Journal of KIIT. Vol. 23, No. 3, pp. 109-115, Mar. 31, 2025. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2025.23.3.109

소프트웨어중심 자동차(SDV) 원격 접속 보안성 강화에 대한 연구

심윤보*, 윤철희**, 박장우***

Research on Strengthening Remote Access Security of Software-Centric Vehicles(SDVs)

Yunbo Sim*, Cheolhee Yoon**, and Jangwoo Park***

본 연구는 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임
(No. RS-2024-00337489, 분석 모델의 성능저하 극복을 위한 데이터 드리프트 관리 기술 개발)

요 약

최근 자동차 산업에서 소프트웨어중심 자동차의 중요성이 증가함에 따라, 소프트웨어를 통한 차량의 제어, 시동 등 모바일 애플리케이션의 보안 강화가 주요 과제로 대두되고 있다. 본 연구에서는 H사의 APK를 대상으로 애플리케이션의 보안 메커니즘을 분석하고 잠재적 취약점을 식별하였다. Jadx-gui, Frida, rootAVD 등의 분석 도구를 활용하여 애플리케이션의 Entry Point 분석, 동적 코드 로딩 기법 분석, 애플리케이션 프로텍터 분석, 그리고 루팅 탐지 우회 가능성을 확인하였다. 추가적인 실험을 통해 시동에 사용되는 모바일 애플리케이션의 구현 현황과 한계점을 확인하였으며, 앱 데이터드리프트 분석을 통해 향후 소프트웨어중심 자동차(SDV) 보안 강화를 위한 실질적인 개선 방안을 제시하였다. 본 논문의 결과를 통해 소프트웨어중심 자동차 애플리케이션의 보안설계 및 구현 설계에 있어 중요한 연구가 될 것으로 기대한다.

Abstract

As the importance of software-centric vehicles have increased in the automobile industry, strengthening the security of mobile applications such as vehicle control and starting through software has emerged as a major task. In this study, we analyzed the security mechanism of the application and identified potential vulnerabilities targeting H Company's APK. Using analysis tools such as Jadx-gui, Frida, and rootAVD, we analyzed the application's entry point, analyzed dynamic code loading techniques, analyzed application protectors, and confirmed the possibility of bypassing rooting detection. Through App datdrifting experiments, we confirmed the implementation status and limitations of mobile applications used for startup, and suggested practical improvement measures for strengthening the security of Software-centric Vehicles(SDVs) in the future. The results of this paper are expected to be an important study in the security design and implementation design of software-centric low-volume vehicle applications.

Keywords

mobile app security, APK analysis, vehicle control application, dynamic code loading, app protector, rooting detection, app data-drifting

* 교보문고 유통시스템 차장
- ORCID: https://orcid.org/0009-0001-3716-7985
** 경찰대학교 치안정책연구소 연구관(공동1저자)
- ORCID: https://orcid.org/0000-0002-4862-4790
*** 아주자동차대학교 미래자동차공학부 교수(교신저자)
- ORCID: https://orcid.org/0000-0003-3904-5882

· Received: Nov. 17, 2024, Revised: Feb. 28, 2025, Accepted: Mar. 03, 2025
· Corresponding Author: Jangwoo Park
Ajou Motor College
Tel.: +82-41-939-3093, Email:pjw@motor.ac.kr

Ⅰ. 서 론

현재 스마트폰에서 사용되는 모바일 애플리케이션 프로그램은 일상생활의 필수적인 부분이 되었으며, 특히 자동차 관련 모바일 애플리케이션은 차량 제어, 상태 모니터링, 원격 시동 등 다양한 기능을 제공하며 급속도로 발전하고 있다. 이러한 발전과 함께 모바일 애플리케이션의 보안 중요성 또한 크게 부각되고 있다. 특히 차량 제어 관련 애플리케이션의 경우, 보안 취약점이 발견될 경우 사용자의 안전과 직결될 수 있어 높은 수준의 보안이 요구된다. 본 논문에서는 실험한 A사의 애플리케이션은 차량 원격 제어, 차량 상태 확인, 주행 기록 관리 등 다양한 서비스를 제공하는 대표적인 커넥티드 카 플랫폼으로 이러한 애플리케이션은 사용자 인증, 데이터 암호화, 애플리케이션 무결성 검증 등 다양한 보안 메커니즘을 적용하고 있다. 관련 중요한 보안 메커니즘의 실효성과 견고성에 대한 분석이 필요한 시점으로 본 연구에서는 A사의 APK를 대상으로 애플리케이션의 구조와 보안 메커니즘을 분석하였다. 잠재적인 취약점을 식별하며, 이에 대한 대응 방안을 제시하려 하였으며, 특히 동적 코드 로딩, 애플리케이션 프로텍터, 루팅 탐지 등의 보안 기술이 실제로 어떻게 구현되어 있는지 심층적으로 분석하였다. 또한, SDV에 포함되는 보안 메커니즘의 데이터 드리프트 방지 및 우회 가능성을 검토함으로써 더욱 강화된 보안 방안을 모색하였다.

Ⅱ. 본 론

2.1 SDV 접속 애플리케이션 분석

본 논문의 분석 대상은 Android B apk 3.880으로 한정하였으며, 분석도구로는 Jadx-gui 1.5, frida 16.5.6, frida-tools 13.6.0, firda-dexdump, rootAVD를 사용하였다. 안드로이드 에뮬레이터 분석 환경 설정의 경우 분석도구로는 Jadx-gui 1.5, frida 16.5.6, frida-tools 13.6.0, firda-dexdump, rootAVD를 사용하였다. 분석을 시도한 기간은 24년 하반기에 릴리즈된 애플리케이션이며, 실행을 위한 안드로이드 에뮬레이터는 설치되어 있다고 가정하고, 개발자 메뉴를 활성화시킨 후 분석이 가능하도록 RootAVD를 이용하였다. 실험을 위해 설치된 AVD(Android Virtual Device) 환경을 확인하여, 분석하고자하는 에뮬레이터 타입 및 API 버전에 맞는 Rooting을 실행 시도하였다. 순차적으로 Rooting이 완료되면 Magisk 애플리케이션 및 Zygisk 활성화를 통해 실험을 수행하였다. 그림 1 개발자 메뉴활성화는 개발자 메뉴를 활성화하기 위해서 API버젼 루팅을 준비하는 시작점으로 SDV 접속 애플리케이션을 취약점을 분석하였다. 단계적으로 AVD 에뮬레이터 내 Frida Server 실행한 다음 Frida Server Process를 확인이 가능하며 이후 분석하고자 하는 A사의 B 애플리케이션을 진입점을 확인할 수 가 있다.

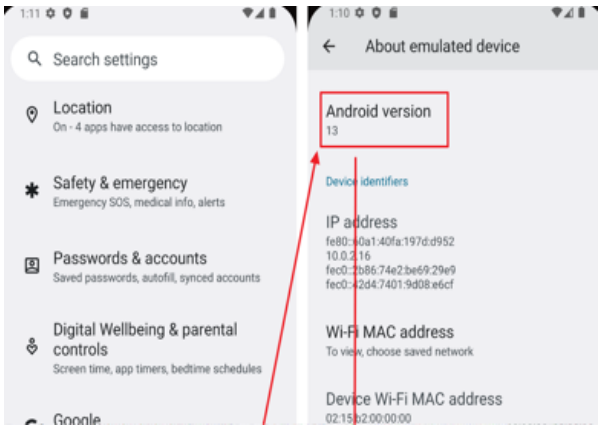

그림 1. 개발자 메뉴 활성화
Fig. 1. Activate developer menu

실험 준비가 되면 Magisk라는 Android 기기를 루팅하는데, 루팅은 모바일 기기에서 구동되는 안드로이드 운영 체제 상에서 최상의 권한을 얻거나 해당 기기의 제약을 해재하는 행위를 통해 본 실험을 수행한다. 그림 2의 에뮬레이터 타임 및 API 버전에 맞추어 Rooting을 실행하게 된다. 본 시스템에서는 리스 루팅 프로그램을 사용하는데 시스템 리스 프로그램이란, 시스템 파티션을 변경하지 않고 루팅을 하는 방식을 뜻한다. 즉, 본 실험에서는 사용자에게 허용되지 않는 루트 권한을 강제로 부여하는 작업을 수행하고, 관련된 시스템리스 루트 프로그램인 Magisk를 사용하였다.

```
Command Examples:
rootAVD.bat
rootAVD.bat ListAllAVDs
rootAVD.bat InstallApps

rootAVD.bat system-images\android-33\google_apis_playstore\x86_64\ramdisk.img
```

그림 2. 에뮬레이터 타입 및 API 버전에 맞는 Rooting 실행
Fig. 2. Execute rooting appropriate for emulator type and API version

Magisk Manager는 안드로이드 OS를 루팅을 할 수 있도록 도와주는 도구로 그림 3 Magisk 실행 화면과 같이 Magisk 애플리케이션을 구동하여 A사의 B 애플리케이션을 진입점을 확인할 수 있다.

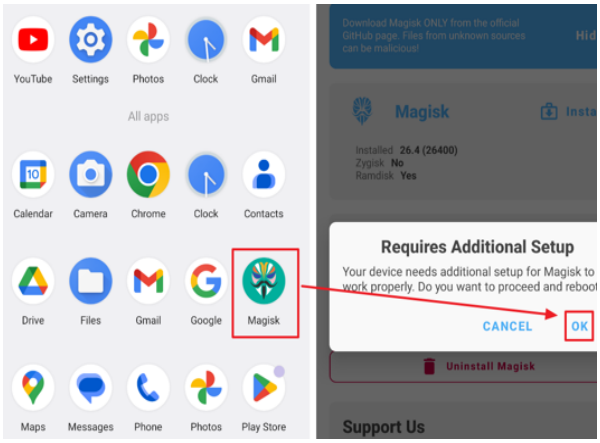

그림 3. Magisk 실행
Fig. 3. Running of magisk

이후 Magisk-frida라는 magisk 모듈을 이용해 frida-server을 안드로이드 기기에 추가 설치 후 순차적으로 파일 실행권한을 부여하고, AVD(Android Virtual Device)를 통해 Android Emulator에서 시뮬레이션하려는 애플리케이션을 AVD 에뮬레이터에 Frida Server를 통해 분석할 수 있도록 준비하는 절차를 거친다.

2.2 Datadrift 통한 원격시동 애플리케이션 분석

AVD는 Android Emulator를 이용해 시뮬레이션 가능케 하는 Android mobile, 태블릿, Wear OS, Android TV, Automotive OS 기기를 구현하는 관리 도구로 앞서 언급하였듯이, AVD를 만들고 관리하는 데 도움을 주는 도구로 Android 스튜디오에서 실행할 수 있다. 이러한 AVD 이용해서 앞에서 설치된 애플리케이션을 실행하면 스플레시 화면과 검은색 화면이 반복적으로 전환되어 나타나며, 이후 순차적으로 APP Entry Point 및 Dex dynamic code loading을 확인하기 위해, 애플리케이션 APK 파일을 jadx-gui로 실행한다. 이후 APK 진입점(Entry point) 확인을 위해 왼쪽 패널의 '리소스'탭의 'AndroidManifest.xml'을 클릭하여 확인하고, 'AndroidManifest.xml' 파일을 통해 MAIN과 LAUNCHER 키워드로 해당 APK의 시작 Activity를 파악하게 된다. 해당 APK 파일은 일부 코드들이 애플리케이션 기동 시 구동되는데 이때 동적 코드로 크게 3가지 메소드(DexClassLoader, PathClassLoader, InMemoryDexClassLoader)가 사용되고, 이 메소드들을 검색하면 PathClassLoader를 사용하는 코드가 존재하는 것을 알 수 있고, 해당 코드 상단에 선언된 내용을 통해 BaseDecClassLoader 등이 그림 4 Loader검색과 같아 메소드가 호출되서 사용되고 있는 것을 확인할 수 있다.

```
6  import android.os.UserHandle;
7  import com.apk_shield.skdb;
8  import dalvik.system.BaseDexClassLoader;
9  import dalvik.system.PathClassLoader;
0  import java.io.ByteArrayOutputStream;
1  import java.io.DataInputStream;
2  import java.io.DataOutputStream;
```

그림 4. Loader 검색
Fig. 4. Loader search

SDV 관련 애플리케이션은 일반적으로 App Protector가 설치되어있기 때문에, lib 폴더에 native 라이브러리들이 존재하는지 확인하고 여러 개의 관련 파일들이 존재함을 확인하는 절차를 확인하게 된다. 존재하는 코드상에서 의심스러운 부분은 패널상 소스코드 트리에 노출된 com.apk_shield 카테고리이며, 그 부분을 지속적으로 분석하여 APP Protector를 정보를 얻을 수 있다. 해당 카테고리 내에는 2개의 클래스와 하나의 인터페이스가 존재하는데 그중 skdb라는 클래스가 코드량이 방대하며 무언가 암호화 또는 인코딩된 문자열들이 보이고 있기 때문에, 루팅에 관련된 상수정의를 하는 것처럼 추정할 수가 있다.

끝이 "==" 또는 "="으로 끝나는 문자열은 Base64로 인코딩되어 있음을 확인할 수 있고, 루팅에 사용되는 su, supersu 등의 바이너리나 루팅 또는 디버깅에 관련된 에러 메세지를 인코딩해놓은 것을 확인할 수가 있다. 즉, 애플리케이션 프로텍터인 것을 알 수 있다. 지속적으로 코드를 확인하면 수많은 리소스 ID와 같은 상수 선언들이 이어지다 코드의 마지막 부근에서 함수와 코드 로직이 나타나는 것을 파악할 수 있으며, 중간지점에 native 메소드를 호출하는 것을 알 수 있다. 해당 함수들이 실제로 애플리케이션 기동시 호출 되는지 간단하게 Frida의 코드를 사용해서 확인할 수 있으며, Frida script를 명령어로 실행했을 때 깜빡임이 사라진 것을 알 수 있다. 이 후 그림 5와 같이 library 상태의 존재를 파악이 가능해 진다.

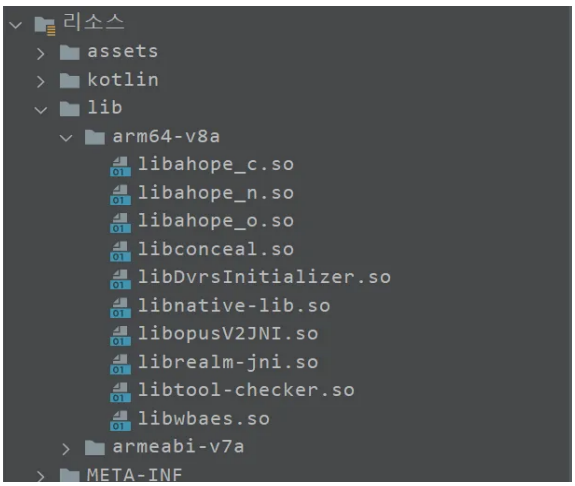

그림 5 lib 폴더상 native 라이브러리 존재 확인
Fig. 5. Check the existence of native library in the lib folder

Rooting Bypass를 위해서, 앞에서 언급한 동적 코드 로딩 기법을 사용하는데 숨겨진 코드를 메모리 덤프(Frida-dexdump)를 통해 복원하고 덤프를 실행한 후, 해당 경로 하위에 b○○link라는 폴더와 함께 덤프된 dex 파일들이 생성하게 된다. ex-dump에서 덤프된 파일을 추가로 로드하면 동적 로드된 코드들이 보여지는데 처음 로드했던 apk 소스보다 많은 양의 숨겨진 코드가 추가되었음을 알 수 있으며, AndroidMenifest.xml 파일에서의 애플리케이션 시작점 코드 및 시작 화면을 확인할 수 있다.

그림 6과 같이 루팅이 제한되어있기 때문에, 실험에서 유의할 점은 루팅을 탐지하는 방법을 고려해야 한다. 대부분 su, superuser, magisk 등 슈퍼유저 권한을 부여하는 권한 상승 바이너리가 존재하는지 검사하는 키워드들이 존재하는 경우가 많기때문에 본 실험에서 처럼 루팅 시 사용했던 방법을 사용하여야한다. 그림 7에서 보듯이 RootAVD에서 설치되는 magisk 바이너리를 검사하는 로직이 있는지 확인 후 우회를 하는데 magisk 키워드 검색 결과를 통해 확인할 수 있다.

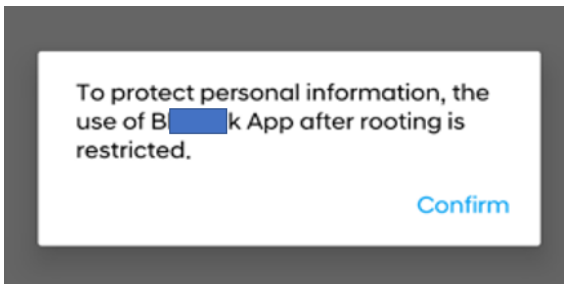

그림 6. 애플리케이션 화면 확인
Fig. 6. Check the application screen

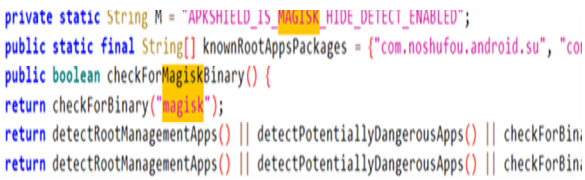

그림 7 "magisk" 키워드 검색 결과
Fig. 7. "magisk" keyword search results

전체 코드의 해당 클래스를 살펴보면 여러가지 루팅 탐지 로직들이 존재함을 알 수 있으며, 루팅된 기기에서 자주 발견되는 바이너리 경로와 파일명 등을 상수로 선언하고, 그 내역을 순회하며 검색하는 로직들도 탐색할 수가 있다. RootBeer 클래스내의 여러가지 루팅 탐지 메소드들을 Frida로 재작성하여 루팅 탐지를 우회할 수가 있다. 하지만, 이때 문제점은 애플리케이션이 실행은 되지만 루팅이 탐지되어, 확인 버튼을 누르면 애플리케이션이 종료되기 때문에 확인된 isRooted()에 나열된 메소드들 전부 Frida 스크립트를 사용하여 재작성해야하는 작업을 추가해 주어야 한다. 이때 주의할 점은 현재 동적 코드 로딩으로 RootBeer 클래스가 로딩됨으로 해당 클래스가 로드될 때까지 계속해서 메모리 영

역을 검색하고, 클래스가 발견되어 지면 발견 플래그를 업데이트하여 더 이상 수행되지 않도록 변경해야 한다는 점으로 해당 클래스의 메소드들을 재작성하면 해결이 된다. 이후 해당 그림 8 루팅된 장비에서 실행확인 처럼 스크립트를 포함하여 재실행하면 깔끔하게 루팅 탐지가 우회되어 루팅된 장비에서 실행된 모습을 확인할 수 있다.

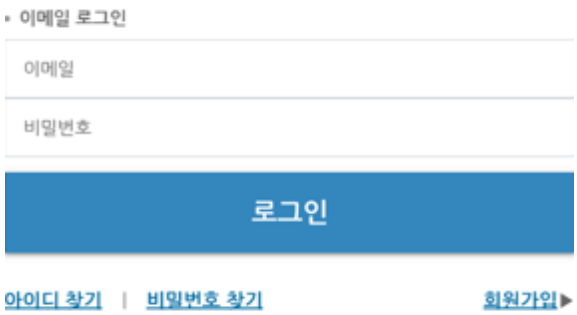

그림 8. 루팅된 장비에서 실행 확인
Fig. 8. Confirm running on rooted device

III. 결 론

현재는 사이버 위협의 증대와 사용자 데이터 보호의 중요성이 대두되면서, 다양한 방법으로 애플리케이션 보안을 강화하려는 노력이 활발히 이루어지고 있다. 불과 얼마 전 과거에는 모바일 애플리케이션(App) 보안에 대한 인식이 낮아, 많은 애플리케이션들이 기본적인 보안조치도 없이 출시되곤 했다. 그러한 문제점들을 보안하기 위해 지속적으로 취약점을 분석하고, 앱플리케이션 보안을 강화하고 있는 활동을 하고 있지만, SDV에 사용되는 애플리케이션의 앱 데이터를 드리프트(Drift)하는 방법을 통해 루팅을 시도하기도 하는 점을 실험으로 확인하였다.

앞서 실험에서 간단한 도구를 이용해 루팅을 하는 방법을 선보였으며, 여전히 애플리케이션 datadrift를 통해 애플리케이션의 보안 메커니즘을 우회하는 방법들이 존재하며, 완벽한 보안 구현은 여전히 도전 과제로 남아 있음을 확인하였다.

향후 많은 애플리케이션들은 데이터 암호화, 코드 난독화, 인증 절차 강화 등 다양한 보안 기술 적용, 사용자 인증을 위해 다중 인증(Multi-factor authentication), 데이터 전송 시 SSL/TLS 암호화, 모바일 플랫폼 자체 보안 강화, Data-Drift 방지 업데이트, 패치 등을 지속적으로 수행하면서 동시에 역공학 및 디버깅 도구를 활용한 보안 우회 방법을 대응하는 연구를 수행해야 한다.

본 논문에서는 APK 파일을 디컴파일하여 소스코드를 분석하거나, 런타임 환경을 조작하여 보안 검사를 회피하는 등의 방법을 사용하여 이러한 우회 기법을 실험하였으며, 상대적으로 쉽게 구할 수 있는 간단한 도구와 지식만으로도 수행할 수 있었다. 결과적으로 시큐어 코딩이 약한 애플리케이션일수록 취약성이 높아지기 때문에, 역공학 분석을 방해하고 우회 시도를 지연시키는 코드 난독화와 동적 분석 방지 및 루팅 및 탈옥 감지 등 다양한 기술을 동시에 적용해야 함도 본 논문의 학술적 의의로, 코드 난독화는 소스 코드를 이해하기 어렵게 만들어 분석 시간을 증가시키고 동적 분석 방지는 디버깅 도구의 접근을 제한하여 실시간으로 보안 검사를 우회하는 것을 어렵게 만들기 때문에 루팅 또는 탈옥된 기기에서의 애플리케이션 실행을 차단할 수 가 있다. 마지막으로 민감 정보를 처리하는 애플리케이션은 기본적인 보안 조치 외에도, 정기적인 보안 감사와 취약점 분석을 통해 지속적으로 보안 수준 향상, SDV 앱 애플리케이션 정보보호를 위해 지속적인 노력, 최신 보안 기술의 적용 등을 통해 보안취약점을 지속적으로 관리하고, 사용자 데이터 안전을 위해 최선의 노력을 기울여야 할 것이다.

References

[1] A. Datta, K. Dutta, S. Kajanan, and N. Pervin, "Mobilewalla: A mobile application search engine", Mobile Computing, Applications, and Services, Vol. 95, pp. 172-187, Oct. 2012. https://doi.org/10.1007/978-3-642-32320-1_11.

[2] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces", ACM CODASPY 2012, pp. 317-326, Feb. 2012. https://doi.org/10.1145/2133601.2133640.

[3] T. W. Hou, H. Y. Chen, and M. H. Tsai, "Three

Control Flow Obfuscation Methods for Java Software", IEE Proceedings-Software, Vol. 153, No. 2, pp. 80-86, Apr. 2006. https://doi.org/10.1049/ip-sen:20050043.

[4] Y. Piao, J. H. Jung, and J. H. Yi, "Server based code obfuscation scheme for APK tamper detection", Security and Communication Networks, Vol. 9, No. 6, pp. 457-467, Mar. 2014. https://doi.org/10.1002/sec.989.

[5] J. Kim and E. Lee, "A strategy of effectively applying a control flow obfuscation to programs", J. Korea Soc. Comput. Inf., Vol. 16, No. 6, pp. 41-50, Jun. 2011. https://doi.org/10.9708/jksci.2011.16.6.041.

[6] K. I. Shin, J. S. Park, J. Y. Lee, and J. H. Park, "Design and implementation of improved authentication system for android smartphone users", IEEE WAINA, pp. 704-707, Mar. 2012. https://doi.org/10.1109/WAINA.2012.29.

[7] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, "Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds", Advances in Cryptology–EUROCRYPT 2010, Vol. 6110, pp. 299-319, May 2010. https://doi.org/10.1007/978-3-642-13190-5_15.

[8] B. Alex and D. Khovratovich, "Related-key cryptanalysis of the full AES-192 and AES-256", Advances in Cryptology–ASIACRYPT 2009, Vol. 5912, pp. 1-18, Dec. 2009. https://doi.org/10.1007/978-3-642-10366-7_1.

[9] J. Y. Kim, N. H. Ko, and Y. S. Park, "A code hiding technique using Java reflection and dynamic loading in Android environment", Journal of the Korea Institute of Information Security & Cryptology, Vol. 25, No. 1, pp. 17-30, Feb. 2015. https://doi.org/10.13089/JKIISC.2015.25.1.17.

[10] J. W. Park and Y. S. Park, "Automatic Detection Techniques of Anti-Debugging Routines according to Analysis Environment", Journal of Korean Society for Internet Information, Vol. 15, No. 6, pp. 47-54, Dec. 2014. https://doi.org/10.7472/jksii.2014.15.6.47.

[11] M. G. Kang, H. Yin, S. Hanna, S. McCamant, and D. Song, "Emulating Emulation-Resistant Malware", Proceedings of the 1st ACM Workshop on Virtual Machine Security (VMSec '09), pp. 11-22, Nov. 2009. https://doi.org/10.1145/1655148.1655151.

[12] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly", Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03), pp. 290-299, Oct. 2003. https://doi.org/10.1145/948109.948149.

[13] J. Kim, Y. H. Woo, and Y. Park, "An automatic identifier renaming technique for Java software protection", Journal of the Korea Institute of Communications and Information Sciences, Vol. 40, No. 4, pp. 709-719, Apr. 2015. https://doi.org/10.7840/kics.2015.40.4.709.

[14] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques", ACM Computing Surveys, Vol. 49, No. 4, pp. 1-41, Jan. 2017. https://doi.org/10.1145/3017427.

[15] L. Xing, X. Pan, R. Wang, K. Yuan, and X. Wang, "Upgrading your Android, elevating my malware: Privilege escalation through mobile OS updating", IEEE Symposium on Security and Privacy, pp. 393-408, May 2014. https://doi.org/10.1109/SP.2014.32.

저자소개

심 윤 보 (Yunbo Shim)

2006년 8월 : 순천향대학교 정보기술공학부(공학학사)
2024년 8월 : 연세대학교 정보대학원 AI빅데이터(공학석사)
2008년 2월 ~ 현재 : 교보문고 유통시스템 차장
관심분야 : 딥러닝, XAI, Python, Reverse engineering, OpenCV, RPA

윤 철 희 (Cheolhee Yoon)

2004년 2월 : 한성대학교 정보시스템학과(공학사)
2016년 8월 : 고려대학교 디지털포렌식학과(공학석사)
2023년 2월 : 연세대학교 기술정책(공학박사)
2024년 8월 : 극동대학교 인공지능보안학과(공학박사)
2017년 6월 ~ 현재 : 치안정책연구소 연구관
관심분야 : 데이터분석, 딥러닝

박 장 우 (Jangwoo Park)

2015년 8월 : 서울과학기술대학교 자동차공학과(공학석사)
2020년 2월 : 호서대학교 기계자동차공학부(공학박사)
2007년 3월 ~ 2021년 3월 : 두원공과대학교 자동차과 겸임교수
2021년 4월 ~ 현재 : 아주자동차대학교 미래자동차공학부 교수
관심분야 : 자동차구조해석, 소음·진동, 자동차서비스