Hindawi
Security and Communication Networks
Volume 2021, Article ID 9946022, 14 pages
https://doi.org/10.1155/2021/9946022

Research Article

GNFCVulFinder: NDEF Vulnerability Discovering for NFC-Enabled Smart Mobile Devices Based on Fuzzing

Zhiqiang Wang,[1,2] **Yuheng Lin**,[1] **Zihan Zhuo**,[3] **Jieming Gu**,[3] **and Tao Yang**[4]

[1]*Beijing Electronic Science and Technology Institute, Cyberspace Security Department, Beijing 100070, China*
[2]*State Information Center, Post-Doctoral Scientific Research Workstation, Beijing 100045, China*
[3]*National Internet Emergency Center, Beijing 100029, China*
[4]*Key Lab of Information Network Security, Ministry of Public Security, Shanghai 200031, China*

Correspondence should be addressed to Zhiqiang Wang; wangzq@besti.edu.cn and Zihan Zhuo; zzh@cert.org.cn

Received 29 March 2021; Accepted 19 June 2021; Published 28 June 2021

Academic Editor: Jinguang Han

Near-field communication (NFC) is a set of communication protocols that enable two electronic devices. Its security and reliability are welcomed by mobile terminal manufactures, banks, telecom operators, and third-party payment platforms. Simultaneously, it has also drawn more and more attention from hackers and attackers, and NFC-enabled devices are facing increasing threats. To improve the security of the NFC technology, the paper studied the technology of discovering security vulnerabilities of NFC Data Exchange Format (NDEF), the most important data transmission protocol. In the paper, we proposed an algorithm, GTCT (General Test Case Construction and Test), based on fuzzing to construct test cases and test the NDEF protocol. GTCT adopts four strategies to construct test cases, manual, generation, mutation, and "reverse analysis," which can detect logic vulnerabilities that fuzzing cannot find and improve the detection rate. Based on GTCT, we designed an NDEF vulnerability discovering framework and developed a tool named "GNFCVulFinder" (General NFC Vulnerability Finder). By testing 33 NFC system services and applications on Android and Windows Phones, we found eight vulnerabilities, including DoS vulnerabilities of NFC service, logic vulnerabilities about opening Bluetooth/Wi-Fi/torch, design flaws about the black screen, and DoS of NFC applications. Finally, we give some security suggestions for the developer to enhance the security of NFC.

1. Introduction

NFC (near-field communication) is a set of ideas and technologies that enable smart phones and other devices to establish radio communication with each other by touching the devices together or bringing them in proximity to a distance of typically 10 cm or less [1–4]. It evolved from the integration of RFID and interconnection technology, which can realize two-way interactive communication between electronic devices. Devices using NFC technology (such as mobile phones) can exchange data when they are close to each other. NFC has many characteristics like short communication distance, one-to-one device connection, and hardware encryption, and it has excellent security and reliability. With the development of mobile Internet and mobile payment, more and more smartphones begin to support the NFC function [5]. NFC payment has become a promising means of mobile payment favored by banks, telecom operators, and mobile phone manufacturers.

However, NFC has also drawn more and more attention from researchers and attackers, and the security of NFC is facing increasing threats and challenges. On February 9, 2012, Zvelo Labs found a security vulnerability in Google Wallet [6], a mobile payment system based on NFC. Knowing that the PIN could only be a 4-digit numeric value, they could get the PIN easily by a brute-force attack, but the premise of this attack was that the attacked phone was rooted. The next day, the SmartPhone Champ found another vulnerability; the vulnerability could calculate the PIN in Google Wallet regardless of whether the phone is rooted or

not. The new PIN could be acquired by going into the application menu to clear the Google Wallet app's date and resetting a new PIN by opening it again [7]. In July 2012, Charlie Miller, the chief research consultant of Accuvant Labs, exposed many vulnerabilities in the NFC protocol stack at Blackhat USA [8]. For example, attackers can automatically cause the affected users to open a malicious URL and embed the malware in victims by exploiting these bugs with an NFC tag. At EUSecWest in September 2012, Corey Benninger and Max Sobell from Intrepidus Group were able to quickly and easily reset the number of journeys on the NFC contactless travel card using an NFC-enabled Android phone. This NFC payment flaw existed in the MIFARE Ultralight chips and it could be used to reset certain data on the NFC contactless card. This defect resulted in the provision of free subway tickets, which affected several cities in subway systems, such as Boston, Philadelphia, and Chicago [9]. In August 2013, the Wall of Sheep revealed that the prepared malware in advance could be installed in attacked phones that touched a crafted malicious NFC tag. This vulnerability, which can copy users' short messages in their phones, is discovered in their NFC Security Awareness Project [10]. In July 2015, Rob Miller and Jon Butler of MWR Labs found that remote attackers can attack Samsung NFC-enabled mobile phones to download any or all images on the vulnerable devices without any notifications or user interactions [11]. In the same year, a video [12] published by a group of hackers demonstrated how a thief could use a stolen Apple Watch to make payments using Apple Pay without authenticating the transactions in any way. In May 2016, Martijn Coenen found that sensitive foreground-application information can be obtained by attackers via a crafted background application, which caused NfcService in NFC-enabled phones. In August 2016, security researcher Salvador Mendoza demonstrated a flaw in Samsung Pay at Black Hat [13], in which the transaction tokens could be predicted and be used to authorize fraudulent payments. In 2017, Xinyi Chen et al. found that attackers can use the vulnerabilities of the card payment transactions to compromise the NFC communication message and then transmit the wrong payment information to the communicators [14]. In 2020, S. Akter et al. identified a potential vulnerability in existing contactless payment protocols due to the separation between the card authentication and the transaction authorization phase. They showed how an attacker could compromise the integrity of contactless payments by a malicious MITM smartcard [15].

For all the NFC-related vulnerabilities and security incidents mentioned above, the fundamental and critical things are security vulnerabilities, which are the source of network attacks and defenses. Therefore, the paper mainly studied the technology to discover NFC security vulnerabilities effectively. The main works and contributions of the paper are as follows:

(i) The test case construction combines manual, generation-based, and mutation-based strategies, which takes advantage of three strategies and overcomes their disadvantages. Also, the known NFC vulnerability data and abnormal data are adopted to construct test cases. This method solves the single data construction strategy problem and enhances the positive detection rate of vulnerabilities.

(ii) To achieve the automation of testing and the mobile operating system's independence, we adopted two manners: simulating tag with the NFC reader device and simulating "touch" by operating the NFC process. Besides, we adopted "logcat/Xapspy" and monitoring process to monitor the abnormal operations and other problems.

(iii) For the NFC Data Exchange Format (NDEF) protocol, we designed and constructed a test case database, which can be used to test all kinds of NFC-related systems and applications, saving time and costs greatly.

(iv) With the database, we tested lots of NFC system applications and third-party applications on the Android and Windows Phone platforms and found many known and unknown vulnerabilities, including "Wi-Fi on," "Bluetooth on," "black screen," "Flashlight on," and DoS vulnerabilities.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 introduces the proposed methodology. Section 4 illustrates the architecture and implementation of our system used to discover NFC vulnerabilities. Section 5 shows the experimental configurations and environments. Section 6 provides the results in detail. Section 7 evaluates the results by comparing the related works, and Section 8 draws conclusions.

2. Related Work

Currently, many researchers and scholars at home and abroad have done lots of studies on vulnerability discovery and defense security strategies of NFC. The research progress of NFC security will be reviewed as follows.

Ernst Haselsteiner and Klemens Breitfu [16] gave a comprehensive analysis of NFC security, including the various aspects of NFC security threats and related solutions to protect against these threats. This paper provides an overview of the traditional NFC security, including eavesdropping, data corruption, data modification, data insertion, man-in-the-middle attack, and secure channels for NFC, which emphasizes communication security. Collin Mulliner put forward a method to perform vulnerabilities analysis of NFC-enabled mobile phones by fuzzing applications with NFC tags [17]. They analyzed NFC subsystem and components (such as the web browser) and they found several vulnerabilities which can be abused for phishing, NFC-based worms, and DoS attacks. Their method constructs test cases and tests NFC-enabled phones manually, which does not support automatic testing, so the method is time-consuming and manpower-consuming. Gauthier et al. [18] presented an NFC offline payment application and applied it to Nokia 6313/6312. The PKI-based security protocol is used to create a secure channel. Their research is done from the viewpoint

of defense strategies. However, data encryptions will increase the time costs. Antonio et al. [19] evaluated the security capabilities of NFC-powered devices to carry out asymmetric cipher based on the public key encryption algorithm RSA. They proposed a hybrid security scheme that combines asymmetric key encryption with a shared key and asymmetric encryption algorithm. The drawbacks of their scheme are the long size of the key and the time to generate the certificate. Jia and Tong [20] adopted threat modeling methodologies to analyze the threats that may damage the assets from the entries during the mobile payment, which are depicted in the data flow diagram and illustrated by some attack scenarios. They had proposed some migration solutions to these threats, respectively. Their works remain theoretical and need to be verified in the future. Miller [8] proposed a method for fuzzing the NFC protocol stack and found many NFC vulnerabilities. Miller constructed test cases based on the tool Sulley and adopted a single strategy for constructing test cases and also used a single monitor, "logcat," which may cause a high false-negative rate. Omkar and Hegde [21] analyzed the threats faced by Google Wallet and evaluated its security measures. Their method is based on the existing mobile payment solutions, including the embedded solutions, the SIM-based solutions, and the card-based solution. This paper also analyzed and evaluated the security of NFC payment from a theoretical point of view. Norbert [22] developed a framework called "fuzzing-to-go" using a fuzzing approach for Android NFC APIs and NFC applications, in which test sets were generated by the tool Sulley. The monitor in fuzzing-to-go was also provided by Sulley. This framework's monitors and data generation strategy are relatively simple and do not output error logs in detail. Besides, the framework is developed to aim for Android OS and specially appointed API version. Thus, its portability is poor. Gummeson et al. designed a passive hardware-based patch called "Engarde" [23] at extremely low power to protect NFC interaction. Engarde can be stuck on the back of an NFC phone and intercept malicious operations and behaviors in the specific blacklisted behaviors. Their solution needs additional hardware, which increases maintenance overhead and hardware cost. Roland [24] designed two scenarios to emulate a secure element of the Android platform. The secure element emulator can be used for debugging and rapid prototyping of secure element applications. Application developers can use it to replace the secure elements for long-term testing, which reduces the development cost and complexity of SE applications.

With the rise of mobile payment, NFC security faces more and more challenges. However, there are many issues and problems in the researches on NFC security. As can be seen from the previous related research works, the research works [16, 20, 21] are purely theoretical and will await further evaluation. The research works [18, 19, 23] propose some NFC security strategies from the point of software or hardware protection. However, the disadvantages are that these strategies will increase time overhead and hardware costs. The research works [17, 22] have some disadvantages, including low automatization, a single strategy for constructing test cases, lack of monitors, and poor extensibility. Reference [24] develops two scenarios for the open platforms emulating a security element of the Android platform, which is helpful to NFC security.

As can be seen from NFC security research progress, the security of the NFC Data Exchange Format protocol has drawn increased attention from security researchers [8, 17, 22]. Security researchers have done a lot of research work and made some progress. However, there are still many problems with NDEF security that need to be solved. NDEF is a standardized data format maintained by the NFC Forum [25] which can exchange information between any compatible NFC device and another NFC device or tag. The research work in [17] constructs test cases and test targets manually, which are of low automation degree and cost much time and labor power. The research work in [8] constructs test cases with a single strategy, namely, generation-based or mutation-based, and monitors targets with a simple "logcat" that will cause a high false-negative rate. The research work in [22] adopts Sulley's process monitoring module. The monitoring effect is not good, and there are no detailed logs available to analyze exceptions. In addition, the study in [22] strongly depends on the versions of the Android operating system and APIs, causing poor extensibility.

Aiming at the problems existing in NFC security research, we proposed and designed an NFC application security system called GNFCVulFinder (General NFC Vulnerability Finder) to study the technologies of discovering vulnerabilities, whose strategies of constructing test cases adopt a generation-based manner combined with the mutation-based manner and manual test.

3. Methodology

In this section, the methodology of vulnerabilities discovery will be introduced.

3.1. Outline of the Methodology. Our methodology adopts three technologies to find vulnerabilities: fuzzing test [26–31], manual test, and "reverse analysis." The combined use of these three methods can make up for each other's shortcomings and improve efficiency and effectiveness. Fuzzing is a method of finding software vulnerabilities by providing unexpected input to the target system and monitoring the results. It is between a complete manual test and a fully automated test. It is an effective automatic test method, which can improve test efficiency and greatly reduce testing costs. However, it has several shortcomings, including low code coverage and inability to discover logic vulnerabilities. Manual test is a traditional technique for software testing used in our methodology to detect logic vulnerabilities. The complete manual testing is penetration testing. Testers simulate hackers maliciously entering the system and find loopholes. This method completely depends on the testers' ability. It can make up for fuzzing's deficiencies. In addition, "reverse analysis," not the traditional meaning of reverse engineering, is also used in our methodology. Some NDEF messages are defined by third-party developers; they do not strictly follow the NDEF

specifications, so these messages will be obscure. These proprietary protocols lead to some difficulties in constructing test cases. Therefore, we have adopted some techniques, including "write in and read back" and "sniffer," to analyze NDEF messages using binary editors, which will improve the recognition rate of test case. We call these techniques "reverse analysis." The procedure of "reverse analysis" will be introduced in Section 4.4.

The methodology is illustrated in Figure 1. We first analyze the vulnerabilities in the tested protocols and then construct test cases based on fuzzing, manual testing, and "reverse analysis." Afterward, we test targets with test cases and monitor the targets. Finally, we validate the exceptions and output test results. In our method, the key algorithm is to construct test cases, which will be introduced in the next section.

3.2. The Algorithm of Test Case Construction and Test. The construction and testing of test cases are the core points of our methodology, as shown in Figure 2. We call this algorithm GTCT (General Test Case Construction and Test). GTCT is described as follows. The input includes **G**, **M**, S_1, and S_2.

G is a generator matrix that denotes a flag whether the corresponding protocol field will be filled with the generated data or not. $\mathbf{G} = (g_{ij})n \times r$, $g_{ij} = 0$ or 1, $1 \le i \le n, 1 \le j \le r$. $g_{ij} = 1$ means generating test cases with malformed data MD_j at the field f_i, and equaling 0 means the opposite. MD_j is a column vector in **MDB** which is a database of malformed data fragments. $\mathbf{MDB} = \{MD_1, MD_2, \ldots, MD_r\}$; MD_i denotes a type of malformed data in database **MDB**, $1 \le i \le r$. f_i is a weak protocol field in a vulnerable field set **F**. $\mathbf{F} = \{f_1, f_2, \ldots, f_n\}$; f_i denotes a weak field in network protocols, $1 \le i \le n$.

M is a mutation matrix that denotes a flag whether the corresponding protocol field will be mutated randomly or not based on a data sample, $\mathbf{M} = (m_{ij})_{n \times q}$, $m_{ij} = 0$ or 1, $1 \le i \le n, 1 \le j \le q$. $m_{ij} = 1$ means mutating f_i field of S_j randomly, and equaling 0 means the opposite.

S_1 and S_2 are the column vectors in **SDB**, which are a set of sample data. $\mathbf{SDB} = \{S_1, S_2\} = \{s_1, s_2, \ldots, s_q\}$, where q is the number of samples. S_1 denotes sample data from the process of manual analysis and testing process, and S_2 denotes sample data collected from NVD, CVE, and other databases (see Algorithm 1).

Firstly, Initialize() is used to initialize mobile phones and applications. Secondly, the strategies for constructing test cases in the algorithm include three steps: manually construct test cases, generate vulnerable protocol field F_{gen}, and mutate samples' vulnerable fields F_{mut}. The function Generator (S_{RFC}, **G**, **MDB**) denotes the generation of test cases based on matrices **G**, **MDB**, and the protocols' fields S_{RFC} based on the RFC specifications. The function Mutator (**SDB**, **M**, **rand**) denotes the mutation samples of the **SDB** based on **M** with a random function. In addition, the "reverse analysis" on the protocols is used to build a sample database **SDB**. Thirdly, execute the test cases T_{mau} (test cases constructed by manual strategy), T_{gen} (test cases constructed by multiple-dimension strategy based on the generation), and T_{mut} (test cases constructed by multiple-dimension strategy based on mutation) with the test function NdefTest(). NdefTest() is a test function; this function sends some data for the mobile phone to read the data for testing. By the way, S_1 is obtained during the previous test, and S_2 is from manual analysis on some vulnerability database, such as NVD. Fourthly, execute the test cases T_{mut}. Finally, output logs and results. The details and advantages of the three strategies will be described in Section 4.3.

4. Design and Implementations of System Architecture

This section will introduce the system architecture design of our vulnerability discovery framework and its implementation process.

4.1. System Architecture. We designed a discovering vulnerability system named GNFCVulFinder for the NDEF protocol, and its architecture is shown in Figure 3. The system architecture includes test case generation, initialization, exception monitor, NFC apps test, exceptions validation, and log output. The architecture contains two entities: the mobile phone and the NFC tag. These two entities do not need to be authenticated during the test. The NFC communication process uses the NDEF protocol, which is used between the NFC mobile phone and the tag and does not require identity verification.

Testcase generation: this module is used to construct test cases about the protocol NDEF, including NFC Forum type, NFC external type, absolute URI type, and MIME type. The strategies for constructing test cases have been introduced in detail in the section titled "Test Case Construction."

Initialization: since two or more NFC applications will listen to NDEF message events and respond to them simultaneously, mobile phone users have to select an application to handle the events, which will cause a serious disturbance to automated testing. So, we need to initialize the tested mobile phone, namely, installing the tested application and uninstalling other unconcerned applications.

NFC apps test: this module is designed to test NFC services and Android and Windows Phone applications. On Android, emulating tags and "touch" operations are adopted to realize automatic tests. The former is achieved by calling APIs in the open-source library libnfc based on the NFC device ACR 122U, and the latter is achieved by controlling the NFC process with the commands "kill" and "start." On Windows Phone, we only need to emulate a tag, because Windows Phone can automatically detect the changes of data in tags and automatically read the data.

Exception monitor: this module adopts monitoring abnormal logs and monitoring NFC services and logs the test cases triggering abnormal conditions and the breakpoints.

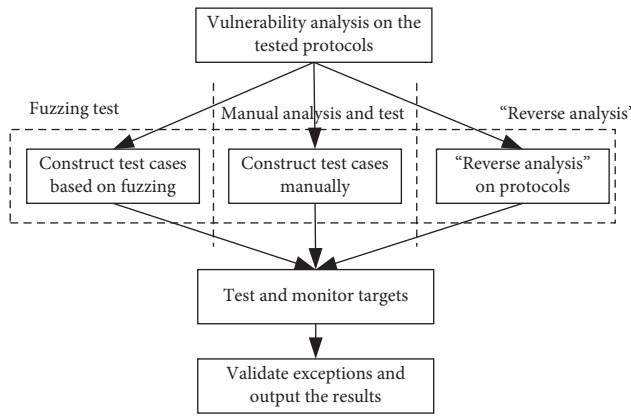

Figure 1: The illustration of our methodology.

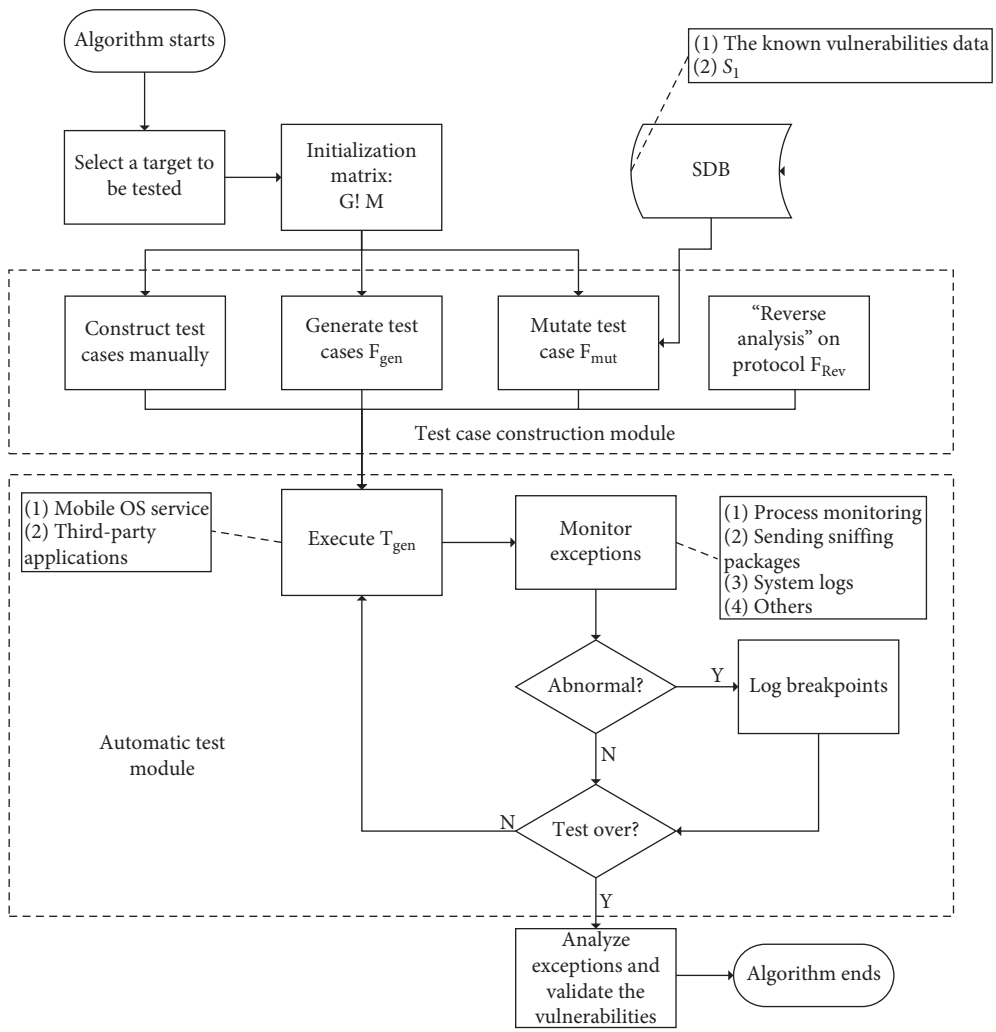

Figure 2: The flowchart of Generalized Test Case Construction and Test.

Exception validation: this module validates the exceptions found in the process of a test by resending the test cases and monitoring the targets. The exceptions, if present, will be analyzed manually and verified whether they could be exploited.

Log output: this module will output the results of the test. The system architecture is independent of any mobile platform because the emulated NFC tag does not depend on the mobile phone. Therefore, the architecture is suitable for testing on

Input: **G**, **M**, S_1, S_2
Output: result
Begin
(1) Initialize();
(2) S_1 = Ndef Test(T_{mau});
(3) T_{gen} = Generator(S_{RFC}, **G**, **MDB**);
(4) S_2 = Ndef Test(T_{gen});
(5) **SDB** += (S_1, S_2);
(6) T_{mut} = Mutator(**SDB**, **M**, **rand**);
(7) Ndeftest(T_{mut})
(8) Output(log);
end

Algorithm 1: The algorithm of GTCT.

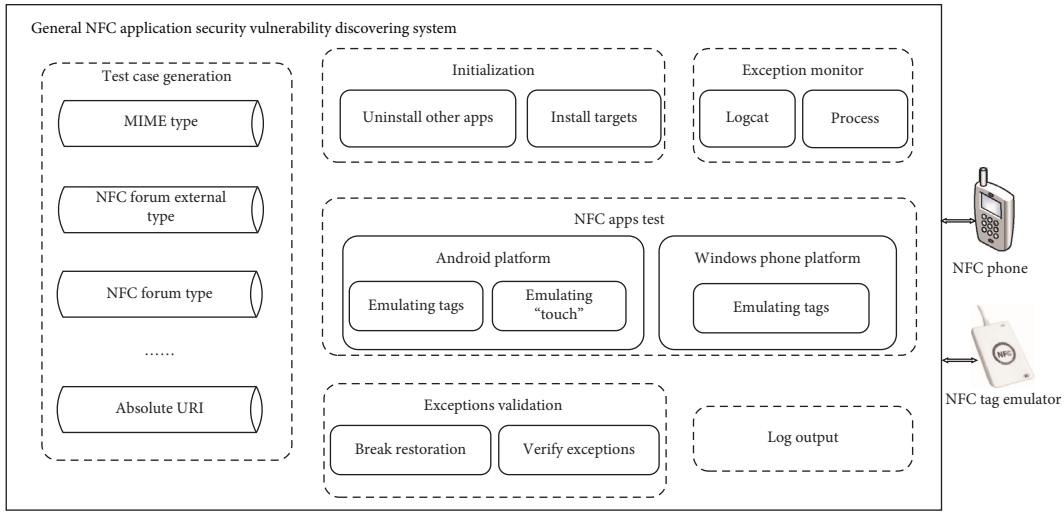

Figure 3: The architecture of GNFCVulFinder.

Android, Windows Phone, and other mobile OS platforms.

The following sections will describe the implementation of some important modules in detail, including "test case construction" and "test and monitor."

4.2. Test Case Construction. NDEF is the NFC Data Exchange Format, which is the data format of the NFC tag agreed by the NFC organization. It is a lightweight and compact binary format with various data types defined by URL, vCard, and NFC. This paper will not detail the detailed definitions of the protocol NDEF, which can be acquired in the NFC Forum. The following will introduce the strategies of test case construction about fuzzing and manual test.

4.3. Strategies

4.3.1. Manual Strategy. Manual strategy constructs test cases by analyzing the NFC forum's NDEF specifications and NDEF vulnerabilities in the popular vulnerability databases. The strategy can improve the validity of test cases, reverify known vulnerabilities, and find some logical vulnerabilities that fuzzing cannot detect. By searching the well-known database NVD [32], we analyzed the following several typical vulnerabilities:

CVE-2008-5825: the SmartPoster implementation on the NFC mobile phone Nokia 6131 does not properly display the URI records when the title record contains a certain combination of space("0 × 20"), CR("0x0D"), and (dot, "0x2E") characters, which allows remote attackers to trick a user into loading an arbitrary URI via a crafted NDEF tag. We can test NFC applications with an NFC tag demonstrated by (a) an http: URI for a malicious website, (b) a tel: URI for a premium-rate telephone number, and (c) an sms: URI that triggers the purchase of a ringtone.

CVE-2008-5826: the NFC phone Nokia 6131 allows attackers to cause a denial of service (device crash) via a large value in the payload length. So, we should test NDEF applications with a large VALUE in the payload length field of an NDEF record or a certain length for a tel or sms: NDEF URI.

CVE-2008-5827: Nokia 6131 automatically installs the software upon completing the download of a JAR file,

making it easier for remote attackers to execute arbitrary code via a crafted URI record in an NDEF tag. We should test it by making the NFC service complete the download of an APK file.

CVE-2015-8041: multi-integer overflows in the NDEF record parser in hostapd before 2.5 and wpa_supplicant before 2.5 allow remote attackers to cause a denial of service (process crash or infinite loop) via a large payload length field value in a WPS or P2P NFC NDEF record, which triggers an out-of-bounds read. So, we can test NFC applications with large payload length values.

CVE-2017-7461: it is a directory traversal vulnerability in the web-based management site on the Intellinet NFC-30ir IP Camera and allows remote attackers to read arbitrary files via a request to a vendor-supplied CGI script that is used to read an HTML text file, but that does not do any URI/path sanitization. So, we can test targets with directory traversal strings.

4.3.2. Multiple-Dimension Strategy Based on the Generation. This strategy first acquires the NFC Forum's specifications and analyzes the format of NDEF messages. The test cases are constructed by adding one or more fields of NDEF with malformed data and other fields filled with normal data. As shown in Table 1, we designed a malformed database to test fields, including header, length, type, and payload. The database consists of integers, formatted strings, directory traversal data, separators, and other nonalphanumeric characters. An example is illustrated in Figure 4. This message is constructed by generating data for four fields, and other fields are filled with normal data.

4.3.3. Multiple-Dimension Strategy Based on Mutation. This strategy does not need to analyze NDEF specifications; it only needs to construct test cases by mutating one or more bits of selected samples. Since the previous vulnerability data is likely to trigger an old or new bug [31], the known vulnerability data and test cases that trigger exceptions during the test process are used as mutation samples. We can construct test cases by mutating one or more bits of the "local name field" of the Bluetooth pairing protocol.

4.4. Reverse Analysis. During the test process, NDEF messages defined by developers in different NFC applications did not follow the NDEF specifications in the NFC Forum. For example, the messages constructed by an NFC application called "YunNFC" cannot be identified by MIUI OS, and MIUI just opens the site of "YunNFC." As a result, lots of test cases may be denied by tested targets. To solve this problem, "reverse analysis" is adopted to analyze the format of NFC application messages. "Reverse analysis" helps to improve the validity of test cases, and it consists of five steps.

Step 1: construct NDEF message with NFC applications, such as Detail!, TagInfo, and TagWriter. For example, we can construct a Bluetooth pairing message just by inputting the information about Bluetooth.

Step 2: write the constructed messages into an NFC tag.

Step 3: "touch" an NFC mobile phone with the tag. The binary file about the message will be stored on the phone.

Step 4: export the binary file by some tools about phone managers or the command "adb pull."

Step 5: analyze the binary file based on the protocol format of NDEF and the information entered in step 1, with some binary editor such as 010editor.

4.5. Test and Monitor. The core issues of automatic tests include simulation of NFC Tags and the "touch" operations. We use NFC reader devices (such as ACS ACR122u and PN512) to simulate NFC Tags. The two devices mentioned above are based on the open-source library libnfc. It is independent of the platforms of mobile terminals to simulate a tag by NFC devices. Therefore, our system is universal to test NFC services and applications. The "touch" is the operation that an NFC device touches an NFC tag from far to near. On the Android platform, "touch" is achieved by the command "kill -s SIGSTOP PID" and the command "kill -s SIGCONT PID" to control the process of the NFC service. The NDEF messages' changes in NFC tags can be detected by the Windows Phone platform, so there is no need to simulate "touch."

Target monitoring is also an important part of an automatic test. For Android, we adopt process monitoring and the typical log command "logcat," which has many types of options and operations and can output detailed logs. For Windows phones, we use a real-time monitor called "Xapspy" to detect exceptions. Xapspy is a dynamic monitor, which is realized by the technologies of program instrumentation and API Hook.

5. Experimental Configurations and Environments

5.1. Experimental Configurations. The configurations of our experiments can be divided into hardware configuration and software configuration. The former covers hardware devices needed by GNFCVulFinder and their configuration information. The latter covers NFC system services, NFC applications, and their configuration parameters.

5.1.1. Hardware Configuration. GNFCVulFinder runs in a virtual machine with the OS version "Linux Ubuntu 2.6.32-21generic"; NFC phones include Samsung GT-I9300, Mi 5S Plus, OnePlus 3T, and Lumia 920. In addition, NFC devices include ACR 122U and Proxmark3. The detailed information is shown in Table 2.

Table 1: The database about malformed data.

Name	Malformed data (hex)
Type	0x02, 0x04, 0x06, 0x05, 0x30, 0x40, 0x41, 0x42, 0x43, 0x44, 0xRR (0xRR is random hex data)
Length	0x0, 0xFF, 0xFFFF, 0xRRRR (0xR-RRR is random hex data)

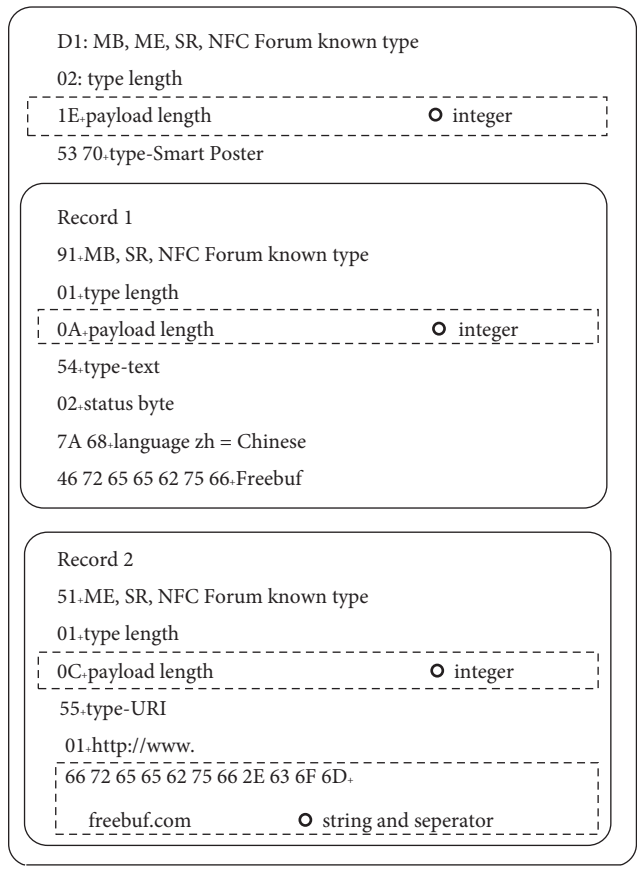

Figure 4: A generation example about SmartPoster message.

Table 2: Our experiments' hardware configurations.

Hardware name	Configuration
PC	Intel(R) Core(TM) i7, single CPU 2.93 GHz, memory 2 GB, Linux Ubuntu 2.6.32-21generic
Samsung GT-I9300	Samsung Exynos 4412, 4-Core Processor, CPU 1.4 GHz, Mali-400 MP4 GPU, 1 GB RAM, 16 GB ROM
Mi 5S Plus	MIUI 8.2, Android 6.0.1 MXB48T, 4-Core Processor, 6 GB RAM, 128 GB ROM
OnePlus 3T	H2 OS 2.5, Android 6.0, RAM 6 GB, 4-Core Processor, 64 GB ROM
Lumia 920	Windows Phone 8, 2-Core Processor, CPU 1.5 GHz, 1 GB RAM, 32 GB ROM
ACS ACR122U	Frequency 13.56 MHz, compatible with ISO 14443 Types A & B, MIFARE, Felica, and so on
Proxmark3	Frequency 125 KHz, 13.56 MHz, compatible with high-frequency and low-frequency card

5.1.2. Software Configurations. The software configurations cover the targets and their information, including NFC system services and third-party applications on Android platform and Windows Phone. The iOS platform does not support reading and writing NDEF messages, so we do not test applications on the iOS platform. By searching the keywords "NFC," "NDEF," and "Tag" on Android and Windows Phone's official and third-party application markets, we currently have collected 42 NFC applications and NFC services as shown in Table 3. Among these targets, the applications (34–42) are NFC writer applications or need human-computer interaction. These applications have not been tested because they can break the automatic test.

5.2. Experimental Environments. The experimental environments are illustrated in Figure 5; (1) in Figure 5 shows the environment of "reverse analysis" and the details are given in

Table 3: The targets.

No.	Package name	Name and version	Developer
1	com.android.nfc	Android	Google
2	com.android.nfc	MIUI	MI Corp.
3	com.android.nfc	MK	MoKee
4	com.android.nfc	H2 OS	OnePlus
5	com.android.nfc	Android (Touch Wiz)	Samsung
6	Windows.Networking.Proximity	Windows Phone	Microsoft/Nokia
7	com.nxp.taginfolite	TagInfo 2.00	NXP
8	com.nxp.nfc.tagwriter	TagWriter 2.3	NXP
9	com.fmsh.appsys.nfctag	NFC Tag helper 1.05	Fudan Microelectronics Group
10	hyundai.uni.nfc	NFC Detail 0.5	U&I Research Lab
11	com.sharemore.nfc.transport	ShareMore 3.0	Share More Studio
12	com.antares.nfc	NFC Developer 2.1.1	Thomas Rorvik Skjolberg
13	com.wakdev.wdnfc	NFC Tools 1.7	Wakdev
14	org.ndeftools.boilerplate	NDEF Tool for Android 1.2	Thomas Rorvik Skjolberg
15	com.boco.nfc.activity	elechong 1.0.09	Beijing Yiyang Huizhi
16	com.kddi.nfc.tag_reader	NFC TagReader 2.2.2	KDDI
17	com.microsoft.tag.app.reader	Microsoft tag 5.6.4.90.7866	Microsoft
18	com.mls.handover.wifi	Wifi Handover 1.0.3	STYL Solutions Pte., Ltd
19	at.mroland.android.apps.nfctaginfo	NFC TagInfo 1.11	NFC Research Lab
20	com.touf.mfclassic	MFClassic 2.0	TouF
21	com.widgapp.NFC_ReTAG_FREE	NFC ReTag Free 2.8.2	WidgApp Mobile Solutions
22	com.yunnfc.nfcaction	YunNFC 1.5	Yunfei
23	de.syss.MifareClassicTool	MIFARE Classic Tool 1.3.3	IKARUS Projects
24	se.anyro.nfc_reader	NFC Reader 0.13	Adam Nyback
25	com.nellon.mifareid	MIFARE ID reader 1.1.0	Nellon
26	N/A (Windows Phone)	NFC Commander 2.2.3.5	JasonP
27	N/A (Windows Phone)	NFC Launchit 2.6.0.0	VinApp
28	N/A (Windows Phone)	NFC Writer Reader 1.2.0.0	PCCON
29	N/A (Windows Phone)	NFC Reader 1.2.0.0	DysonChan
30	N/A (Windows Phone)	NFC Tag Writer 2.0.5.9	Mike Francis
31	N/A (Windows Phone)	NFCsms 1.2.0.0	Mopius
32	N/A (Windows Phone)	Nokia NFC Writer 1.0.0.59	Microsoft Mobile
33	jp.co.menox.caffeetime.toiki.nfc.mifare2ndef	MIFARE NDEF TOOL 1.1	N/A
34	com.sony.easyconnect	NFC Easy Connect1.0.02	Sony
35	com.hkphka.mifaredoctor	MIFARE Doctor 2.7	Kings Studio
36	com.tagstand.writer	Tagstand Writer 2.0.14	Egomotion
37	tw.com.method.rfidtool	RFID Tool 0.3	Method
38	com.nfcquickactions.appfree	NFC Actions 2.0.3	Flomio
39	com.anytag.android	AnyTag NFC Launcher 1.3.0	XtraSEC
40	com.skjolberg.nfc.clone	NFC Tag Cloner 1.2.4	Thomas Rorvik Skjolberg
41	nz.intelx.send.lite	Send! 2.0.2	Billy Lam
42	com.Samsung.android.app.gearnfcwriter	NFC Tagwriter 1.4.131022	Samsung

Section 4.4. By the way, Proxmark3 has been used to sniffer the messages between NFC phones and NFC tags. (2) in Figure 5 illustrates our test process. Firstly, simulate NFC tags with ACS ACR122u based on libnfc. Secondly, "touch" tags with NFC phone. Finally, monitor the exceptions and output the results.

6. Results

We test the targets in Figure 3 with GNFCVulFinder and find eight new vulnerabilities of mobile OS and applications, as shown in Table 4. We divide the vulnerabilities into five types according to the ways they can be exploited to describe how to use these vulnerabilities to attack. The specific descriptions are as follows:

(1) NFC service DoS: this vulnerability is triggered by a Bluetooth pairing message with its field "local name length" filled by 0b0000 0000 or 0b1xxx xxxx (x denotes 0 or 1). NFC service will crash, report the exception "java.lang.NegativeArraySizeException," and stop the service. The results are shown in Figure 6.

(2) Opening torch: this vulnerability is caused by starting a package "com.android.systemui," which leads the torch in an NFC phone to be opened automatically. "com.android.systemui" is a core system application in MIUI, which is used to manage battery, configure MIUI OS, and so on. After decompiling the torch apk, we found that the activity of the torch is the only activity kept in the root directory, which will cause the torch to be activated.

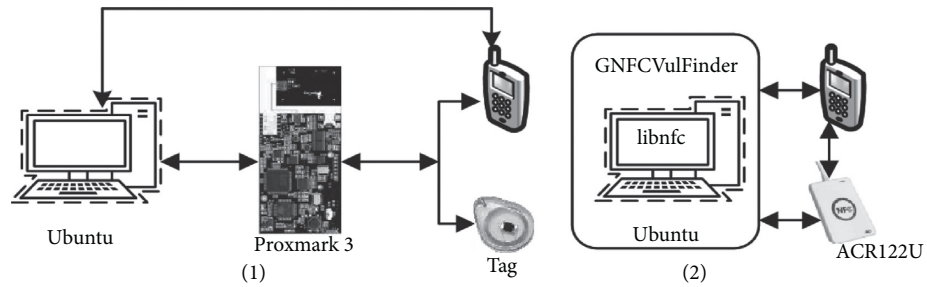

Figure 5: Experimental environments.

(3) Opening Bluetooth and opening Wi-Fi (logic vulnerability): the two vulnerabilities are both logic vulnerabilities. Suppose that Bluetooth and Wi-Fi are closed by default. Bluetooth or Wi-Fi will be closed normally if the pairing or connection fails. However, they are opened, respectively, after paring or connection failure. The result of Bluetooth opening is shown in Figure 7. No matter which choice is to be selected, Bluetooth will be opened. The result of Wi-Fi opening is shown in Figure 8. When a phone connects to a Wi-Fi SSID "UCAS" successfully or not, the Wi-Fi of the phone will be opened. For the type of logic vulnerability, we give an attack scene as an example; attackers firstly open the Wi-Fi of an NFC phone and then connect NFC phones by NFC or Wi-Fi to get sensitive information combined with other vulnerabilities. Since the messages are normal NDEF packets, we do not need to describe them specially.

(4) Denial of Service (TagInfo DoS, NFC Tag helper DoS, and NFC Detail DoS): we find lots of DoS vulnerabilities when testing TagInfo, Detail!, and NFC Tag helper. The messages that can cause a Denial of Service of these applications, including smart posters, URLs, and Wi-Fi connections. For the reason that these applications do not handle NDEF messages properly, they will crash when reading these types of messages, as shown in Figure 9. Two types of exceptions are captured including "java.lang.IllegalStateExcepton," "java.lang.IndexOutOfBoundsException," "java.lang.ArrayIndexOutOfBoundsException," and "java.lang.NullPointerException." These exceptions cannot be exploited by analyzing the logs, and they just lead to a Denial of Service attack.

(5) Black screen: this vulnerability existing in the NFC Tag helper is a type of design defect. One of the functions of this application is to set the brightness of the Android phone screen. The value scale is between 5 and 100. However, we can break the limit and set the brightness to 0 by writing 0x00 to the NFC tag. The screen of the Android NFC phone will turn black until rebooting the phone or rewriting a new brightness value.

The message that triggers the black screen vulnerability is defined as follows: D1 01 12 54 02 65 6E 10 02 0C 01 04 6E 69 70 63 02 01 1E 03 01 00.

The above test results belong to the Android platform, and no exceptions were found on Windows Phone. The reasons for the results are summarized as follows:

(i) Many applications need user interaction; namely, when an NFC phone "touches" an NFC tag, the operating system Windows Phone leaves the decision to the users, which interrupts the user interaction testing process. If the user does not make any decision, the messages will be denied by Windows Phone, which results in the low efficiency of the test.

(ii) The number of NFC applications is small, and most applications just print the binary data of NDEF messages. The operations embedded in the messages have no chance to be executed.

7. Evaluations and Measures

7.1. Evaluations. To illustrate our work's value and efficiency, we compared it with some related research results in five parts: test case construction, monitoring, portability, automation, and vulnerabilities. The results shown in Table 5 are analyzed as follows:

(i) Test case construction: Mulliner [17] constructed test cases manually, the efficiency of which is low and has a lot to do with the testers' experience. Reference [8] and Wiedermann [22] adopt a single strategy: generation strategy (G, for short) (Sulley) or the mutation strategy (M, for short). The shortcomings of single generation are randomness and blindness, resulting in low detection rate. GNFCVulFinder adopts manual generation and mutation to construct malformed data and proposes a method called "reverse analysis" as an auxiliary method for analyzing messages. Our strategy is based on the known vulnerability knowledge of NDEF, which can help find bugs quickly; a manual strategy can detect logical vulnerabilities that fuzzing cannot find. In addition, a multidimensional strategy can detect multidimensional vulnerabilities, which can increase the detection rate.

(ii) Monitoring: Mulliner [17] monitors exceptions manually; the efficiency is low and the false-negative rate is high. Miller [8] uses logcat to monitor exceptions, and Wiedermann uses Sulley; a single monitor may omit some exceptions.

Table 4: Experimental results.

No.	Name	Type	Impact
1	NFC service DoS	Denial of Service	Android 4.1.1, 4.1.2, 4.4
2	Opening torch	Design defect	MIUI-3.6.21, 4.1.17, 4.5.30
3	Opening Bluetooth	Logic vulnerability	MIUI-3.6.21, 4.1.17, 4.5.30
4	Opening Wi-Fi	Logic vulnerability	MIUI-3.6.21, 4.1.17, 4.5.30
5	TagInfo DoS	Denial of Service	TagInfo 2.00
6	NFC Detail DoS	Denial of Service	NFC Detail 0.5
7	Black screen	Design defect	NFC Tag helper1.05
8	NFC Tag helper DoS	Denial of Service	NFC Tag helper1.05

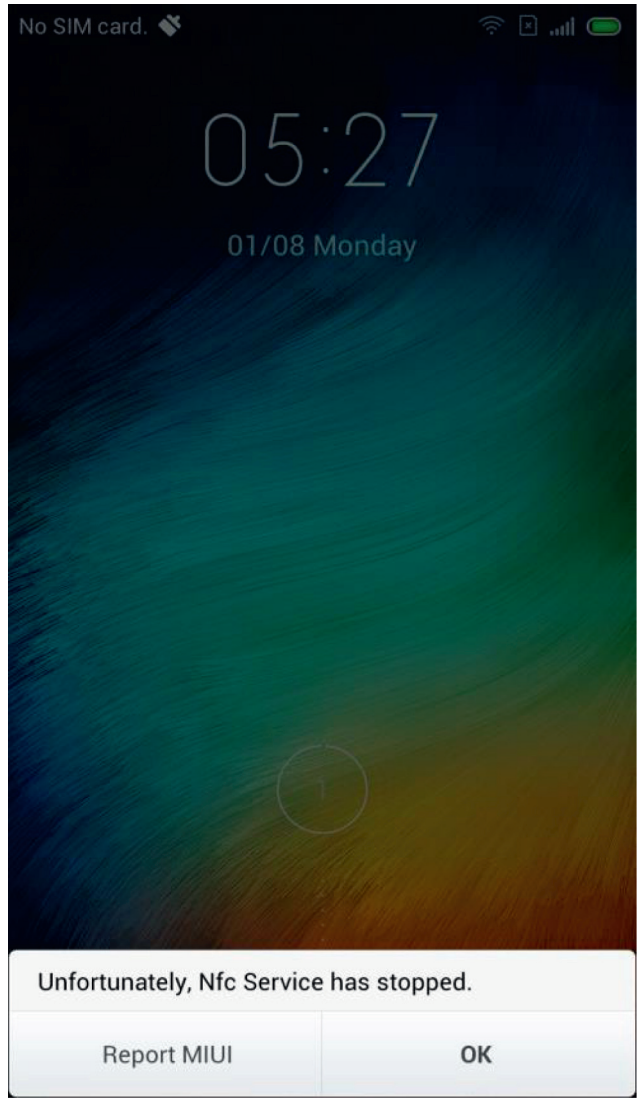

Figure 6: NFC service crash.

GNFCVulFinder monitors the targets by logcat, monitoring the NFC process, which increases the detection rate.

(iii) Portability: the portability of GNFCVulFinder and Mulliner's work is good; others are dependent on Android.

(iv) Automation (auto): except for Mulliner's work, the automation level of others is high.

(v) Vulnerability: GNFCVulFinder finds eight vulnerabilities and covers all others' vulnerabilities, including some new vulnerabilities such as opening Wi-Fi/Bluetooth and black screen.

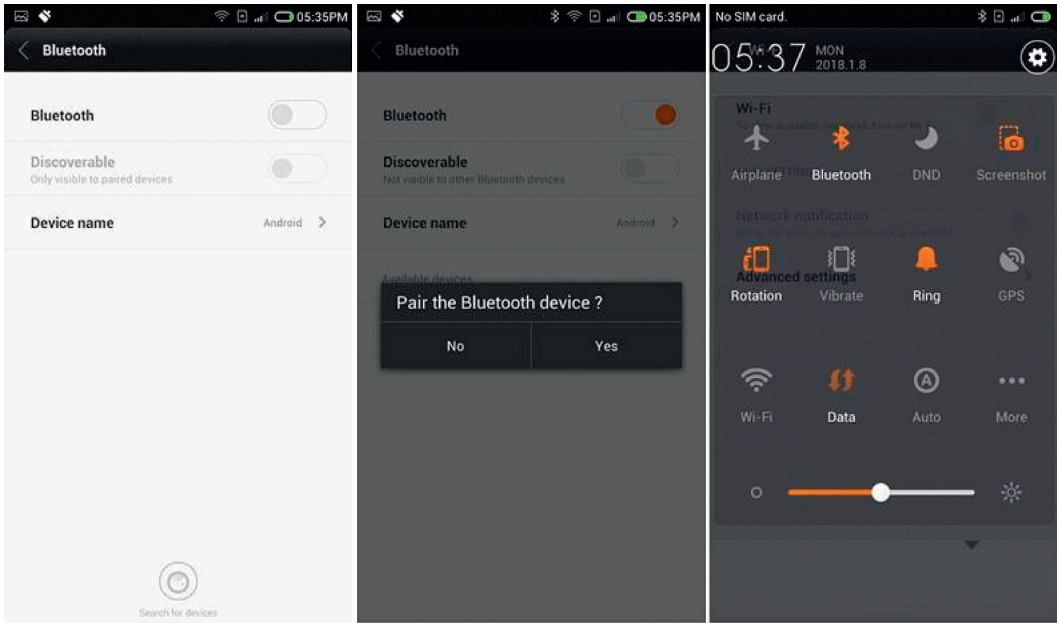

Figure 7: The Bluetooth is opened.

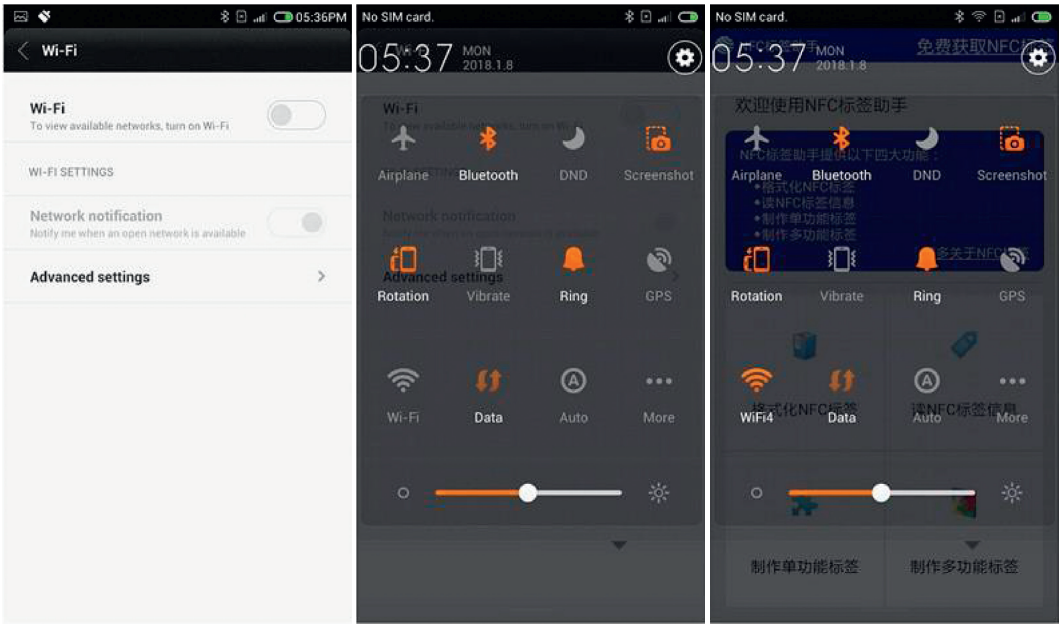

Figure 8: The Wi-Fi is opened.

7.2. Measures. To solve the above-mentioned problems and vulnerabilities, we propose the following specific measures and suggestions:

(1) Detect fields about length in all kinds of messages, and ensure that the value of 0 and the negative value can be resolved correctly.

```
E/AndroidRuntime (20745): FATAL EXCEPTION: main
E/AndroidRuntime (20745): java. lang. NullPointerException
E/AndroidRuntime (20745):        at com. fmsh. appsys. nfctag. nfc. base. 1. a (Unknown Source)
E/AndroidRuntime (20745):        at com. fmsh. appsys. nfctag. nfc. base. e. a (Unknown Source)
E/AndroidRuntime (20745):        at com. fmsh. appsys. nfctag. nfc. EntranceActivity. b (Unknown Source)
E/AndroidRuntime (20745):        at com. fmsh. appsys. nfctag. nfc. EntranceActivity. onNewIntent (Unknown Source)
E/AndroidRuntime (32443): FATAL EXCEPTION: main
E/AndroidRuntime (32443): java. lang. IllegalStateException: Couldn't read row 2, col 0 from CursorWindow.
Make sure the Cursor is initialized correctly before accessing data from it.
E/AndroidRuntime (32443):        at android. database. CursorWindow. nativeGetLong (Native Method)
E/AndroidRuntime (32443):        at android. database. CursorWindow. getLong (CursorWindow. java: 511)
E/AndroidRuntime (32443):        at android. database. AbstractWindowedCursor. getLong (AbstractWindowedCursor. java: 75)
E/AndroidRuntime (32443):        at android. database. AbstractCursor. moveToPosition (AbstractCursor. java: 219)
E/AndroidRuntime (32443):        at android. database. CursorWrapper. moveToPosition (CursorWrapper. java: 162)
E/AndroidRuntime (21432): FATAL EXCEPTION: main
E/AndroidRuntime (21432): java. lang. RuntimeException: Unable to start activity ComponentInfo
(hyundai. uni. nfc/hyundai. uni. nfc.ReadTagMain_Activity): java. lang. IndexOutOfBoundsException: Invalid index 0, size is 0
E/AndroidRuntime (21432):        at android. app. ActivityThread. performLaunchActivity (ActivityThread. java: 2100)
E/AndroidRuntime (21432):        at android. app. ActivityThread. handleLaunchActivity (ActivityThread. java: 2125)
E/AndroidRuntime (21432):        at android. app. ActivityThread. access$600 (ActivityThread. java: 140)
E/AndroidRuntime (21432):        at android. app. ActivityThread$H. handleMessage (ActivityThread. java: 1227)
E/AndroidRuntime (21432):        at android. os. Handler. dispatchMessage (Handler. java: 99)
E/AndroidRuntime (29559): FATAL EXCEPTION: main
E/AndroidRuntime (29559): java. lang. ArrayIndexOutOfBoundsException: lenght = 69; index = 69
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. base. a. a (Unknown Source)
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. wifi. e. b (Unknown Source)
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. wifi. f. a (Unknown Source)
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. base. 1. a (Unknown Source)
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. base. e. a (Unknown Source)
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. EntranceActivity. b (Unknown Source)
E/AndroidRuntime (29559):        at com. fmsh. appsys. nfctag. nfc. EntranceActivity. onNewIntent (Unknown Source)
```

Figure 9: The DoS of NFC Detail.

Table 5: Comparison results.

Research work	Construction	Monitoring	Portability	Automation	Vulnerability
Mulliner [17]	Manual	Manual	n/a	n/a	2
Miller [8]	G or M	Logcat	Good	High	2
Wiedermann [22]	G	Sulley	Bad	High	2
GNFCVulFinder	Manual, G, M, and R	Logcat, process monitor	Good	High	8

(2) Modify the logic of Bluetooth pairing and Wi-Fi connection, and restore their original condition when operations fail.

(3) When developing NFC applications, follow NDEF specifications strictly.

(4) Add some means to filter special characters, such as directory traversal characters, to prevent these characters from triggering security vulnerabilities.

8. Conclusions

This paper proposed an algorithm named GTCT to construct test cases and test the protocol NDEF based on fuzzing. The GTCT adopts four strategies to generate test cases: manual, generation, mutation, and "reverse analysis." The manual strategy is helpful to find logic vulnerabilities that fuzzing cannot detect. In addition, the known vulnerability data is also used to construct test cases, which can improve test efficiency. Based on the algorithm, we designed an NFC vulnerability discovery framework and developed a tool called GNFCVulFinder. By testing lots of NFC system services and applications, we find 8 NDEF vulnerabilities, which can cover all previous vulnerabilities and prove the effectiveness of GNFCVulFinder.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Zhiqiang Wang and Yuheng Lin designed the system architecture and conceived the experiments; Zihan Zhuo and Jieming Gu conducted the experiments; and Tao Yang analyzed the results. All of the authors reviewed the manuscript.

Acknowledgments

This research was financially supported by China Postdoctoral Science Foundation funded project (2019M650606), National Key R&D Program of China (2017YFC1201204), First-Class Discipline Construction Project of Beijing Electronic Science and Technology Institute (3201012), and the Fundamental Research Funds for the Central Universities (328201909).

References

[1] V. Coskun, B. Ozdenizci, and K. Ok, "A survey on near field communication (NFC) technology," *Wireless Personal Communications*, vol. 71, no. 3, pp. 2259–2294, 2013.

[2] K. Markantonakis and K. Mayes, *Secure Smart Embedded Devices, Platforms and Applications*, Springer, New York, NY, USA, 2013.

[3] R. Want, "Near field communication," *IEEE Pervasive Computing*, vol. 10, no. 3, pp. 4–7, 2011.

[4] K. Curran, A. Millar, and C. Mc Garvey, "Near field communication," *International Journal of Electrical and Computer Engineering*, vol. 2, p. 371, 2012.

[5] IHS Technology, "NFC-enabled cellphone shipments to soar fourfold in next five years," April 2017, https://technology.ihs.com/490062/nfc-enabled-cellphone-shipments-to-soar-fourfold-in-next-five-years.

[6] J. Rubin, *Google Wallet Security: Pin Exposure Vulnerability*, ZveloBLOG, Greenwood Village, CO, USA, 2012.

[7] T. Huynh, "Second major security flaw found in Google Wallet," December 2017, http://thesmartphonechamp.com/second-major-securityflaw-found-in-google-wallet-rooted-or-not-no-one-is-safe-video/.

[8] C. Miller, "Exploring the NFC attack surface," in *Proceedings of the Blackhat*, Las Vegas, NV, USA, August 2012.

[9] C. Benninger, M. Sobell, "NFC for free rides and rooms (on your phone)," in *Proceedings of EUSecWest Conference*, Amsterdam, Netherland, September 2012.

[10] WallofSheep, "NFC security awareness project," December 2017, http://www.wallofsheep.com/pages/nfc-security-awareness-project.

[11] Z. D. Initiative, "Samsung SBeam image remote information disclosure vulnerability," December 2017, http://www.zerodayinitiative.com/advisories/ZDI-15-257.

[12] GadgetHacks, "Apple watch exploit," December 2017, https://www.youtube.com/watch?v=2blTo-Ej6mo&feature=youtu.be.

[13] S. Mendoza, "Samsung Pay: tokenized numbers, flaws and issues," in *Proceedings of the Blackhat*, Las Vegas, NV, USA, July-August 2016.

[14] C. Xinyi, C. Kyung and K. Choi, "A secure and efficient key authentication using bilinear pairing for NFC mobile payment service," *Wireless Personal Communications*, vol. 97, 2017.

[15] S. Akter and S. Chellappan, "Man-in-the-middle attack on contactless payment over NFC communications: design, implementation, experiments and detection," *IEEE Transactions on Dependable and Secure Computing*, p. 1, 2020.

[16] E. Haselsteiner and K. Breitfuß, "Security in near field communication (NFC)," in *Proceedings of the Workshop on RFID Security*, Gratkorn, Austria, January 2006.

[17] C. Mulliner, "Vulnerability analysis and attacks on NFC-enabled mobile phones," in *Proceeding of the 2009 International Conference on Availability, Reliability and Security*, Fukuoka, Japan, March 2009.

[18] V. Gauthier, K. Damme and Wouters, "Practical experiences with NFC security on mobile phones," in *Proceedings of 2009 International Workshop on RFID Security and Privacy Issue*, Leuven, Belgium, January 2009.

[19] J. Antonio, Z. Migual, F. A. Alberto, and F. G. A. Skarmeta, "Evaluation of the security capabilities on NFC-powered devices," in *Proceedings of European Workshop on Smart Objects: Systems, Technologies and Applications*, Ciudad, Spain, June 2010.

[20] F. Jia and X. Tong, "Threat modeling for mobile payments using NFC phones," *Journal of Tsinghua University & Technoogy*, vol. 52, pp. 1460–1464, 2012.

[21] O. Ghag and S. Hegde, "A comprehensive study of google wallet as an NFC application," *International Journal of Computing and Applications*, vol. 58, 2012.

[22] W. Norbert, *Fuzzing-to-go: A Test Framework for Android Devices*, Technische University München, Munich, Germany, 2012.

[23] J. Gummeson, D. Ganesan, B. Priyantha, D. Thrasher, and P. Zhang, "Engarde: protecting the mobile phone from malicious NFC interactions," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 445–458, ACM, Taipei, Taiwan, June-2013.

[24] M. Roland, "Debugging and rapid prototyping of NFC secure element applications," in *Proceeding of the International Conference on Mobile Computing, Applications, and Services*, pp. 298–313, Springer, Paris, France, November 2013.

[25] N. Forum, "NFC forum technical specifications," August 2017, Available at: http://nfc-forum.org/our-work/specifications-and-applicationdocuments/specifications/nfc-forum-technical-specifications/.

[26] Z. Wang, Q. Liu and Y. Zhang, "A research on vulnerability discovering for router protocols based on fuzzing," in *Proceedings of 2012 7th International ICST Conference on Communications and Networking in China (CHINACOM)*, IEEE, Kunming, China, August 2012.

[27] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerabilty Discovery*, Pearson Eduation, London, UK, 2007.

[28] A. Takanen, *Fuzzing: The Past, the Present and the Future*, SSTIC, Rennes, France, 2009.

[29] M. B. S. Maryam and M. Zolfaghari, "A smart fuzzing method for detecting heap-based vulnerabilities in executable codes," *Security and Communication Networks*, vol. 9, 2016.

[30] J. Yan, Y. Zhang and Y. Dingning, "Structurized grammar-based fuzz testing for programs with highly structured inputs," *Security and Communication Networks*, vol. 6, pp. 1319–1330, 2013.

[31] Z. Wang, Q. Liu and Y. Zhang, "A framework for discovering router protocols vulnerabilities based on fuzzing," *KSII Transactions on Internet and Information System*, vol. 7, 2013.

[32] NVD, "National vulnerability database," December 2017, https://nvd.nist.gov/.