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
2024

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, 23 de Junho de 2024

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 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 a ferramenta LibScout, permitindo a identificação da origem de warnings relacionados a bibliotecas externas. Essa abordagem qualitativa representa um avanço 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. Após a análise de mais de 200 aplicativos Android, foi possível observar que para a ferramenta Cognicrypt tivemos 6798 warnings totais com 2436 pertecendo a bibliotecas externas e para o Cryptoguard tivemos 2710 warnings totais com 1394 pertecendo a bibliotecas externas.

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

Abstract

This study presents an approach to enhance the detection of vulnerabilities in Java cryptographic APIs, aiming to strengthen the security of applications based on this technology. For this purpose, we integrated the CogniCrypt and CryptoGuard tools with LibScout, allowing the identification of the origin of warnings related to external libraries. This qualitative approach represents an advancement in promoting security in Java applications, contributing to a more resilient digital ecosystem protected against potential cyber threats. By incorporating the identification of the origin of warnings, we also enable direct suggestions to the developers of the libraries, optimizing the vulnerability correction process. After analyzing more than 200 Android applications, we observed that for the CogniCrypt tool, we had a total of 6,798 warnings, with 2,436 belonging to external libraries, and for CryptoGuard, we had a total of 2,710 warnings, with 1,394 belonging to external libraries.

Keywords: CogniCrypt, Eclipse, Code Security, Analisys

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Introdução

A criptografia é uma disciplina essencial da segurança da informação, fundamental para proteger sistemas digitais e dados sensíveis de ameaças cibernéticas [1]. Com a crescente complexidade das aplicações e a variedade de bibliotecas e frameworks disponíveis, o desenvolvimento de ferramentas automatizadas para detectar falhas e vulnerabilidades nas interfaces de programação de aplicações (APIs) criptográficas tornou-se crucial [2]. No entanto, muitos desenvolvedores enfrentam dificuldades significativas ao utilizar essas APIs devido à sua complexidade e documentação inadequada [3].

O estudo de Nadi et al. [3] destacou que os desenvolvedores frequentemente recorrem a fontes como StackOverflow para resolver problemas relacionados à criptografia, pois as APIs não são bem documentadas e frequentemente são complexas demais para o uso direto. A pesquisa também indicou que muitos exemplos disponíveis online contêm erros, levando os desenvolvedores a implementar soluções inadequadas. Nadi e seus colegas realizaram uma investigação empírica que incluiu a análise de 100 posts no StackOverflow, 100 repositórios no GitHub e respostas de 48 desenvolvedores a uma pesquisa. Eles descobriram que, enquanto os desenvolvedores se sentem confiantes na seleção dos conceitos de criptografia corretos, eles enfrentam dificuldades significativas ao usar as APIs de criptografia do Java devido à sua complexidade e ao baixo nível de abstração oferecido [3].

Para mitigar essas dificuldades, foram desenvolvidas várias ferramentas de criptografia. A linguagem CrySL, por exemplo, permite a definição precisa de regras para o uso seguro de APIs criptográficas em código Java [4], estabelecendo padrões rigorosos para a implementação de métodos de criptografia seguros. Ferramentas como CogniCrypt e CryptoGuard ainda apresentam limitações em sua precisão e não identificam a origem dos alertas, sejam eles de bibliotecas nativas ou externas. Atualmente, um estudo está

em desenvolvimento para investigar mais a fundo as percepções dos desenvolvedores sobre essas limitações, bem como as dificuldades enfrentadas ao lidar com vulnerabilidades em APIs criptográficas.

Neste contexto, o presente estudo realiza uma análise qualitativa abrangente da detecção de vulnerabilidades em APIs criptográficas, utilizando as ferramentas CogniCrypt [4] e CryptoGuard [5], integradas com os resultados do LibScout, uma ferramenta desenvolvida para o mapeamento de bibliotecas de um aplicativo [6]. Esta abordagem permitiu não apenas a detecção de potenciais vulnerabilidades, mas também a identificação precisa de correspondências associadas a bibliotecas externas.

Durante este estudo, encontramos a complexidade inerente à análise de código ofuscado, o que ressaltou a importância de considerar uma variedade de contextos de implementação ao avaliar a eficácia das ferramentas de detecção de vulnerabilidades [7]. Observou-se também que as ferramentas CogniCrypt e CryptoGuard, apesar de sua capacidade de detectar possíveis vulnerabilidades, não identificam a origem dos alertas. Essa limitação pode resultar em falsos positivos ou negligenciar alertas críticos provenientes de bibliotecas essenciais. Para superar esse desafio, utilizamos o estudo "Automated Third-Party Library Detection for Android Applications: Are We There Yet?"[7], propondo a integração dos resultados do LibScout ao contexto do CryptoGuard e CogniCrypt. Esta abordagem permitiu a detecção precisa de correspondências associadas a bibliotecas externas, resultando na criação de uma flag adicional, "external_library", para sinalizar a presença de uma biblioteca externa quando uma correspondência era identificada.

Também mapeamos as bibliotecas encontradas nos resultados do LibScout [6], identificando tanto as bibliotecas externas quanto aquelas que poderiam ser externas, resultando na flag "possible_external". No entanto, enfrentamos desafios ao usar o LibScout, como limitações na definição de clusters com diferentes graus de granularidade e a atualização do conjunto de dados, que pode afetar a precisão das correspondências. A inclusão desses recursos aumentou a precisão da detecção de falhas e abriu novas perspectivas. Agora podemos fornecer recomendações diretamente aos desenvolvedores das bibliotecas, permitindo uma intervenção mais direta e eficaz na resolução de vulnerabilidades. Este trabalho representa um avanço significativo na promoção da segurança de aplicações Java, visando 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 desta abordagem qualitativa e da integração de ferramentas de detecção.

1.2 Objetivos

O objetivo geral deste estudo é adaptar as ferramentas de análise estática para identificar se os warnings são provenientes do código de origem da aplicação ou de bibliotecas externas. Para alcançar este objetivo, diversas etapas foram delineadas:

- **Revisão da literatura**: Identificar métodos e abordagens existentes para a detecção de vulnerabilidades em bibliotecas externas.
- **Comparação de ferramentas**: Avaliar a eficácia das ferramentas CogniCrypt e CryptoGuard na detecção de vulnerabilidades em APIs criptográficas.
- **Integração dos resultados à saída**: Incorporar os resultados obtidos do LibScout ao contexto do CogniCrypt e CryptoGuard.
- **Estudo empírico**: Conduzir uma análise empírica para estimar a quantidade e origem dos warnings detectados.

Inicialmente, o objetivo do estudo era fornecer aos desenvolvedores uma forma de identificar vulnerabilidades em APIs criptográficas. No entanto, percebeu-se que a detecção de vulnerabilidades utilizando apenas CogniCrypt e CryptoGuard não era suficiente para identificar a origem dos alertas. Assim, o objetivo foi ampliado para incluir a identificação precisa da origem dos alertas, o que é crucial para os desenvolvedores, pois possibilita ações direcionadas e específicas para corrigir vulnerabilidades, economizando tempo e recursos valiosos no processo de desenvolvimento e garantindo a segurança efetiva das aplicações.

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. À medida que as aplicações se tornam mais sofisticadas e interconectadas, a superfície de ataque aumenta, expondo dados sensíveis a ameaças cibernéticas cada vez mais avançadas. Diversos estudos demonstraram que o uso inadequado dessas APIs é uma das principais fontes de vulnerabilidades em software, resultando em falhas de segurança que podem ser exploradas por atacantes [2, 3].

A segurança das APIs criptográficas é particularmente crucial, pois elas são frequentemente utilizadas para proteger dados confidenciais e comunicações seguras. No entanto, devido à complexidade inerente dessas APIs e à documentação muitas vezes inadequada,

os desenvolvedores enfrentam desafios significativos para usá-las corretamente [3]. O estudo de Nadi et al. destacou que a complexidade das APIs e a falta de abstração adequada dificultam o uso correto por parte dos desenvolvedores, levando a implementações inseguras [3].

Além disso, muitos desenvolvedores não estão cientes se os *warnings* são provenientes de código nativo ou de bibliotecas externas. Em muitos casos, esses *warnings* podem ser de bibliotecas externas, e os desenvolvedores podem não dar prioridade a esses problemas, o que pode comprometer a segurança da aplicação como um todo.

Diante desse cenário, a presente pesquisa se propõe a aprimorar a detecção de vulnerabilidades em APIs criptográficas, proporcionando um melhor entendimento sobre a origem das vulnerabilidades. 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 de fortalecer a segurança das aplicações Java em um contexto onde as ameaças cibernéticas estão em constante evolução. A contribuição da abordagem proposta, que inclui a identificação da origem dos alertas de vulnerabilidade, oferece uma ferramenta poderosa para os desenvolvedores, permitindo uma resposta mais rápida e precisa a problemas de segurança.

Capítulo 2

Trabalhos Correlatos e Revisão de Literatura

2.1 Trabalhos Correlatos e Revisão de Literatura

2.1.1 Criptografia

O conceito do dicionário de criptografia é a prática de proteger informações e comunicações por meio do uso de códigos, hashes, assinaturas, para que apenas aqueles para quem as informações são destinadas possam lê-las e processá-las. No contexto de ciência da computação, de acordo com Kathleen Richards [1] 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. A criptografia moderna preocupa-se com quatro objetivos principais: confidencialidade, integridade, não repúdio e autenticação.

- confidencialidade. A informação não pode ser entendida por ninguém além de quem a mensagem foi destina.
- Integridade. A informação não pode ser alterada de forma alguma entre o remetente e o destinatário sem que isso seja detectado.
- Não Repudiação. O remetente não pode negar o envio da informação.
- Autenticação. O remetente e o destinatário devem poder confirmar a identidade de cada um.

Existem diversos algoritmos de criptografia, dentre eles podemos citar o Advanced Encryption Standard (AES), o RSA, o Elliptic Curve Digital Signature Algorithm (ECDSA) e o Digital Signature Algorithm (DSA) [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. [8] [2] 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.[2]

2.1.2 Análise dinâmica de APIs criptográficas

O estudo realizado por Torres et al. [9] fornece uma importante 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 RV-Sec), juntamente com notáveis analisadores estáticos como CogniCrypt e CryptoGuard, além da ferramenta CryLogger.

Este estudo desempenha um papel complementar para a nossa própria investigação. Ele não apenas fornece uma visão sobre as técnicas de detecção de vulnerabilidades em APIs criptográficas, mas também motiva a explorar uma perspectiva complementar em relação as ferramentas de análises estáticas.

2.1.3 Detecção de vulnerabilidades em APIs criptográficas Java

O 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.

Na análise feita por Zhang, é destacado questões como a complexidade das APIs, a falta de documentação adequada e a falta de treinamento em segurança cibernética por parte dos desenvolvedores [2] como as principais causas de vulnerabilidades nos códigos.

Ele ainda destaca que 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.

Nos exemplos citados no artigo temos os desenvolvedores não estarem completamente cientes dos valores adequados para serem utilizados como parâmetros de funções criptográficas, das sequências de chamadas corretas ou da lógica de substituição, o que pode resultar na implementação insegura de funcionalidades de segurança [2], a prática de

copiar e colar trechos de código de fontes online, como o StackOverflow sem ter total entendimento do que está sendo copiado.

O estudo tenta identificar a percepção dos desenvoledores em relação a vulnerabilidades em seus códigos.

Houveram visões divergentes em relação às ferramentas existentes para detectar usos incorretos de APIs.

Dos feedbacks recebidos, a maioria dos desenvolvedores rejeitou as vulnerabilidades relatadas, indicando que não as consideravam válidas ou relevantes. Menos desenvolvedores demonstraram disposição para abordar os problemas reportados e fazer as correções necessárias. Ainda menos desenvolvedores efetivamente substituíram os usos incorretos de APIs com base nas orientações fornecidas pelas ferramentas [2].

Os fatores que contribuíram para a relutância dos desenvolvedores em lidar com os problemas reportados incluem:

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.

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.

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.

Assim, nosso estudo se fundamenta na análise de Zhang, e visa compreender se as vulnerabilidades identificadas pelas ferramentas CryptoGuard e CogniCrypt têm sua origem associada a bibliotecas de caráter nativo ou externo.

2.1.4 Detecção de bibliotecas externas em aplicações Android

Em outro artigo do Zhang, é realizado um estudo com foco na detecção de bibliotecas externas em aplicações android. [7] Esse estudo é importante para o nosso trabalho pois ele aborda a detecção de bibliotecas externas em aplicações Android, e a partir dele

foi possível identificar a ferramenta LibScout, que será utilizada para a identificação de bibliotecas externas em aplicações Android.

Durante o artigo é destacado que muitas vezes a introdução de bibliotecas de terceiros podem resultar em introdução de vulnerabilidades das quais o desenvolvedor nem está ciente.

O estudo foi realizado com várias ferramentas para detectar bibliotecas de terceiros (TPLs) em aplicativos Android, temos:

- LibID, que utiliza análise estática e dinâmica para identificar TPLs.
- LibPecker concentra-se na detecção de TPLs ofuscadas por diferentes técnicas de ofuscação de código.
- ORLIS que emprega uma combinação de análise estática e dinâmica, fornecendo análise de vulnerabilidades para as TPLs identificadas.
- LibRadar que utiliza análise baseada em API e é conhecido por sua rápida detecção.
- LibD2, semelhante ao LibID e ORLIS, usa análise estática e dinâmica, além de fornecer análise de vulnerabilidades para as TPLs detectadas.

Os programas LibID, LibRadar, LibScout, LibPecker e ORLIS foram avaliados em quatro categorias: eficiência, 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 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 [7].

Devido a eficiência e escalabilidade, inicialmente optamos seguir com o LibScout e com o LibRadar. As outras ferramentas tem tempo de execução muito grande para uma precisão não tão alta.

Os resultados do LibRadar e do LibScout para o piloto não só não foram semelhantes como o LibScout identificou muito mais TPLs do que o LibRadar. Por conta disso, optamos por seguir com o LibScout.

2.1.5 LibScout

A ferramenta LibScout é resultado de um projeto de pesquisa cujo objetivo principal é analisar quais são as bibliotecas externas e quais são bibliotecas nativas em aplicativos Android.

A ferramenta permite analisar chamadas de API de aplicativos Android diretamente do bytecode java.

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. [6]

Assim, para o estudo desse artigo, a comparação dos resultados das ferramentas CryptoGuard e CogniCrypt aos resultados gerados pelo LibScout visa solucionar o problema de 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. [7]

O LibScout é excepcionalmente resistente a aplicativos obfuscados, porém, o CryptoGuard e o CogniCrypt não. 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.

O problema acima é um dos motivos dos quais os aplicativos utilizados para montar os testes do estudo não serem obfuscados.

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 [4].

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.

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 [4].

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. [4]

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.

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. [5]

O CRYPTOGUARD alcança isso por meio da implementação de um conjunto de novos algoritmos de análise que realizam uma análise estática do código-fonte.

Ele proporciona detecção de alta precisão de vulnerabilidades criptográficas e oferece insights de segurança aos desenvolvedores. [5]

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. [5]

Resumindo, o Cryptoguard é uma ferramenta, assim como o CogniCrypt, para detectar vulnerabilidades criptográficas em projetos Java.

Capítulo 3

Estudo Experimental

3.1 Estudo Exprimental

Para alcançar o objetivo de identificar quantos warnings das ferramentas CogniCrypt e CryptoGuard advém ou não de uma biblioteca extena, adotamos a abordagem GQM (Goal, Question, Metric), conforme descrito a seguir:

3.1.1 Objetivo

Identificar a quantidade de *warnings* das ferramentas CogniCrypt e CryptoGuard que advém de bibliotecas externas.

Questões de Pesquisa

Definimos as seguintes perguntas baseadas nos dados coletados:

- **RQ1:** Qual é a quantidade de *warnings* no *dataset* de aplicativos Android analisados pelas ferramentas CogniCrypt e CryptoGuard?
- **RQ2:** Qual a quantidade de *warnings* específicos de bibliotecas externas?
- **RQ3:** Dado as limitações de análise da ferramenta LibScout, qual a quantidade de *warnings* que possivelmente são de bibliotecas externas?

Métricas

Para responder as questões RQ1, RQ2 e RQ3 as seguintes métricas foram estabelecidas:

- **M1: Número total de warnings detectados pela ferramenta CogniCrypt.** Nos 307 aplicativos analisados, após executar o CogniCrypt, a ferramenta identificou 11.036 *warnings* de crypto api missuses.

- **M2: Número total de warnings detectados pela ferramenta CriptoGuard.** Nos mesmos aplicativos analisados pelo CogniCrypt, a ferramenta CryptoGuard identificou 4.964 *warnings*. Essas métricas são essenciais para quantificar a presença geral de vulnerabilidades no conjunto de dados de aplicativos Android.

- **M3: Número total de bibliotecas externas na ferramenta CogniCrypt.**

  No CogniCrypt, de um total de 6.798 bibliotecas contabilizadas dos 307 aplicativos, a ferramenta LibScout identificou que 1.710 *warnings* advém de bibliotecas externas.

- **M4: Número total de bibliotecas externas na ferramenta CriptoGuard.**

  No CryptoGuard, de um total de 2.710 bibliotecas dos mesmos 307 aplicativos, o LibScout identificou que 1.149 *warnings* advém de bibliotecas externas.

  Isso demonstra que uma parte considerável dos *warnings* detectados pode ser atribuída a bibliotecas externas, o que sugere a importancia da integração dessas ferramentas na identificação da origem dos *warnings* e na detecção de vulnerabilidades específicas de bibliotecas externas.

  Com o resultado da integração do resultado do LibScout com os *warnings* identificados pelo CogniCrypt e CriptoGuard criamos uma abordagem capaz de identificar os *warnings* associados a bibliotecas externas daqueles associados a bibliotecas nativas.

- **M5: Número total de bibliotecas possivelmente externas na ferramenta CogniCrypt**

  Após a integração dos resultados do LibScout com os *warnings* identificados por ambas as ferramentas, observamos que apesar do LibScout gerar resultados com precisão de algumas bibliotecas externas, a ferramenta não é capaz de identificar todas as bibliotecas externas presentes nos aplicativos [7].

  Para melhorar a precisão dos resultados, identificamos também as bibliotecas que estão presentes nos aplicativos, mas que não foram detectadas pelo LibScout como bibliotecas externas. Uma biblioteca reconhecida pelo LibScout e presente nos resultados das ferramentas de análise estática foram marcadas como possivelmente externas e foram incluídas na análise.

  Das 6.798 bibliotecas identificadas pelo CogniCrypt, 726 foram marcadas como possivelmente externas.

- **M6: Número total de bibliotecas possivelmente externas na ferramenta CriptoGuard**

  Já das 2.710 bibliotecas identificadas pelo CryptoGuard, 245 foram marcadas como possivelmente externas.

3.2 Metodologia do Estudo

Para conseguir essas métricas e responder às questões de pesquisa, a pesquisa foi dividida em várias fases, conforme descrito a seguir:

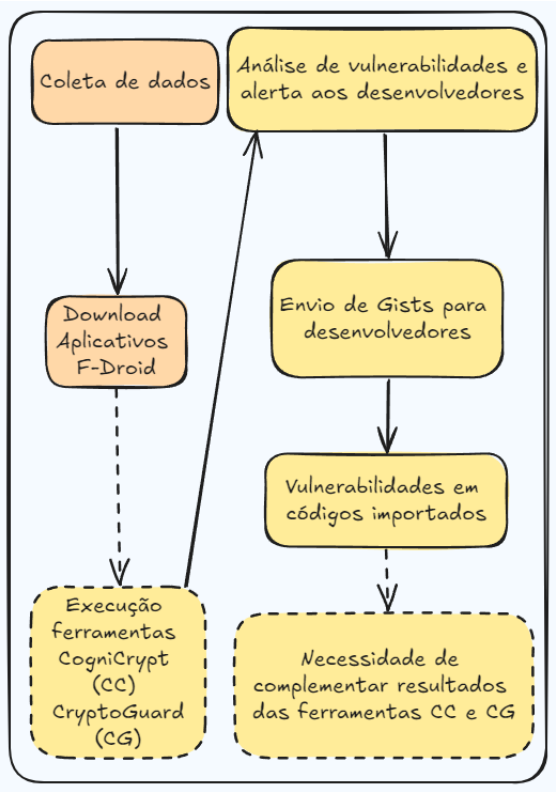


Figura 3.1: Fases da pesquisa

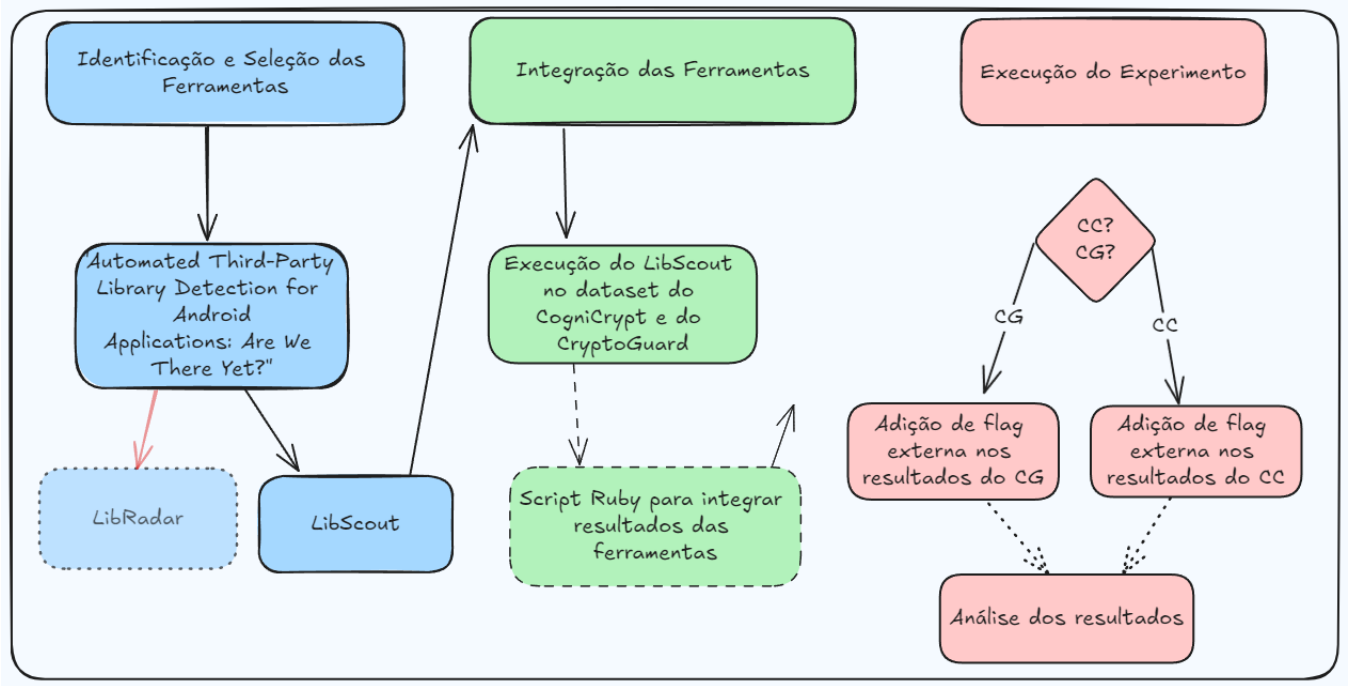


Figura 3.2: Fases da pesquisa

3.2.1 Seleção de dataset

Inicialmente, foi coletado um conjunto de aplicativos Android de código aberto para análise. O conjunto de dados foi obtido do repositório F-Droid, que contém aplicativos de código aberto disponíveis para download. O dataset foi separado em categorias: Connectivity, Finances, Internet, Security, SMS e System. O repositório foi escolhido devido à sua natureza de código aberto e à disponibilidade de aplicativos para download.

3.2.2 Análise de vulnerabilidades e alerta aos desenvolvedores

Em seguida, foram utilizadas as ferramentas CogniCrypt e CryptoGuard para analisar os aplicativos e identificar vulnerabilidades em APIs criptográficas. As ferramentas foram escolhidas devido à sua capacidade de detectar vulnerabilidades em APIs criptográficas e fornecer alertas aos desenvolvedores sobre possíveis problemas de segurança. Nessa etapa, identificamos que vários dos *warnings* detectados pelas ferramentas estavam associados a código que os desenvolvedores não tinham escrito e sim a códigos de bibliotecas externas. Ambas as ferramentas foram executadas em um ambiente docker rodando scripts estruturados para garantir a reprodutibilidade dos resultados.

CogniCrypt

```
basePath = "/home/cryptoRunner/"
# basePath = '/Users/Amaral/tools/crypto-analysis-runner/fse2022/'

outputPath = basePath + "results/cryptoAnalysisOutput/"
reportPath = basePath + "results/cryptoReportOutput/"
summaryPath = basePath + "results/cryptoSummaryOutput/"
sarifPath = basePath + "results/cryptoSarifOutput/"
apksPath = basePath + 'projects/apks/'
jarPath=  basePath + 'src/CryptoAnalysis-Android-2.8.0-SNAPSHOT-jar-with-dependencies.jar'
rulesPath = basePath + 'src/JCA/'
platformsPath = '/usr/lib/android-sdk/platforms/'

os.chdir(apksPath)

extension = 'apk'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]

for file in all_filenames:
    subprocess.call(['java', '-Xmx8g', '-Xss128m', '-cp', jarPath, 'de.fraunhofer.iem.crypto.CogniCryptAndroidAnalysis',
                apksPath + file, platformsPath,  rulesPath, outputPath])

    print("Analysis finished for prject: " + file)

    file = unidecode.unidecode(file)
    subprocess.call(['mv', outputPath + "CryptoAnalysis-Report.csv", reportPath + file + "-report.csv"])
    subprocess.call(['mv', outputPath + "CryptoAnalysis-Summary.csv", summaryPath + file + "-summary.csv"])
    subprocess.call(['mv', outputPath + "CryptoAnalysis-Report.json", sarifPath + file + "-sarif.json"])
```

Figura 3.3: Script utilizado para rodar o CogniCrypt


```
{
  "sarifVersion" : "2.0.0",
  "runs" : [ {
    "files" : {
      "okhttp3/internal/platform/ConscryptPlatform.java" : {
        "mimeType" : "text/java"
      },
      "okhttp3/internal/platform/BouncyCastlePlatform.java" : {
        "mimeType" : "text/java"
      },
      "okhttp3/internal/platform/OpenJSSEPlatform.java" : {
        "mimeType" : "text/java"
      },
      "okio/ByteString.java" : {
        "mimeType" : "text/java"
      },
      "okhttp3/internal/platform/Platform.java" : {
        "mimeType" : "text/java"
      },
      "okio/SegmentedByteString.java" : {
        "mimeType" : "text/java"
      }
    },
    "resources" : {
      "rules" : {
        "ConstraintError-1" : {
          "id" : "ConstraintError",
          "fullDescription" : {
            "text" : "A constraint of a CrySL rule is violated, e.g., a key is generated with the wrong key size."
          }
        },
        "ConstraintError-2" : {
          "id" : "ConstraintError",
          "fullDescription" : {
            "text" : "A constraint of a CrySL rule is violated, e.g., a key is generated with the wrong key size."
          }
        },
        "ConstraintError-3" : {
```


Figura 3.4: Output da ferramenta Cognicrypt para o apk Ademar.bitac_5

```
"results" : [ {
  "locations" : [ {
    "physicalLocation" : {
      "fileLocation" : {
        "uri" : "okio/ByteString.java"
      },
      "region" : {
        "startLine" : 85
      }
    },
    "fullyQualifiedLogicalName" : "okio::ByteString::digest$okio"
  } ],
  "ruleId" : "ConstraintError",
  "message" : {
    "text" : "First parameter (with value \"MD5\") should be any of {SHA-256, SHA-384, SHA-512}.",
    "richText" : "ConstraintError violating CrySL rule for java.security.MessageDigest."
  }
}, {
  "locations" : [ {
    "physicalLocation" : {
      "fileLocation" : {
        "uri" : "okio/ByteString.java"
      },
      "region" : {
        "startLine" : 85
      }
    },
    "fullyQualifiedLogicalName" : "okio::ByteString::digest$okio"
  } ],
  "ruleId" : "ConstraintError-1",
  "message" : {
    "text" : "First parameter (with value \"SHA-1\") should be any of {SHA-256, SHA-384, SHA-512}.",
    "richText" : "ConstraintError violating CrySL rule for java.security.MessageDigest."
  }
}, {
```


Figura 3.5: Output da ferramenta Cognicrypt para o apk Ademar.bitac_5

CriptoGuard

```
start_time = time.time()

basePath = "/home/cryptoRunner/"
outputPath = basePath + "results/cryptoGuardOutput/"
jarsPath = basePath + 'projects/jars/'
jarPath=  basePath + 'src/cryptoguard.jar'

os.chdir(jarsPath)

extension = 'jar'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]

# print(all_filenames)

os.chdir(outputPath)
# AnalysisOutput = open(outputPath + 'AnalysisResults.txt', 'w')
for file in all_filenames:
    subprocess.call(['java', '-Xmx8g', '-Xss128m', '-jar', jarPath,'-in', 'jar', '-s', jarsPath + file , '-m', 'D'])
    print("Analysis finished for project: " + file)

print("Analysis finished for: ")
```

Figura 3.6: Script utilizado para rodar o CryptoGuard

Figura 3.7: Output da ferramenta Cryptoguard para o apk Ademar.bitac_5

3.2.3 Identificação e Seleção das Ferramentas

Para identificar as bibliotecas externas, fizemos uma análise da literatura e através do estudo realizado por [7], identificamos o LibScout como uma ferramenta eficaz para identificar bibliotecas em aplicativos Android. O LibScout foi escolhido devido à sua capacidade de identificar bibliotecas em aplicativos Android e fornecer informações detalhadas sobre as bibliotecas encontradas. O estudo cita outras ferramentas que poderiam ser utilizadas, como o LibRadar, LibID, libPecker. O LibRadar foi um dos aplicativos inicialmente considerados para podermos comparar com os resutlados do LibScout, no entanto, o número de bibliotecas externas encotradas em um subset dos quais já sabiamos os resultados não foi suficiente. O LibRadar compara os resultados da investigação com um dataset base, o qual não foi atualizado nos últimos 7 anos. Cada uma das ferramentas destacadas no paper de Zhang tem suas limitações, como a falta de atualizações de datasets, tempo exagerado para análise de apenas um aplicativo, baixa confiabilidade nos resultados. De todas as métricas descritas no artigo, o LibScout foi a que apresentava resultados mais confiáveis em tempo hábil para análise de um grande número de aplicativos.

Para a execução do LibScout, foi necessário instalar o Android SDK, além de configurar o ambiente de desenvolvimento. O LibScout foi executado em um ambiente docker através de scripts estruturados para garantir a reprodutibilidade dos resultados.

```
libScoutLogsPath = '/home/' + user + customPath + '/libScout/logs/'
libScoutJsonLogsPath = '/home/' + user + customPath + '/libScout/jsonlogs/'
tccResults = '/home/' + user + customPath + '/tccResults/'
libScoutResultsJson = 'libscout-result/json'
cryptoSASTRunnerPath = '/home/' + user + customPath
sarifOutput = cryptoSASTRunnerPath + 'cryptoRunner/results/cryptoSarifOutput'
cryptoguardSarifResults = 'cryptoguard-sarif'
extension = 'apk'

start_time = time.time()

os.chdir(sarifOutput)
crypto_results = [i for i in glob.glob('*.{}'.format('json'))]
for crypto_file in crypto_results:
    subprocess.call(['cp', crypto_file, tccResults + cryptoguardSarifResults])

os.chdir(apksPath)
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
os.chdir(libScoutPath)


for file in all_filenames:
    subprocess.call(['java', '-jar', 'build/libs/LibScout.jar', '-o',
                     'match', '-a', 'build/libs/Android.jar', apksPath + file, '-d'])
    print("Analysis finished for project: " + file)

os.chdir(libScoutLogsPath)
subprocess.call(['ruby', 'libscout-log-to-json.rb'])


os.chdir(libScoutJsonLogsPath)
json_filenames = [i for i in glob.glob('*.{}'.format('json'))]
for json_file in json_filenames:
    subprocess.call(['cp', json_file, tccResults + libScoutResultsJson])

```

Figura 3.8: Script utilizado para rodar o LibScout

```
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.lifecycle  (androidx.lifecycle::lifecycle-runtime)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.lifecycle  (androidx.lifecycle::lifecycle-service)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.lifecycle  (androidx.lifecycle::lifecycle-viewmodel)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.lifecycle  (androidx.lifecycle::lifecycle-viewmodel-ktx)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.loader  (androidx.loader::loader)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4.media  (androidx.media::media)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.recyclerview.widget  (androidx.recyclerview::recyclerview)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.slidingpanelayout.widget  (
androidx.slidingpanelayout::slidingpanelayout)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.swiperefreshlayout.widget  (
androidx.swiperefreshlayout::swiperefreshlayout)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.transition  (androidx.transition::transition)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.vectordrawable.graphics.drawable  (
androidx.vectordrawable::vectordrawable)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.vectordrawable.graphics.drawable  (
androidx.vectordrawable::vectordrawable-animated)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.versionedparcelable  (
androidx.versionedparcelable::versionedparcelable)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.viewpager.widget  (androidx.viewpager::viewpager)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4  (com.android.support::customview)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4  (com.android.support::loader)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4  (com.android.support::support-compat)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4  (com.android.support::support-core-ui)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4  (com.android.support::support-core-utils)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4.app  (com.android.support::support-fragment)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4.media  (com.android.support::support-media-compat)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package android.support.v4  (com.android.support::support-v4)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package androidx.versionedparcelable  (
com.android.support::versionedparcelable)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package com.google.android.material  (com.google.android.material::material)
16:20:18 INFO  LibraryIdentifier         :     - Found lib root package okio  (okio)
16:20:18 INFO  LibraryIdentifier         :
16:20:18 INFO  LibraryIdentifier         : = Match profiles =
16:20:58 INFO  LibraryIdentifier         :     >> profile matching done (39 sec, 131 ms)
16:20:58 INFO  LibraryIdentifier         :
```

Figura 3.9: Output da ferramenta LibScout para o apk Ademar.bitac_5

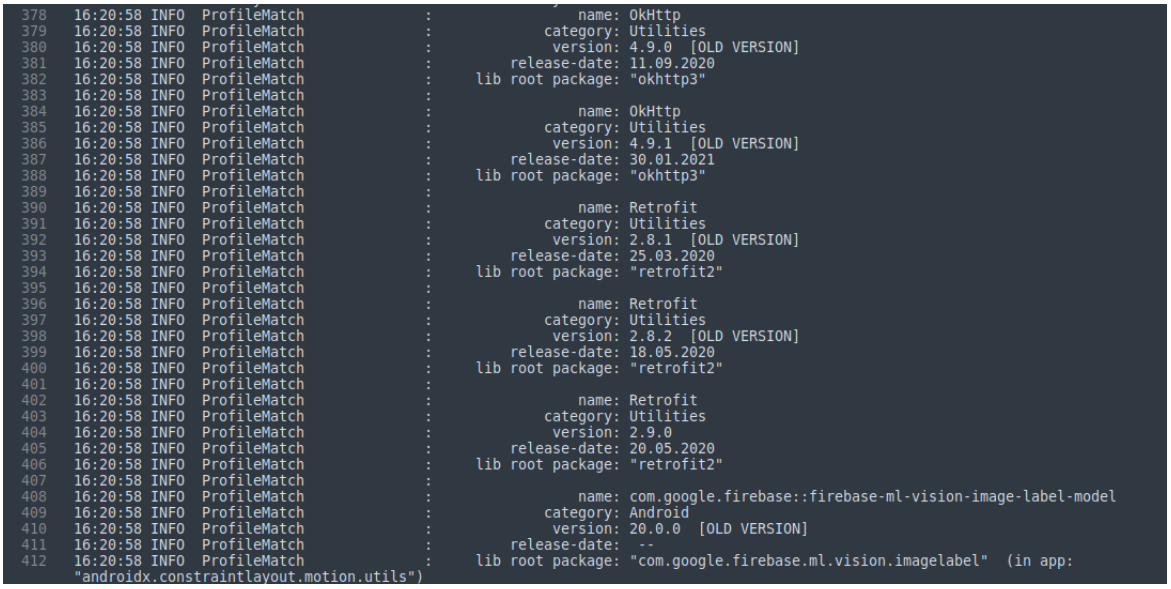
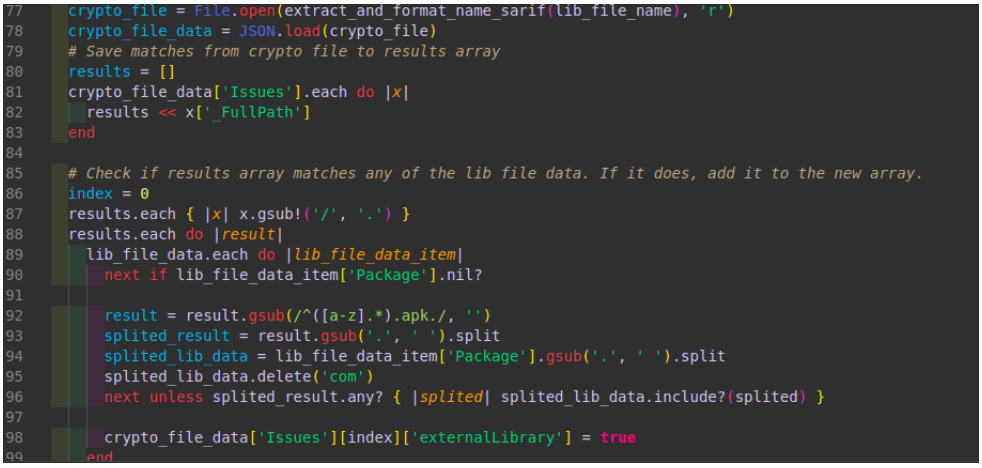


Figura 3.10: Output da ferramenta LibScout para o apk Ademar.bitac_5

As bibliotecas identificadas pelo LibScout que não foram marcadas como externas mas que apareceram nos resultados do CogniCrypt e do CryptoGuard foram marcadas como possivelmente externas e incluídas na análise.

3.2.4 Integração das Ferramentas

Após a seleção, integramos os resultados do LibScout aos contextos das ferramentas CryptoGuard e CogniCrypt. Essa integração permitiu uma análise mais detalhada e contextualizada das vulnerabilidades encontradas, especialmente em bibliotecas externas. A integração se deu por scripts que cruzam os resultados das ferramentas de análise estática com os resultados do LibScout.



```ruby
crypto_file = File.open(extract_and_format_name_sarif(lib_file_name), 'r')
crypto_file_data = JSON.load(crypto_file)
# Save matches from crypto file to results array
results = []
crypto_file_data['Issues'].each do |x|
  results << x['_FullPath']
end

# Check if results array matches any of the lib file data. If it does, add it to the new array.
index = 0
results.each { |x| x.gsub!('/', '.') }
results.each do |result|
  lib_file_data.each do |lib_file_data_item|
    next if lib_file_data_item['Package'].nil?

    result = result.gsub(/^([a-z].*).apk./, '')
    splited_result = result.gsub('.', ' ').split
    splited_lib_data = lib_file_data_item['Package'].gsub('.', ' ').split
    splited_lib_data.delete('com')
    next unless splited_result.any? { |splited| splited_lib_data.include?(splited) }

    crypto_file_data['Issues'][index]['externalLibrary'] = true
  end
```

Figura 3.11: Script utilizado para integrar os resultados do LibScout com os resultados do CogniCrypt

```ruby
# Save matches from crypto file to results array
results = []
crypto_file_data['runs'][0]['results'].each do |x|
  results << x['locations'][0]['physicalLocation']['fileLocation']['uri']
end

# Check if results array matches any of the lib file data. If it does, add it to the new array.
index = 0
results.each { |x| x.gsub!('/', '.') }
results.each do |result|
  lib_file_data.each do |lib_file_data_item|
    next if lib_file_data_item['Package'].nil?

    splited_result = result.gsub('.', ' ').split
    splited_lib_data = lib_file_data_item['Package'].gsub('.', ' ').split
    splited_lib_data.delete('com')
    next unless splited_result.any? { |splited| splited_lib_data.include?(splited) }

    crypto_file_data['runs'][0]['results'][index]['locations'][0]['physicalLocation']['fileLocation']['externalLibrary'] =
      true
  end
  index += 1
```

Figura 3.12: Script utilizado para integrar os resultados do LibScout com os resultados do CryptoGuard

A estratégia para adicionar a flag de possible external foi semelhante. Para cada biblioteca identificada pelo LibScout que não foi marcada como externa, mas que apareceu nos resultados do CogniCrypt e do CryptoGuard, adicionamos a flag de possivelmente externa.

3.2.5 Análise do Experimento

Durante a análise, geramos gráficos e tabelas para ilustrar a distribuição de *warnings* e a prevalência de vulnerabilidades. Os resultados foram gerados utilizando scripts para contar o número de *warnings* e bibliotecas externas e possivelmente externas identificadas.

```ruby
csv_file = File.new(LOADED_NAME_PATH, 'w')
csv_file.write("apk_name,external_libs_count,native_libs_count,possible_external\n")
loaded_path.each do |lib_file_name|
  counter += 1
  external_libs_count = 0
  possible_external = 0

  # Open lib file, parse it and count number of external and native libs
  lib_file = File.open(lib_file_name, 'r')
  apk_name = File.basename(lib_file_name)
  lib_file_data = JSON.load(lib_file)
  libs_count = lib_file_data['runs'][0]['results'].count
  lib_file_data['runs'][0]['results'].each do |lib|
    external_libs_count += 1 if lib['locations'][0]['physicalLocation']['fileLocation'].include?('externalLibrary')
    if lib['locations'][0]['physicalLocation']['fileLocation'].include?('PossibleExternalLibrary')
      possible_external += 1
    end
  end
  native_libs_count = libs_count - external_libs_count
  csv_file.write("#{apk_name},#{external_libs_count},#{native_libs_count},#{possible_external}\n")
  p "Merged #{counter} of #{files_count} files"
  lib_file.close
```

Figura 3.13: Script utilizado para gerar os arquivos de saída para análise

3.3 RQ1. Quantidade de warnings encontrados pelas ferramentas CogniCrypt e CryptoGuard

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

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 3.1: Aplicativos por categoria sem warning das ferramentas CogniCrypt e CryptoGuard

Como visto, o número de aplicativos sem *warnings* no CogniCrypt é bem maior do que no CryptoGuard. A categoria Sistema é a que apresenta a maior diferença entre eles, com 34 aplicativos. As categorias Conectividade e Finanças também têm um número elevado de aplicativos sem *warnings*, 20 e 25, respectivamente.

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

Como podemos observar na tabela acima, o CogniCrypt conseguiu encontrar 4964 *warnings*, enquanto o CryptoGuard encontrou 11 036 *warnings*. A diferença numérica é

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	11 036 (100.00%)	4964 (100.00%)	6072 (100.00%)

Tabela 3.2: Warnings encontrados nas ferramentas CogniCrypt e CryptoGuard

de 6072 (ou 122.32%) *warnings* entre as duas ferramentas. As categorias de Finanças e Internet concentraram 58.8% (6494) dos *warnings* do CogniCrypt, enquanto as categorias de Finanças e Conectividade concentraram 56.6% (2811) dos *warnings* do CryptoGuard. A maior diferença entre os *warnings* encontrados foi notada nas categorias Internet (2491) e Finanças (1400), representando 64.1% de diferença.

3.4 RQ2 e RQ3. Quantidade de warnings de bibliotecas externas e possivelmente externas

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 3.3: 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	17 439.9	118.2	1747.7	15 895.3

Tabela 3.4: 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 3.5: 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 3.6: Resultados da integração do CogniCrypt com o LibScout na categoria System

Os resultados para a tabela de conectividade (3.3) 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 (3.4), 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 (3.5), 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 (3.6), 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. O LibScout pode não ter mapeado a biblioteca com *warning* como externa, e o CogniCrypt pode ter encontrado *warnings* em bibliotecas que não foram mapeadas pelo LibScout ou ainda não ter encontrado vulnerabilidade no aplicativo selecionado.

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

Analisando a tabela de conectividade (3.7), 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 (3.8), 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 (3.9), a média foi de 9

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 3.8: 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 3.9: 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 3.10: Resultados da integração do CryptoGuard com o LibScout na categoria System

alertas, com 0.78 potencialmente externos e 2.73 definitivamente externos, e 5.47 nativos. E para sistema (3.10), 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. O LibScout pode não ter mapeado a biblioteca com *warning* como externa, e o CryptoGuard pode ter encontrado *warnings* em bibliotecas que não foram mapeadas pelo LibScout ou ainda não ter encontrado vulnerabilidade no aplicativo selecionado.

-	CogniCrypt	Cryptoguard
Total Aplicativos	246	253
Total Bibliotecas	6798	2710
Total Bibliotecas Externas	1710	1149
Total Bibliotecas Externas Potenciais	726	245
Total Bibliotecas Nativas	4362	1316

Tabela 3.11: Resultados da integração do CryptoGuard com o LibScout na categoria System

A tabela 3.11 mostra um resumo dos resultados obtidos nas categorias analisadas. O CogniCrypt analisou 246 aplicativos, com 6798 bibliotecas, das quais 1710 são externas, 726 potencialmente externas e 4362 nativas. O CryptoGuard analisou 253 aplicativos, com 2710 bibliotecas, das quais 1149 são externas, 245 potencialmente externas e 1316 nativas.

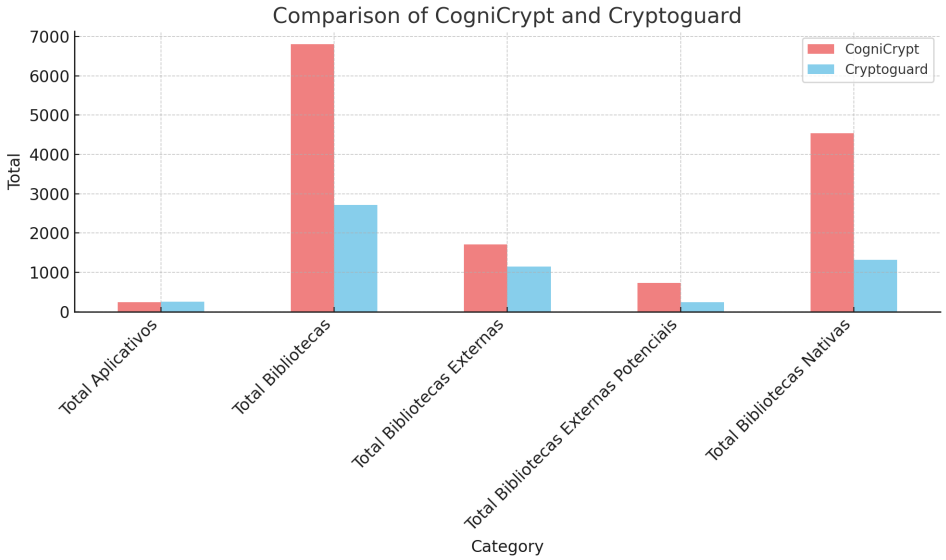


Figura 3.14: Comparação total entre as ferramentas CogniCrypt e CryptoGuard

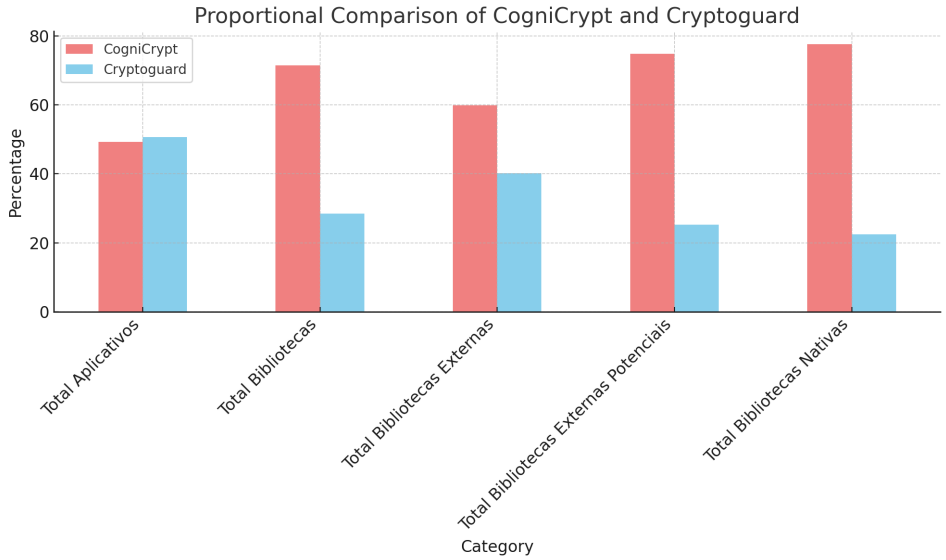


Figura 3.15: Comparação proporcional total entre as ferramentas CogniCrypt e CryptoGuard

Os resultados para a ferramenta CogniCrypt se destacaram em relação aos resultados para o CryptoGuard.

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

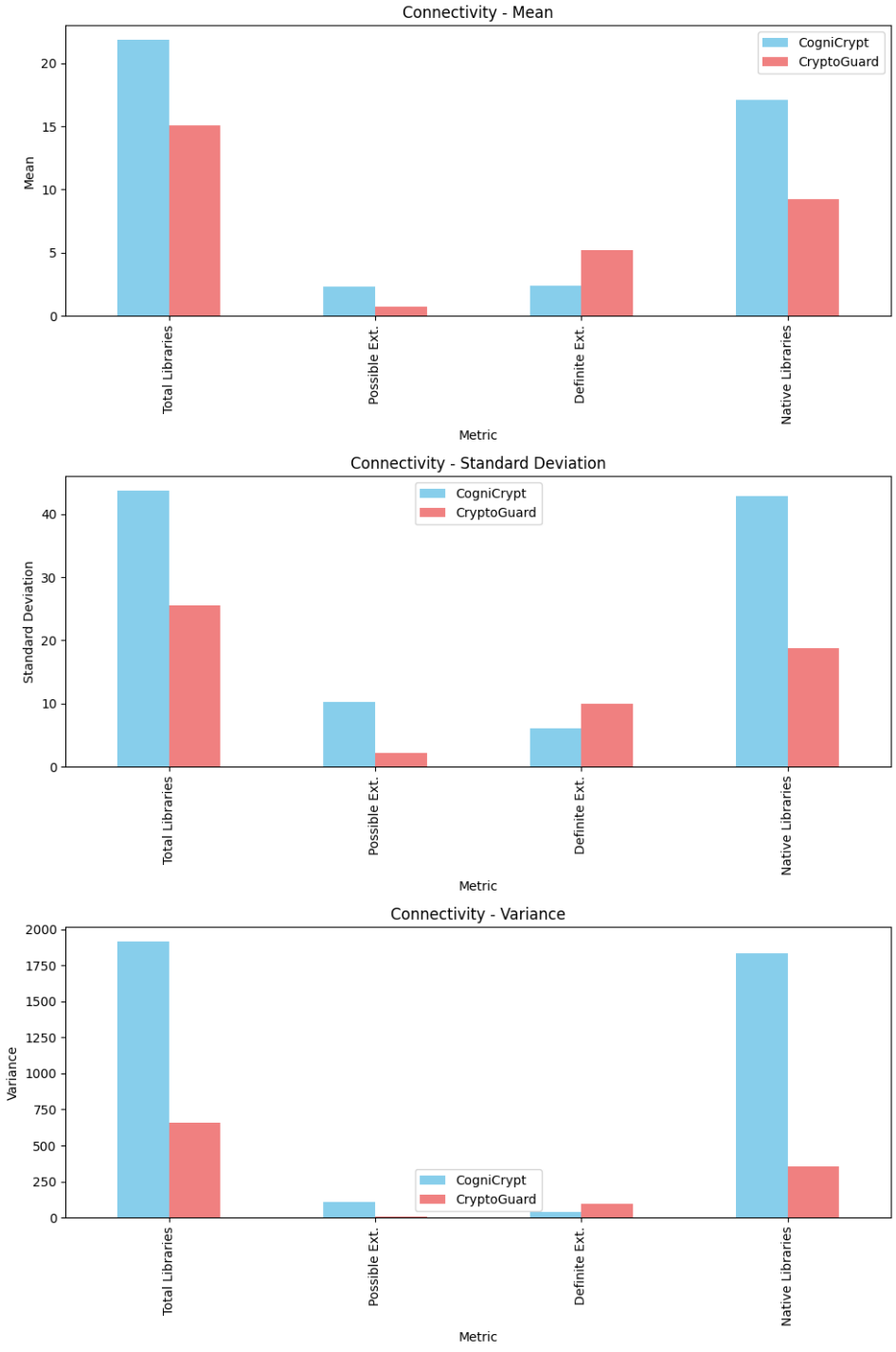


Figura 3.16: Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria Connectivity

Na categoria de conectividade (3.16), 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.

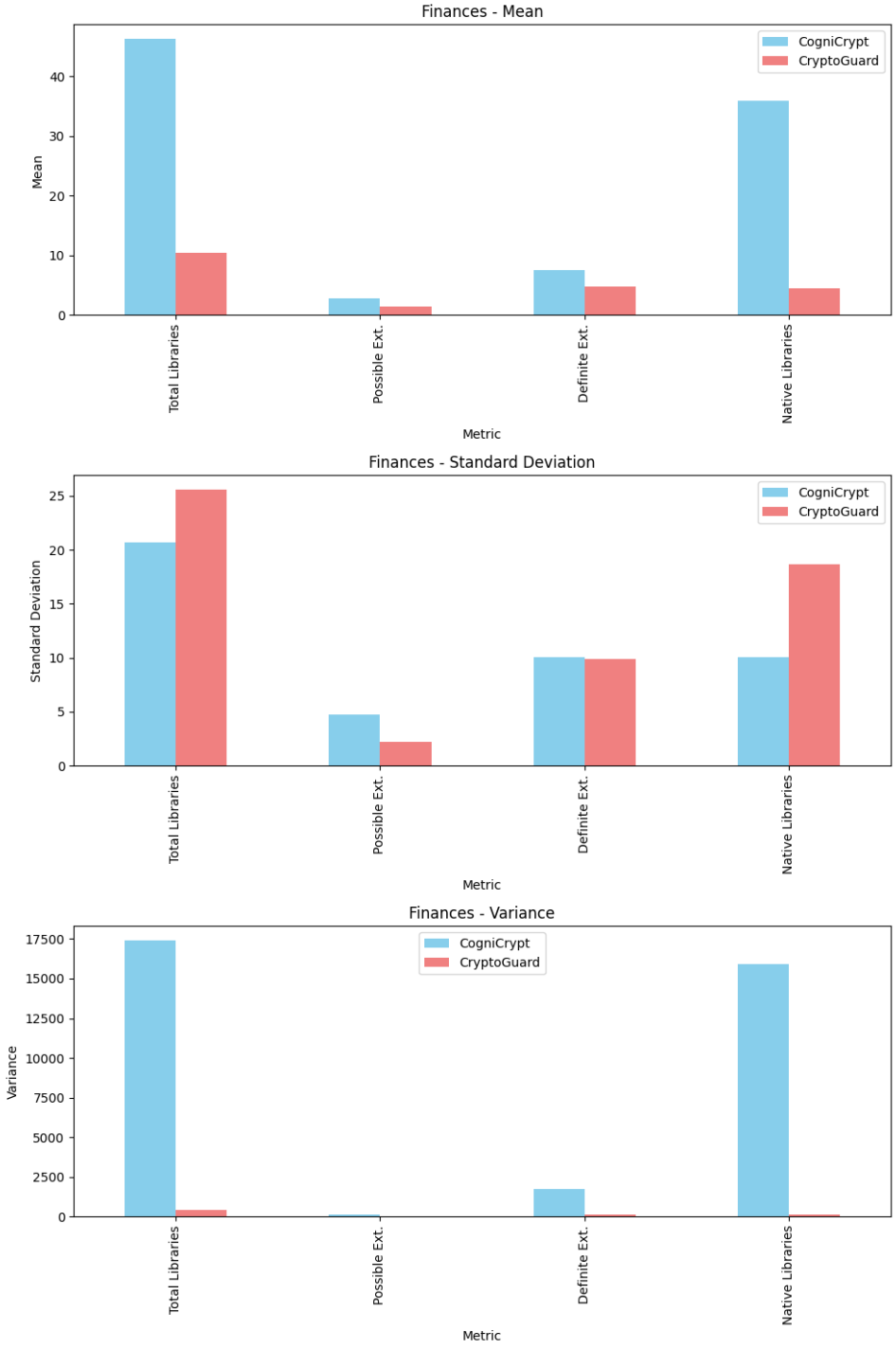


Figura 3.17: Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria Finances

Na categoria financeira (3.17), 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.

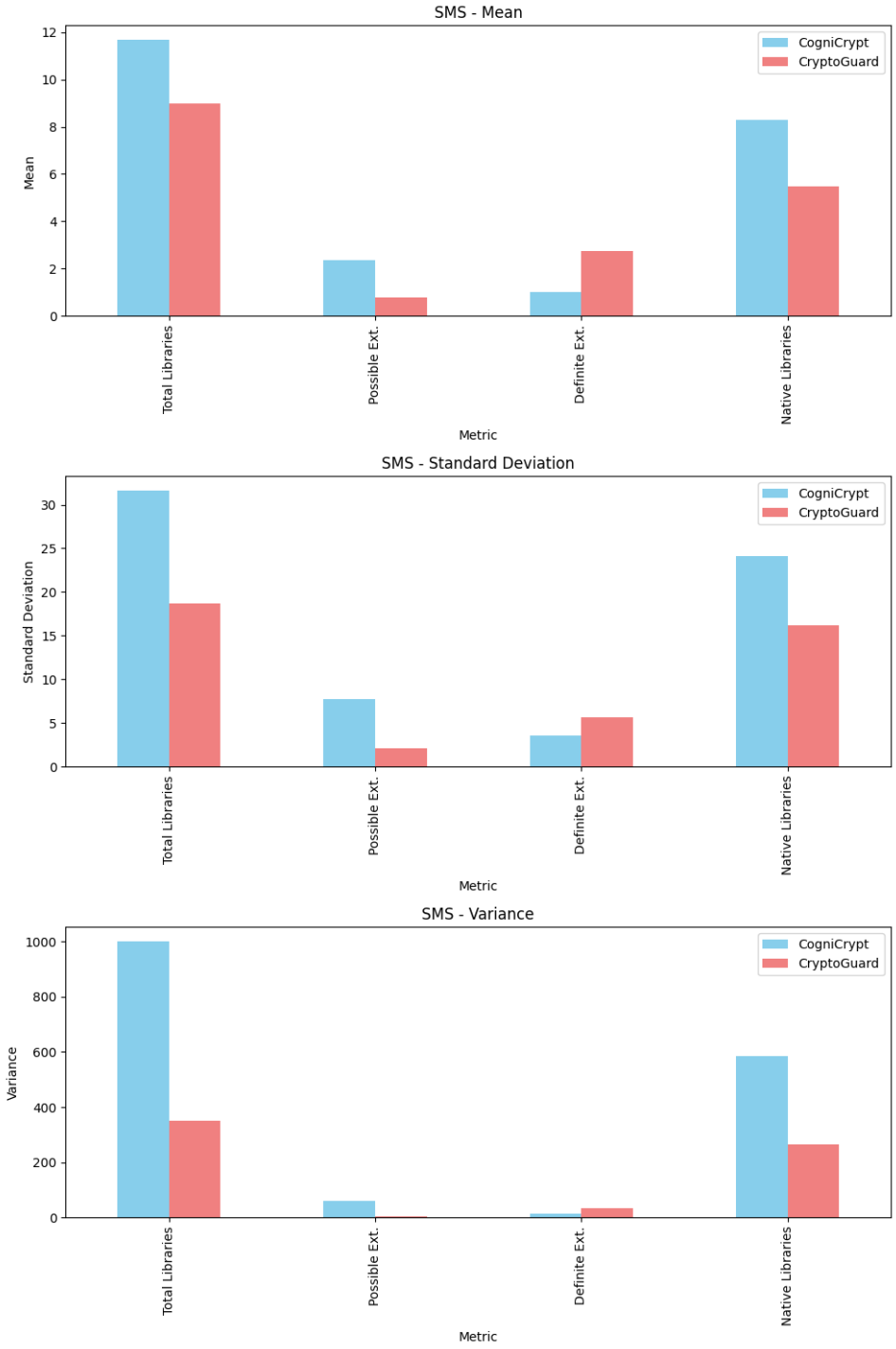


Figura 3.18: Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria SMS

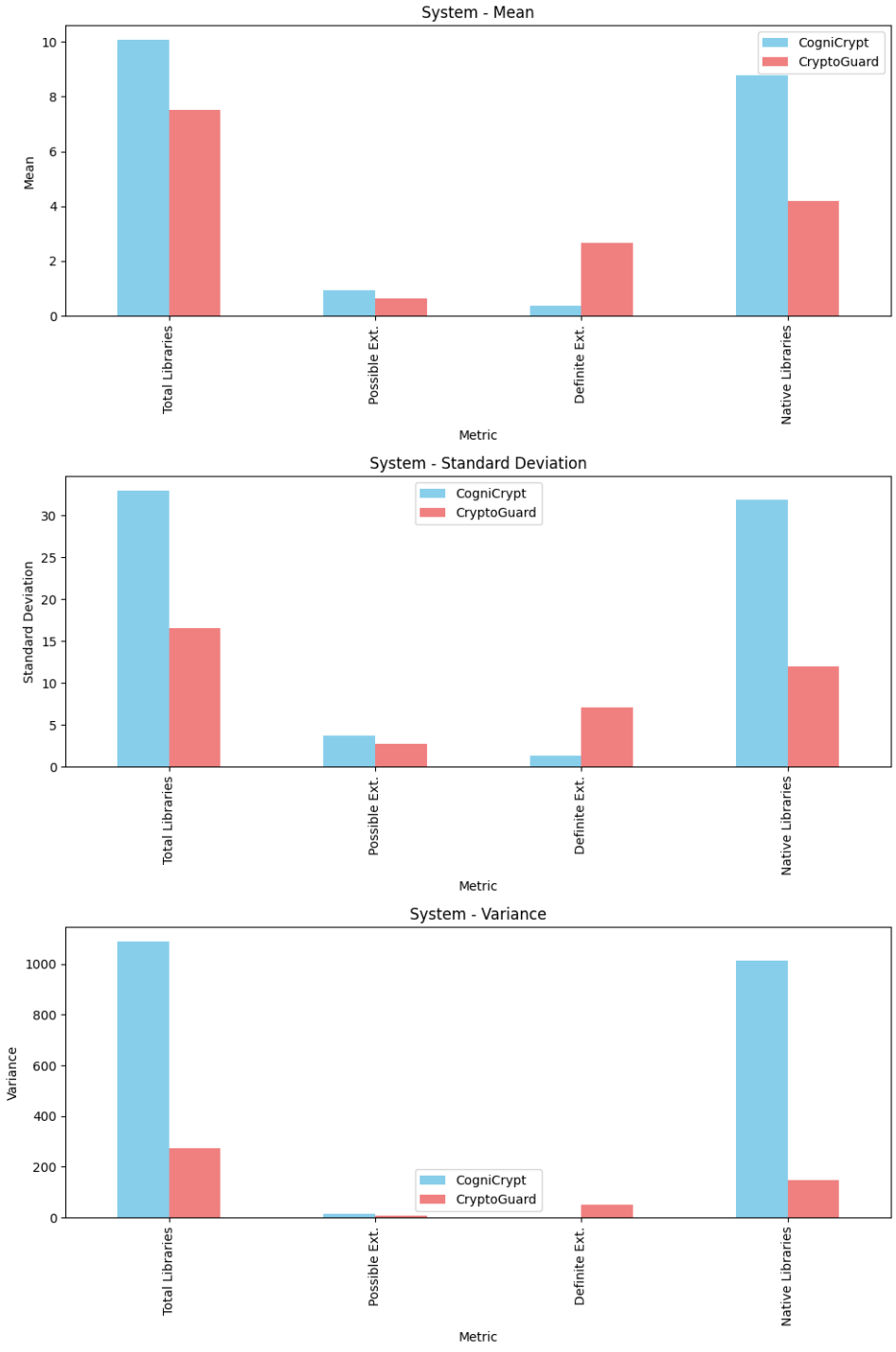


Figura 3.19: Comparação entre as ferramentas CogniCrypt e CryptoGuard na categoria System

Nas categorias de SMS (3.18) e Sistemas (3.19), 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.

Capítulo 4

Conclusão

4.1 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 a eficácia na identificação de falhas de segurança, oferecendo uma abordagem holística e mais robusta para a segurança de aplicações Java.

A ferramenta CryptoGuard identificou menos bibliotecas externas em comparação com o CogniCrypt, 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.

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, 5

[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, 3, 6, 7, 32

[3] Nadi, Sarah, Stefan Krüger, Mira Mezini e Eric Bodden: *"jumping through hoops": Why do java developers struggle with cryptography apis?* páginas 935–946, 2016. 1, 3, 4

[4] 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, 2017. 1, 2, 9, 10

[5] 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. 2, 10

[6] Derr, Erik: *Libscout*. `https://github.com/reddr/LibScout`, 2019. 2, 9

[7] 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, páginas 919–930, New York, NY, USA, 2021. Association for Computing Machinery, ISBN 9781450367684. `https://doi.org/10.1145/3324884.3416582`. 2, 7, 8, 9, 12, 17

[8] Lazar, David, Haogang Chen, Xi Wang e Nickolai Zeldovich: *Why does cryptographic software fail? a case study and open problems*. Em *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys '14, New York, NY, USA, 2014. Association for Computing Machinery, ISBN 9781450330244. `https://doi.org/10.1145/2637166.2637237`. 6

[9] Torres, A., P. Costa, L. Amaral, J. Pastro, R. Bonifacio, M. d'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. 6