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.

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

Contents

Chapter 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. 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.

O surgimento da linguagem CrySL permitiu a definição precisa de regras para o uso seguro de APIs criptográficas em código Java. 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 e CryptoGuard.

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.

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. 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?". 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 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 aumegntou 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.

Chapter 2

Trabalhos Correlatos e Revisão de Literatura

2.1 Trabalhos Correlatos e Revisão de Literatura

2.1.1 Análise dinâmica de APIs criptográficas

O estudo intitulado "Runtime Verification of Crypto APIs: An Empirical Study" oferece uma meticulosa investigação comparativa de métodos de detecção de uso inadequado de APIs criptográficas em projetos Java. A pesquisa conduzida fornece 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.

2.1.2 Automatic Detection of Java Cryptographic API Misuses: Are We There Yet?

O estudo apresentado no artigo acima traz que ao se trabalhar com APIs criptográficas Java, os desenvolvedores se deparam com diversos desafios. Em primeiro lugar, algumas dessas APIs tendem a ser excessivamente complexas, o que torna sua compreensão e utilização eficaz uma tarefa árdua. A situação é frequentemente agravada pela escassez de documentação adequada, o que dificulta ainda mais para os desenvolvedores o uso correto dessas APIs.

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. 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.

Outro desafio comum decorre da falta de compreensão e treinamento, levando os desenvolvedores a fazerem uso inadequado das APIs criptográficas. 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.

Adicionalmente, é comum que os desenvolvedores recorram à prática de copiar e colar trechos de código de fontes online, como o StackOverflow. 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.

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. Portanto, a importância de uma utilização adequada das APIs criptográficas torna-se evidente.

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.

Também foi identificado pelo estudo que diversas vulnerabilidades relacionadas ao uso inadequado de APIs criptográficas Java. 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.

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.

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.

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.

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.

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.

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.

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 segu-

rança. 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.

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.

2.1.3 Automated Third-Party Library Detection for Android Applications: Are We There Yet?

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

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.

O artigo menciona diversas ferramentas para detectar bibliotecas de terceiros (TPLs) em aplicativos Android. Algumas dessas ferramentas incluem o LibID, que utiliza uma combinação de análise estática e dinâmica para identificar TPLs em aplicativos Android. O LibPecker se concentra na detecção de TPLs que são ofuscadas utilizando diferentes técnicas de ofuscação de código. O ORLIS também emprega uma combinação de análise estática e dinâmica para detectar TPLs, fornecendo ainda análise de vulnerabilidades para as TPLs identificadas. O LibRadar utiliza análise baseada em API para detectar TPLs em aplicativos Android, sendo conhecido por sua rápida velocidade de detecção. O LibD2, semelhante ao LibID e ORLIS, utiliza uma combinação de análise estática e dinâmica para detectar TPLs, oferecendo também análise de vulnerabilidades para as TPLs detectadas. O LibScout é destacado por sua sensibilidade à ofuscação do fluxo de controle, achatamento de pacotes e remoção de código inativo. Ele utiliza uma árvore de Merkle para gerar perfis de TPLs e depende da estrutura hierárquica de pacotes. Observa-se ainda que o LibScout possui uma alta taxa de detecção de TPLs, mas é afetado por técnicas de ofuscação de código, levando a uma redução na taxa de detecção em mais de 70

Em resumo, o artigo destaca o LibScout como uma ferramenta sensível a técnicas de ofuscação de código, que depende da estrutura hierárquica de pacotes para a detecção de TPLs. Embora possua uma alta taxa de detecção, o LibScout é influenciado por determinadas técnicas de ofuscação, o que resulta em uma diminuição na precisão da detecção.

Neste estudo, baseamos nossa escolha da ferramenta LibScout para identificação de bibliotecas de terceiros (TPLs) em aplicativos Android. 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.

2.1.4 CogniCrypt: Supporting Developers in Using Cryptography

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

2.1.5 CRYPTOGUARD: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects

O objetivo do CRYPTOGUARD é detectar vulnerabilidades criptográficas em projetos Java. 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. 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. 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.

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. 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. O CRYPTOGUARD auxiliou na identificação e relato dessas vulnerabilidades para as equipes do Apache, resultando em correções rápidas em alguns casos. 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. Bibliotecas de empresas como Google, Facebook, Apache, Umeng e Tencent foram identificadas com violações em diversas categorias, incluindo senhas hard-coded de keyStore e vulnerabilidades SSL/TLS. O CRYPTOGUARD detectou múltiplas vulnerabilidades SSL/TLS (MitM) que a triagem automática do Google Play aparentemente deixou passar. FONTES:

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.

Chapter 3

Metodologia e Fundamentos

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 Fundamentação Teórica

3.3 Criptografia

A criptografia é um componente essencial para a segurança da informação, desempenhando um papel fundamental ao garantir a confidencialidade e a integridade dos dados. Essa técnica consiste em transformar informações em um formato ilegível, conhecido como cifrado, que somente uma pessoa autorizada pode reverter ao seu estado original. No âmbito da criptografia, dois tipos de abordagens principais são empregados: a simétrica, que utiliza uma única chave para tanto cifrar quanto decifrar dados, e a assimétrica, que envolve o uso de pares distintos de chaves – uma pública, para cifragem, e outra privada, para decifragem.

Diversos algoritmos criptográficos, cada qual com suas particularidades e aplicações, estão disponíveis. O Advanced Encryption Standard (AES), por exemplo, é largamente empregado na criptografia simétrica para salvaguardar dados sensíveis, sendo reconhecido pela sua segurança e eficácia. Já o RSA, um dos primeiros algoritmos de criptografia

assimétrica, fundamenta-se na complexidade de fatorar números primos extremamente grandes e é amplamente utilizado em trocas seguras de chaves e assinaturas digitais.

A segurança de um sistema criptográfico depende da robustez do algoritmo empregado e da gestão adequada das chaves utilizadas. Em virtude do constante avanço das tecnologias de informação e das técnicas de ataque, é imperativo recorrer a algoritmos criptográficos confiáveis e adotar métodos atualizados. Este é um procedimento crucial para manter a integridade e a confidencialidade dos dados em um ambiente dinâmico e em contínua transformação.

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.

3.4 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.

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 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.

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.

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.

Essa combinação de ferramenta e linguagem apresenta uma solução valiosa para as preocupações de segurança no desenvolvimento de aplicações Java, permitindo que os desenvolvedores tomem medidas proativas para proteger dados e sistemas contra ameaças cibernéticas.

3.4.1 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.

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.

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.

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.

3.5 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. 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. A ferramenta é projetada para ser leve e eficiente, executando mais rapidamente do que técnicas de análise existentes. 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. 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.

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.

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.

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.

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.

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.

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.

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.

3.5.1 CryptoGuard vs CrySL

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

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). 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%.

3.6 Ferramentas para análise de bibliotecas externas

Na 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.

3.6.1 LibScout

LibScout é uma ferramenta que visa extrair dados das APIs de bibliotecas de aplicativos android. 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. 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. 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. 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. 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

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

A ferramenta conta com técnicas de clusterização para encontrar bibliotecas externas em aplicativos Java. Este método baseia-se na análise de diversas aplicações Java como base. Isso permite que o LibScout identifique padrões comuns e classifique se a instancia é ou não uma biblioteca externa ou nativa.

Usando esta estratégia de agrupamento, 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.

Ao adicionar clustering ao seu processo de identificação, o LibScout melhora sua capacidade de distinguir entre chamadas de API para bibliotecas externas e nativas. Esta abordagem melhora o desempenho e a precisão do LibScout em bibliotecas de terceiros

encontradas em aplicações Java, mesmo com os desafios de implementação de ofuscação de código.

3.6.2 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. A exemplo dos resultados apresentados pelas ferramentas de análise estática de código, a ofuscação em ambas as ferramentas impossibilitou a identificação dos nomes das bibilotecas com vulnerabilidades.

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. Mesmo com ferramentas como o LibScout, isso dificulta a extração precisa de informações sobre bibliotecas de terceiros.

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.

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.

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.

3.7 Metodologia

- 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. Este último se destaca como um catálogo de aplicativos de código aberto e livre (FOSS), especialmente concebidos para a plataforma Android.

  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. 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.

- 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 visava 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.

- Análise de Resultados

  Em um estágio subsequente, procedeu-se com a análise dos resultados obtidos. Esta etapa envolveu a comparação minuciosa dos alertas inicialmente gerados com aqueles que surgiram após a integração dos dados provenientes do LibScout. Tal comparação foi essencial para discernir o impacto da abordagem proposta na detecção e identificação de vulnerabilidades em APIs criptográficas. A avaliação criteriosa dessa eficácia se configura como um passo crucial na validação do método adotado.

- Discussão e Conclusão

  A etapa subsequente englobou a interpretação meticulosa dos resultados obtidos, seguida de uma análise aprofundada sobre a contribuição significativa da abordagem para a segurança de aplicações Java que fazem uso de operações criptográficas.

Essa análise proporcionou insights valiosos sobre o impacto positivo da integração proposta.

Além disso, com base nas conclusões extraídas, foram delineadas sugestões pertinentes para possíveis melhorias futuras no método adotado. Estas considerações visam aprimorar ainda mais a eficácia da abordagem, bem como expandir suas aplicações em cenários diversos, promovendo uma segurança ainda mais robusta para aplicações Java que incorporam operações criptográficas.

Referências

Appendix A

Fichamento de Artigo Científico

UnB Departamento de Ciência da Computação 2017/1

Fichamento de Artigo Científico

Prof. Guilherme N. Ramos

Um fichamento reúne elementos relevantes do conteúdo, apresentando a estrutura do texto, e deve seguir a seqüência do pensamento do autor, destacando suas ideias, argumentos, justificativas, exemplos, fatos, etc.

1 Artigo Científico

Geralmente, um *artigo científico* é escrito com a seguinte estrutura (buscando responder algumas questões):

I. Introdução

- Qual o contexto do problema? (O que? Onde? Quando?)
- Qual a principal questão ou problema colocado? (Por quê? Como? Qual?)
- Qual o objetivo visado? O que se pretende constatar ou demonstrar? (investigar, analisar, refletir, contribuir,...)

II. Referencial Teórico

- Quais são os autores/teorias/conceitos que já estudaram os principais assuntos abordados e que sustentam ao texto?
- Quais os resultados mais recentes relacionados a eles?

III. Metodologia/Desenvolvimento

- Quais os procedimentos metodológicos adotados? (natureza do trabalho: empírico, teórico, histórico) – (coleta de dados: questionário, entrevista, levantamento bibliográfico).
- Como a pesquisa foi desenvolvida? Quais as principais relações entre teoria e prática?
- Havendo artefato proposto, ele está disponível para utilização e/ou modificação?

IV. Resultados

- Houve validação (por meio de experimentação)? Como foi feita?
- Os resultados obtidos são corretos/válidos?

V. Conclusões

- Qual o problema atacado?
- Quais os resultados obtidos para os objetivos propostos?
- Quais conclusões podem ser tiradas destes resultados?
- Quais as limitações da metodologia utilizada?
- Quais as possibilidades de trabalhos futuros para o problema?

2 Fichamento

Neste contexto, um fichamento deve conter a seguinte estrutura:

1. ***Identificação do aluno:*** indicação precisa de quem é o autor do fichamento.
2. ***Identificação do texto:*** indicação precisa de quem são os autores do texto analisado e dos detalhes do documento, de modo que se possa buscá-lo para uma leitura completa.
3. ***Pontos-chave:*** noções mais relevantes do texto analisado. *Proposta* (o que é apresentado?), *mérito* (por que é relevante?), *validação* (como verificar a utilidade?), e *perspectivas* (o que pode ser melhorado?).
4. ***Palavras-chave:*** expressões que identificam o assunto abordado.
5. ***Sinopse do texto:*** resumo *com suas palavras*. Deve ser mais detalhado que um *abstract*, geralmente apresentando pelo menos um parágrafo por seção do texto original. No caso de inclusão de trechos, o texto deve ser identificado entre "aspas" e concatenado através de suas próprias palavras.
6. ***Análise crítica:*** posicionar-se em relação as seguintes questões: pertinência do assunto; forma como foi abordado; comparação com outras abordagens do mesmo assunto (caso conheça). Junto ao *resumo*, é a parte mais interessante para o leitor, pois apresenta uma avaliação do conteúdo apresentado.

2.1 Exemplo

1. ***Identificação do aluno:***Alan Mathison Turing, 00/000000
2. ***Identificação do texto:***Guilherme N. Ramos, Yutaka Hatakeyama, Fangyan Dong, and Kaoru Hirota, Hyperbox clustering with Ant Colony Optimization (HACO) method and its application to medical risk profile recognition, Applied Soft Computing, Vol. 9, Issue 2, pp 632-640, 2009. (doi:10.1016/j.asoc.2008.09.004)
3. ***Pontos-chave:***

   Proposta: HACO - método para aglomeração de dados utilizando hipercaixas com posicionamento otimizado via algoritmo de colônia de formigas.

   Mérito: apresenta uma nova forma de fazer agrupamentos considerando a topologia do espaço de dados e fornecendo resultados intuitivos e facilmente utilizáveis.

   Validação: comparação com algoritmos conhecidos em testes com dados padrões e com dados de infecção viral para diagnóstico auxiliado por computador.

   Perspectivas: adequação das dimensões das hipercaixas, diminuição de parâmetros.

4. ***Palavras-chave:***colônia de formigas, hipercaixa, otimização, reconhecimento de padrões.

5. ***Sinopse do texto:*** A *Colônica de Formigas* (ACO) é um método de otimização que pode ser utilizado para agrupar dados. *Hyperbox clustering with Ant Colony Optimization* (HACO) é um método de agrupamento que utiliza ACO tentar posicionar hipercaixas no espaço de forma a agrupar a maior quantidade de dados possível, e ainda gera uma forma simples de classificar novos dados.

   ACO é baseado no comportamento de formigas reais, que otimizam o caminho percorrido entre o alimento e o formigueiro. Hipercaixas definem de forma muito simples uma região em um espaço n-dimensional, combinadas para definir regiões de topologia complexa, e utilizadas como um classificador de forma trivial.

   HACO busca encontrar uma partição de dados, efetivamente definindo grupos. Primeiro, aplica ACO para tentar posicionar hipercaixas de forma que estas contenham a maior quantidade possível de dados. A seguir, se não há conhecimento prévio da quantidade de classes, considera-se que as hipercaixas que se sobrepõem representam uma mesma classe de dados, e [grupos de] hipercaixas distintas representam classes diferentes. Caso o número de classes seja conhecido, HACO aplica o algoritmo *Nearest-neighbor* (NN) para definir a quantidade correta de grupos. Uma consequência de se usar hipercaixas é que o resultado do agrupamento define também um classificador: se um novo dado está dentro de uma hipercaixa, sua classe será a mesma da definida por esta hipercaixa.

   Os resultados experimentais de HACO fora, comparados a três algoritmos que têm o mesmo fim: testado em NN, *Fuzzy C-Means* (FCM), e o próprio ACO (com uma abordagem diferente para agrupamento). O primeiro teste foi em conjuntos de dados sintéticos, e serviu como prova de conceito, oferecendo diversas informações sobre o comportamento do método em função de certas configurações. Um segundo experimento foi realizado com dados reais de pacientes para agrupá-los em "saudáveis" e "não saudáveis", e HACO obteve o melhor resultado dentre os algoritmos testados. A análise da estrutura do classificador gerado possibilita descobrir informações relativas às características das classes, indicando um "perfil de risco" para os pacientes.

   Foi apresentado o método HACO para agrupar dados, utilizando a meta-heurística ACO e hipercaixas, que possibilita a extração de informações inerentes a estrutura dos dados. HACO foi validado com experimentos, e demonstrou grande potencial. Os resultados são muito influenciados pela configuração dos parâmetros, que será investigada.

6. ***Análise crítica:*** ~~Este é o melhor artigo de todos os tempos.~~ O artigo apresenta uma forma inovadora de agrupar dados, de forma não-supervisionada (embora possa aproveitar informações se houver). O resultado pode ainda ser utilizado como classificador de novos dados, e - o mais interessante - analisado para descobrir informações sobre as classes. Além disso, explora as vantagens de cada elemento que compõe o método, obtendo melhores resultados e diminuindo o custo computacional. A aplicação em um caso real, cujos resultados podem ser utilizados para auxiliar o diagnóstico de pacientes, dá mais destaque ao trabalho.

   O problema de agrupamento de dados é muito pertinente e, em tempos de excesso de dados, a possibilidade de análise intuitiva da estrutura e descoberta de conhecimento é bastante interessante. Além disso, a solução proposta é de uso geral, oferecendo mais possibilidades de uso.

   Os experimentos realizados foram coerentes e suficientes para demonstrar o que foi afirmado. Entretanto, o método só foi comparado a outros algoritmos simples, seria interessante uma comparação com algoritmos mais avançados, bem como específicos para aplicação. A comparação também foi em uma única aplicação específica, seria melhor que houvesse mais testes com outros dados para conclusões melhor embasadas. Além disso, é preciso uma análise mais profunda quanto às configurações de HACO, que influenciam muito o resultado.

Anexo I

Documentação Original UnB-CIC (parcial)

```
% -*- mode: LaTeX; coding: utf-8; -*-
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% File    : unb-cic.cls (LaTeX2e class file)
%% Authors : Flávio Maico Vaz da Costa
%%               (based on previous versions by José Carlos L. Ralha)
%% Version : 0.96
%% Updates : 0.5  [??/11/2004] - Initial release. don't remember the day.
%%         : 0.75 [04/04/2005] - Fixed font problems, UnB logo
%%                               resolution, keywords and palavras-chave
%%                               hyphenation and generation problems,
%%                               and a few other problems.
%%         : 0.8  [08/01/2006] - Corrigido o problema causado por
%%                               bancas com quatro membros. O quarto
%%                               membro agora é OPCIONAL.
%%                               Foi criado um novo comando chamado
%%                               bibliografia. Esse comando tem dois
%%                               argumentos onde o primeiro especifica
%%                               o nome do arquivo de referencias
%%                               bibliograficas e o segundo argumento
%%                               especifica o formato. Como efeito
%%                               colateral, as referências aparecem no
%%                               sumário.
%%         : 0.9 [02/03/2008]  - Reformulação total, com nova estrutura
%%                               de opções, comandos e ambientes, adequação
%%                               do logo da UnB às normas da universidade,
%%                               inúmeras melhorias tipográficas,
```

```
%%                              aprimoramento da integração com hyperref,
%%                              melhor tratamento de erros nos comandos,
%%                              documentação e limpeza do código da classe.
%%          : 0.91 [10/05/2008] - Suporte ao XeLaTeX, aprimorado suporte para
%%                              glossaries.sty, novos comandos \capa, \CDU
%%                              e \subtitle, ajustes de margem para opções
%%                              hyperref/impressao.
%%          : 0.92 [26/05/2008] - Melhora do ambiente {definition}, suporte
%%                              a hypcap, novos comandos \fontelogo e
%%                              \slashedzero, suporte [10pt, 11pt, 12pt].
%%                              Corrigido bug de seções de apêndice quando
%%                              usando \hypersetup{bookmarksnumbered=true}.
%%          : 0.93 [09/06/2008] - Correção na contagem de páginas, valores
%%                              load e config para opção hyperref, comandos
%%                              \ifhyperref e \SetTableFigures, melhor
%%                              formatação do quadrado CIP.
%%          : 0.94 [17/04/2014] - Inclusão da opção mpca.
%%          : 0.95 [06/06/2014] - Remoção da opção "mpca", inclusão das opções
%%                              "doutorado", "ppginf", e "ppca" para identificar
%%                              o programa de pós-graduação. Troca do teste
%%                              @mestrado por @posgraduacao.
%%          : 0.96 [24/06/2014] - Ajuste do nome do curso/nome do programa.
%%
```