PROJECT DOCUMENT

코드 난독화 도구 한계점 분석

문제점 개요서

Project Name	코드 난독화 도구 제작

15 조

202002562 조인우
202002508 손지웅
201902686 노형우

지도교수: 조은선 교수님 (서명)

Document Revision History

Rev#	Date	Affected Section	Author
1	2025/03/17	초안 및 Survey Paper 작성	조인우
2	2025/03/21	Limitations and Research Gaps 작성	손지웅
3	2025/03/21	오탈자 수정	노형우

Table of Contents

1.Survey Paper - Limitations Focus

번호	연구 제목(저자)	저널/컨퍼런스(연도)	주요 내용 요약	한계점
1	OBFUS: An Obfuscation Tool for Software Copyright and Vulnerability Protection(Seoyeon Kang, Sujeong Lee, Yumin Kim, Seong-Kyun Mok, Eun-Sun Cho)	11th ACM Conference on Data and Application Security & Privacy (CODASPY '21)(2021)	이 논문은 고수준(C 언어)과 저수준(x86 어셈블리) 프로그래밍 언어를 대상으로 난독화 기술을 제공하는 웹 기반 도구인 OBFUS를 제안한다. OBFUS는 고수준 언어의 경우 ANTLR 4.8을 사용해 소스 코드를 파싱하고 난독화를 수행하며, 저수준 언어의 경우 RetDec 디컴파일러와 LLVM Pass를 활용해 바이너리 프로그램을 LLVM IR로 변환한 뒤 난독화를 적용하고 다시 컴파일한다. 주요 난독화 기법으로는 MBA 표현식, 불투명 조건(Opaque Predicates), 변수 이름 변경, 코드 가독성 제거, 가상화 등이 포함되며, 사용자는 원하는 옵션을 선택해 난독화를 적용할 수 있다. 실험 결과, OBFUS는 기존 도구(ADVobfuscator, Obfuscator-LLVM)와 비교해 소스 코드와 바이너리를 모두 난독화할 수 있는 유연성을 제공하며, 다양한 프로그래밍 언어(C, C++)와 플랫폼(x86, ARM 등)에서 호환성을 입증했다. 또한 웹 기반으로 설계되어 접근성과 사용 편의성이 높아 소프트웨어 저작권 보호 및 취약점 방지를 위한 효과적인 도구로 평가된다.	첫째, 지원되는 언어와 플랫폼이 제한적이며, Java나 Python과 같은 다른 언어는 지원하지 않는다. 둘째, 실험은 제한된 데이터셋과 환경에서 수행되어 결과의 일반화 가능성이 낮고, 대규모 소프트웨어 프로젝트에 대한 적용 가능성을 충분히 검증하지 못했다.
2	Obfuscapk: An Open-source Black-box Obfuscation Tool for Android(Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, Alessio Merlo)	SoftwareX, Vol. 11,(2020)	Obfuscapk는 원본 소스 코드 없이 APK 파일을 변환할 수 있는 오픈소스 안드로이드 앱 난독화 도구로, 다양한 자동화된 난독화 기법을 통해 앱 분석을 어렵게 만드는 특징을 가진다. 이 도구는 모듈형 구조로 설계되어 있어, 사용자는 추가적인 난독화 기법을 쉽게 확장할 수 있다. 또한, APK 테스트에서 Google Play Store에서 다운로드한 상위 1000개 앱을 난독화한 결과, 83%의 성공률을 보였다. Obfuscapk는 단순 기법(예: 앱 서명 변경, Manifest 파일 수정, APK 재빌드)과 복잡 기법(예: 문자열 암호화, 네이티브 라이브러리 암호화, 리플렉션 활용)을 지원하며, 메서드 호출 경로 변경, Goto 삽입, 메서드 오버로딩 등을 포함한 코드 난독화 기법을 적용한다. 특히, 악성코드 탐지	Obfuscapk의 주요 한계점은 APK의 안정성 문제로, 17%의 앱이 난독화 후 정상적으로 실행되지 않는다는 점이다. 또한, 머신러닝 기반 악성코드 탐지 기법의 발전에 따라 Obfuscapk가 생성한 난독화 코드가 장기적으로 효과를 유지할지 불확실하다. 마지막으로, 난독화된 앱이 실행 중 동적 분석(예: 후킹 및 디버깅)에 대해 얼마나 강한지에 대한 평가가 부족하여 동적 분석에 취약할 수 있다는 문제가 존재한다.

			회피 테스트에서는 CometBot 악성코드 샘플에 특정 난독화 기법을 적용한 결과, 탐지율이 55%에서 0%로 감소하는 효과를 나타냈다.	
3	강화된 난독화 기법을 통한 안드로이드 애플리케이션의 보안 강화(이동호, 조해현, 손기욱)	한국사이버안보학회(2024)	이 논문은 역공학 및 악성 공격에 대응하기 위해 안드로이드 애플리케이션을 보호하는 강화된 난독화 기법을 제안하고, 정적 및 동적 역난독화 기술에 대한 저항력을 검증한 연구이다. 제안된 난독화 기법은 ① Sensitive Code Split(민감한 코드를 탐지하여 분리된 .dex 파일로 나누고 원격 서버에 저장해 분석을 어렵게 함), ② Screen Element Protection (SEP)(UI 요소를 보호하여 자동화된 동적 분석 도구로부터 앱을 보호함), ③ Android Data Storage Space Utilization (ADSSU)(데이터를 안드로이드 저장 공간에 숨기는 스테가노그래피 기법을 활용해 정보 보호), ④ Source Code Obfuscation(문자열 난독화, 클래스 난독화, API 은닉 기법을 적용해 코드 분석을 어렵게 만듦)으로 구성되며, 이를 통해 기존 난독화 기법보다 높은 보안성을 제공하는 것을 목표로 한다.	첫째, 성능 저하 문제로, Sensitive Code Split 기법은 분리된 코드(SC.dex)를 서버에서 불러오는 과정이 필요하기 때문에 실행 속도가 느려질 수 있으며, 네트워크 환경이 좋지 않을 경우 앱 실행에 지연이 발생할 가능성이 있다. 둘째, 숙련된 역공학자에 의한 분석 가능성으로, 자동화된 분석 도구에 대해서는 저항력을 갖추었지만, Frida 및 Android Open Source Project (AOSP)와 같은 도구를 활용하면 일부 난독화 기법이 무력화될 가능성이 있다. 셋째, 난독화 기법에 대한 표준화된 데이터셋 부재로, 연구에서는 다양한 난독화 기법을 적용한 후 성능을 평가했지만, 공식적으로 인정된 표준 난독화 평가 데이터셋이 부족하여 연구의 재현성을 높이기 위해 공개 데이터셋을 활용한 추가 검증이 필요하다. 넷째, 고급 난독화 기술(LLVM 기반 등)의 부재로, LLVM-Obfuscation과 같은 최신 기법을 적용하지 않아 기존 난독화 기법과의 비교가 제한적이며, 최신 AI 기반 난독화 우회 기술을 고려한 추가적인 보안 기법 연구가 필요하다.

2.Limitations and Research Gaps

번호	기존 연구	한계점	연구 필요성	본 연구의 기여
1	A Systematic Literature Review of Code Obfuscation Techniques and Tools (Samuel Ardianto, Bagus Jaya Santosa, et al. , 2023)	Ardianto et al. (2023)은 최근 10년간 발표된 코드 난독화 기술과 도구들을 체계적으로 검토하며 다양한 기법을 유형별로 분류하고 주요 도구들을 비교 분석하여 기존 연구의 흐름을 파악하는 데 기여하였다. 그러나 이 연구는 몇 가지 한계점을 지닌다. 첫째, 난독화 기술의 보안성을 주장하면서도 공격자 모델에 따른 정량적 평가가 부족하여 리버스 엔지니어링 저항성에 대한 객관적 비교가 어렵다. 둘째, 성능 저하나 유지보수성 문제 등 실제 소프트웨어 개발 환경에서의 적용 가능성에 대한 고려가 미흡하다. 셋째, 난독화 효과를 평가하는 기준이 연구마다 달라 기술 간 비교의 신뢰성이 떨어지며, 이를 위한 표준화된 벤치마크나 평가 프로토콜이 부재하다. 마지막으로, 난독화 도구들의 자동화 수준이나 Java, Android, .NET 등의 개발 툴체인과의 호환성에 대한 분석이 부족하고, 도구 간 비교도 이론 중심으로 이루어져 실사용 관점에서의 통찰이 부족하다.	정량적 보안성 평가 모델의 필요성이 대두됨에 따라, 공격자의 수준(스킬, 도구, 시간 등)에 따른 다층적인 난독화 저항성 평가 기준이 요구되며, Symbolic Execution이나 Decompiler 기반 분석 도구에 대한 저항력을 계량화할 수 있는 방법이 필요하다. 또한, 난독화 기술의 보안 효과뿐 아니라 실제 소프트웨어 시스템에 미치는 실행 성능 저하 및 유지보수 비용을 함께 고려한 통합 평가 지표 개발이 중요하며, 다양한 난독화 도구를 동일한 조건에서 비교할 수 있도록 표준화된 테스트베드와 벤치마크(예: ObfuscationBench) 구축이 필요하다. 더불어, 클라우드 네이티브, 컨테이너 기반 환경, 웹어셈블리 등 현대 개발 환경에 적합한 새로운 난독화 기법의 고안도 요구된다.	Ardianto et al. (2023)의 연구는 최근 10년간 발표된 코드 난독화 기법과 도구에 대해 체계적인 문헌 조사를 수행함으로써, 난독화 기술의 분류 체계(문법적, 제어 흐름, 데이터 변형 등)를 정립하고, 각 기술의 적용 사례 및 도구(예: ProGuard, Allatori 등)를 종합적으로 비교한 점에서 큰 기여를 한다. 특히 난독화 기술의 목적, 방식, 효과를 기술적 관점에서 정리함으로써, 후속 연구자들이 난독화 기술을 객관적으로 비교·분석할 수 있는 기반을 마련하였으며, 다양한 도구의 특징 및 한계점을 기술하여 실무적 활용 가능성에 대한 통찰도 함께 제공하였다.
2	Deobfuscation Processing and Deep Learning-Based Detection Method for PowerShell-Based	해당 연구에서는 PowerShell 기반 악성코드의 역난독화 기법과 딥러닝을 활용한 탐지 방법을 제안하였다. 그러나 본 연구는 주로 악성코드 탐지에 초점을 맞추고 있어, 일반적인	다양한 프로그래밍 언어와 플랫폼에서 적용 가능한 범용적인 코드 난독화 도구의 개발이 필요하다. 또한, 난독화된 코드의 역공학을 방지하면서도 성능 저하를 최소화하는	본 연구에서는 다양한 프로그래밍 언어와 플랫폼에서 적용 가능한 범용적인 코드 난독화 도구를 개발하고, 이를 통해 코드의 보안성을 향상시킨다. 또한, 성능 저하를 최소화하면서도 역공학을 방지할 수 있는 효율적인 난독화

	Malware(Ho-jin Jung, Hyo-gon Ryu, Kyu-whan Jo, Sangkyun Lee, 2022)	코드 난독화 도구의 개발 및 적용에 대한 직접적인 논의는 부족한 실정이다. 또한, 제안된 기법이 다른 스크립트 언어나 바이너리 코드에 적용될 수 있는지에 대한 검증이 이루어지지 않았다.	효율적인 난독화 기법에 대한 연구가 요구된다.	기법을 제안한다. 정량적으로는 코드 분석 시간의 증가와 성능 저하율을 측정하고, 정성적으로는 보안성 향상에 대한 평가를 수행한다.
	OBFUS: An Obfuscation Tool for Software Copyright and Vulnerability Protection(Seoyeon Kang, Sujeong Lee, Yumin Kim, Seong-Kyun Mok, Eun-Sun Cho, 2021)	이 연구는 C 언어와 x86 바이너리에 한정되어 있어 다양한 언어 및 플랫폼을 지원하지 못하며, 제공하는 난독화 기법도 비교적 단순하여 최신 역공학 기법에 대한 방어력이 부족하다. 또한, 웹 기반 도구로 설계되어 있어 대규모 프로젝트에서 자동화하기 어렵고 성능 최적화 및 확장성 면에서 한계를 보인다.	보다 강력한 난독화 도구를 개발하기 위해서는 Java, Python, JavaScript 등 다양한 언어를 지원하는 기능이 필요하며, 기존의 기본적인 난독화 기법을 넘어 제어 흐름 평탄화(Control Flow Flattening), 명령어 가상화(Instruction Virtualization) 등 최신 난독화 기법을 적용할 필요가 있다. 또한, 대규모 프로젝트에서도 활용할 수 있도록 CLI 기반 도구를 추가하여 CI/CD 환경에서도 쉽게 통합할 수 있는 확장성이 요구된다.	본 연구에서는 기존 연구의 한계를 극복하기 위해 다양한 프로그래밍 언어를 지원하는 범용 난독화 도구를 개발하고, 최신 난독화 기법을 적용하여 보안성을 향상시킨다. 또한, 웹 기반 도구뿐만 아니라 CLI 기반 도구도 함께 제공하여 자동화를 지원하고, 대규모 프로젝트에서도 적용할 수 있도록 확장성을 고려하여 설계한다.