por parte dos desenvolvedores de bibliotecas de terceiros também é uma questão a ser considerada. Isso pode deixar a aplicação vulnerável a novas ameaças de segurança ou a dificuldades de compatibilidade [5].

Adicionalmente, o emprego de determinadas bibliotecas de terceiros pode acarretar em violação de acordos de licenciamento ou direitos de propriedade intelectual, potencialmente resultando em consequências legais para o desenvolvedor da aplicação [5].

A complexa cadeia de dependências que frequentemente acompanha as bibliotecas de terceiros é outra preocupação destacável. Caso alguma dessas dependências apresente vulnerabilidades ou outros problemas, isso pode afetar a segurança e a estabilidade global da aplicação [5].

Outro aspecto crítico refere-se à possibilidade de bibliotecas de terceiros coletarem e transmitirem dados do usuário sem o devido conhecimento ou consentimento. Esta prática pode resultar em violações de privacidade e no uso não autorizado de informações pessoais [5].

Por fim, o desempenho da aplicação pode ser comprometido caso bibliotecas de terceiros não tenham sido otimizadas de maneira adequada ou apresentem eficiência limitada. Isso pode resultar em tempos de carregamento mais lentos, uso elevado de recursos e, consequentemente, numa experiência de usuário insatisfatória [5].

O impacto das Bibliotecas de Terceiros (TPLs) na detecção de malware em aplicativos móveis é que as TPLs podem introduzir ameaças à segurança se contiverem código malicioso. Quando essas TPLs são integradas a aplicativos populares, elas podem rapidamente infectar um grande número de dispositivos móveis. Além disso, as TPLs também podem afetar os resultados da detecção de malware, já que podem agir como ruído e potencialmente interferir no processo de detecção. Portanto, é importante contar com técnicas eficazes de detecção de TPLs para identificar e mitigar os riscos associados às TPLs em aplicativos móveis [5].

As ferramentas e métodos existentes para detectar e mitigar os riscos associados a Bibliotecas de Terceiros (TPLs) em aplicações Android abrangem diversas abordagens. Um desses métodos é o Baseado em Lista Branca, que emprega uma lista pré-definida de TPLs confiáveis para filtrar bibliotecas conhecidas. No entanto, esse método apresenta limitações, pois não é resiliente a renomeações de pacotes e pode deixar de abranger algumas TPLs, especialmente as mais recentes.

Além disso, diversas Ferramentas de Detecção foram desenvolvidas para identificar TPLs em aplicações Android. Estas ferramentas extraem características como APIs do Android, grafos de fluxo de controle e assinaturas de métodos variantes para representar as TPLs. Elas fazem uso de técnicas diversas, como métodos baseados em agrupamento e métodos de comparação de similaridade, a fim de identificar TPLs dentro do aplicativo.

Outra estratégia é a Revisão Sistemática da Literatura (RSL), na qual pesquisadores conduzem análises comparativas de técnicas de detecção de TPLs existentes. Esses estudos avaliam a eficácia, eficiência, capacidade de resistência à ofuscação de código e facilidade de uso das diferentes ferramentas [5].

Paralelamente, são adotadas Estratégias de Ofuscação de Código para proteger o software contra engenharia reversa, embora isso possa complicar a detecção de TPLs. Para contornar esse desafio, foram desenvolvidas ferramentas que são resilientes à ofuscação e capazes de lidar com renomeações de identificadores e pacotes.

Alguns pesquisadores disponibilizaram Conjuntos de Dados de Aplicativos Repacotados, os quais são utilizados para estudar e replicar abordagens de detecção de TPLs já

existentes.

É crucial ressaltar que as vantagens, desvantagens e desempenho dessas ferramentas variam, tornando a seleção da ferramenta apropriada um processo dependente do cenário de aplicação específico e dos requisitos envolvidos [5].

O artigo destaca várias ferramentas para detectar bibliotecas de terceiros (TPLs) em aplicativos Android, incluindo o LibID, que utiliza análise estática e dinâmica para identificar TPLs. O LibPecker concentra-se na detecção de TPLs ofuscadas por diferentes técnicas de ofuscação de código. O ORLIS também emprega uma combinação de análise estática e dinâmica, fornecendo análise de vulnerabilidades para as TPLs identificadas. O LibRadar utiliza análise baseada em API e é conhecido por sua rápida detecção. O LibD2, semelhante ao LibID e ORLIS, usa análise estática e dinâmica, além de fornecer análise de vulnerabilidades para as TPLs detectadas. O LibScout se destaca por sua sensibilidade à ofuscação do fluxo de controle, achatamento de pacotes e remoção de código inativo, dependendo da estrutura hierárquica de pacotes. No entanto, é afetado por técnicas de ofuscação de código, resultando em uma redução na taxa de detecção [5].

Em resumo, o artigo enfatiza o LibScout como uma ferramenta sensível a técnicas de ofuscação de código, que depende da estrutura hierárquica de pacotes para detectar TPLs. Embora tenha uma alta taxa de detecção, o LibScout pode ser influenciado por certas técnicas de ofuscação, o que afeta a precisão da detecção [5].

Além disso, os programas LibID, LibRadar, LibScout, LibPecker e ORLIS foram avaliados em quatro categorias: eficiência, eficácia/escalabilidade, resistência à obfuscação e facilidade de uso. O LibScout se destacou na eficiência, identificando 49 bibliotecas de terceiros com uma precisão de 97%. Em eficiência/escalabilidade, o LibRadar foi o mais eficaz, capaz de analisar cada aplicação em cerca de 5 segundos. No quesito resistência à obfuscação, o LibPecker mostrou-se o mais eficaz. Quanto à facilidade de uso, tanto o LibScout quanto o LibRadar superaram os concorrentes [5].

Os resultados do LibRadar e do LibScout para o piloto não foram semelhantes. O LibScout identificou muito mais TPLs do que o LibRadar devido ao primeiro analisar profundamente a aplicação, enquanto o último utiliza um método de agrupamento para categorizar se é ou não uma TPL. Para executar o LibRadar, tivemos que configurar um servidor Redis e carregar uma amostra para que o LibRadar possa comparar se a aplicação em teste utiliza ou não bibliotecas externas. A última vez que a amostra foi atualizada com um bom exemplo de agrupamento foi em 2017. A maioria dos aplicativos analisados eram de 2017 em diante. Por essa razão, o foco se voltou para o LibScout [5].

Assim para realizar a escolha de quais ferramentas seriam utilizadas para a identificação e mapeamento de bibliotecas nativas e externas em aplicações Android, foram consideradas as ferramentas LibScout, LibRadar, LibSoft, LibPecker, LibId e ORLIS. Foi

observado que através dos resultados apresentados no artigo Automatic Detection of Java Cryptographic API Misuses: Are We There Yet? que as ferramentas LibScout e LibRadar apresentaram os melhores resultados. O artigo aborda algumas categorias para a classificação das ferramentas, sendo elas:

Efetividade, Eficiência/Escalabilidade, Capacidade de resiliência à código obfuscado e Facilidade de uso. Para a escolha nos atentamos particularmente à eficiência. Tanto o LibRadar quanto o LibScout apresentaram resultados satisfatórios, porém o LibScout apresentou um desempenho melhor visto que a base utilizada para clusterização do libRadar é de 2016 e o LibScout utiliza uma base mais atualizada. As outras ferramentas apresentadas tinham baixo recall e precisão, além disso, o tempo de execução para um único aplicativo era muito alto.

Por fim, foi optado a escolha da ferramenta LibScout para identificação de bibliotecas de terceiros (TPLs) em aplicativos Android para esse estudo. Um dos maiores motivadores foi o fato de sua robustez e a apresentação dos resultados fornecendo uma visão hierarquizada das bibliotecas. Dessa forma, integramos o LibScout à nossa abordagem qualitativa, complementando as ferramentas CogniCrypt e CryptoGuard para tentar obter resultados mais abrangentes na detecção de vulnerabilidades. O tempo de execução, baixa precisão e amostras fora de data foram parte do porque não termos escolhido outras ferramentas.

2.1.5 LibScout

LibScout é uma ferramenta que visa extrair dados das APIs de bibliotecas de aplicativos Android. [7] Este resultado faz parte de um projeto de pesquisa cujo objetivo principal é analisar quais são as bibliotecas externas e quais são bibliotecas nativas em aplicativos Android. [7] Essa presença é fundamental para compreender e avaliar a segurança e a integridade dessas aplicações.

Esta ferramenta permite analisar chamadas de API de aplicativos Android diretamente do bytecode java. [7] A ferramenta coleta informações detalhadas sobre bibliotecas implantadas, incluindo nomes e definições, e fornece uma visão abrangente do ecossistema de bibliotecas de cada aplicativo analisado. [7] Essa funcionalidade é particularmente útil para desenvolvedores e pesquisadores que desejam melhorar sua compreensão a cerca de bibliotecas que pertencem à aplicativos específicos.

O LibScout funciona bem com aplicativos Android, independentemente de sua finalidade ou complexidade. [7] Assim, a ferramenta nos ajuda a identificar se o uso de práticas de desenvolvimento seguras ou inseguras está associado à integração de bibliotecas externas, facilitando a identificação e compreensão de bibliotecas de terceiros na aplicação [7].

A ferramenta conta com técnicas como 'renomeação de identificador'[7] e 'ofuscações baseadas em código', como ocultação de API baseada em 'reflexão' ou 'randomização de fluxo de controle' para encontrar bibliotecas externas em aplicativos Java mesmo com os desafios de implementação de ofuscação de código [7]. [7]

Usando essas estratégias, o LibScout pode encontrar bibliotecas de terceiros em vários contextos de aplicativos Java. Ao coletar dados de múltiplas aplicações, o LibScout é capaz de buscar padrões de chamadas de API que vão além das especificações de cada aplicação, fornecendo uma forma confiável de busca em bibliotecas externas [7].

A adição dos resultados das ferramentas CryptoGuard e CogniCrypt aos resultados gerados pelo LibScout pode melhorar significativamente a capacidade de avaliar a segurança de aplicativos Android e identificar possíveis vulnerabilidades relacionadas ao uso de bibliotecas de terceiros.

2.1.6 Aplicativos obfuscados

Uma dificuldade significativa na localização e extração de informações sobre bibliotecas de terceiros é a análise de aplicativos obfuscados. O uso comum da técnica de ofuscação de código torna a compreensão e análise do código-fonte mais difíceis, tornando a localização de bibliotecas externas ainda mais complicada. [5]

A ofuscação pode incluir a inserção de código adicional, bem como a renomeação de classes, métodos e variáveis, tornando as chamadas de API menos identificáveis. [5] Mesmo com ferramentas como o LibScout, isso dificulta a extração precisa de informações sobre bibliotecas de terceiros. [7]

O LibScout é excepcionalmente resistente a aplicativos obfuscados, sendo capaz de identificar bibliotecas mesmo diante de vários tipos de ofuscação comuns, como o ProGuard. Essa capacidade é essencial para garantir a precisão e confiabilidade na identificação de bibliotecas de terceiros em aplicativos obfuscados. [7]

Além disso, a ofuscação pode criar novos níveis de complexidade que requerem métodos sofisticados de análise de bytecode para desembaraçar o código obfuscado e identificar as chamadas de API pertinentes. Portanto, é fundamental usar abordagens e técnicas específicas ao lidar com aplicativos obfuscados para superar os problemas relacionados à prática da ofuscação de código. [5]

Para garantir a precisão e a confiabilidade na identificação de bibliotecas de terceiros, é necessário levar em consideração esses problemas ao trabalhar na análise de aplicativos obfuscados. Mesmo diante das complexidades criadas pela prática da ofuscação de código, isso permite uma avaliação completa da segurança dos aplicativos.

Um desafio significativo surgiu ao integrar os resultados do LibScout com as ferramentas CryptoGuard e CogniCrypt. Embora essas ferramentas mais recentes detectem

problemas e erros de segurança com sucesso, elas têm dificuldade em encontrar os nomes originais das bibliotecas e classes que são usadas. O processo de correlacionar os resultados e combinar os scripts de identificação de bibliotecas externas é mais difícil devido a essa restrição.

Assim, os problemas com a apresentação da classe da vulnerabilidade pelas duas ferramentas impediram que os resultados do LibScout fossem usados para melhorar a análise de segurança do CryptoGuard e CogniCrypt para esse estudo. No entanto, a capacidade do LibScout de localizar bibliotecas de terceiros em aplicativos Android é vital para avaliar a segurança desses aplicativos e relacionar os resultados das duas ferramentas com os do LibScout.

2.1.7 O que é a ferramenta CogniCrypt?

O CogniCrypt, desenvolvido no centro de pesquisa CROSSING da Technische Universität Darmstadt, é uma ferramenta projetada para auxiliar desenvolvedores na identificação e correção de usos inseguros de bibliotecas criptográficas em software. Estudos recentes têm apontado que muitos aplicativos que empregam procedimentos criptográficos o fazem de maneira inadequada, o que destaca a relevância do CogniCrypt [3].

Essa ferramenta integra-se ao ambiente de desenvolvimento Eclipse e oferece dois principais componentes. Primeiramente, um assistente de geração de código que auxilia os desenvolvedores na produção de código seguro para tarefas criptográficas comuns. Além disso, realiza uma análise estática contínua do código do desenvolvedor, notificando sobre possíveis usos incorretos de APIs criptográficas [3].

O CogniCrypt representa um avanço significativo na segurança de aplicações Java que fazem uso de operações criptográficas. Os desenvolvedores podem empregar a linguagem CrySL, na qual a ferramenta se baseia, para definir as melhores práticas para o uso seguro das APIs criptográficas disponíveis na arquitetura Java Cryptography (JCA). Desde a seleção de algoritmos até a gestão adequada de chaves de criptografia, as CrySL Rules fornecem um conjunto abrangente de diretrizes [3].

Além das análises em tempo real durante o processo de escrita, o CogniCrypt facilita a criptografia de dados, oferecendo um conjunto de ferramentas para implementar práticas de segurança de forma transparente e eficaz [3].

A colaboração entre a linguagem CrySL e o CogniCrypt oferece uma abordagem abrangente para identificar e reforçar a segurança de códigos vulneráveis. Ao seguir as regras e especificações definidas em CrySL, os desenvolvedores podem identificar potenciais pontos fracos na implementação de criptografia e receber recomendações precisas para aprimorar a segurança de seus sistemas [3].

Assim, o CogniCrypt pode oferecer uma abordagem abrangente para abordar a identificação de vulnerabilidades no código por meio de dois recursos fundamentais: a geração de código e a aplicação de análises estáticas [3].

A funcionalidade de geração de código do CogniCrypt destaca-se ao produzir implementações seguras para tarefas de programação comumente associadas à criptografia. Por meio desta característica, os desenvolvedores recebem exemplos de uso orientados por tarefas específicas das APIs criptográficas em Java. Esses exemplos são gerados com base em configurações selecionadas, que incluem o algoritmo criptográfico desejado e seus parâmetros correspondentes. Ao empregar esta capacidade, o CogniCrypt pode desempenhar um papel crucial em auxiliar os desenvolvedores na prevenção de vulnerabilidades comuns, garantindo a integração segura de componentes criptográficos em seus projetos [3].

Adicionalmente, o CogniCrypt incorpora uma funcionalidade de análise estática que opera em segundo plano, aplicando uma série de análises ao projeto do desenvolvedor. Estas análises têm por objetivo assegurar que todas as utilizações das APIs criptográficas permaneçam seguras, mesmo que o desenvolvedor venha a modificar o código gerado ou utilize as APIs diretamente, sem recorrer à geração de código. O CogniCrypt se vale do framework de análise de estados TS4J, implementado como um plugin do Eclipse, para efetuar a inspeção do projeto. Ele reporta discrepâncias na utilização por meio da geração de marcadores de erro diretamente no ambiente de desenvolvimento Eclipse IDE. Esta funcionalidade pode apoiar os desenvolvedores na identificação e correção de vulnerabilidades em seu código [3].

Por meio da sinergia entre geração de código e análise estática, o CogniCrypt pode conceder aos desenvolvedores uma abordagem completa para lidar com vulnerabilidades em seu código, promovendo o uso seguro de APIs criptográficas [3].

Os desafios enfrentados pelos desenvolvedores ao criar código são diversos e envolvem uma série de complexidades, especialmente quando lidam com sistemas extensos e intricados. Compreender os requisitos, desenhar a arquitetura e implementar o código são tarefas que demandam habilidade e atenção minuciosa. [3]

A identificação e correção de bugs e erros são etapas cruciais, muitas vezes exigindo um esforço considerável em termos de depuração e resolução de problemas. Além disso, a gestão do tempo é uma preocupação constante, pois os desenvolvedores frequentemente trabalham sob prazos apertados. Esta pressão adicional pode tornar desafiador o cumprimento desses prazos, o que, por sua vez, demanda a entrega de código de alta qualidade dentro dos limites estabelecidos. [3]

A dinamicidade dos requisitos de um projeto ao longo do processo de desenvolvimento pode requerer adaptações e modificações no código, o que, por sua vez, pode gerar trabalho adicional e até mesmo conflitos com o código já existente. [3]

Em ambientes colaborativos, a efetiva colaboração e comunicação entre membros da equipe são de vital importância. Coordenar esforços, resolver conflitos e assegurar que todos os membros estejam alinhados com os objetivos e metas do projeto pode se configurar como uma tarefa desafiadora, exigindo habilidades de comunicação e gestão de equipe. [3]

A aprendizagem contínua é um componente essencial no universo do desenvolvimento de software. A necessidade de se manter atualizado em relação a novas tecnologias, linguagens de programação, frameworks e ferramentas é premente. Este processo, embora vital, pode ser demandante em termos de tempo e esforço, requerendo um investimento contínuo por parte dos desenvolvedores. [3]

Uma vez que o código é desenvolvido, a manutenção contínua se torna imperativa. Esta etapa envolve a correção de bugs, adição de novos recursos e otimização de desempenho. No entanto, a complexidade desta tarefa pode ser exacerbada quando o código não está adequadamente documentado ou quando os desenvolvedores originais não estão mais disponíveis para prestar suporte. [3]

Por fim, a segurança e garantia de qualidade do código são aspectos cruciais. Os desenvolvedores devem assegurar que seu código seja imune a vulnerabilidades e que siga as melhores práticas para codificação segura. A condução de testes rigorosos se torna essencial para identificar e corrigir possíveis problemas de segurança, garantindo assim a integridade e segurança do software desenvolvido. [3]

Ao integrarmos o CogniCrypt em nossa abordagem, complementando-o com outras ferramentas como o CryptoGuard e o LibScout, queremos fornecer aos desenvolvedores uma estratégia poderosa e abrangente para detectar e corrigir vulnerabilidades em APIs criptográficas Java, seja ela de código nativo ou externo. Isso pode contribuir significativamente para a segurança e integridade dos sistemas desenvolvidos.

2.1.8 O que é a linguagem CrySL?

A linguagem de especificação criptográfica, ou CrySL, é um componente essencial do ecossistema do CogniCrypt. Ele foi desenvolvido para especificar boas práticas para o uso seguro de APIs criptográficas em Java. A CrySL, que foi desenvolvida como parte integrante do CogniCrypt, permite que os desenvolvedores expressem as regras de segurança de forma simples e fácil de entender, o que facilita a identificação de possíveis vulnerabilidades em códigos que envolvem operações criptográficas. [3]

A seleção adequada de algoritmos criptográficos, o gerenciamento seguro de chaves e o tratamento adequado de dados sensíveis estão entre as construções de alto nível fornecidas pelo CrySL para descrever cenários comuns de uso de criptografia. Além disso, a

linguagem foi desenvolvida para ser flexível, o que permite a inclusão de novas regras à medida que novos padrões e práticas de segurança surgem. [3]

Os desenvolvedores podem verificar automaticamente se um código está em conformidade com as boas práticas de segurança antes mesmo da execução ao definir regras em CrySL. Isso incentiva uma abordagem proativa para a segurança da informação, evitando brechas de segurança potenciais quando o software é desenvolvido em estágio inicial. [3]

A linguagem CrySL e o CogniCrypt criam um ambiente poderoso e fácil de entender para o desenvolvimento seguro de aplicações Java. Eles fornecem um conjunto abrangente de diretrizes e ferramentas para proteger dados e sistemas críticos de ameaças cibernéticas. [3]

2.1.9 O que é a ferramenta CryptoGuard?

CRYPTOGUARD é uma ferramenta de verificação de código estático projetada para detectar usos incorretos de APIs criptográficas e SSL/TLS em projetos Java de grande porte. [4] Seu propósito é auxiliar os desenvolvedores na identificação e correção de vulnerabilidades relacionadas a algoritmos criptográficos, exposição de segredos, geração previsível de números aleatórios e verificações de certificados vulneráveis. [4] O CRYPTOGUARD alcança isso por meio da implementação de um conjunto de novos algoritmos de análise que realizam uma análise estática do código-fonte. [4] Ele proporciona detecção de alta precisão de vulnerabilidades criptográficas e oferece insights de segurança aos desenvolvedores. [4] A ferramenta é projetada para ser leve e eficiente, executando mais rapidamente do que técnicas de análise existentes. [4] Suas funcionalidades incluem identificação de violações de propriedades criptográficas, realização de fatiamento para frente e para trás, e geração de alertas de segurança para potenciais vulnerabilidades. [4] O CRYPTOGUARD foi avaliado em 46 projetos Apache e 6.181 aplicativos Android, fornecendo descobertas de segurança valiosas e auxiliando projetos na melhoria de seu código. [4]

O CRYPTOGUARD utiliza algoritmos especializados de fatiamento de programa para sua análise estática. Esses algoritmos de fatiamento são implementados utilizando técnicas de análise de fluxo de dados sensíveis a fluxo, contexto e campo. Os algoritmos de fatiamento são projetados para identificar o conjunto de instruções que influenciam ou são influenciadas por uma variável de programa. [4]

Os algoritmos de fatiamento utilizados pelo CRYPTOGUARD incluem:

Fatiamento interprocedural retroativo: Este algoritmo parte de um critério de fatiamento e se propaga retroativamente pelo programa, identificando as instruções que contribuem para o valor do critério de fatiamento. Ele constrói uma coleção ordenada de instruções de todos os métodos visitados. [4]

Fatiamento retroativo intra-procedural: Semelhante ao fatiamento interprocedural retroativo, este algoritmo opera dentro de um único método. Ele identifica as instruções dentro do método que contribuem para o valor do critério de fatiamento. [4]

Fatiamento interprocedural progressivo: Este algoritmo identifica as instruções que são influenciadas por um critério de fatiamento em termos de relações de definição e uso. Ele opera nos recortes obtidos a partir do fatiamento retroativo interprocedural. [4]

Fatiamento progressivo intra-procedural: Este algoritmo é utilizado para sensibilidade de campo sob demanda de classes apenas com dados. Ele identifica as instruções dentro de um método que são influenciadas por um critério de fatiamento, especificamente para classes apenas com dados onde os campos são visíveis apenas em invocações de método ortogonais. [4]

Esses algoritmos de fatiamento permitem ao CRYPTOGUARD analisar eficientemente projetos Java de grande porte e detectar vulnerabilidades de uso indevido de APIs criptográficas e SSL/TLS. [4]

Ao usar o Cryptoguard, podemos relatar vários problemas preocupantes de codificação criptográfica em projetos de código aberto Apache e Android. Além disso, incorpora um padrão para comparar a qualidade das ferramentas de detecção de vulnerabilidades criptográficas. [4]

Em resumo o objetivo do CRYPTOGUARD é detectar vulnerabilidades criptográficas em projetos Java. [4] Ele alcança isso por meio do uso de técnicas de análise estática de programas para analisar o código e identificar possíveis usos incorretos de APIs criptográficas. [4] O CRYPTOGUARD emprega um conjunto de algoritmos de "slicing"(recorte) rápidos e altamente precisos que refinam recortes de programas ao identificar elementos irrelevantes específicos da linguagem Java. [4] Esses refinamentos auxiliam na redução significativa de alertas falsos. Ao executar o CRYPTOGUARD em projetos Java de grande escala, são gerados insights de segurança e auxilia na identificação de vulnerabilidades no código. [4]

As conclusões de segurança obtidas a partir dos testes com o CRYPTOGUARD em projetos Apache e aplicativos Android incluem o seguinte:

Projetos Apache: Dos 46 projetos Apache avaliados, 39 projetos apresentaram pelo menos um tipo de uso incorreto de criptografia, e 33 projetos tinham pelo menos dois tipos. [4] As vulnerabilidades comuns encontradas nos projetos Apache incluíam o uso de chaves previsíveis, funções hash inseguras, geradores de números aleatórios inseguros e a utilização de URLs HTTP. [4] O CRYPTOGUARD auxiliou na identificação e relato dessas vulnerabilidades para as equipes do Apache, resultando em correções rápidas em alguns casos. [4]

Aplicativos Android: A avaliação em 6.181 aplicativos Android demonstrou que cerca

de 95% das vulnerabilidades totais originaram-se de bibliotecas empacotadas com o código do aplicativo.[4] Bibliotecas de empresas como Google, Facebook, Apache, Umeng e Tencent foram identificadas com violações em diversas categorias, incluindo senhas hardcoded de keyStore e vulnerabilidades SSL/TLS.[4] O CRYPTOGUARD detectou múltiplas vulnerabilidades SSL/TLS (MitM) que a triagem automática do Google Play aparentemente deixou passar.[4]

O estudo sobre o CRYPTOGUARD, ao evidenciar a complexidade na identificação da percepção de vulnerabilidades, assume um papel catalisador para a pesquisa em questão. Ao abordar a detecção de falhas criptográficas em projetos Java, o CRYPTOGUARD não apenas destaca a necessidade premente de compreensão e correção de vulnerabilidades, mas também delineia um terreno propício para a investigação correlata. A pesquisa em pauta visa justamente tentar elucidar as origens das vulnerabilidades identificadas por meio de ferramentas análogas, oferecendo um avanço significativo no entendimento das fragilidades inerentes a sistemas criptográficos. Deste modo, o estudo sobre o CRYPTOGUARD se revela não apenas como um contributo intrínseco ao domínio da segurança cibernética, mas também como um impulso fundamental para a empreitada que visa discernir as fontes subjacentes às vulnerabilidades apresentadas por ferramentas congêneres.

2.1.10 CryptoGuard vs CrySL

A comparação é baseada na precisão e no tempo de execução das ferramentas. [4] Durante os experimentos, o CrySL travou e saiu prematuramente de 7 dos 10 subprojetos raiz do Apache selecionados aleatoriamente. [4] Para os 3 projetos concluídos, o CrySL é mais lento, mas comparável em 2 projetos (5 vs. 3 segundos, 25 vs. 19 segundos). [4] No entanto, é 3 ordens de magnitude mais lento que o Cryptoguard no codec Kerbaros. [4]

Os falsos positivos do CrySL devem-se principalmente ao fato de suas regras serem excessivamente rígidas e ele não conseguir reconhecer 4 usos corretos da API na avaliação (de 9). [4] Por outro lado, o Cryptoguard usa algoritmos de fatiamento rápidos e altamente precisos para refinar as fatias do programa e reduzir alertas falsos em até 80%. [4]

Capítulo 3

Metodologia

3.1 Hipótese de Trabalho

Ao integrar os resultados do LibScout ao contexto das ferramentas CryptoGuard e CogniCrypt, será possível não apenas detectar potenciais vulnerabilidades em APIs criptográficas, mas também identificar com precisão as correspondências associadas a bibliotecas externas, proporcionando uma abordagem mais abrangente e eficaz para a segurança de aplicações Java que utilizam operações criptográficas.

3.2 Metodologia

A metodologia abordada nestre trabalho serve como um instrumento complementar às metodologias e mecanismos descritas e abordadas no artigo 'Perceptions of Software Practitioners Regarding Crypto-API Misuses and Vulnerabilities'. [6] O trabalho realizado [6] apresenta um estudo macro em relação à percepção das vulnerabilidades dos códigos dos desenvolvedores.

A imagem acima descreve a metodologia utilizada. [6] A metodologia abordada neste trabalho entra na etapa de 'Análise de Dados' em específico 'Análise externa / nativa', onde é feita a integração dos resultados obtidos pelo LibScout com os contextos fornecidos pelo CryptoGuard e CogniCrypt. Essa integração tenta proporcionar uma visão mais minuciosa e contextualizada de onde as vulnerabilidades identificadas se encontram. Este trabalho propõe a seguinte integração:

- Coleta de Dados A metodologia adotada para a constituição do conjunto de dados envolveu uma cuidadosa seleção de aplicativos Java provenientes do renomado repositório F-Droid. [6] Este último se destaca como um catálogo de aplicativos de código aberto e livre (FOSS), especialmente concebidos para a plataforma Android.

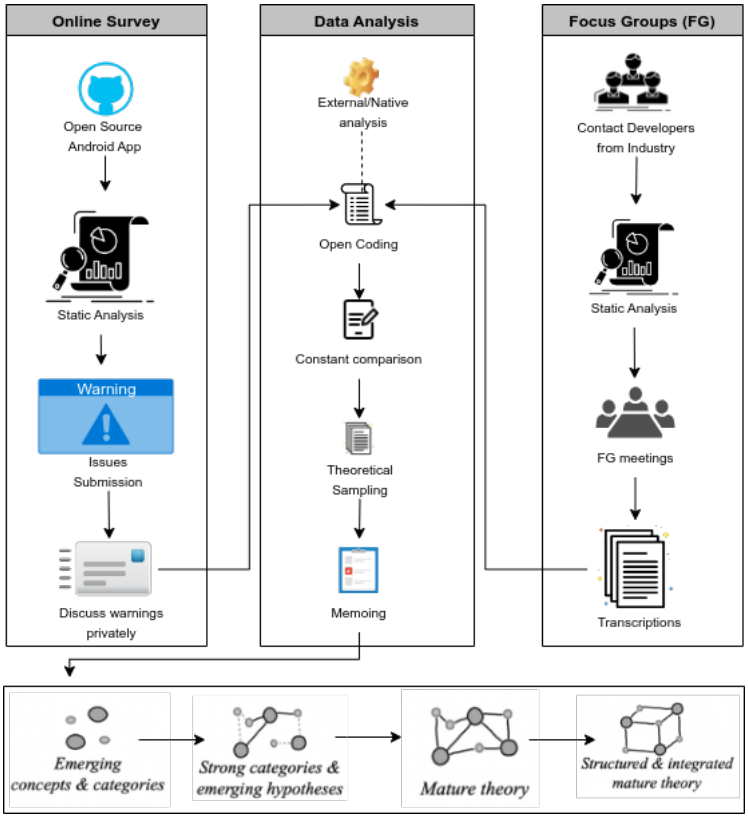


Figura 3.1: Metodologia adotada no artigo 'Perceptions of Software Practitioners Regarding Crypto-API Misuses and Vulnerabilities'

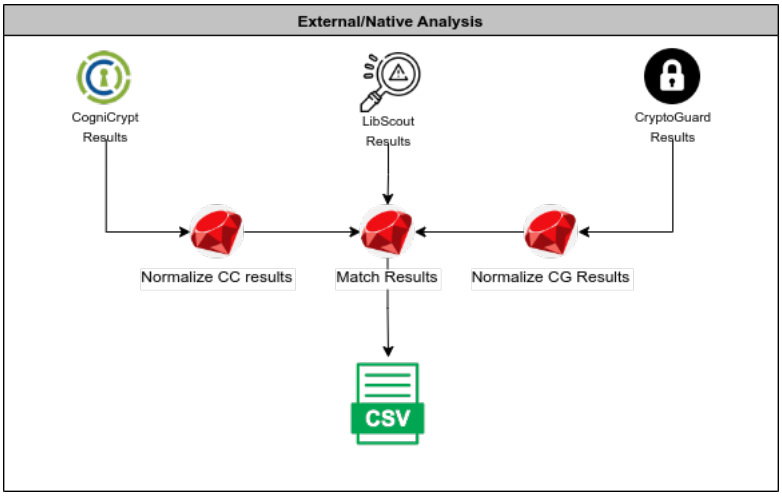


Figura 3.2: Metodologia adotada neste trabalho'

Nesse processo, buscou-se uma representativa diversidade de categorias de aplicativos, abrangendo áreas vitais como conectividade, finanças, segurança, mensagens de texto (SMS) e funcionalidades de sistema. Tal abordagem foi implementada com o intuito de assegurar uma abrangência abarcadora de contextos e finalidades, enriquecendo assim a robustez e representatividade do conjunto de dados analisado.

- Análise Estática

  A etapa subsequente consistiu na aplicação das ferramentas CryptoGuard e CogniCrypt para conduzir uma análise estática detalhada do código fonte dos aplicativos selecionados. Essa abordagem permitiu a identificação minuciosa de possíveis vulnerabilidades relacionadas às APIs criptográficas empregadas nos aplicativos avaliados. O uso dessas ferramentas especializadas proporcionou uma avaliação precisa e abrangente das práticas de segurança adotadas, visando aprimorar a integridade e robustez dos aplicativos em questão.

- Identificar a percepção de vulnerabilidade dos desenvoledores

  Após a conclusão da análise estática, foi possível identificar um conjunto de vulnerabilidades que não foram reconhecidas pelos desenvolvedores, bem como aquelas que foram identificadas, porém não receberam intervenção corretiva. Para facilitar a comunicação e o entendimento das questões de segurança identificadas, procedeu-se à criação de GISTS individuais para cada vulnerabilidade. [6] Um GIST é um recurso que permite compartilhar trechos de código, arquivos inteiros ou até mesmo aplicações, e também possibilita a preservação e compartilhamento de saída de console ao executar, depurar ou testar o código. Cada GIST representa um repositório que pode ser clonado ou bifurcado por outras pessoas, promovendo assim a colaboração e a discussão ativa em busca do aprimoramento da segurança nos aplicativos avaliados.

- Analisar origem das vulnerabilidades

  A etapa seguinte consistiu na análise da origem das vulnerabilidades identificadas. Para realizar essa análise, empregou-se a ferramenta LibScout, a qual desempenhou um papel crucial ao extrair informações detalhadas sobre as APIs criptográficas utilizadas nos aplicativos, permitindo, assim, a identificação de bibliotecas externas empregadas. A utilização do LibScout proporcionou um panorama abrangente das dependências externas dos aplicativos, fornecendo uma visão clara das fontes potenciais de vulnerabilidades no código. Esta abordagem foi essencial para direcionar os esforços na mitigação das ameaças identificadas e fortalecer a segurança das aplicações avaliadas.

  A princípio, considerou-se a utilização do LibRadar devido à sua reputação pela rapidez de execução. Contudo, logo se constatou que a ferramenta estava baseada em dados disponibilizados até 2016, o que não condizia com nossa necessidade de informações atualizadas e abrangentes sobre as bibliotecas utilizadas nos aplicativos. Diante dessa constatação, optou-se por descartar o uso do LibRadar e buscar uma alternativa mais alinhada com os objetivos do estudo.

CogniCrypt (report 1)

- Class:
- Method: md5
- Line: 351
- Issue details: ConstraintError
  - ConstraintError violating CrySL rule for java.security.MessageDigest.
  - First parameter (with value "MD5") should be any of {SHA-256, SHA-384, SHA-512}.

Code

```
public static String md5(String s) {
  try {
    // Create MD5 Hash
    MessageDigest digest = java.security.MessageDigest.getInstance("MD5");
    digest.update(s.getBytes());
    byte[] messageDigest = digest.digest();

    // Create Hex String
    StringBuilder hexString = new StringBuilder();
    for (byte b : messageDigest) hexString.append(Integer.toHexString(0xFF & b));
    return hexString.toString();

  } catch (NoSuchAlgorithmException e) {
    Timber.e(e);
  }
  return "";
}
```

Questions

1. How likely might this warning reveal a security threat to this app?
   a. Very unlikely;
   b. Unlikely;
   c. I cannot evaluate this;
   d. Likely;
   e. Very likely;
2. Are you likely to accept a patch that fixes this particular issue?

Figura 3.3: 'Exemplo de uma Gist'

- Integração de Resultados

  Foi empreendido um esforço no sentido de desenvolver um processo de integração que possibilitasse a unificação dos resultados obtidos por meio do LibScout com os contextos fornecidos pelo CryptoGuard e CogniCrypt. Essa iniciativa tenta criar uma visão mais abrangente e contextualizada das vulnerabilidades identificadas. Em paralelo, foi realizada uma avaliação da eficácia dessa abordagem, no que tange à habilidade de determinar a origem dos alertas gerados pelas mencionadas ferramentas.

Capítulo 4

Resultados

4.1 Percepção dos desenvolvedores em relação as vulnerabilidade em aplicativos open source

O artigo motivador deste trabalho [6] aborda a percepção dos desenvolvedores e profissionais de segurança em relação às vulnerabilidades em aplicativos open source. O estudo não se limitou apenas à análise estática das ferramentas utilizadas, mas também incluiu uma investigação sobre aplicativos Android de código aberto, com o objetivo de obter a opinião dos desenvolvedores envolvidos. [6]

Uma descoberta relevante foi que muitas das questões identificadas pelo CogniCrypt não estavam diretamente relacionadas ao código do aplicativo Android em si, mas sim às bibliotecas de terceiros utilizadas por esses aplicativos. [6] Dentre os projetos de código aberto analisados, aproximadamente 60% apresentaram questões levantadas pelo CogniCrypt que se originavam de códigos de terceiros. Os desenvolvedores abordaram essas preocupações de segurança de maneiras diversas. Alguns optaram por atualizar imediatamente as dependências, enquanto outros confiaram implicitamente nas grandes empresas de tecnologia fornecedoras das bibliotecas e não consideraram os problemas como ameaças reais.

Surpreendentemente, em alguns casos, os desenvolvedores tiveram dificuldades em compreender completamente as questões apresentadas pelo CogniCrypt. Para eles, não estava claro por que determinados trechos de código eram considerados problemas apenas a partir das explicações fornecidas pela ferramenta. Diante disso, os desenvolvedores expressaram o desejo de receber explicações mais detalhadas sobre os problemas identificados, bem como sugestões diretas sobre como corrigi-los. Além disso, alguns sugeriram a categorização dos problemas levantados pelo CogniCrypt com base na origem, indicando se pertencem ao aplicativo digitalizado ou a uma biblioteca de terceiros. Essa observação

proporciona insights valiosos sobre a necessidade de uma comunicação mais eficaz entre as ferramentas de análise e os desenvolvedores, visando uma compreensão mais precisa e eficiente das vulnerabilidades detectadas. [6]

4.2 Análise quantitativa em aplicativos android

Foram analisados 307 aplicativos de 6 diferentes categorias do repositório de aplicativos de código aberto F-Droid. [6] A execução do CogniCrypt reportou 195 Warnings de uso indevido de criptografia enquanto o CryptoGuard reportou 298. [6] A tabela abaixo mostra a quantidade de aplicativos analisados por categoria e a quantidade de Warnings reportados por cada ferramenta. [6]

Categoria	**Número de Aplicativos**	**CogniCrypt**	**CryptoGuard**
Connectivity	58	20	3
Finances	90	25	2
Internet	39	7	0
Security	47	16	1
Sms-Phone	18	10	2
System	55	34	0
Total	307	112	8

Tabela 4.1: Aplicativos por categoria sem warning das ferramentas CogniCrypt e CryptoGuard

Como visto, o número de aplicativos sem warning no cognicrypt é bem maior do que no cryptoguard. A categoria sistema é a que tem a maior diferença entre eles, 34. As categorias conectividade e finanças também tem um número alto de aplicativos sem warnings, 20 e 25, respectivamente. [6]

Apesar disso, o número de warnings fundados pelo CogniCrypt é maior do que no CryptoGuard, descrito na tabela abaixo.

Categoria	**CogniCrypt (a)**	**CryptoGuard (b)**	**Diferença (a - b)**
Connectivity	1768 (16.02%)	1124 (22.64%)	644 (10.61%)
Finances	3087 (27.97%)	1687 (33.98%)	1400 (23.06%)
Internet	3407 (30.87%)	916 (18.45%)	2491 (41.02%)
Security	1780 (16.13%)	553 (11.14%)	1227 (20.21%)
Sms-Phone	428 (3.88%)	171 (3.44%)	257 (4.23%)
System	566 (5.13%)	513 (10.33%)	53 (0.87%)
Total	11036 (100.00%)	4964 (100.00%)	6072 (100.00%)

Tabela 4.2: Warnings encontrados nas ferramentas CogniCrypt e CryptoGuard

Como podemos observar na tabela acima, o CogniCrypt consegue encontrar 4964 warnings enquanto o Cryptoguard encontrou 11036 warnings. A diferença numérica é

de 6072 (ou 122.32%) de warnings entre as duas ferramentas. A categoria de finanças e internet concentrou 58.8% (6494) dos warnings do CogniCrypt e as categorias de finanças e conectividade concentram 56.6% (2811) dos warnings do CryptoGuard. A maior diferença entre os warnings fundados foi notada nas categorias internet (2491) e finanças (1400), representando 64.1% de diferença. [6]

Considerando os resultados acima, analisamos a quantidade média de warnings por aplicativo. Os resultados são mostrados no gráfico abaixo.

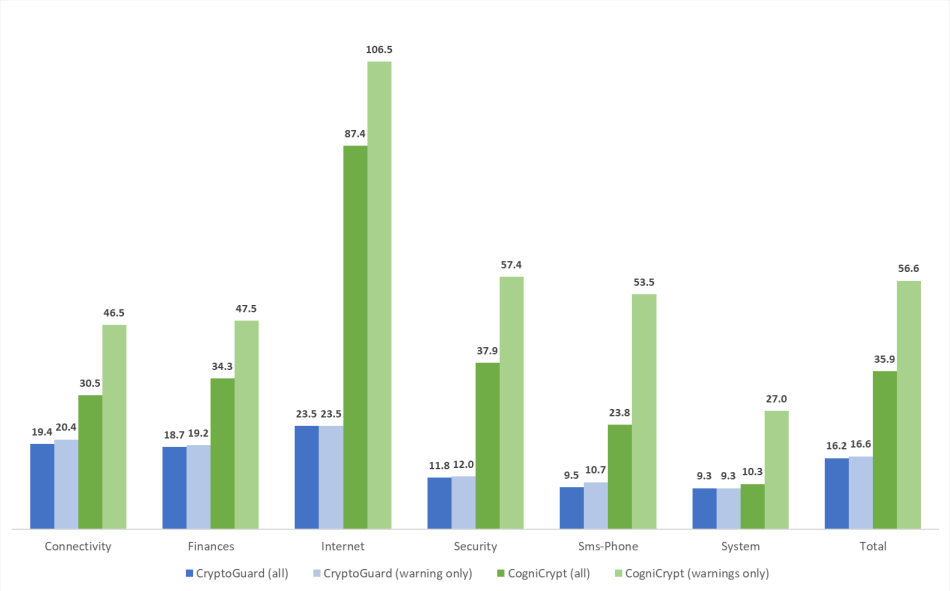


Figura 4.1: Quantidade média de alertas por aplicativo

Como esperado, a média de warnings é maior no CogniCrypt do que no CryptoGuard. A diferença é ainda maior quando desconsideramos os aplicativos sem warnings. A categoria internet tem a maior média de warnings nas duas ferramentas, com um valor maior no CogniCrypt: 106.5 warnings por aplicativo. Em segundo e terceiro lugares temos as categorias segurança e sms-telefone no CogniCrypt e internet e conectividade, considerando a média de warnings fundados pelo CryptoGuard. [6]

4.2.1 Análise qualitativa da integração do Cryptoguard e do Cognicrypt com o LibScout

A fim de facilitar a análise, os aplicativos foram categorizados. Em cada categoria, serão exibidos os resultados relativos ao número de aplicativos com alertas de vulnerabilidade, diferenciando aqueles que são potencialmente externos dos que são definitivamente externos, para cada ferramenta utilizada. As tabelas a seguir apresentam os resultados dessas integrações.

CogniCrypt

CC/Connectivity	Warnings (a)	Possible ext (b)	Definite ext (c)	Native-ext (a-b-c)
Média	21.9	2.3	2.4	17.1
Desvio Padrão	43.8	10.3	6.1	42.8
Variância	1919.8	107.8	38.2	1831.9

Tabela 4.3: Resultados da integração do CogniCrypt com o LibScout na categoria Connectivity

CC/Finances	Warnings (a)	Possible ext (b)	Definite ext (c)	Native-ext (a-b-c)
Média	46.4	2.8	7.5	35.9
Desvio Padrão	132.06	10.8	41.8	126.07
Variância	17439.9	118.2	1747.7	15895.3

Tabela 4.4: Resultados da integração do CogniCrypt com o LibScout na categoria Finances

CC/SMS	Warnings (a)	Possible ext (b)	Definite ext (c)	Native-ext (a-b-c)
Média	11.7	2.36	1	8.3
Desvio Padrão	31.67	7.78	3.69	24.1
Variância	1003.03	60.54	12.94	585.4

Tabela 4.5: Resultados da integração do CogniCrypt com o LibScout na categoria SMS

CC/System	Warnings (a)	Possible ext (b)	Definite ext (c)	Native-ext (a-b-c)
Média	10.09	0.93	0.36	8.79
Desvio Padrão	33.01	3.74	1.28	31.82
Variância	1090.2	14.05	1.66	1012.5

Tabela 4.6: Resultados da integração do CogniCrypt com o LibScout na categoria System

Os resultados para a tabela de conectividade (4.3) mostram que em média, por aplicativo, o CogniCrypt encontrou 21.9 warnings de vulnerabilidade. Desses, 2.3 são possivelmente externos e 2.4 são definitivamente externos. A quantidade de warnings nativos é de 17.1. Para finanças (4.4), a média de warnings por aplicativo é de 46.4. Desses, 2.8 são possivelmente externos e 7.5 são definitivamente externos. A quantidade de warnings nativos é de 35.9. Para SMS (4.5), a média de warnings por aplicativo é de 11.7. Desses, 2.36 são possivelmente externos e 1 é definitivamente externo. A quantidade de warnings nativos é de 8.3. E para sistema (4.6), a média de warnings por aplicativo é de 10.09. Desses, 0.93 são possivelmente externos e 0.36 são definitivamente externos. A quantidade de warnings nativos é de 8.79. Em todos os exemplos, o desvio padrão e a variância são altos, indicando que os valores estão bem dispersos. Isso é explicado tanto pelas limitações do LibScout quanto pelas limitações do CogniCrypt. [6] O LibScout

pode não ter mapeado a biblioteca com warning como externa [7] e o CogniCrypt pode ter encontrado warnings em bibliotecas que não foram mapeadas pelo LibScout ou ainda não ter encontrado vulnerabilidade no aplicativo selecionado. [6]

CryptoGuard

CG/Connectivity	Total Libraries	Possible Ext.	Definite Ext.	Native Libraries
Média	15.1	0.73	5.21	9.22
Desvio Padrão	25.62	2.17	9.91	18.7
Variância	656.6	4.71	98.3	352.4

Tabela 4.7: Resultados da integração do CryptoGuard com o LibScout na categoria Connectivity

CG/Finances	Total Libraries	Possible Ext.	Definite Ext.	Native Libraries
Média	10.45	1.33	4.68	4.43
Desvio Padrão	20.71	4.72	10.03	10.05
Variância	429.13	22.2	100.72	101.1

Tabela 4.8: Resultados da integração do CryptoGuard com o LibScout na categoria Finances

CG/SMS	Total Libraries	Possible Ext.	Definite Ext.	Native Libraries
Média	9	0.78	2.73	5.47
Desvio Padrão	18.72	2.09	5.66	16.22
Variância	350.6	4.37	32.08	263.19

Tabela 4.9: Resultados da integração do CryptoGuard com o LibScout na categoria SMS

CG/System	Total Libraries	Possible Ext.	Definite Ext.	Native Libraries
Média	7.53	0.65	2.67	4.2
Desvio Padrão	16.49	2.69	7.07	12
Variância	272.21	7.27	50.07	146.4

Tabela 4.10: Resultados da integração do CryptoGuard com o LibScout na categoria System

Analisando a tabela de conectividade (4.7) observa-se que o CryptoGuard, em média por aplicativo, detectou 15.1 alertas de vulnerabilidade, dos quais 0.73 são potencialmente externos e 5.21 definitivamente externos, com 9.22 alertas nativos. Na categoria de finanças (4.8), a média foi de 10.45 alertas por aplicativo, com 1.33 potencialmente externos e 4.68 definitivamente externos, além de 4.43 nativos. Para SMS, (4.9), a média foi de 9 alertas, com 0.78 potencialmente externos e 2.73 definitivamente externos, e 5.47 nativos. E para sistema (4.10), a média foi de 7.53 alertas, com 0.65 potencialmente externos e 2.67 definitivamente externos, além de 4.2 nativos. Os altos desvios padrão e variância

indicam uma dispersão significativa, influenciada pelas limitações do LibScout e do CryptoGuard. [6] Novamente o LibScout pode não ter mapeado a biblioteca com warning como externa [7] e o CryptoGuard pode ter encontrado warnings em bibliotecas que não foram mapeadas pelo LibScout ou ainda não ter encontrado vulnerabilidade no aplicativo selecionado. [6]

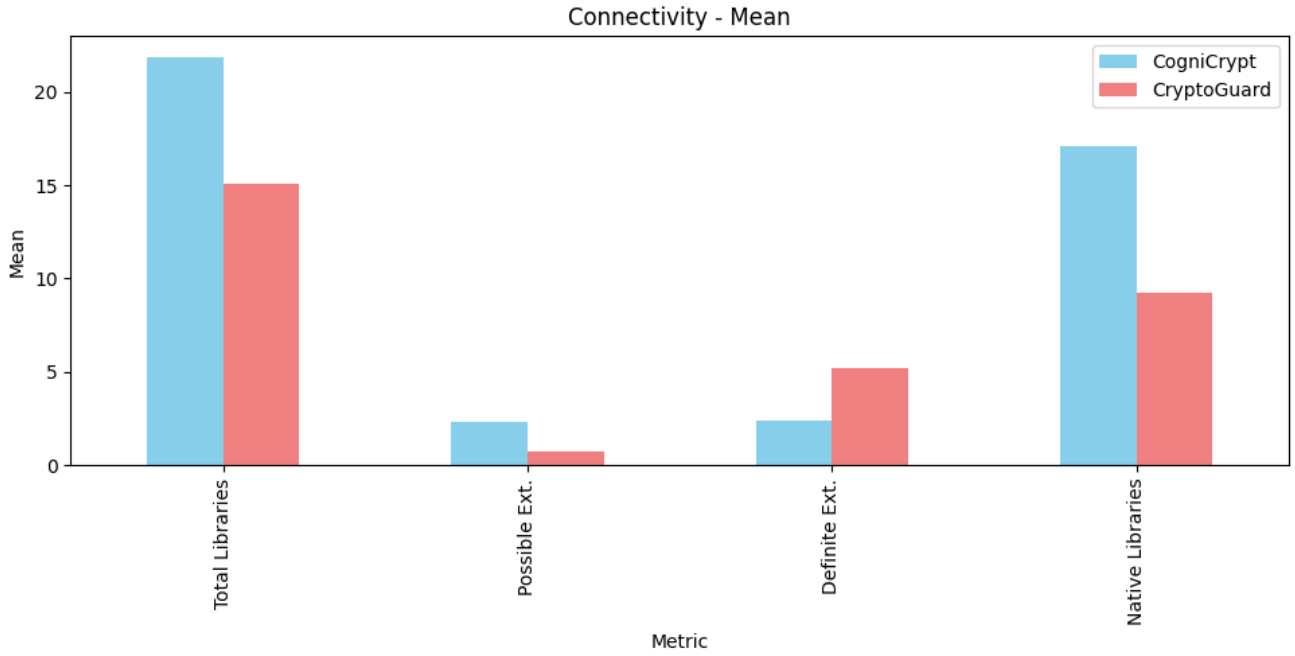
Comparação entre as ferramentas

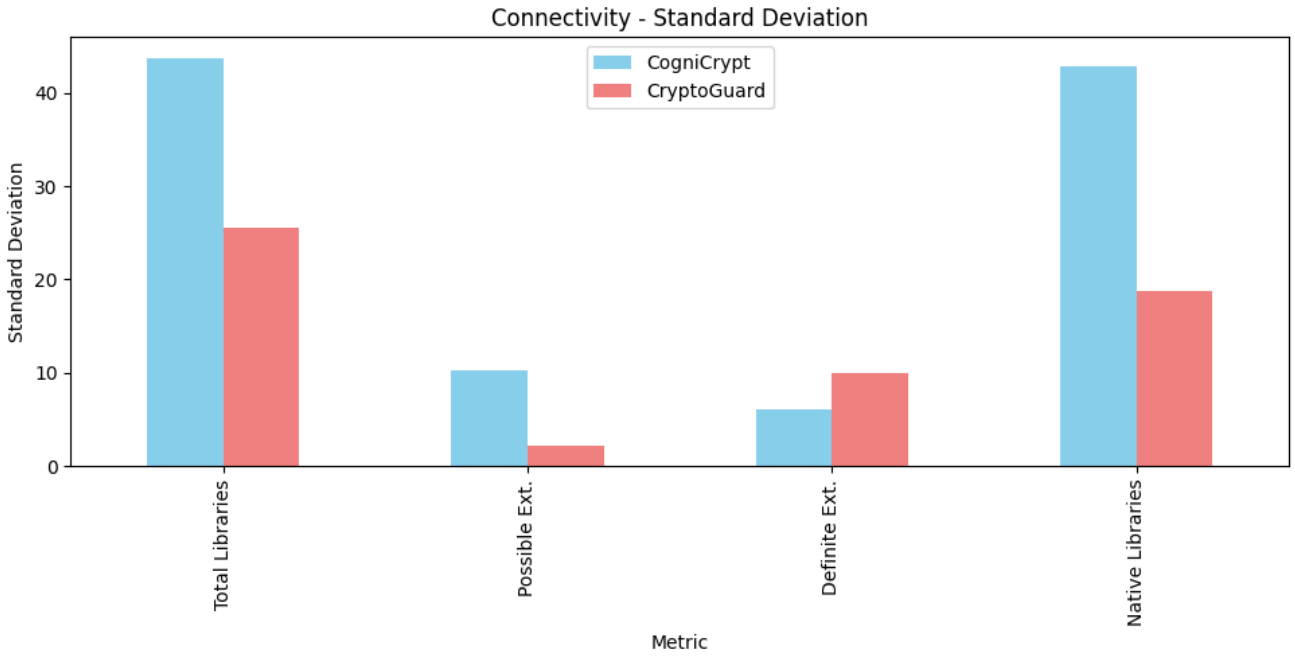
As figuras subsequentes ilustram uma análise comparativa categorizada entre as ferramentas CogniCrypt e CryptoGuard

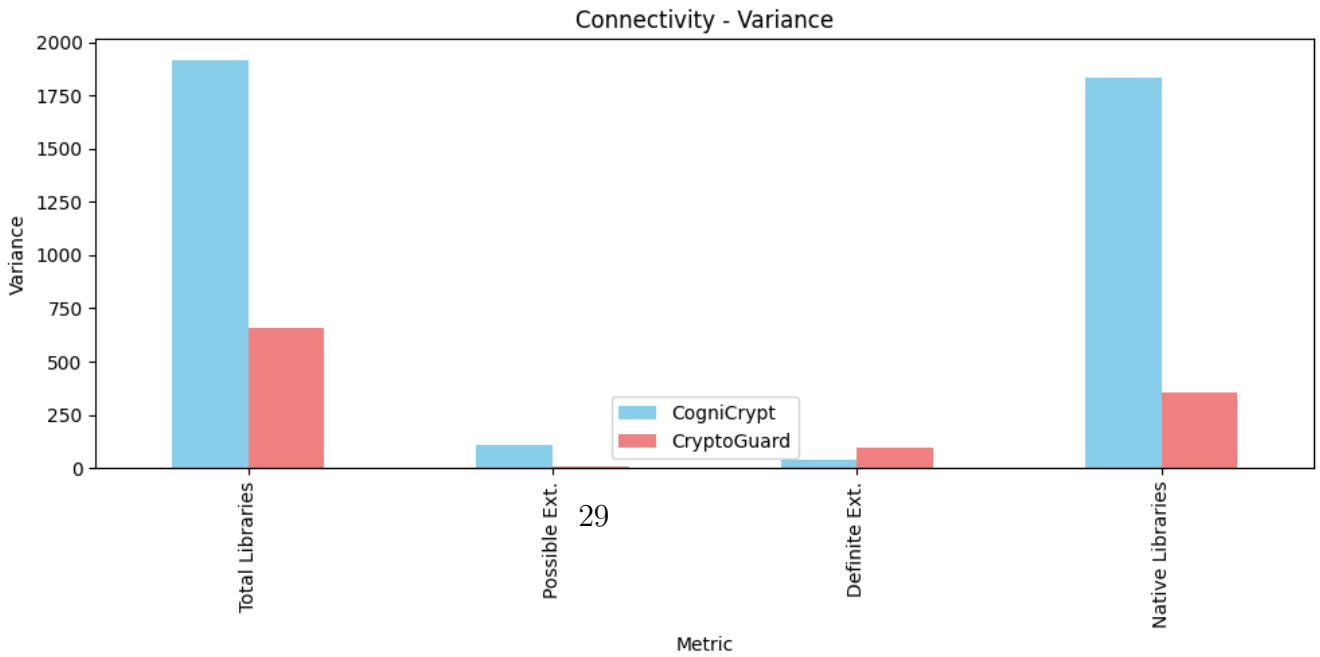
Na categoria de conectividade (4.2), a média de alertas emitidos pelas ferramentas indica que o CogniCrypt gera um número maior de alertas por aplicativo, incluindo alertas nativos e potencialmente externos. Contudo, o CryptoGuard excede no número de alertas definitivamente classificados como externos. O CogniCrypt apresenta um desvio padrão e uma variância superiores, com exceção da quantidade de alertas de bibliotecas categorizadas como definitivamente externas. A eficiência na integração com o LibScout, para a identificação de bibliotecas definitivamente externas, é mais pronunciada no CryptoGuard.

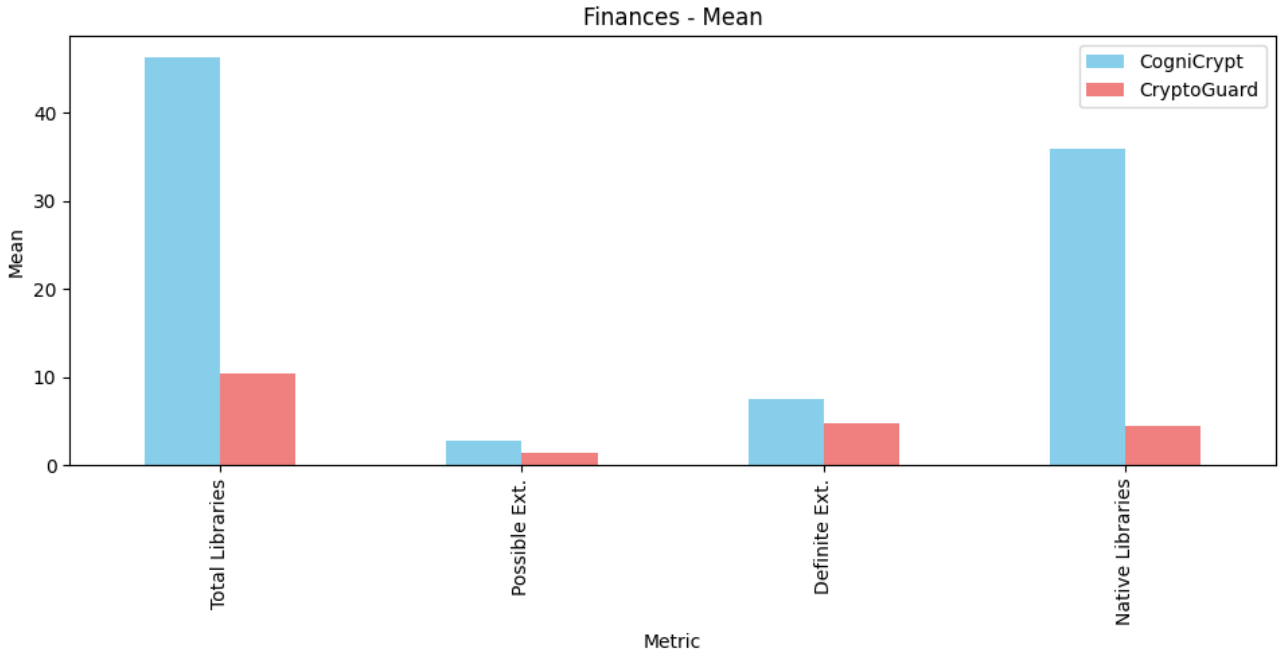
Na categoria financeira (4.3), o CogniCrypt demonstrou superioridade em relação ao CryptoGuard. Isso indica que o CryptoGuard tem uma dispersão maior nos valores relativos aos alertas por aplicativo e aos alertas de bibliotecas nativas, em contraste com a maior dispersão do CogniCrypt nos valores de alertas possivelmente de bibliotecas externas e definitivamente externas.

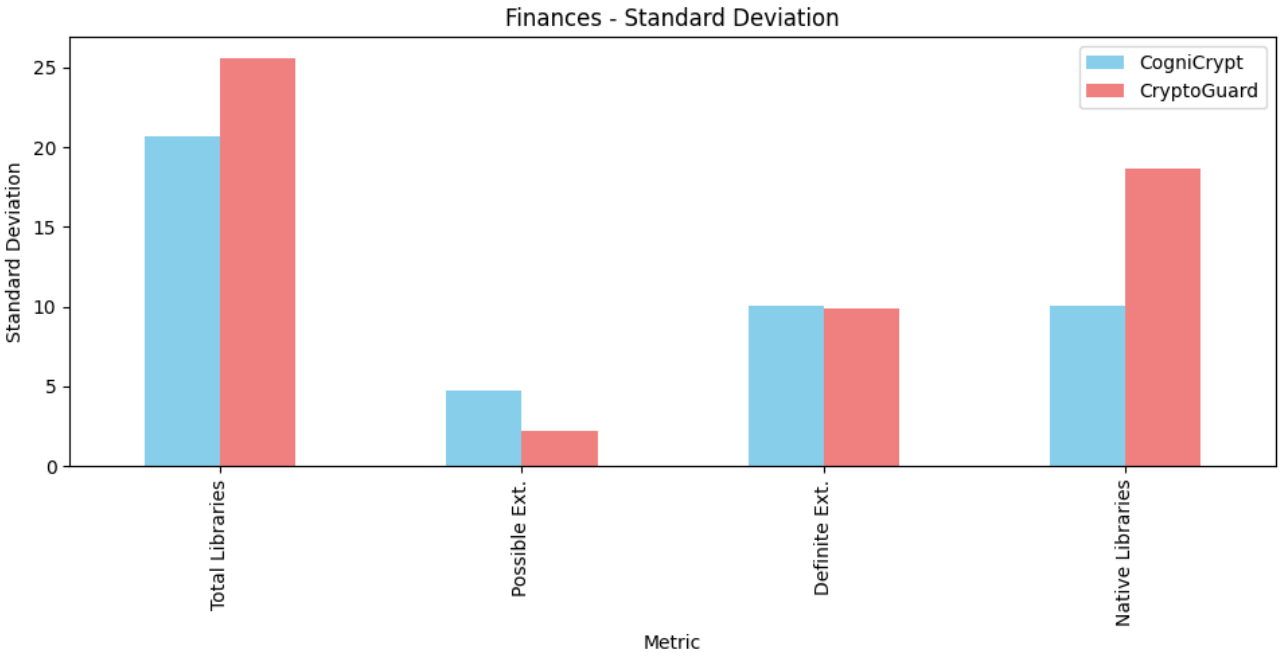
Nas categorias de SMS (4.4) e Sistemas (4.5), observa-se um padrão análogo ao da categoria de conectividade. O CogniCrypt gera uma quantidade superior de alertas por aplicativo, incluindo uma maior frequência de alertas nativos e potencialmente externos. Em contrapartida, o CryptoGuard excede no número de alertas categorizados como definitivamente externos. Esta tendência também se reflete no desvio padrão e na variância, onde o CogniCrypt mostra maior dispersão de dados, à exceção dos alertas definitivamente externos.

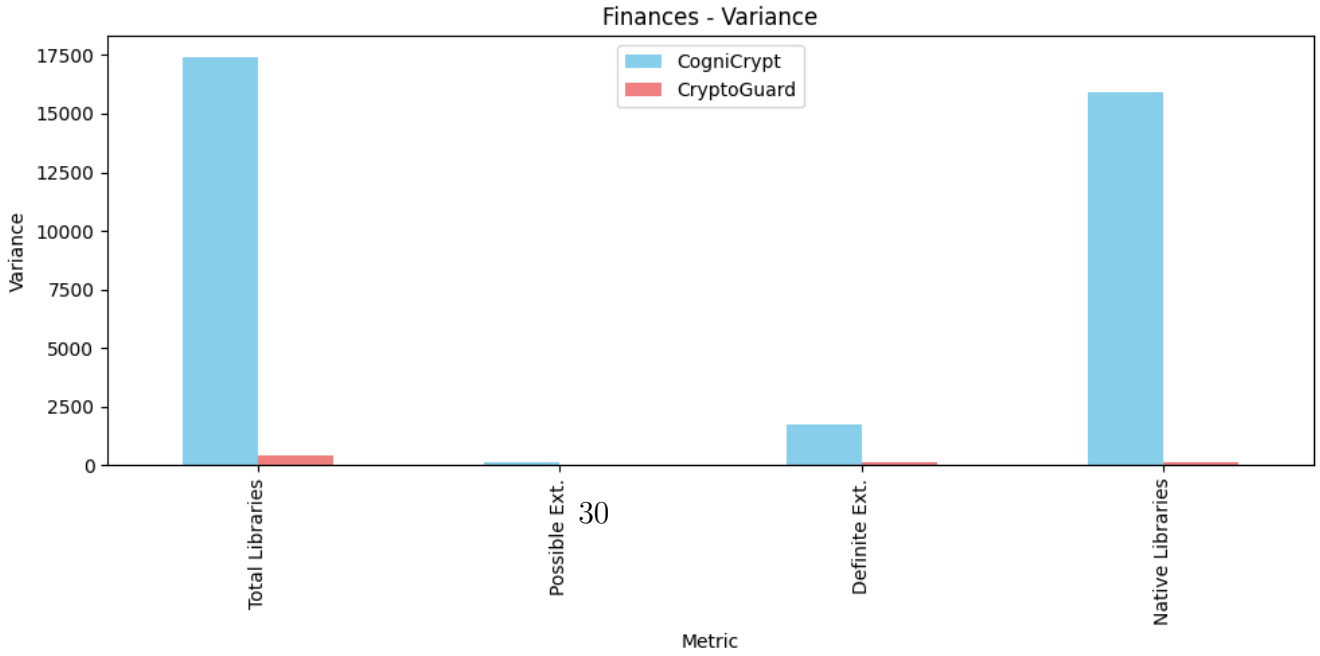

Connectivity - Mean
CogniCrypt
CryptoGuard
Mean
0
5
10
15
20
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

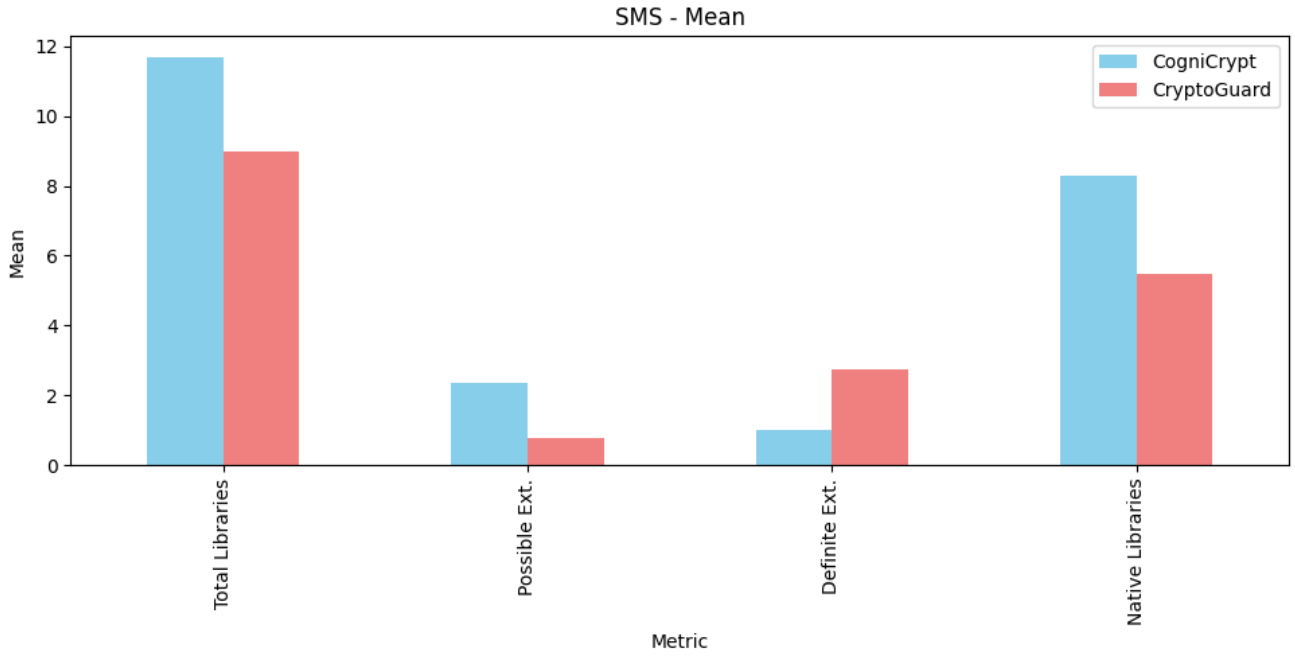

Connectivity - Standard Deviation
CogniCrypt
CryptoGuard
Standard Deviation
0
10
20
30
40
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

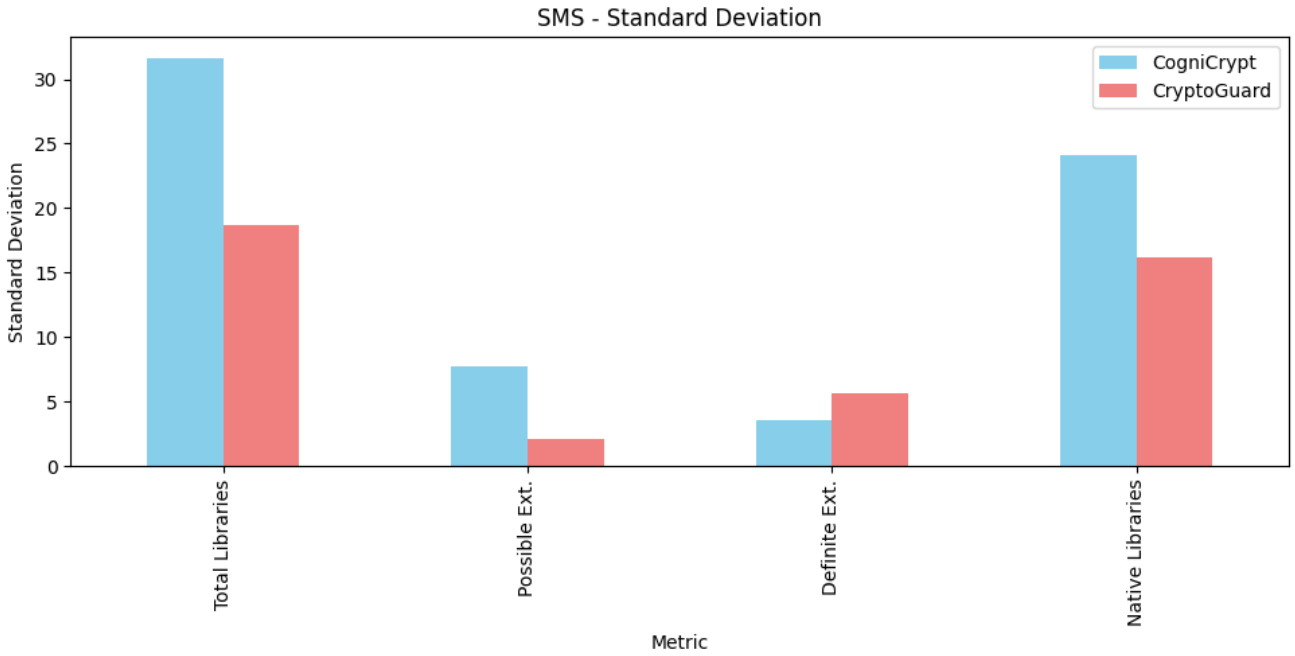

Connectivity - Variance
CogniCrypt
CryptoGuard
Variance
0
250
500
750
1000
1250
1500
1750
2000
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

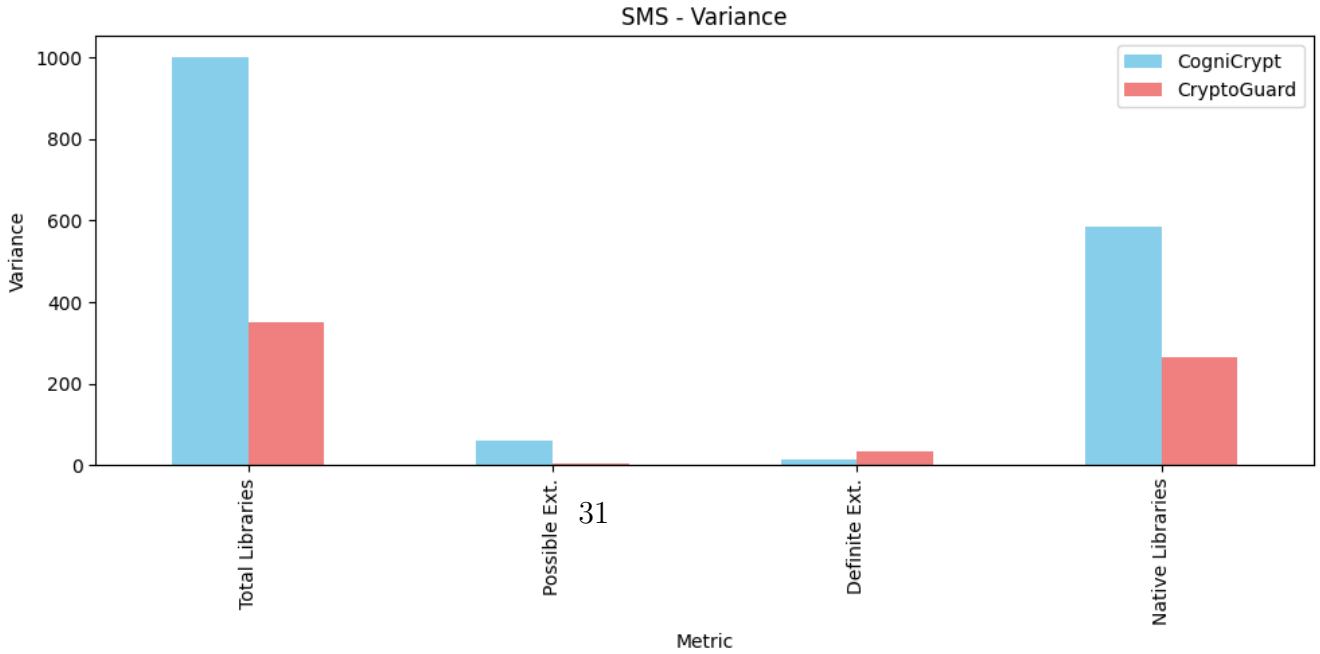

Finances - Mean
CogniCrypt
CryptoGuard
Mean
0
10
20
30
40
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

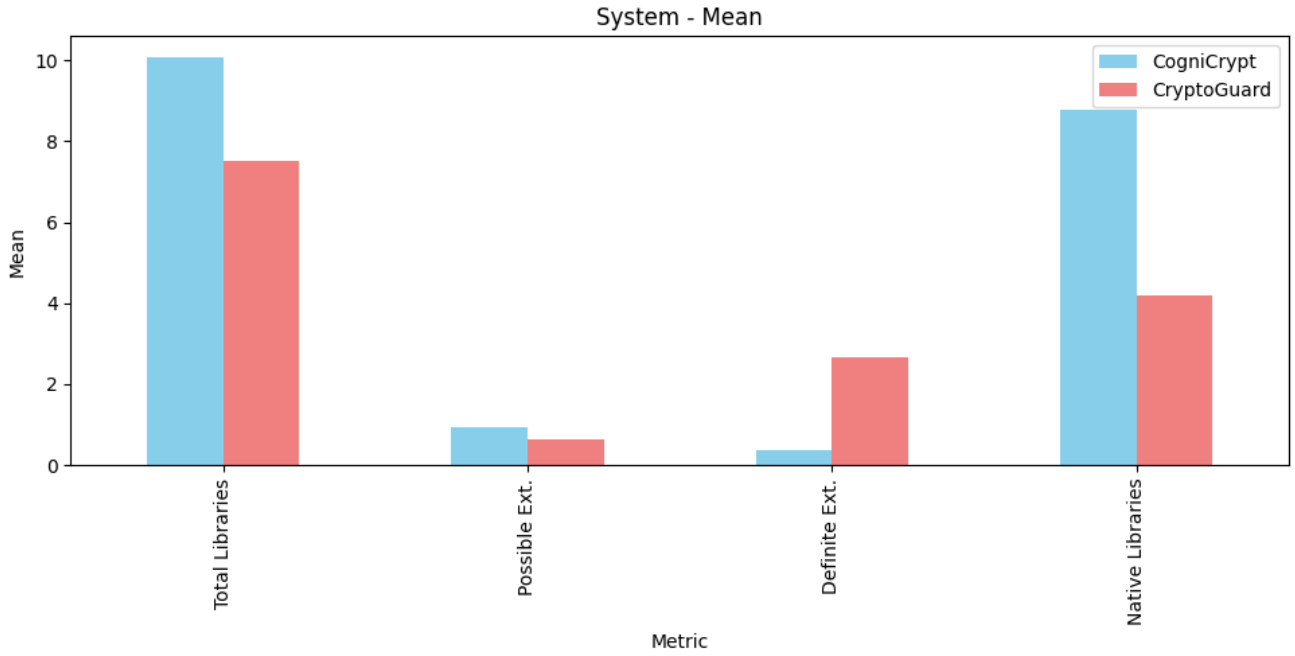

Finances - Standard Deviation
CogniCrypt
CryptoGuard
Standard Deviation
0
5
10
15
20
25
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

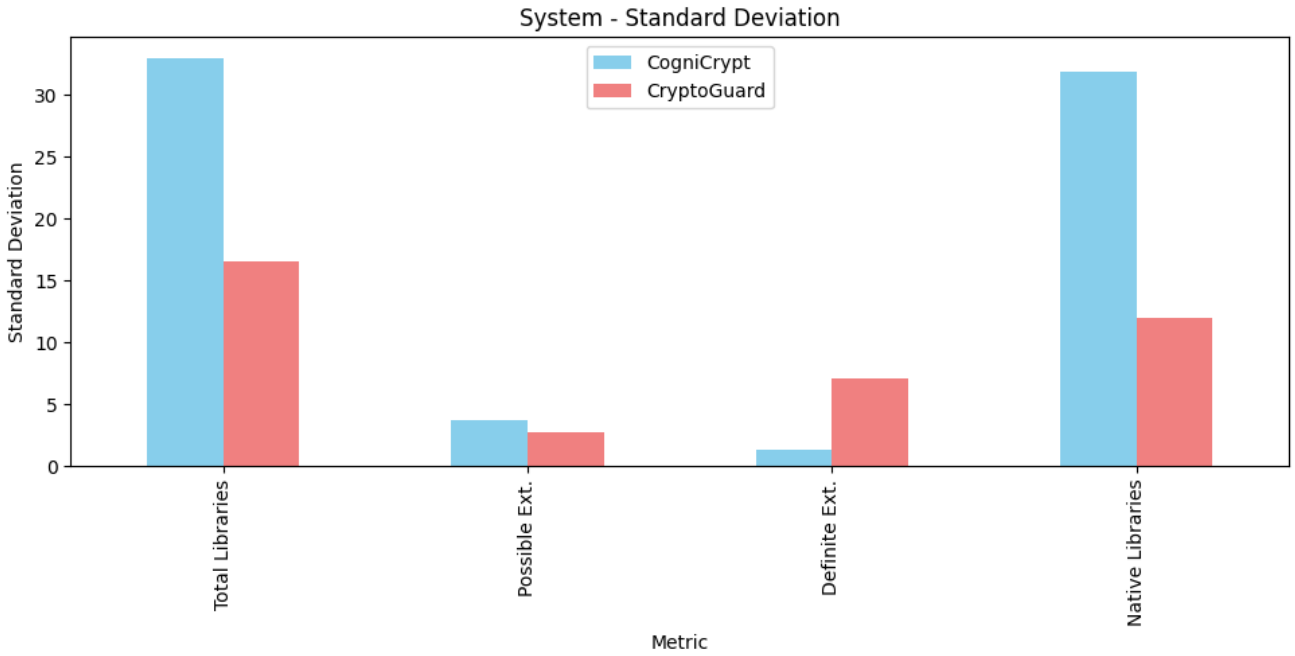

Finances - Variance
CogniCrypt
CryptoGuard
Variance
0
2500
5000
7500
10000
12500
15000
17500
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

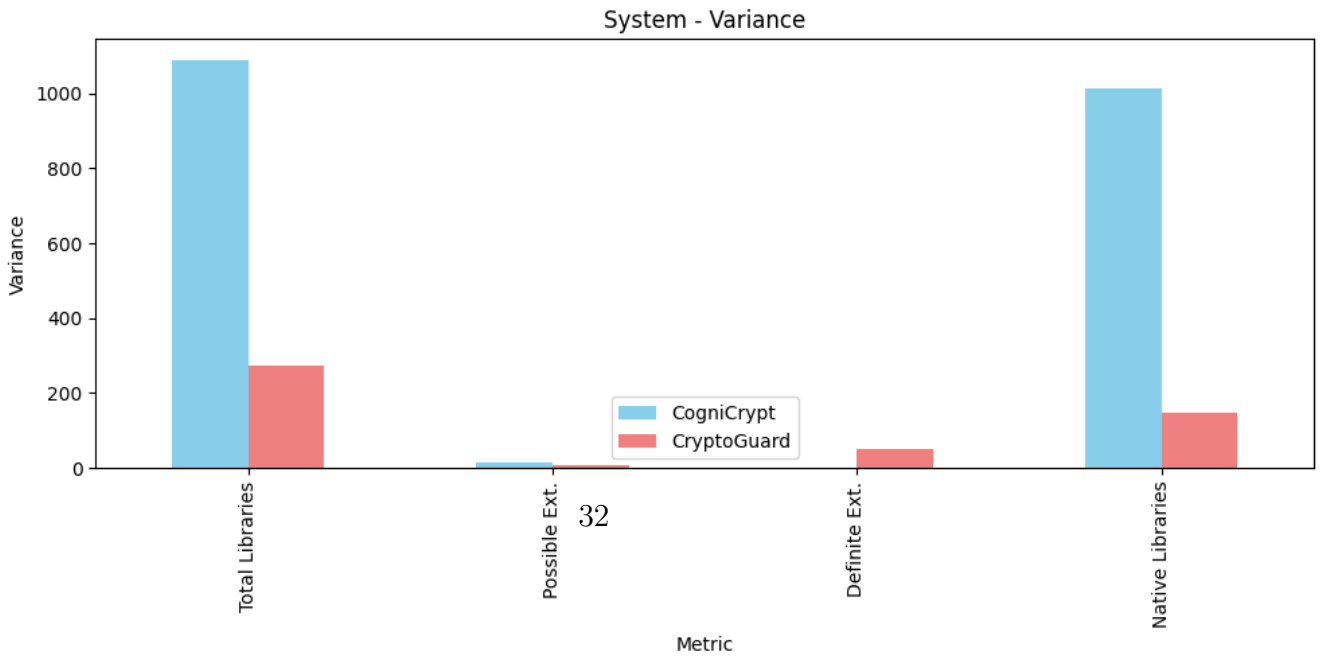

SMS - Mean
CogniCrypt
CryptoGuard
Mean
0
2
4
6
8
10
12
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric


SMS - Standard Deviation
CogniCrypt
CryptoGuard
Standard Deviation
0
5
10
15
20
25
30
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric


SMS - Variance
CogniCrypt
CryptoGuard
Variance
0
200
400
600
800
1000
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric


System - Mean
CogniCrypt
CryptoGuard
Mean
0
2
4
6
8
10
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric


System - Standard Deviation
CogniCrypt
CryptoGuard
Standard Deviation
0
5
10
15
20
25
30
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric


System - Variance
CogniCrypt
CryptoGuard
Variance
0
200
400
600
800
1000
Total Libraries
Possible Ext.
Definite Ext.
Native Libraries
Metric

Capítulo 5

Conclusão

5.1 Conclusão

Este estudo se concentrou na avaliação comparativa entre as ferramentas CogniCrypt, CryptoGuard e LibScout para melhorar a detecção de vulnerabilidades em APIs criptográficas Java. As descobertas evidenciam que a integração destas ferramentas aprimora significativamente a precisão e a eficácia na identificação de falhas de segurança, oferecendo uma abordagem holística e mais robusta para a segurança de aplicações Java.

A ferramenta CogniCrypt identificou menos bibliotecas externas em comparação com o CryptoGuard, porém, ambas as ferramentas trouxeram resultados favoráveis para a detecção da origem do código malicioso.

As implicações destas descobertas podem ser úteis para a comunidade de desenvolvedores Java. A utilização integrada destas ferramentas pode contribuir para as práticas atuais de desenvolvimento seguro, permitindo uma identificação mais rápida e precisa de vulnerabilidades. Isso não apenas melhora a segurança das aplicações, mas também otimiza o processo de desenvolvimento, economizando tempo e recursos. [6]

O estudo enfrentou limitações, como a complexidade na análise de código obfuscado [2], que impactam a eficácia das ferramentas. Estas limitações destacam a necessidade contínua de aprimoramento na tecnologia de detecção de vulnerabilidades, reforçando a importância de abordagens adaptativas e inovadoras na segurança cibernética.

Para pesquisas futuras, sugere-se o desenvolvimento de metodologias mais avançadas e aprimoramento das ferramentas existentes para abordar novos desafios de segurança. A expansão do escopo para outras linguagens de programação e plataformas pode oferecer uma contribuição mais abrangente para a segurança de aplicações. Também é recomendado um set de dados mais amplo e diversificado para avaliar a eficácia das ferramentas.

A segurança em APIs criptográficas Java é de suma importância no cenário digital atual. Este estudo contribui para este campo, oferecendo insights valiosos e abrindo

caminho para futuras inovações. A necessidade de pesquisa contínua e desenvolvimento de novas soluções de segurança é clara, dada a evolução constante das ameaças cibernéticas.

Referências

[1] Richards, Kathleen: *Cryptography.* `https://www.techtarget.com/searchsecurity/definition/cryptography`, acesso em 2023-10-03. 1, 4

[2] Zhang, Ying, Md Mahir Asef Kabir, Ya Xiao, Danfeng Yao e Na Meng: *Automatic detection of java cryptographic api misuses: Are we there yet?* IEEE Transactions on Software Engineering, 49(1):288–303, 2023. 1, 5, 6, 7, 8, 33

[3] Krüger, Stefan, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler e Ram Kamath: *Cognicrypt: Supporting developers in using cryptography.* Em *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, páginas 931–936, Oct 2017. 1, 13, 14, 15, 16

[4] Rahaman, Sazzadur, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Danfeng, Yao e Murat Kantarcioglu: *Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects*, 2019. 1, 16, 17, 18

[5] Zhan, Xian, Lingling Fan, Tianming Liu, Sen Chen, Li Li, Haoyu Wang, Yifei Xu, Xiapu Luo e Yang Liu: *Automated third-party library detection for android applications: Are we there yet?* Em *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, página 919–930, New York, NY, USA, 2021. Association for Computing Machinery, ISBN 9781450367684. `https://doi.org/10.1145/3324884.3416582`. 1, 8, 9, 10, 12

[6] Bonifacio, Amaral, Monteiro: *Perceptions of software practitioners regarding crypto-api misuses and vulnerabilities.* IEEE Transactions on Software Engineering, 49, 2023. 1, 2, 19, 21, 23, 24, 25, 26, 27, 28, 33

[7] Derr, Erik: *Libscout.* `https://github.com/reddr/LibScout`, 2019. 2, 11, 12, 27, 28

[8] Torres, A., P. Costa, L. Amaral, J. Pastro, R. Bonifacio, M. dx27;Amorim, O. Legunsen, E. Bodden e E. Dias Canedo: *Runtime verification of crypto apis: An empirical study.* IEEE Transactions on Software Engineering, (01):1–16, aug 5555, ISSN 1939-3520. 5