Generic Vulnerability Analysis Based on Large-Scale Automotive Software

Chenya Bian(✉), Yuqiao Ning, Qingyang Wu, Longhai Yu, and Yang Chen

CATARC Intelligent and Connected Technology Co., Ltd., Tianjin, China
{bianchenya,ningyuqiao,wuqingyang,yulonghai,
chenyang2022}@catarc.ac.cn

Abstract. With the continuous development of the intelligent connected vehicle, the scale of automotive software system structure is expanding, and the possibility of security vulnerability is increasing. To improve the low adaptability of traditional vulnerability scanning tools in the ICV system environment and the inaccurate vulnerability results, this paper proposes a vulnerability scanning technology for large-scale ICV software programs. By extracting the software program, performing feature extraction and component analysis, and matching the vulnerability information of the open source vulnerability database, the technology achieves more accurate identification and judgment of the components and vulnerability information in the automotive software program, and can meet the vulnerability scanning requirements of various automotive software programs.

Keywords: Intelligent Connected Vehicle · Automotive Software · Vulnerability Scan · Software Composition Analysis

1 Introduction

The development and popularization of the ICV (Intelligent Connected Vehicle), not only brings convenience to people's production and life but also poses a great challenge to automobile information security. The ICV is a complex intelligent computing system with a large number of structural modules, including many electronic devices such as IVI, T-BOX, Gateway, etc. Each structural module may introduce various vulnerabilities to the vehicle system.

Vulnerability Scanning technology is a security detection behavior based on a vulnerability database to detect the security vulnerability of the designated remote or local system and discover exploitable vulnerabilities by identification, scanning, analysis, and other means. [1]. The application scenarios and features of some of the vulnerability scanning tools widely used in the IT field are shown in Table 1 [2]. However, traditional vulnerability scanning tools are only suitable for vulnerability detection of traditional IT architecture systems. If we use the tools directly in automotive vulnerability scanning, false positives and false negatives of vulnerability results will increase, and there will be a precipitous decrease in the accuracy of vulnerability analysis.

D. Kumar et al. (Eds.): IEIT 2023, AHSSEH 10, pp. 43–56, 2023.
https://doi.org/10.2991/978-94-6463-230-9_7

Table 1. Traditional vulnerability scan tools

Tools	Scanned Objects	Scanning Features
AppScan	Web	Crawl and scan all URLs of the website to automate testing for web vulnerabilities.
Nessus	Host	Use client/server mode. The server side performs security scanning checks on the specified network environment, the client side uses the configuration management server [3].
MobSF	APP	Perform static, dynamic, and malware analysis for Android, iOS, and Windows mobile applications.

The common feature of many modular electronic devices in ICV is the existence of a large number of software programs. Software programs are large in file size, have many components, and introduce third-party components, which inevitably create vulnerabilities.

In addition, the in-vehicle network and radio that provide internal and external communication for the vehicle are also integrated into the software, with similar vulnerability risks [4]. Therefore, it is particularly important to design vulnerability scanning tools for the software programs of each module of the entire ICV to meet the vulnerability scanning needs of large-scale automotive software. It is also important to ensure that the time dimension is acceptable to the business and security teams.

In this paper, a vulnerability scanning technology for large-scale automotive software is proposed. The feature of the software program is extracted with its components analyzed. The vulnerability information of the open source vulnerability database is matched, thus getting the vulnerabilities in the software program. Based on this technology, we implemented a software system to conduct vulnerability scanning experiments on five different types of automotive software to verify the accuracy of the technology for scanning automotive software vulnerabilities.

2 Technical Solutions

2.1 Vulnerability Scanning System Processes

The vulnerability scanning technology proposed in this paper focuses on the automotive software program, analyzes the components of the software program by decompressing, unpacking, and feature extraction, and then matches the vulnerability information to get the vulnerabilities in the software. The vulnerability scanning process of the automotive software program is shown in Fig. 1.

First of all, add the software file to be scanned into the scanning system and determine the different component types of software files by software component analysis. If the components cannot be determined, the system will re-run the software component analysis for the files that failed in one software component analysis. In this way, a more fine-grained component analysis can be performed using the results of the previous component analysis. If the software component is not determined after N cycles of analysis,

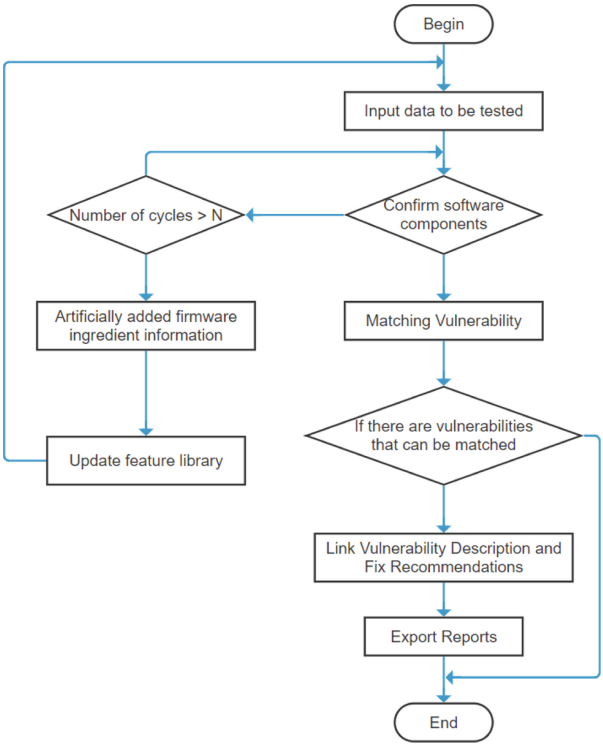

Fig. 1. ICV software vulnerability scanning process

the component database may not have the feature data of the corresponding component entered into it. N is the number of cycle analyses, which can be adjusted by the actual scanning situation. It is necessary to add component samples and feature data to the component database by analyzing component features manually. After identifying the software components, the vulnerabilities are matched by category, name, version, and other information. Then, the vulnerability scan results are output.

The vulnerability scanning technology proposed in this paper has two main contributions: (1) Construction of a software feature database. (2) software component analysis. The following describes the vulnerability scanning technology for automotive software programs by detailing the process of building a software feature database and the process of software component analysis for ICV.

2.2 Constructing the Feature Database

The most important part of the process of building a feature database is sample collection and feature extraction [5]. In this paper, the proposed software vulnerability scanning technology for ICV uses five databases, including the software database, the development platform information database, the code segment history database, the component database, and the vulnerability database. Among them, the development platform information database uses open source data because the development platform information

store the popular and common development platform compilation options and compilation features. The application development software platforms used in the ICV and IT domains are generally the same [6]. Using leading information databases will improve the accuracy of vulnerability scanning.

The software database stores the software data of each device in the automobile. The ICV generally consists of the modules shown in Fig. 2.

Through historical project accumulation and cooperation with auto enterprises and equipment manufacturers, our team has gradually built a database covering the full architecture modules of ICV. The software data now contain the full-stack software of IVI, TBOX, RSU, OBU, BMS, etc. from dozens of car brands. Under the premise of complying with laws and regulations and protecting enterprise data assets, we use the data to help the development of security for ICV.

The code segment history database stores the code segments in software files, in various languages. The code files are split into code fragments by features such as global variables, function names, ad source code file names to form a code segment history database applicable to the field of ICV [7].

The component database holds the components referenced in the ICV software, also known as third-party components. They are usually.so and.ko files under various file paths. The component database currently in use contains traditional IT system components. Our team is currently accumulating data on the automotive software and gradually adding ICV specific components. The features of the designed component database are shown in Table 2.

The Package Manager is a tool that allows users to install, delete, upgrade, configure, and manage packages on the operating system. Examples include apt-get, pip, Pacman, etc. In the field of ICV, the Package Manager of some modules is probably a private tool and the installation standard developed and used by the auto enterprise, which needs to be improved continuously [8].

The vulnerability information database stores traditional IT vulnerability information and proprietary vulnerability information of the ICV field. The system identifies the components used in the ICV software through component analysis and then matches them with the component vulnerability information in the vulnerability database to

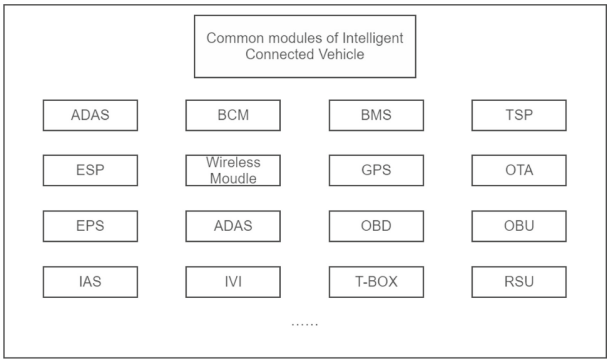

Fig. 2. Common modules of ICV

Table 2. Characteristics table of component data

No.	Features	Descriptions
1	Database name	Component names
2	Suppliers	The company that published the component
3	Programming Languages	Programming languages used in the source phase of the database
4	Package Manager	Name of the package manager that manages the component
5	Versions information	Component version information
6	Date of publication	Publication date of components
7	Licenses	License information of components
8	Architecture	The CPU platform architecture on which the component runs
9	Attached file	A detailed description of the components

determine the software vulnerability [9]. The main features in the vulnerability database are shown in table 3.

In this paper, the ICV software vulnerability scanning technology is compatible with existing vulnerabilities in the IT field. On the basis of traditional IT vulnerabilities, the software and vulnerability data of the ICV field are continuously added. The current threats in the ICV field are shown in Fig. 3.

Analogous to the IoT 'Cloud-Sea-Fog' architecture [10], this paper divides the security threats of ICV into four levels. During the construction of a database, data including software, components, patches, and vulnerabilities are added to improve the coverage and accuracy of vulnerability scanning technology.

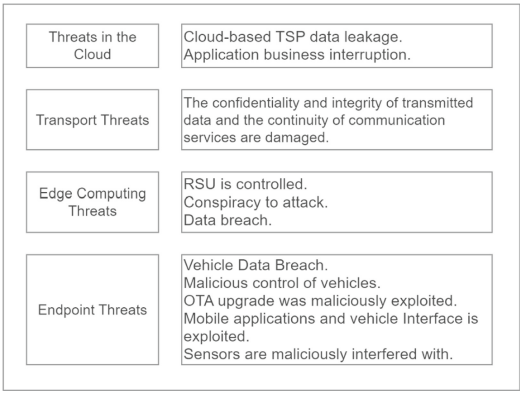

Fig. 3. Threats in the ICV field

Table 3. FEATURE table of Vulnerability dataBASE

No.	Features	Descriptions
1	Vulnerability identification	Public number of vulnerabilities, e.g. CVE
2	Description of the vulnerability	Description of the vulnerability
3	Range of components affected by the vulnerability	Components Information attributed to this vulnerability
4	Database name	Component names
5	Version	Versions of components with vulnerabilities
6	Date of publication	Publication date of components
7	Links to patches	Locations where vulnerability patches are available

2.3 Analysis of the Software Components

The system matches the software components by software component analysis using the constructed data sets such as the software database and component database [11]. The technical flow of software component analysis presented in this paper is shown in Fig. 4.

Firstly, input the ICV software file into the software component analysis system, and the system decompresses the software file. After decompression, according to the file type, the system classifies the executable files into one category and other scripts, configurations, and documents into another category. The system classifies the files with the path of lib and the files with the suffix with.so and.ko feature as dynamic link library files according to the file path and part of the suffix information. The system also filters out the program files in scripts, configurations, and documents by the keywords of the programming language and categorizes them as application source code.

Then the system decompresses the software data by recursion into three types of data: dynamic link library, binary file, and application source code. Different types of data are analyzed by different processes to obtain software component information [12]. Among them, the dynamic link library class data, which mainly relies on the development platform information database, improve the matching accuracy of dynamic link libraries through dependency locking and dependency inference. For the software component analysis of the binary file class, the system splits the file to form fragments by symbolic information such as static variables, global variables, function names, assembly instruction categories, numbers, jump relationships, and other information. Then the software components are determined by means of code fragment HASH matching, code fragment list structure BOM analysis, code fragment summary analysis, etc. For the application source code class, the system determines the software component composition by the syntax tree and lexical tree comparison, and then by the code segment HASH matching. In the process of feature matching of three data classes, we use the M-gram statistical algorithm with sliding window M, which is used to improve the accuracy of data matching.

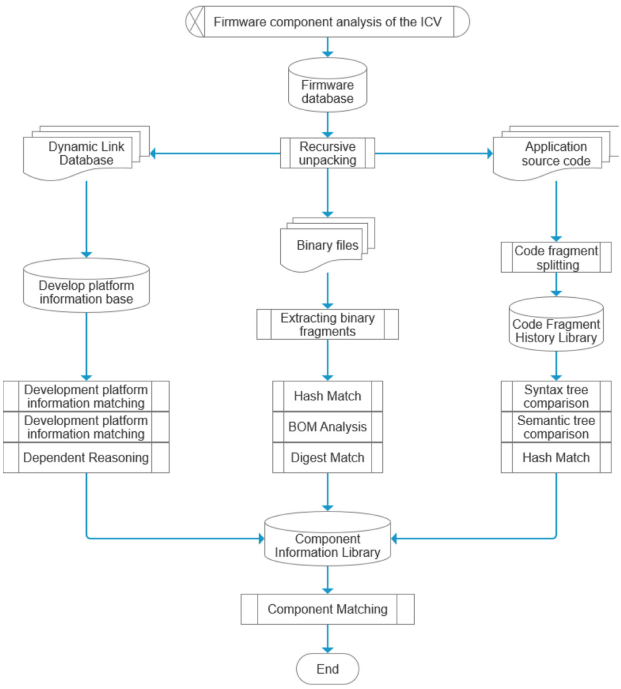

Fig. 4. Component analysis of ICV software components

Feature database construction and software component analysis are the main innovation points of this paper. This paper improves the security capability in the field of ICV and reduces the security cost of the industry through research on the vulnerability scanning technology of ICV.

2.4 Technical Characteristics

The vulnerability scanning technology for ICV studied in this paper is designed to meet the generality and adaptability of vulnerability scanning technology in the field of ICV. The technology can be adapted to scan source code of different languages, including but not limited to C/C + +, Java, Python, Shell, etc. For scanning compressed and executable file formats, including but not limited to tar, tar.gz, jar, bin, deb, dylib, apk, ko, etc. The vulnerability data sources of this system include open source vulnerability databases such as CVE. It helps the security development of the ICV industry and protects the development of ICV through comprehensive feature data, combined with high-precision component matching algorithms.

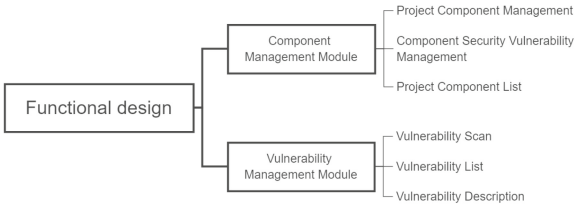

Fig. 5. Functional design of the automotive software vulnerability scanning system

3 System Design

3.1 System Architecture

This paper builds a terminal scanning system, the automotive software vulnerability scanning system, based on vulnerability scanning technology for ICV software, which realizes the scanning and analysis functions for generic vulnerabilities of large-scale automotive software. The system is designed with a 3-tier B/S architecture: the presentation layer realizes the functional interaction between the user and the system; the logic layer implements business logic through a call engine and interface to support system services; the data layer is used to store vulnerability data, component data, project data, and user data, and process the data after receiving the request [13].

3.2 Functional Design

The system proposed in this paper has two functional modules, including a component management module and a vulnerability management module. Each module contains related sub-functions, as shown in Fig. 5.

The functions of the automobile software vulnerability scanning system achieved in this paper include calling relevant functions through the command terminal mode to scan the vulnerabilities of automotive software programs. Security managers can use this system to track the running progress of vulnerability scanning projects and the results of component vulnerability analysis in real time. At the same time, according to the scan results of different versions of the same software program, security managers can also obtain the changing trend of the security problems of the software program of the automobile component in the development process [14], which can help project managers to control the overall security quality of the project.

4 Experimental Design and Analysis of Results

4.1 Experimental Environment

In this experiment, we upload the automotive software files to the cloud server for vulnerability scanning. The experimental environment of this system adopts the system environment of the server, as shown in Table 4.

The parameters are described as follows: TP (true positive), the number of components or vulnerabilities identified that exist; FP (false positive), the number of components or vulnerabilities identified but not exist; FN (false negative), the number of components or vulnerabilities not identified but exist.

Table 4. Experimental environment

Composition	Parameters
CPU	16 cores, 64 bits
RAM	64GB
SSD	1TB
Operating systems	CentOS6.7

4.2 Evaluation Indicators

In order to judge the quality of the scanning results of the system for the automobile software program, this paper uses the precision and recall rates to judge the quality of the component scanning results and vulnerability scanning results [15]. The calculation formulas are as follows.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

The parameters are described as follows: TP (true positive), the number of components or vulnerabilities identified that exist; FP (false positive), the number of components or vulnerabilities identified but not exist; FN (false negative), the number of components or vulnerabilities not identified but exist.

4.3 Experimental Results

In this paper, the experiments and results analysis are carried out from two aspects of component identification and vulnerability identification.

4.3.1 Component Identification Experiments.

In order to verify the component identification capability of this system for the software program of ICV, we uploaded the software files of each of the five experimental objects to this system for testing and analysis. The testing results were summarized as shown in Table 5.

The OBU scanning experiment is taken as an example to show the analysis results of some of the vulnerability-containing components after the software was scanned, as shown in Table 6. Taking the vulnerability component busybox in the table as an example, the component contains a full shell, a DHCP client-server, and some small utilities. These utilities are packaged into a single executable file. Version 1.24.1 may contain vulnerabilities that result in denial of service (DoS), information leakage, and remote code execution (RCE).

Table 5. Component identification results

Software types	System Versions	software sizes (MB)	Number of software files	Testing time (min)	Identified components	Vulnerable components
APP	/	136	5631	5	44	1
TBOX	Linux 3.18.44	85.3	1407	13	19	15
IVI	Linux 4.14.117	329	974	37	34	2
OBU	Linux 4.1.44	196	4525	68	56	40
RSU	Linux 4.9.11	76.6	4929	32	24	18

Table 6. OBU component scan results (partial)

Components	Versions	Depth	Latest version	Number of Vulnerabilities (Ultra-high/high/medium/low risk)	Licenses
linux:linux_kemel	4.1.44	1	2.6.24:rc1	61 / 417 / 548 / 40	GNU General Public License v3.0 or later
gnu:glibc	2.4	1	1.09	15 / 29 / 41 / 4	GNU Library General Public License v2 or later
busybox: busybox	1.24.1	1	1.33.2	3 / 13 / 2 / 0	GNU General Public License v2.0 only
isc: bind	8	1	9.17.4	2 / 6 / 7 / 0	Mozilla Public License2.0
openssl_project: openssl	1.0.20	1	3.0.0-beta1	1 / 3 / 8 / 3	OpenSSL License

Table 7. Results of component identification experiments

Software	True positive component (TP)	Component False Positives (FP)	Component False Negative (FN)	Component accuracy rate	Component Recall Rate
APP	22	4	18	84%	55%
TBOX	10	2	7	83%	59%
IVI	18	4	12	81%	60%
OBU	28	8	20	78%	58%
RSU	13	4	7	76%	65%

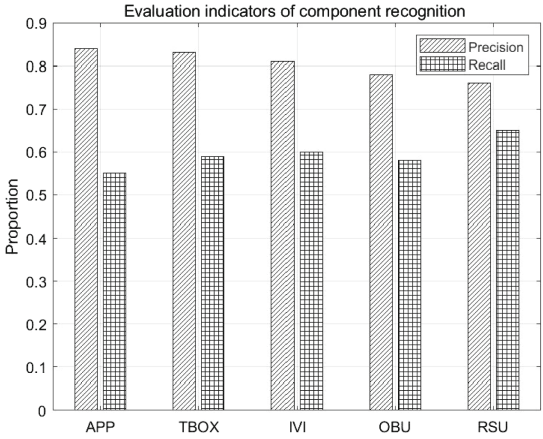

Fig. 6. Component identification accuracy and recall rates

In order to calculate the accuracy of the software program component identification of the system, we will check and verify component identification after getting the results of scanning. The statistical results are shown in Table 7, and the precision and recall rates are shown in Fig. 6.

The experimental results prove that the system can successfully identify components and component details in the software program. The accuracy of component identification is high and stable at more than 75%, and the recall rate is maintained at more than 55%.

4.3.2 Vulnerability Identification Experiments.

In order to verify the vulnerability identification ability of the system for the software program of ICV, the software files of five experimental objects were uploaded to the system for testing and analysis, and the testing results were summarized, as shown in Table 8.

Table 8. Vulnerability identification results

Software types	Total number of vulnerabilities	Ultra High Risk Vulnerabilities	High-risk vulnerabilities	Medium Risk Vulnerabilities	Low Risk Vulnerabilities
APP	6	1	2	2	1
TBOX	380	57	132	175	16
IVI	13	1	1	7	4
OBU	1676	131	671	809	65
RSU	195	33	79	70	13

Table 9. The accuracy rate of vulnerability identification

Software types	True positive for CVE	False positive for CVE	CVE false negative	CVE accuracy rate	CVE recall rate
APP	4	1	1	80%	80%
TBOX	320	38	22	89%	93%
IVI	9	2	2	82%	82%
OBU	1421	131	124	92%	92%
RSU	154	28	13	85%	92%

In order to calculate the accuracy of the system for software program vulnerability identification, the system checks and verifies the vulnerability identification after getting the scan results. The statistical results are shown in Table 9, and the precision and recall rates are shown in Fig. 7.

The experiment proves that this system can successfully identify the known vulnerabilities of components in the software programs. Since the system reports the vulnerabilities only when the components and version coexist, the accuracy of the vulnerability identification is high, and the precision and recall rates are steadily maintained at more than 80%.

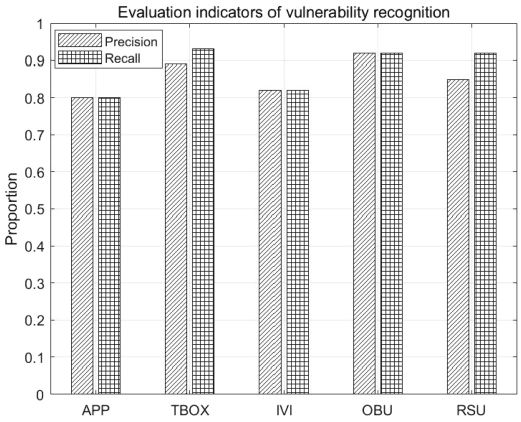

Fig. 7. Accuracy and recall rate of vulnerability identification

5 Conclusion

This paper designs and implements a vulnerability scanning technology for large-scale ICV software programs. The technology extracts the automotive software programs and unpacks them, extracts feature information to construct a feature database, analyzes the components of the software program files, matches the vulnerability information of the open source vulnerability database, and obtains the vulnerabilities existing in the software programs. Based on this technology, a vulnerability scanning terminal system for large-scale ICV software is implemented.

This paper uses this system to conduct vulnerability scanning experiments on five different types of automotive software, proving that the vulnerability scanning technology proposed in this paper can identify the components and vulnerability information in automotive software programs accurately. The accuracy of the component identification of the system reaches more than 75%, and the accuracy of the vulnerability identification of the system reaches more than 80%, which can better meet the vulnerability scanning needs of various common automotive system software programs, help security managers to control the security vulnerabilities of each module of ICV. The information security needs for ICV will be covered by all components of the automobile.

The next step of this paper is to check and analyze other types of software in automobiles to further verify the accuracy of the system's components and vulnerability identification, and to improve the system's advancedness and usability by comparing it with mainstream software scanning tools in the market.

References

1. Wufei Wu, Renfa Li, Guoqi Xie, et al. A Survey of Intrusion Detection for In-Vehicle Networks. IEEE Transactions on Intelligent Transportation Systems, 2019.
2. I. Oz, H. R. Topcuoglu, O. Tosun. A user-assisted thread-level vulnerability assessment tool[J]. Concurrency and Computation: Practice and Experience, 2019, 31(13): 23-27.

3. K. Kritikos, K. Magoutis, M. Papoutsakis, et al. A survey on vulnerability assessment tools and databases for cloud-based web applications[J]. Array, 2019, 7(5): 3-4
4. V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal and B.
5. Sikdar, A Survey on IoT Security: Application Areas, Security
6. Threats, and Solution Architectures[J]. IEEE Access, 2019, vol. 7, pp. 82721–82743.
7. B. Elisa, N. Walter, C. Senni. A Framework for Vehicle Penetration Tests[C]. Embedded Security in Cars, American, 2017
8. Kyong-Tak Cho, Kang G Shin. Error handling of in-vehicle networks makes them vulnerable. In: Proc of Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016, 1044–1055.
9. Tobias Hoppe, Stefan Kiltz, Jana Dittmann. Security threats to automotive CAN networks–practical examples and selected short-term countermeasures. In: Proc of International Conference on Computer Safety, Reliability, and Security. Springer, 2008, 235–248
10. Dan C T L, Kai Q, Wei C. Hardware Attacks and Security Education[C]. IEEE, Computer Software and Applications Conference. IEEE, 2016:253–257.
11. Zou, Bowei and Gao, Mingqiu and Cui, Xiaochuan. Research on Information Security Framework of Intelligent Connected Vehicle[C].Association for Computing Machinery,2017,3:91–95.
12. X. Shao, C. Dong and L. Dong.Research on Detection and Evaluation Technology of Cybersecurity in Intelligent and Connected Vehicle[C]. 2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM), 2019, pp. 413-416.
13. Samuel Lv, Sen Nie, Ling Liu. Car Hacking Research: Remote Attack Tesla Motors. Keen Security Lab of Tencent, 2016.
14. Karl Koscher, Alexei Czeskis, Franziska Roesner, et al. Experimental security analysis of a modern automobile. In: Proc of 2010 IEEE Symposium on Security and Privacy. IEEE, 2010, 447–462
15. Stephen Checkoway, Damon McCoy, Brian Kantor, et al. Comprehensive experimental analyses of automotive attack surfaces. In: Proc of USENIX Security Symposium, volume 4. San Francisco, 2011, 447–462