2020 16th International Conference on Mobility, Sensing and Networking (MSN) | 978-1-7281-9916-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/MSN50589.2020.00095

NoPTPeer:Protecting Android Devices from Stealthy Spoofing and Stealing in WLANs without Privilege

Shuying Wei
Institute for Network Sciences and Cyberspace
Tsinghua University
Beijing, China
wei-sy18@mails.tsinghua.edu.cn

Xiaoliang Wang
Information Engineering College
Capital Normal University
Beijing, China
wangxiaoliang@cnu.edu.cn

Ke Xu
1Department of Computer Science and Technologyy
2Beijing National Research Center for Information Science and Technology
1,2Tsinghua University
3Peng Cheng Laboratory
Beijing, China
xuke@tsinghua.edu.cn

***Abstract*—Android devices are prone to spoofing attacks in Wireless Local Area Networks (WLANs), and many of them access numerous unknown networks in daily use. Moreover, because of the weak authentication between Android smartphones, attackers can steal data in a stealthier way based on Address Resolution Protocol (ARP) spoofing. These facts bring a gap in the study of device-side spoofing defense for Android devices. So in this paper a framework is proposed which requires No Privilege but can guarantee the True identity of Peers (NoPTPeer), to protect Android's device-to-device communication in WLANs. Its main features include realizing strong authentication between Android devices, controlling dangerous outgoing connections, and monitoring suspicious incoming connections. These features are realized by an Identity-Based-Signature (IBS) scheme, an Android base class VpnService, and information read from Android system files, which all require no root privilege and are independent of network infrastructures. We implement this framework as an Android smartphone application. The experiments show its effectiveness in detecting spoofing and monitoring stealing, as well as acceptable overhead in memory, Central Processing Unit (CPU) usage and communication latency.**

Keywords—Android, Wireless Local Area Network, ARP Spoofing, Identity Based Signature

I. Introduction

In a Wireless Local Area Network (WLAN), one node can be identified using an Internet Protocol (IP) address or a Media Access Control (MAC) address, while identity spoofing is an easy-to-launch attack. In theory, any attacker capable of programming raw packets can send fabricated packets with fake source IP or MAC addresses and spoof other users in the same WLAN [1]. As the vulnerability lies in the nature of weak authentication in protocol design, this kind of attack is difficult to defend, especially when WLANs and smartphones have formed an indispensable part in people's daily life. Compared to the traditional computer environment, smartphones' defense against attacks is backward while private information is more abundant. And the portability of smartphones makes the situation even worse. They are more likely to access a variety of unpredictable or dangerous WLANs with their careless users. Taking Address Resolution Protocol (ARP) spoofing as example, it aims to attack ARP cache to modify the actual IP and MAC address mapping and is considered one of the easiest attacks. Many smartphones are vulnerable to it nowadays. When a smartphone user accesses the Internet or transfers files locally using the WLAN in a restaurant, a café or even their homes, anyone in the same network equipped with a computer or a rooted phone can send fabricated ARP Reply packets to force the phone's ARP cache to change and then kick the phone out of the network or monitor its communication and steal information in a further Man-in-the-Middle (MITM) attack.

Android as the most prevailing mobile OS claims a market share of over 80% in smartphones. But unfortunately, many Android devices do not perform any check to detect possible spoofing attacks. In addition, it turns out that if without enough security considerations, some Android applications' seemingly admirable features will provide a broader attack surface in WLANs. For example, the open ports contained in applications can make the device act as a server and listen to incoming connections from other devices. After establishing device-to-device communication in local networks, they can transfer big files without access to the Internet. But in practice, many open ports with weak or no authentication have become a door for attackers [2]. Attackers conducting simple ARP spoofing can exploit open ports with weak authentication to steal files from victims more easily and stealthily. More detailed discussion will be given in Section III.

Although there have been continuous efforts in detecting and mitigating ARP spoofing [3], [4], [5], [6], the defense on Android devices remains a neglected area. The few studies about defending spoofing attacks in Android systems suffer from their special requirements for networks or devices [7], [8], because any privilege required, including the root privilege of an Android device and the administrative privilege of a network, will prevent an approach from functioning in daily use or in unknown networks. So our goal is to implement an effective framework against spoofing attacks among Android devices which requires no privilege and is independent of the networking configurations. The main contributions of this paper are as follows:

- We propose a threat model in which an attacker can steal files from a victim's open ports based on a simple ARP Spoofing attack and prove this attack is easy to launch and difficult to notice;
- We introduce a framework NoPTPeer to defend spoofing attacks among devices in WLANs, which mainly contains three features, i.e., stronger authentication between devices, control of outgoing connections and check for suspicious incoming connections to open ports;
- We implement the framework in Android devices as an application, and the experiments show its effectiveness with acceptable overhead;
- Finally, our framework maintains mappings among MAC address, IP address, identity and outgoing traffic of Android devices, which can be further analyzed to make Android environment more secure and trusted.

The rest of this paper is organized as follows: Section II discusses the gap in related work on Android devices' defense against spoofing attacks. Section III introduces the details of our threat model. Section IV describes the design of our framework, including the necessary features and the corresponding methods. Then the implementation details are provided in Section V. Section VI proves our implementation's effectiveness through experiments and evaluation.

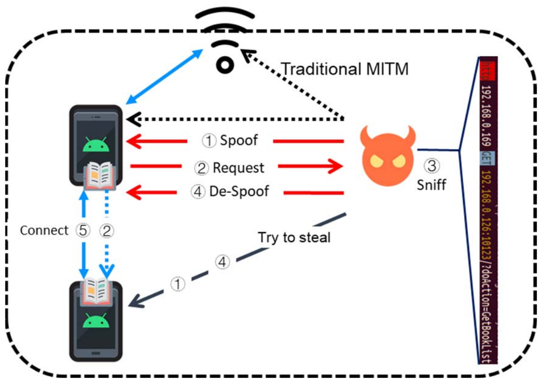
II. Related Work

The detection and mitigation of spoofing attacks has always been an important issue for the safety and trust in cyberspace. But finding a perfect solution to address them is difficult.

Despite years of research, ARP spoofing is still one of the easiest-to-launch attacks in WLANs. Traditional ARP spoofing detection and mitigation techniques analyzed in [3], [4] focus on the computer environment and a substantial portion of them is network-side solutions, i.e., introduce custom network protocols or infrastructure, such as the Secure ARP Protocol (S-ARP) which signs all the ARP messages with the public key [9], or the ARP Central Server (ACS) which validates ARP entries of all the hosts within a network [10]. As for the host-based methods, they may be difficult to be transplanted to the Android system. Taking the Internet Control Message Protocol (ICMP) probe packets based technique as example, it sends ICMP packets to validate ARP update packets before updating ARP cache [11]. But for common Android users, the update of ARP cache is out of control.

Some prevention methods for the Internet of Things (IoT) take Android phones into consideration as IoT nodes [5], [6]. But they actually work on the router/switch, thus they can also be seen as network-side approaches.

Besides, a series of studies about source address validation based on IPv6 in China's Next Generation Internet (CNGI) have proposed mature mechanisms to enhance trustworthiness and accountability of the identity in networks [12] – [16]. Generally they bind a link-layer property which is called the trust anchor and is often the MAC address, to the IP address and the identity based on trusted infrastructure, and prevent IP, MAC, as well as identity spoofing attacks. But firstly, they are network side approaches and cannot be promoted widely in the short term. Secondly, the study on Android environment is always one step behind that on the computer environment.



```
http 192.168.0.169 GET 192.168.0.126:10123/?doAction=GetBookList
```

Fig. 1. An example of the attack scenario.

To our best knowledge, there is little work about defending ARP spoofing attacks on Android devices. The strategy presented in [7] uses a *tcpdump* based application at the device side to capture and verify Dynamic Host Control Protocol (DHCP) and ARP packets. After verification, they run "arp -s" in *BusyBox* to store the gateway's IP and MAC address mapping in a static way. In [8], an Access Point (AP) collects IP and MAC address mappings of all the connected devices and broadcasts them to devices. Devices save the mappings in the application memory and check periodically against their real ARP cache to detect spoofing.

As far as we are concerned, they both need privileges to make their schemes work, i.e., either root privilege of an Android device (*tcpdump* and *BusyBox*) or the administrative privilege of the network (DHCP server and AP). The root privilege will incur great difficulties in promoting as rooting an Android device is complex and may affect its normal use. The administrative privilege means dependence on the network infrastructure or configurations, and thus it may disagree with the portability of Android devices .

III. Threat Model

A. Scenario Description

As Fig. 1 shows, we assume a WLAN allows intra-network connectivity. A legitimate Android device can start device-device communication with the other device by connecting to the other's open ports. We define devices that trust each other as *peers*, the connections devices start as *outgoing connections*, and the connections from other devices as *incoming connections*.

In this threat model, the attacker's objective is to steal information from a victim's legitimate device whose open ports perform weak or no authentication when transmitting information, and the attacker's ability is to send fabricated ARP packets and sniff on its own device. While the legitimate Android devices' objective is to always make outgoing connections to a trustworthy peer, detect and block the traffic to a peer that may be under attack, and alert when there are suspicious incoming connections. What these devices have is only user privileges and information shared among peers.

B. Attack Process

The attack process is presented in Fig.1. Firstly, the attacker sends ARP spoofing packets to legitimate devices to make them mistaken the attacker's MAC address for other peers'. Then when one victim device starts an outgoing connection to its peer, it will send requests to the attacker wrongly. Once the attacker sniffs a request, it will send de-spoofing packets to recover peers' normal connection and use information in the request to access the open port and try to steal data.

C. Feasibility Analysis

From our experiment, this attack mode makes sense. We test features Book Transferring in WLAN of a system application iReader in Android, and Visiting from PC of a popular file management application, ES File Explorer. When their file transferring feature is on, they will open a fixed port and others can access files using a link, i.e., http://IP:10123/ and ftp://IP:3721/, respectively. It seems that they barely authenticate and the only "protection" is the port's open status is unknown to others. So sniffing the request is enough for the attacker to steal files directly from victim devices.

We also test common MITM attacks launched between a device and a gateway or two devices using hacking tools. The result shows this threat model is easier to launch and more difficult to notice for either users or network administrators.

IV. NoPTPeer

In this section, we propose our framework, NoPTPeer, which requires No Privilege but guarantees the True identity of Peers on Android side. We will first describe our design principles and then introduce main features which are necessary to reach an efficient defense. At last, we will talk about the challenges in realizing them and the countermeasures.

A. Design Principles

1) No need for root: Root user has the highest permission in Android OS, but to get it is so complex that the need for root often simply means impossible to promote in practice. So our framework must operate with ordinary user permissions. We can neither capture all connections nor configure static ARP.

2) Independent of network: As discussed before, an important fact making Android's denfense more difficult is that Android phones are likely to access more unfamiliar WLANs. Therefore, no network administrative privilege should be assumed in our method. Moreover, our framework focus on the authentication between two devices but not the device and the gateway, as we cannot get enough and verifiable information about the gateway from the device side.

3) Transparent for other applications and low resource overhead: Our framework should not affect the devices' daily use. It neither requires other applications to be specially configured nor generates considerable resource consumption.

B. Necessary Features

Firstly, the basic function of our framework is to detect ARP spoofing timely, which means there must be ways to verify if one peer's true MAC address is the same as that in the ARP cache. Our framework will periodically check the ARP cache

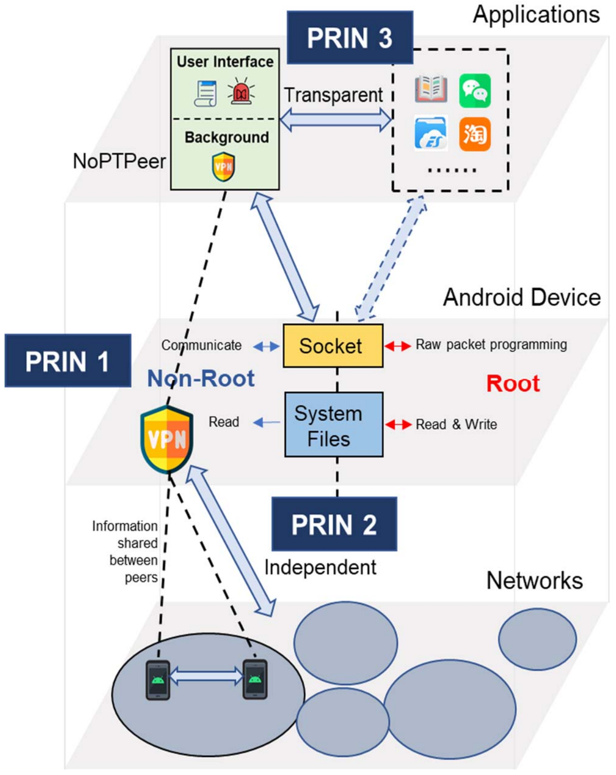


Fig. 2. The three principles in the design of NoPTPeer.

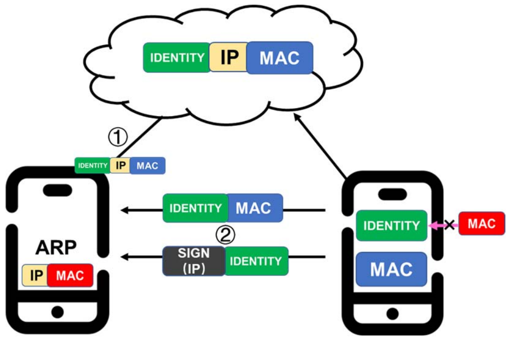


Fig. 3. The two methods for identity authentication in our framework.

and authenticate the mappings between IP and MAC address. Thus the authentication methods are essential.

And further, as our threat model shows, only detecting and warning is not enough, because once the victim starts a connection using the spoofed MAC address, the peer's open port information will leak immediately. So to avoid information theft, the outgoing and incoming connections must be managed in a device's best effort. The minimum is to stop the outgoing connection to an attacker address.

C. Challenges and Countermeasures

1) How to achieve strong authentication: As Fig.3 depicts, we propose 2 methods to authenticate the mapping between IP and MAC address in the ARP cache.

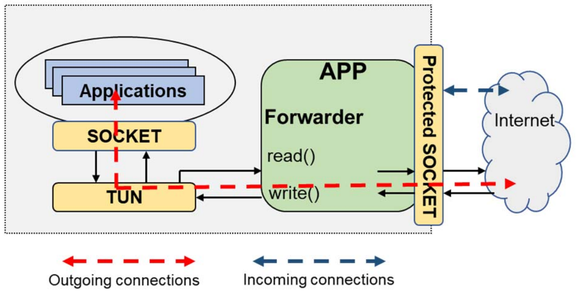


Fig. 4. Outgoing and Incoming connections in an Android device after an APP starts the VpnService.

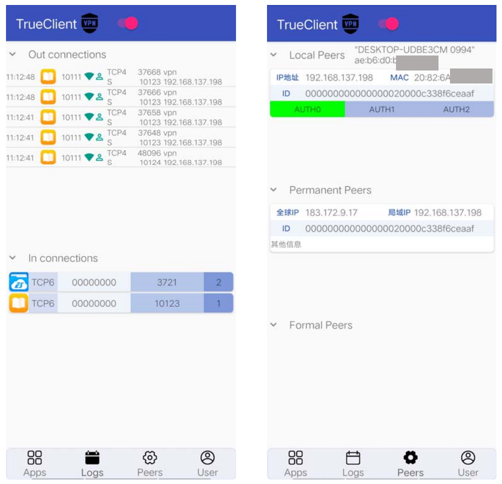
The first method is direct, i.e., every device in our framework uploads its MAC address and the current IP address to our central server. We store these reords and provide query service to get peers' IP and MAC address mappings in the same WLAN. Then the device can check if the queried mappings match its own ARP cache. But this method requires access to the Internet. In a closed network environment, which is a main scenario for file transferring among devices in WLANs, it may not work well.

So in our second method, we use Identity-Based-Signature (IBS) to achieve stronger authentication. IBS is a type of public-key cryptography in which one's identity is also its public key. The others can use one's identity to verify its signature conveniently. Our scheme is to generate an IDENTITY from the MAC address. When a signature on an IP address is verified using the IDENTITY, the mapping between the IDENTITY and the IP address is confirmed. At this point, once the mapping between IDENTITY and the MAC address is verified, the IP and MAC address mapping is also authentiacted. So our method's key falls to design an algorithm to generate and verify the mapping between the MAC address and IDENTITY. We will introduce an algorithm meeting the requirements in details in Section IV.

2) How to manage outgoing and incoming connections: The most powerful and non-root means we find to manage traffic is an Android base class, *VpnService* [17], [18]. As shown in Fig.4, when an application starts a VpnService, a virtual interface, TUN, is created and returned as a file descriptor. Then traffic will be routed to it. The application can write the incoming packets to or read the outgoing packets from the file descriptor. By implementing a *Forwarder* to excahnge and handle packets between TUN and its tunnel socket, the application has a control over the outgoing connections. So when we add our filter logic in the forwarding process and block the outgoing connection to a suspicious address, we can realize the minimum necessary feature.

However, to avoid a loop connection, the application has to *protect* its tnnuel socket, which means the socket is outside of the VPN and will not route to TUN. Therefore, the incoming connections are out of VpnService's reach.

Inspired by [2] which identifies all the TCP open ports through reading files under the */proc/net/* directory, we devise


TrueClient

Out connections

```
11:12:48  10111  TCP4 S  37668 vpn  10123 192.168.137.198
11:12:48  10111  TCP4 S  37666 vpn  10123 192.168.137.198
11:12:41  10111  TCP4 S  37658 vpn  10123 192.168.137.198
11:12:41  10111  TCP4 S  37648 vpn  10123 192.168.137.198
11:12:41  10111  TCP4 S  48096 vpn  10124 192.168.137.198
```

In connections

TCP6	00000000	3721	2
TCP6	00000000	10123	1

TrueClient

Local Peers "DESKTOP-UDBE3CM 0994" ae:b6:d0:b[redacted]

IP地址	192.168.137.198	MAC	20:82:6A[redacted]
ID	00000000000000020000c338f6ceaaf		

AUTH0 AUTH1 AUTH2

Permanent Peers

全球IP	183.172.9.17	局域IP	192.168.137.198
ID	00000000000000020000c338f6ceaaf		

其他信息

Formal Peers

Fig. 5. User interfaces in our application showing the log of outgoing connections, current state of incoming connections, and peers in the same WLAN.

a simple scheme to warn users about suspicious incoming links to open ports, making the attacker's access non-stealthy. Firstly, we maintain a real-time list of open TCP ports through the */proc/net* directory, and then we periodically query the established connections number of every open port. Thus we can monitor if there comes any new connection to open ports and notify the users. We believe this can protect users from stealthy information theft, as in practice open ports are usually started and closed manually by users, and they are often clear about how many connections are to come in.

V. Implementation Details

We have implemented this framework in an Android application and a central server. In this section, we will discuss what algorithms need to be realized, how to implement them in an Android environment and what service or commands we use to realize our schemes.

A. Algorithm to Generate ID from a MAC address

Since exposure, revocation and regeneration of keys is essential for a public-key cryptography, instead of using a MAC address as an ID or public key directly, an algorithm that has the following features is needed: (1) able to generate many IDs from one MAC address, (2) able to verify if an ID is generated from the MAC address, and (3) difficult to generate the same ID from two different MAC addresses. We find the Cryptographically Generated Address (CGA) algorithm [19] meets these features well. It is used in the SEND protocol to generate the IPv6's 64-bit Interface Identifier from a public key, and the binding between the public key and the IPv6 address can be verified. So we can just consider our MAC address and ID as the public key and Interface Identifier in the algorithm. The modified algorithm's details are as follows:

1) Generation: Input the MAC address and a 3-bit Sec(i.e., 0-7).

a) Set the modifier to a pseudo-random 128-bit value.

b) Concatenate the modifier, 9 zero octets and the MAC address. Execute the SHA-1 on the concatentation. Take the 112 leftmost bits of SHA-1 hash value as Hash2

c) Compare the 16*Sec leftmost bits of Hash2 with Zero. If they are all zero, continue with Step d. Otherwise, increment the modifier by one and go back to Step b.

d) Set the 8-bit collision count to zero.

e) Concatenate the final modifier, the collision count and the MAC address. Execute SHA-1 on the concatentation. Take the 64 leftmost bits of SHA-1 hash value as Hash1.

f) Form an ID from Hash1 by writing the value of Sec into the three leftmost bits and setting bits 6 and 7 to zero.

g) Check if there is a duplicate ID, increment the collision count by one and go back to Step e. After 3 collisions, stop and report the error.

2) Verification: Input ID, the modifier and the collision count. The detailed steps are omitted here. In short, it first computes HASH1 and compares it with ID before computing HASH2. Compare 16*Sec leftmost bits of Hash2 with Zero.

We test the time needed to generate a valid ID from a given MAC address on an Android device, Vivo NEX S. First of all, the Sec value 0 actually makes no sense in verification and thus we do not consider it. When the Sec is set to 1, it takes several seconds to generate a valid ID, while for the value 2, the time may range from tens of seconds to more than 10 minutes. As for greater Sec values, the time required is totally unacceptable for common users. Therefore, we set the Sec to 1 in our implementation of NoPTPeer.

B. *Pairing-based IBS in Android*

We focus on how to implement IBS in an Android device but not the comparison or selection of algorithm designs. We have implemented a lightweight IBS scheme proposed by [20], which is a mainstream *pairing-based* IBS.

In the general framework of IBS schemes, the core is the Private Key Generator (PKG), which is responsible for keeping a master secret key, publishing public system parameters, and generating private keys [21]. More specifically, in our chosen pairing-based IBS scheme, the PKG first chooses a *pairing* according to the given security parameters when setting up an new IBS system. Then it chooses a master key which will always be kept secret randomly and computes public parameters which are open to all [20]. Every time there comes a request to register an ID, we check for ID collisions and verify the binding between the ID and the MAC address. After the check and verification, the PKG generates the corresponding private key for the verified ID and return it together with public system parameters to the device, which then stores them in permanent files. The process is partially shown in Fig.6.

In addition, to use *pairing* correctly in Android and set up an unified IBS system among the server and devices, we must generate a same *pairing* using the same initialization parameters, which can be exported into a file *pairing.properties*. It should be noticed that using the same security parameters *rbits* and *kbits* can generate different *pairings*. So we run the server using the desired security parameters first and get its *pairing.properties*.

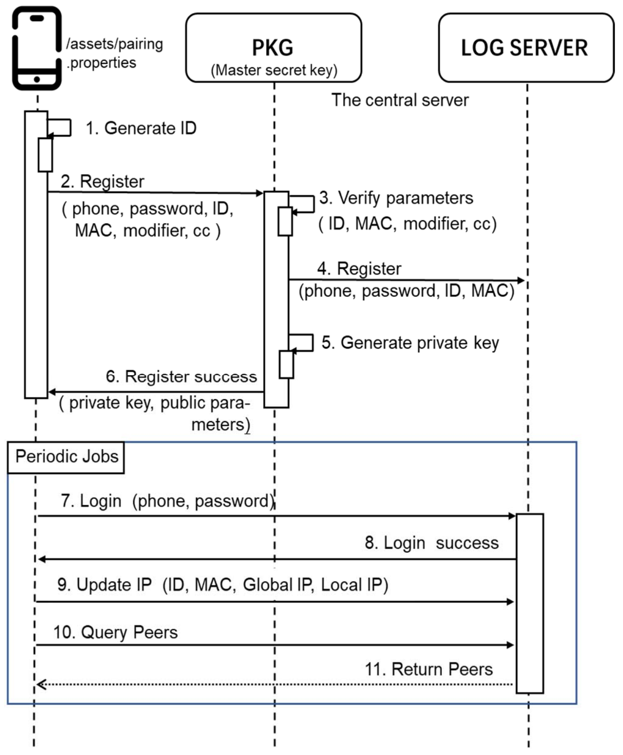


Fig. 6. The interactions between an Android device and the central server.

Then we put the file in Android project's *assets* directory, which will be contained into an Android application package (APK) and installed on devices. With the same pairing, public system parameters and a private key, every device can sign and verify independently.

In our implementation of NoPTPeer, we set *rbits* to 160 and *kbits* to 1024 according to [20], and store all the keys, parameters and signatures in hex format. We test the performance of IBS in aspects of processing time and storage size on the Android device, Vivo NEX S. As the IBS scheme [20] splits the signature into two phases, *OffSign* which preloads heavy computations without the knowledge of the message to sign and *OnSign* which only performs light computations on the message, the signature time can be reduced to 1 milliseconds with the *OffSign* taking about 80 milliseconds. The verification time is about 100ms no matter the length of the signed message. Besides, the length of a signature is fixed to 296 bytes and the total length of all the parameters and the private key is less than 4 KB, which is negligible for an Android device.

C. *Commands Users can Access*

To make up for the lack of powerful means to manage incoming connections, we must try to get as much as possible information about them. Android provides many channels to get information about connections, and it allows users to execute commands in the code. We find the "*netstat*" command that provides very admirable information. It can show active TCP connections, ports on which device is listening, and statistics for IP, ICMP and so on.

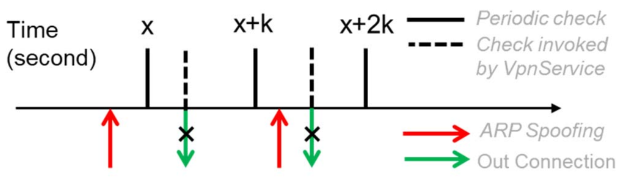


Fig. 7. Two cases about the ARP spoofing attack and an outgoing connection's happening time.

So even with common users permission, we can construct a command as below for every found TCP open port to get the number of its established connections:

netstat -nat | grep -i port | grep ESTABLISHED | wc -l

D. JobService to do Periodic Work

There are many operations in our framework that need to be launched every a short period, for example, to detect the ARP spoofing immediately, we must check ARP cache frequently. Android provides a base class, JobService, that can handle tasks previously scheduled. Or to say, using JobService we can set a task that should be done in a given time. JobService can also set periodic tasks, but Android has forced the minimum periodic interval to be 15 minutes which is too long to detect attacks.

So we reschedule a same job to be handled in the given interval every time the current job is to end to set periodic tasks with any interval. Using this method, we can realize almost real-time detection by setting a short interval to conduct those check tasks.

E. VpnService to Check and Block Suspicious Address

NetGuard Project [22] implements its VpnService which can block individual address per application, and log all the outgoing traffic. So we use its code in our application and add our detection logic to block the connections to suspicious addresses. As shown in Fig. 7, the ARP spoofing attack and one outgoing connection may happen in a same interval between two spoofing checks. Therefore every time we detect an outgoing connection to peers in VpnService, we must invoke a check immediately. Only after passing this check, the outgoing connection is allowed.

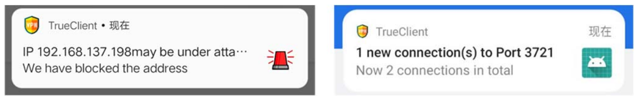
In addition, we decide to record all the TCP outgoing connections and DNS responses in detail with the help of VpnService. We believe these records have richer semantics and can be further used to detect more complex malicious behaviors in Android [23], [24], [25].

VI. Experiment and Evaluation

In this section, we experiment on Android smartphones to show basic features of NoPTPeer. Then we will analyze the possibility of a replay or a MITM attack during the IBS-based authentication in a closed network. Finally, we evaluate the performance overhead in CPU usage, memory and communication delay NoPTPeer may incur.

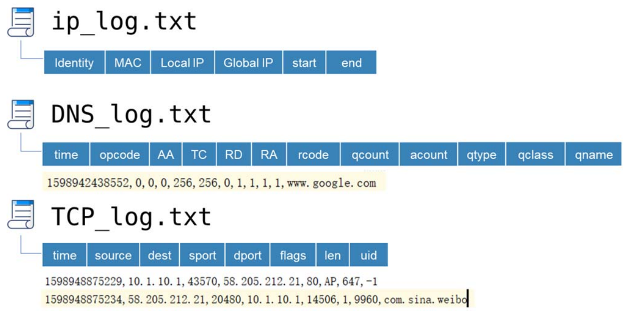
A. Experiments when the Internet is accessible

We set up our experiments with two Android phones, Vivo NEX S and OPPO PCHM 10 as peers, and an Ubuntu VMware as an attacker.


TrueClient • 现在
IP 192.168.137.198may be under atta…
We have blocked the address

TrueClient 现在
1 new connection(s) to Port 3721
Now 2 connections in total

Fig. 8. Notifications made by NoPTPeer.



```
1598942438552, 0, 0, 0, 256, 256, 0, 1, 1, 1, 1, www. google. com
```

```
1598948875229, 10. 1. 10. 1, 43570, 58. 205. 212. 21, 80, AP, 647, -1
1598948875234, 58. 205. 212. 21, 20480, 10. 1. 10. 1, 14506, 1, 9960, com. sina. weibo
```

Fig. 9. Logs recorded by NoPTPeer.

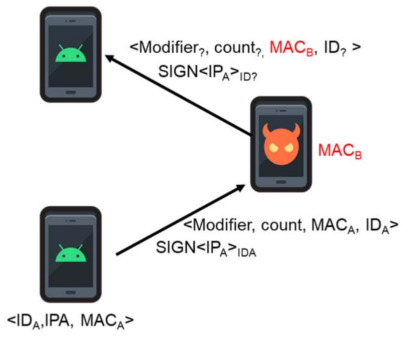
For NoPTPeer, we mainly set 3 periodic tasks: (1) get peers information in the same WLAN from the server every 5 minutes, (2) read the file */proc/net/arp* (where the ARP cache stores) and check against local peer information every 5 seconds, (3) get the list of TCP open ports and the number of established connections for per open port every 5 seconds. As for the VpnService's block logic, we check every new session, and if it is going to one of peers, we invoke the job (2) once to check the destination's MAC address. If the MAC address in ARP cache is inconsistent with that in local peer information, we block the connection immediately and notify the user.

In Ubuntu VMware , we use *scapy* to fabricate spoofing ARP Reply packets and *bettercap* to sniff the traffic. The attack's process has been introduced in Section III. We first keep sending spoofing packets to two victims, and both phones can detect the attack in 5 seconds. Then we control the spoofing and the request of *Book Transferring in WLAN* from Vivo to OPPO in a same check interval. NoPTPeer detects the connection to a peer, invokes an ARP cache check, and blocks this connection directly. We also run NoPTPeer in the background on OPPO. When the feature *Visit from PC* of ES File Explorer is on, we visit its link from two computers, and it can notify these new connections in the foreground in 5 seconds.

We also record the IP history, DNS responses and TCP connections of the device shown in Fig.9. Using VpnService, we can identify by which application every flow is generated. This semantics is difficult to get from networks.

B. Possibility of destroying authentication in closed networks

The strong authentication between peers in a closed network is based on the signature verification of an IP address and the binding between ID and the MAC address. If an attacker wants to destroy the authentication and achieve ARP spoofing successfully, he must get a signature of the victim IP address signed with certain ID, and parameters to verify the binding between his own MAC address and the certain ID.



<Modifier?, count?, MAC_B, ID? >
SIGN<IP_A>_ID?
MAC_B
<Modifier, count, MAC_A, ID_A>
SIGN<IP_A>_IDA
<ID_A,IPA, MAC_A>

Fig. 10. When an attacker tries to spoof the IP A's MAC address.

Obviously, if the attacker is one of the IBS system and has a pair of legitimate ID and private key, he can generate signatures of any IP address and should not be verified as invalid. But we consider this as a real-name attack which can be traced easily.

For attackers out of the IBS system, although the signature and parameters used to verify are not kept secret for them in the same WLAN, we believe it is still difficult to conduct a replay or a MITM attack in the authentication process. Taking the signature as example, if an attacker wants to replay a signature from other peers to claim an ownership of the victim IP address, he has to compute the required parameters to pass the verification on his own MAC address and the victim ID. Recalling our *Verification* algorithm, the attacker first needs to find a 128-bit modifier and a collision count so that the hash value, *SHA-1(modifier||MAC address||collision count)*, can be compared with the victim ID. This is quite time and computation consuming.

But it is still possible. And once an attacker computes successfully the verification parameters on his own MAC address and the victim ID, he can launch replay or MITM attacks against the victim. So a regular ID-updating scheme may make our framework's authentication stronger.

C. Evaluation of performance overhead

We test the performance overhead of NoPTPeer in communication delay, memory and CPU usage on OPPO PCHM10 which is equipped with an 8-core processor.

1) Communication Delay: As *VpnService* handles every packet and checks every connection, our framework do introduce additional delay. Firstly, we measure an application's time spent to connect to a peer in the same WLAN. The result shows VpnService causes 3 to 5 more milliseconds every connection. Then we see the sum of bytes sent and received (provided by file */proc/net/dev*) every second as network speed of the phone and compute it for 3 minutes separately when NoPTPeer is on and off. Through comparing results shown in Fig.11, we hold that the delay introduced by NoPTPeer is not quite noticeable.

2) Memory: We run QQ music in the background and browse Taobao in the foreground. Then we record the total memory used by phone (Fig.12) and respective memory used by

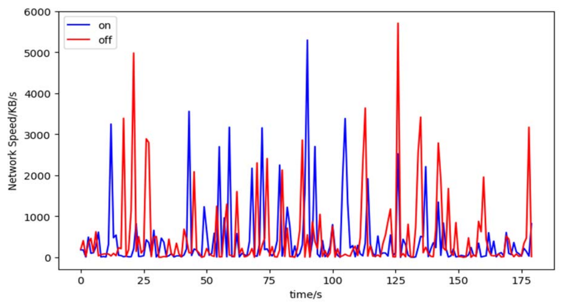


Fig. 11. Network speed of phone when NoPTPeer is on and off.

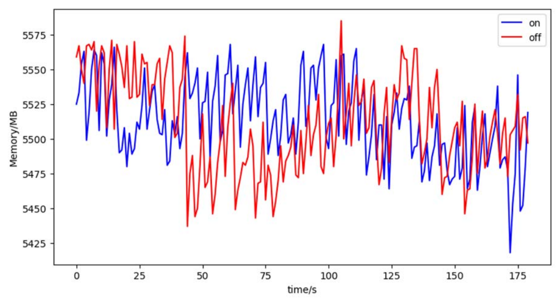


Fig. 12. Total memory used by phone when NoPTPeer is on and off.

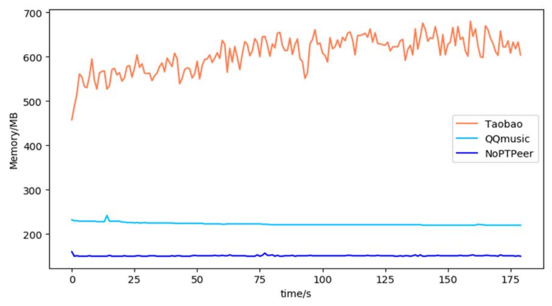


Fig. 13. Memory used by 3 applications when NoPTPeer is on.

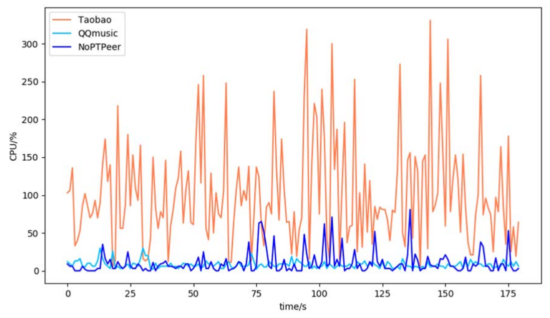


Fig. 14. CPU usage of 3 applications when NoPTPeer is on.

3 applications (Fig.13) for 3 minutes when NoPTPeer is on and off. We find NoPTPeer does not introduce significant memory usage. Maybe this is because VpnService mainly needs limited memory for buffering traffic when forwarding and will not increase the total memory.

3) CPU Usage: In the same test in *2)*, we also measure the individual CPU usage of 3 applications. As shown in Fig.14, although NoPTPeer's CPU usage may rise sharply when doing heavy network activities, it keeps around 60% and is roughly consistent with QQ music. Thus we believe the performance overhead is acceptable when NoPTPeer works as a background application in daily use.

VII. Conclusion

In this paper, we demonstrate a threat of being spoofed and stolen stealthily which most Android phones may be prone to in WLANs. To defend it, we propose and implement a framework, NoPTPeer, to help guarantee the true identity of peers in WLANs requiring no privilege. We use IBS to realize stronger authentication among Android devices that can work even in a closed network. We also detect and block all the dangerous outgoing connections with the help of *VpnService*. Moreover, a method to monitor open ports and their connections is proposed to fill the lack of means to manage the incoming connections. The experiments prove the great effectiveness and small performance overhead of the framework.

In fact, the threats faced by Android devices in WLANs are far more than ARP spoofing. In the future, we plan to analyze traffic logs generated by VpnService to detect other more complex malicious network activities.

Acknowledgment

This work is supported by the National Key R&D Program of China with No.2018YFB1800402, National Science Foundation for Distinguished Young Scholars of China with No.61825204, National Natural Science Foundation of China with No.61932016, 61572278 and U1736209, Beijing Outstanding Young Scientist Program with No.BJJWZYJH-01201910003011, and the project "PCL Future Greater-Bay Area Network Facilities for Largescale Experiments and Applications (LZC0019)". Ke Xu is the corresponding author of this paper.

References

[1] S. Bhirud and V. Katkar, "Light weight approach for IP-ARP spoofing detection and prevention," *2011 2nd Asian Himalayas International Conference on Internet (AH-ICI)*, Kathmandu, Nepal, 2011, pp. 1-5.

[2] D. Wu, D. Gao, R. Chang, E. He, E. Cheng, and R. Deng, "Understanding open ports in Android applications: Discovery, diagnosis, and security assessment," *2019 26th Network and Distributed System Security Symposium (NDSS)*, San Diego, USA, 2019, Proceedings 1-14.

[3] N. Tripathi and B. Mehtre, "Analysis of various ARP poisoning mitigation techniques: A comparison," *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kanyakumari, India, 2014, pp. 125-132.

[4] J. Meghana, T. Subashri and K. Vimal, "A survey on ARP cache poisoning and techniques for detection and mitigation," *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, Chennai, India, 2017, pp. 1-6.

[5] A. Dua, V. Tyagi, N. Patel and B. Mehtre, "IISR: A secure router for IoT networks," *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, Mathura, India, 2019, pp. 636-643.

[6] M. Al-Shaboti, I. Welch, A. Chen and M. Mahmood, "Towards secure smart home IoT: Manufacturer and user network access control framework," *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, Krakow, Poland, 2018, pp. 892-899.

[7] Y. Zhao, Y. Lei, T. Yang and Y. Cui, "A new strategy to defense against SSLStrip for Android," *2013 15th IEEE International Conference on Communication Technology*, Guilin, China, 2013, pp. 70-74.

[8] P. Phetchai, J. David Ndibwile, D. Fall, S. Kashihara and S. Tuarob, "Securing low-computational-power devices against ARP spoofing attacks through a lightweight Android Application," *2017 21st International Computer Science and Engineering Conference (ICSEC)*, Bangkok, Thailand, 2017, pp. 1-6.

[9] D. Bruschi, A. Ornaghi and E. Rosti, "S-ARP: a secure address resolution protocol," *2003 19th Annual Computer Security Applications Conference*, Las Vegas, USA, 2003, pp. 66-74.

[10] S. Kumar and S. Tapaswi, "A centralized detection and prevention technique against ARP poisoning," *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, Kuala Lumpur, Malaysia, 2012, pp. 259-264.

[11] P. Pandey, "Prevention of ARP spoofing: A probe packet based technique," *2013 3rd IEEE International Advance Computing Conference (IACC)*, Ghaziabad, India, 2013, pp. 147-153.

[12] G. Hu, K. Xu, J. Wu, Y. Cui and F. Shi, "A general framework of source address validation and traceback for IPv4/IPv6 transition scenarios," in *IEEE Network*, vol. 27, no. 6, pp. 66-73, November-December 2013.

[13] D. Zhou, J. Bi, G. Yao, et al., "Trustworthy identity system based on IPv6 source address validation, " in *Journal on Communications*, vol. 35, no. Z1, pp. 20-26, October 2014.

[14] G. Hu, W. Chen, Q. Li, Y. Jiang and K. Xu, "TrueID: A practical solution to enhance Internet accountability by assigning packets with creditable user identity code," in *Future Generation Computer Systems*, vol. 72, pp. 219-226, September 2016.

[15] B. Wu, H. Li and Q. Wu, "Extending authentication mechanism to cooperate with accountable address assignment," *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakesh, Morocco, 2019, pp. 1-7.

[16] X. Wang, K. Xu, W. Chen, Q. Li, M. Shen and B. Wu, "ID-Based SDN for the Internet of Things," in *IEEE Network*, vol. 34, no. 4, pp. 76-83, July/August 2020.

[17] Y. Song and U. Hengartner. "PrivacyGuard: A VPN-based platform to detect information leakage on Android devices," *2015 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, Colorado, USA, 2015, pp.15-26.

[18] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou, "Antmonitor: Asystem for monitoring from mobile devices," *2015 SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big Internet Data (C2BID)*, London, UK, 2015, pp.1-15.

[19] T. Aura, " Cryptographically Generated Addresses (CGA), " RFC 3972, DOI 10.17487/RFC3972, March 2005.

[20] F. Li, D. Zhong and T. Takagi, "Practical identity-based signature for wireless sensor networks," in *IEEE Wireless Communications Letters*, vol. 1, no. 6, pp. 637-640, December 2012.

[21] R. Wan, B. Da, R. Li, C. Wang and H. Li, "Identity based security for authentication and mobility in future ID oriented networks," *2018 International Conference on Information Networking (ICOIN)*, Chiang Mai, Thailand, 2018, pp. 402-407.

[22] M66B. NetGuard, https://github.com/M66B/NetGuard, 2020.

[23] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. v. d. Veen and C. Platzer, "ANDRUBIS -- 1,000,000 Apps later: A view on current Android malware behaviors," *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, Wroclaw, Poland, 2014, pp. 3-17.

[24] G. Zhao, K. Xu, L. Xu and B. Wu, "Detecting APT malware infections based on malicious DNS and traffic Analysis," in *IEEE Access*, vol. 3, pp. 1132-1142, July 2015.

[25] F.A. Narudin, A. Feizollah, N.B. Anuar *et al.,* " Evaluation of machine learning classifiers for mobile malware detection," in *Soft Comput*, vol. 20, no. 1, pp. 343–357, November 2014.