2016 IEEE Symposium on Security and Privacy

Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf

Xiaolong Bai*[,1], Luyi Xing*[,2],
Nan Zhang[2], XiaoFeng Wang[2], Xiaojing Liao[3], Tongxin Li[4], Shi-Min Hu[1]
[1]*TNList, Tsinghua University, Beijing,* [2]*Indiana University Bloomington,*
[3]*Georgia Institute of Technology,* [4]*Peking University*
bxl12@mails.tsinghua.edu.cn, {luyixing, nz3, xw7}@indiana.edu,
xliao@gatech.edu, litongxin@pku.edu.cn, shimin@tsinghua.edu.cn

Abstract*—With the popularity of today's usability-oriented designs, dubbed *Zero Configuration* or *ZeroConf*, unclear are the security implications of these automatic service discovery, "plug-and-play" techniques. In this paper, we report the first systematic study on this issue, focusing on the security features of the systems related to Apple, the major proponent of ZeroConf techniques. Our research brings to light a disturbing lack of security consideration in these systems' designs: major ZeroConf frameworks on the Apple platforms, including the Core Bluetooth Framework, Multipeer Connectivity and Bonjour, are mostly unprotected and popular apps and system services, such as Tencent QQ, Apple Handoff, printer discovery and AirDrop, turn out to be completely vulnerable to an impersonation or Man-in-the-Middle (MitM) attack, even though attempts have been made to protect them against such threats. The consequences are serious, allowing a malicious device to steal the user's SMS messages, email notifications, documents to be printed out or transferred to another device. Most importantly, our study highlights the fundamental security challenges underlying ZeroConf techniques: in the absence of any pre-configured secret across different devices, authentication has to rely on Apple-issued public-key certificate, which however cannot be properly verified due to the difficulty in finding a unique, nonsensitive and widely known identity of a human user to bind her to her certificate. To address this issue, we developed a suite of new techniques, including a conflict detection approach and a biometric technique that enables the user to *speak out her certificate* through 6 distinct, rare but pronounceable words to let those who know her voice verify her certificate. We performed a security analysis on the new protection and evaluated its usability and effectiveness using two user studies involving 60 participants. Our research shows that the new protection fits well with the existing ZeroConf systems such as AirDrop. It is well received by users and also providing effective defense even against recently proposed speech synthesis attacks.

I. Introduction

With the proliferation of portable computing systems such as tablet, smartphone, Internet of Things (IoT), etc., ordinary users are facing the increasing burden to properly configure those devices, enabling them to work together. In response to this utility challenge, major device manufacturers and software vendors (e.g., Apple, Microsoft, Hewlett-Packard) tend to build their systems in a "plug-and-play" fashion, using techniques dubbed *zero-configuration* (*ZeroConf*). For example, the AirDrop service on iPhone, once activated, automatically detects another Apple device nearby running the service to transfer documents. Such ZeroConf services are characterized by automatic IP selection, host name resolving and target service discovery. Prominent examples include Apple's *Bonjour* [3], and the Link-Local Multicast Name Resolution (LLMNR) and Simple Service Discovery protocols (SSDP) built into Windows, etc. In addition to those working on the IP network, similar techniques are developed for automatic service discovery on other channels, Bluetooth in particular [29]. The underlying idea of minimizing user involvements in system setup further influences security designs, in which less intrusive solutions such as certificate-based authentication are preferred to those requiring manual configuration, like QR code scan for secret sharing, an approach considered to be inconvenient [40].

Challenges and findings. However, when the design pendulum swings towards usability, concerns may arise whether the system has been adequately protected. Indeed, even though ZeroConf systems are supposed to be deployed in a friendly, collaborative environment, in the real world, they are often operating in the presence of untrusted parties at public locations such as airport, hotel etc. To understand whether the protection those systems receive is commensurate with the threats they are facing, we performed a security analysis on popular ZeroConf systems on the Apple platforms (iOS, Mac OS X). We focus on Apple because it is a main advocate of ZeroConf techniques and also known for its rigorous security control, mostly based upon its public-key infrastructure in which Apple-signed certificates can be used to secure communication. In our study, we inspected popular apps and system services, in an attempt to understand whether they are properly guarded against the realistic adversary in the environment they are supposed to run.

The study brought to light surprising lack of protection in even high-profile ZeroConf services and apps (Section III): security measures are either nonexistent there or incorrectly

*The two lead authors are ordered alphabetically.

DOI 10.1109/SP.2016.45

designed and implemented. More specifically, service discovery on the Bluetooth Low Energy (BLE) channel provided by Apple is typically utilized by apps in an insecure way, leaving the door widely open to a man-in-the-middle (MitM) attack. Even for *Handoff*, Apple's system service, which securely links one's Mac laptop (running OS X) to her iPhone, all iOS services advertised through the BLE channel on the phone are found to be exposed to an unauthorized app (without system privilege) on the laptop, disclosing sensitive user data such as all iOS notifications (SMS messages, emails, instant messages and so on) from Apple Notification Center Service (ANCS). Over the wireless channel, the discovery mechanisms of popular apps like *QQ* and *Filedrop* turn out to be exploitable, even when attempts are made to protect their communication.

Also interesting are the security risks we discovered in Bonjour, Apple's service discovery protocol, which has been utilized in security-sensitive Apple services without proper safeguard in place. As an example, our research shows that this problem allows the adversary in a local network (e.g., a compromised Mac desktop) to silently intercept the document the victim sends to a printer, an attack completely invisible to the victim from her print dialog. Most importantly, we found that the *AirDrop* service on iOS and OS X contains a subtle design flaw: although the service uses TLS to deliver a file between two individuals, the device certificates used in the communication cannot be properly verified, as the binary content they include (e.g., Apple ID) cannot be easily linked to the identifiable information of the persons in the communication (name, appearance, voice, etc.). To establish such a link, upon receiving the file recipient's certificate, AirDrop tries to retrieve her photo from the sender's contacts on his device when possible, according to the hash value of the recipient's email registered in her Apple account (a unique identifier from Apple's perspective) that comes with her certificate. This attempt is often futile given the fact that people tend to have multiple emails and their friends or colleagues may not have their Apple-related emails in the contacts. As a result, a MitM attack can often succeed even in the presence of such TLS protection.

Such vulnerabilities are pervasive in Apple ZeroConf: we analyzed 67 popular Apple apps and services using ZeroConf (e.g. Bonjour, BLE service discovery, etc.), including Airdrop, QQ and Filedrop. The vast majority of them (60 out of 67) are found to be unprotected and vulnerable to the MitM threat. The consequences of the attack are serious, causing the leakage of highly sensitive user information, such as messages, emails, files and more. We reported our findings to Apple and related app developers, who acknowledged that what we found are real and significant. As an example for the impact of the study, based upon our report, Apple has removed the supports for transferring iOS notifications to Mac OS after iOS 8.3 and OS X 10.10.4. Demos of our attacks are posted online [22].

The most important finding of our study is that protecting real-world ZeroConf services can be much more complicated than it appears to be. Even when Apple or app developers take security seriously, often there is little they can do without resorting to manual configuration of shared secrets, particularly in the absence of the iCloud support (which is not supposed to be present for the services like AirDrop that are designed to work in a Wi-Fi ad-hoc network without going through the Internet). Fundamentally, unlike the TLS certificate of a website, which is linked to the site through its domain name, the certificate used in cross-device communication often cannot be trivially connected to its owner (note that device IDs are generated dynamically and can be forged): for example, in AirDrop, often one cannot tell whether a certificate indeed belongs to the person he wants to talk to, as what he knows about the person (e.g., her company email address) can be very different from what is on her certificate (e.g., the hash value of her Apple account email) and even what she uploads to her Apple account. Actually, as evidence for the challenge in protecting ZeroConf, QQ chooses to roll back to manually sharing secrets for protecting its file transfer and Apple still has not found any effective solution to our attack on AirDrop, even after we informed them of the threat months ago.

Securing ZeroConf on Apple. Given the strong demand for ZeroConf techniques, it is of critical importance to come up with usable solutions to address their security risks. In our research, we made a first step toward this end. For service discovery in an environment where all devices can always receive others' service requests (e.g., in the case of Apple Airdrop), we designed a simple mechanism that binds the service name a party announces to its certificate only when no other party claims the same service name, since such a conflict is a necessary condition for a MitM attack (Section IV-A). A more generic solution is a new technique that binds an Apple account certificate to its human owner. Underlying our approach is an idea called *speak out your certificate* (SPYC), which allows anyone who knows the owner's voice to verify her certificate, based upon the intuition that the owner will not speak out the adversary's Apple certificate to give the adversary the opportunity to impersonate her. Key to this technique is a design that maps a certificate to 6 unique, unambiguous and pronounceable *rare or fake* words whose voice samples are hard to collect in daily life. One only needs to say those words *once* (for her certificate), called *SPYC vouch*, during her first authentication with another party and can then send the voice recording together with the certificate when interacting with other parties (e.g., in a TLS connection). For each human contact, the recipient of a certificate only needs to verify the sender's voice *once* to bind her to her certificate. The whole SPYC mechanism is carefully designed to ensure that the vouch is difficult to forge and easy to use. Particularly, we show that

our technique is resilient to known speech synthesis attacks (Section IV-B).

We analyzed the security design of SPYC and further implemented it within the AirDrop services on iOS and OS X. The usability and security of the mechanism was evaluated through two human subject studies with 60 participants. The studies show that identification of a speaker from her SPYC vouch is reliable, resilient to vouch forging and the technique is considered more convenient than joint configuration of a shared secret across devices, e.g., scanning QR code (in the case of AirDrop, one may have to walk about 200 feet to reach her colleague staying in a different office or even on a different floor). We further evaluated the performance of our implementations on two Macbook Pro (Mid 2014 model) and two iPhone 5, which was found to be very efficient, incurring a negligible delay.

Contributions. The contributions of the paper are outlined as follows:

• *New findings and new insights*. We conducted the first study on the security protection of Apple ZeroConf, discovering serious design and implementation flaws in high-profile Apple services (e.g., Handoff, automatic printer discovery and AirDrop) and popular apps (e.g., QQ). Our research shows that those services and apps can be easily exploited in an information stealing or MitM attack, putting sensitive user data in danger. Highlighted in our study is the insight that certificate verification can be difficult in cross-device communication, particularly for linking a human subject's certificate to the information that others use to identify her.

• *New techniques*. We developed a suite of innovative techniques to address the challenges in certificate verification for ZeroConf, including a "no-conflict" approach and a voice based certificate vouch. Particularly, our SPYC design allows one to speak 6 words one time to authenticate the whole certificate to others, which is convenient for establishing a secure channel in ad-hoc communication and robust against the attacks on voice authentication.

• *Implementation and evaluation*. We implemented our designs on the Apple platforms and evaluated them through security analysis, human subject studies and performance tests. Our study shows that the new techniques work effectively in practice, enhancing security protection and largely preserving the usability of ZeroConf systems.

II. Background

The concept of zero configuration is first defined over the IP network [28], for the purpose of setting up a network automatically without configuration of the services like DHCP and DNS. To this end, techniques are developed to self-assign IP addresses to networked devices, resolve conflicts, announce a host name and its IP address through a multicast, e.g., using the Multicast Domain Name System (mDNS), which updates each host's DNS cache, and automatically discover services of interest broadcasted by other devices and let the user choose through service browsing. Later the idea of automatic service discovery has been applied to bootstrap the devices running on other channels. Below, we present two examples to show how ZeroConf works.

BLE service discovery. Bluetooth low energy (BLE) is a wireless technology based on the Bluetooth 4.0 specifications for new applications in the healthcare and home entertainment. Different from the classic Bluetooth, BLE reduces power consumption and is featured by its convenient configuration. The technique has been incorporated into iOS and OS X as part of their Core Bluetooth frameworks. The BLE communication involves two main actors: *peripheral* that provides a service and *central* that discovers and uses the service. Each of such actors is identified by a universally unique identifier (UUID). Further, on the peripheral, individual services running also have their own UUIDs. Here a service is described by a set of characteristics, e.g., whether it is readable, writable or both. During the BLE operation, each peripheral advertises its services through its own UUID. Once it is discovered by a central, a connection between the two devices reveals the services on the peripheral and their characteristics.

To get services from the peripheral, the two devices often need to *pair*, a process that establishes a link-layer secret key between them. Traditionally, this is considered to be a configuration step, at which the user is often required to manually enter a PIN. BLE comes with the Secure Simple Pairing model (SSP) that makes such configuration easier or even removes the whole step. Specifically, it provides four pairing mechanisms: *Just Works*, *Numeric Comparison*, *Passkey Entry* and *Out of Band* (OOB). Among them, Numeric Comparison requires the user to compare two numbers displayed on two devices and confirm that they are identical and Passkey Entry is the legacy approach, asking the user to type a 6-digit PIN. Just Works and OOB, however, are ZeroConf approaches: the former enables a central to directly pair with a peripheral (assuming that the UUID is already known) and the latter requires additional information from different out-of-band mechanisms: e.g., authentication through near-field communication (NFC) or Apple's certificates. Apple's Core Bluetooth framework further hides the details of the pairing from both users and developers, which by default takes care of the whole pairing process easing both usage and development burdens (section III-A).

IP service discovery. A prominent example of ZeroConf on the IP network is Apple's Bonjour, which utilizes mDNS to publish and discover services in local-area network and Wi-Fi direct network. As mentioned earlier, mDNS updates the host-name/IP mapping through broadcast. The design of Bonjour further enables automatic assignment of IP addresses and host names, and direct service access through a service name: all the user needs to do is to choose from a list the service she wants to use.

Like other Internet services, a Bonjour service carries a name in the format of _ServiceType._TransportProtocol._DomainName, where _ServiceType is an identifier for the service type such as _airdrop, _TransportProtocol is the transport protocol like _tcp and _DomainName indicates the domain in which the service is provided, which is often a host name and when it is set to local, the host needs to be found through a broadcast across the local network. Such a broadcast is also used to register a service and create mDNS records. Further, a service instance (e.g., the airdrop service launched by the user) also has an instance name, with a unique identifier preceding the whole service name, e.g., 9c5e3d2._airdrop._tcp.local.

Bonjour publishes services through mDNS record packages. To register a service, three mDNS records need to be announced: a pointer (PTR) record, a service (SRV) record, and a text (TXT) record. The PTR record includes the name of a specific service instance, e.g., 9c5e3d2._airdrop._tcp.local, which enables service discovery from other devices to map the service type to instance names. The SRV record further maps the instance name to the information the client needs to actually use the service, including the service's host name and port. The TXT record contains some additional information about the service instance, which could be left empty. When publishing (registering) a service instance, a host should announce at least a PTR record (from the service type to its instance name) and a SRV record (from the instance name to the host name).

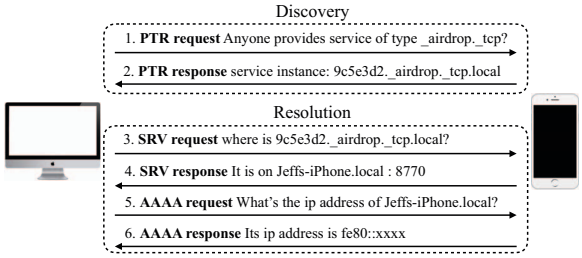


Figure 1: Bonjour Service Discovery and Host Resolution

To discover a service of interest, a Bonjour client first broadcasts PTR requests to look for a service type, e.g.,_airdrop._tcp. The service instances providing this service type then responds with their instance names (through PTR response). These names are further resolved by the client to the IP address of the host running the instance (*instance server*): more specifically, it sends out an SRV request and the instance server responds with an SRV response that provides its host name and port number; the host name is further resolved to the server's IPv4 or IPv6 address through A or AAAA mDNS queries. This resolution process takes places each time the instance name is used to find the service's current address and port number. Apple recommends that the service instance name discovered (e.g., "HP Printer [928FE5]" of a HP printer) is saved, since it is relatively stable, unlike host names, IP addresses, etc. that change frequently. The process of Bonjour service discovery is illustrated in Figure 1.

Adversary model. We assume that the adversary has already infected a device with malware, in an attempt to utilize the device to collect sensitive information from other uninfected devices. Such an adversary could not only listen on the communication channel (e.g., BLE, local-area network, Wi-Fi direct), but also actively send out messages to impersonate a legitimate and uninfected device. In Section III, we demonstrate that such an adversary is capable of performing a MitM attack, intercepting data transferred between other uninfected devices nearby, though the infected device is not the right recipient of data. On the other hand, we do not consider a targeted attack on the owner of an uninfected device, in which the adversary studies the owner's behavior and background, and can even utilize social engineering to collect information about her (e.g., cheating her into speaking out specific words).

III. Understanding Apple ZeroConf

In our research, we conducted a security analysis on popular ZeroConf Apple services and apps, in an attempt to understand whether they are properly protected, and if not, what technical hurdle needs to overcome to put the protection in place. The study reveals that most Apple ZeroConf systems, including Handoff, printer discovery, AirDrop and high-profile apps, are unguarded, subject to various MitM or data-stealing attacks. Note that the discoveries were made through an in-depth analysis on how those systems work, which typically has not been reported by Apple and corresponding app developers. Such technical details were uncovered in our research through inspecting communication traffic and binary code of the apps or system libraries involved.

A. Breaking Bluetooth ZeroConf

Apple's Core Bluetooth framework allows iOS and Mac apps to automatically discover and pair with other BLE devices. This framework is an abstraction of the Bluetooth 4.0 specification which hides BLE low-level details from developers, e.g., which pairing mode to choose, easing the development process. Its default pairing mechanism (an abstraction of `Just Works`) is also designed to reduce users' burden, avoiding the step of entering PINs on different devices. Hence, the framework has been adopted by many popular apps and services to improve their usability. However, our study shows that its service discovery and pairing mechanisms are often problematic, making many Apple apps vulnerable to our MitM attacks (Section III-D). Here we elaborate two examples.

Insecure pairing. We found that the default mode of Apple's Core Bluetooth framework (working through `Just Works`) does not authenticate the central (the client) and the peripheral (the server) on the link layer, and therefore requires application-layer authentication. However, such protection, as revealed in our research, typically has not been integrated

into apps, leaving them mostly unguarded. Also, the usability-oriented design of the framework also allows automatic connection to a UUID *without reporting any conflict* (the same service UUID claimed by multiple devices), further making an attack on the channel hard to detect. As an example, let us look at Scribe [20], a free app that transfers a copied item from Mac to iPhone, which we found is vulnerable to a MitM attack.

During the data transfer, the Scribe app on iOS acts as a peripheral (the server) and the one on OS X is a BLE central (the client). What the adversary wants to do is to become a central for the iOS app and a peripheral for the OS X app. This is actually a little bit tricker than it appears to be, depending on which app runs first. Specifically, if the user activates the OS X Scribe first, the attack app can open a service with the iOS peripheral's service UUID, cheating the Mac app into connecting to it before the real iOS service is announced. Once the service shows up, the attacker can connect it to become the man in the middle. In our research, we found that this attack always succeeds when legitimate apps are launched in that order.

When the user runs the iOS Scribe peripheral before the OS X central, things become a bit complicated. In this case, the attacker can still connect to the peripheral as a client. However, when the legitimate central on the Mac tries to discover services, it will find two peripherals with the same service UUIDs. Once this happens, the BLE client (the central) randomly picks up one of the two services to connect. Therefore, the MitM attack stands a 50% of chance to succeed. Also interestingly, the name of the central as shown on the peripheral is given by the client device, which the attacker can take advantage of to mislead the user with the legitimate device's name.

Attacking Handoff. Unlike `Just Works`, the Out-Of-Band mechanism allows ZeroConf devices to authenticate each other over the BLE channel. A prominent example is Apple Handoff, a service that lets iOS and OS X synchronize data through BLE without configuration. Pairing between the devices happens through OOB: when the user logs in her iCloud account on her Mac and iPhone, the UUIDs of the devices and credentials are exchanged through her account, to ensure that only authorized devices are paired.

The problem is that data synchronization should only happen between specific peripheral/central apps, while the Apple's ZeroConf design does not provide authentication at the app level. As a result, any advertised BLE service on the iPhone is completely exposed to any BLE capable app on the Mac. Specifically, in our research, we successfully exploited Apple Notification Center Service on the iPhone using a sandboxed Mac app. In the attack, as soon as a Bluetooth connection is established between the Mac and the iPhone (which happens when the user launches a Handoff process, with the Handoff setting on and her iCloud account logged-in), the attack app calls `discoverServices:` to discover the advertised ANCS service on the phone and `discoverCharacteristics:forService:` to find out its characteristics, particularly its notification source and data source. By registering with them, the attacker is informed whenever a notification comes into the iPhone from the former and then acquires the notification from the latter. In this way, we found that the sandboxed app, with only the Bluetooth entitlement, stole all notifications from the iPhone, including SMS, emails, Instant Messages and others. Our attack app was successfully published on Apple's Mac App Store. A demo is online [22].

Our attack was implemented on iOS 8.3, OS X 10.10.3 and 10.10.4, the most up-to-date versions when we discovered the problem. After reporting to Apple our findings, they decided to discontinue the support for transferring iOS notifications to Mac OS in the later version (posterior to 10.10.4), only allowing the data synchronization between iPhone and Apple Watch. We further discovered that other third-party cross platform services over BLE tend to have the same problem. Particularly, Pushbullet, a popular cross-device synchronization app, was found to be equally vulnerable to the attack.

B. Exploiting File-Sharing Apps

An important support provided by ZeroConf techniques is to enable file sharing between devices (e.g., Macbook and iPhone) across an ad-hoc network (local WiFi network or peer-to-peer WiFi direct connections), when the Internet is not available or considered to be less economic for the amount of data to be transferred. Such a capability is offered by popular apps such as Tencent QQ [18], FileDrop [6], etc. and Apple system services like AirDrop. These apps or services perform automatic service advertisement, discovery and target host resolution, completely eliminating the burden of manual configuration. However, just like the ZeroConf systems on BLE, such file-sharing techniques are inadequately protected and difficult to be secured. Below, we elaborate our study on popular ZeroConf apps. Our findings about the AirDrop service is reported in Section III-C.

Exploiting MC Framework and QQ. Tencent QQ is an extremely popular instant messaging app, with 829 million active accounts [23]. Its iOS version features a capability that allows an Apple device to share files with other Apple devices nearby. This capability is built on top of an Apple ZeroConf framework called *Multipeer Connectivity* (*MC*) [2], which wraps Bonjour with additional supports for service discovery and file transfer between devices across infrastructure Wi-Fi networks, peer-to-peer Wi-Fi, and Bluetooth personal area networks. Note that Bonjour is not a secure ZeroConf mechanism, which we elaborate in Section III-C. Here our focus is on the security issues within the MC wrapper and the app using the framework.

Apps built on top of the MC framework uses the framework to discover services and establish connections. The technical

details about how the framework works, however, have never been public by Apple. In our research, we looked into the framework through analyzing the binary code of its libraries (`MultipeerConnectivity.framework`) and the traffic traces. Here is what we found. In the case of file sharing, typically the receiver device runs an MC interface `MCNearbyServiceAdvertiser` to advertise its `peerID` and other information, which is picked up by the sender running another MC interface `MCBrowserViewController`. The `peerID`, acting as an identifier of an app using this framework, is an object with a few properties, e.g., a public `displayName` (the user's customized name in the app) and a private `uniqueID` (a random string). The private property `uniqueID`, generated by MC, is transparent to developers. The sender's interface `MCBrowserViewController` provided by MC, displays `displayName` of all discovered peerIDs for the sender user to select from. What we found from the MC libraries is that for each new receiver (i.e., the peer) discovered, the MC interface (`MCBrowserViewController`) checks whether its `peerID` has been seen before, i.e., whether its `uniqueID` matches that of a known peer. If it is matched, the MC interface thinks the peer is an existing one. However, it still updates the peer's IP address it saved, because it thinks the peer might legitimately change its IP.

In legitimate scenario, even if different `peerID`s are created with a same `displayName`, their `uniqueID`s are different. The problem here is that an attack device can also browse and acquire the advertised `peerID` (of a victim receiver) and its integral properties, and then launches a service using exactly the same `peerID` object, to impersonate the receiver to the sender. Further, the MC interface (`MCBrowserViewController`) on the sender side considers the discovered `peerID` from the attacker as an update to the existing `peerID` from the victim receiver. Consequently, it will map this `peerID` to the attacker's IP address. This enables a successful MitM attack in our study. We reported the problem to Apple and are working with them to address it.

In the case of QQ, our analysis of the app shows that it takes a different approach to support service discovery: instead of advertising `peerID`, a QQ receiver announces its service type `qq-qlink` through `MCNearbyServiceAdvertiser`, together with a `discoveryInfo` parameter. Within `discoveryInfo` is one's QQ ID, which is unique across all QQ users. On the sender side, an API `[MCNearbyServiceBrowser startBrowsingForPeers]` is invoked to discover all the devices with the type `qq-qlink`, from which their individual QQ IDs are extracted from `discoveryInfo` and used to retrieve the avatar (a graphic profile) of the user on the receiver side. All such avatars are then displayed to the owner of the sender for choosing the right party to communicate with. Behind the scene, the host and IP resolution is handled by the MC framework: when browsing the `qq-qlink` service, the sender automatically records the receiver's host name, port and resolves its IP automatically, a process transparent to both the app developers and users.

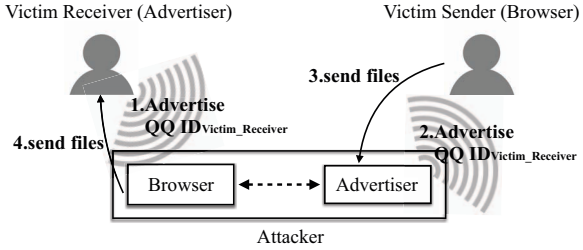


Figure 2: Attack on QQ

This treatment turns out to be equally problematic. Specifically, here the QQ ID serves as a unique identifier for the sender to find the right receiver. However, there is no protection in place to bind the ID to the receiver or the sender to ensure that the person there is indeed the owner of the ID. In our research, we implemented an attack, as illustrated in Figure 2, in which an attack device advertises the receiver's QQ ID and waits for the connection from the sender. Our research shows that in the presence of two receivers with the same QQ ID, the sender randomly chooses one to connect. Therefore, with a 50% of chance, the attacker can successfully impersonate the legitimate receiver to the sender. Once this happens, the attacker can directly connect to the receiver as the sender and become a man in the middle to intercept the file transferred between the two parties. A demo of the attack is posted online [22]. Our research further shows that many other apps utilizing the MC framework are also unprotected (Section III-D).

We performed the attack on iOS 8.4 and QQ v5.4.0.454, the latest versions when we reported the problem to Tencent in March, 2015. They acknowledged the seriousness of the problem we discovered. With our help, they fixed the problem in the latest version of QQ on iOS, in which they now require the receiver to scan a QR code from sender's screen, a shared secret before actual file transfer begins. This essentially rolls back from the ZeroConf feature of the app and resorts to exchange of secret, an approach considered to be inconvenient by mobile users [40], [42].

Breaking Filedrop. Unlike QQ that takes advantage of Apple's ZeroConf framework, some apps implement their own ZeroConf capabilities, which become necessary when file transfer needs to happen across platforms and therefore cannot solely rely on Apple's service. A prominent example is *Filedrop*, a popular paid app designed to quickly share documents between iOS, Mac, Android and Windows devices in a Wi-Fi ad-hoc network. Filedrop has its own service publishing, discovery and target resolving and other mechanisms expected from a ZeroConf system. On top of that, it provides cryptographic protection for the file-transfer process. However, our research shows that despite such effort, the app is still vulnerable to a MiTM attack, which highlights the

challenge in protecting an automatic, self-configured service in the absence of pre-shared secret.

Again, there is no publicly available information about how the app works. We studied its operations by analyzing its binary code and traffic traces. What we found is summarized as follows. Our analysis shows that all the Filedrop apps within the Wi-Fi transmission range form an ad-hoc peer-to-peer network and discover each other and exchange files through a public-key based secure channel. Specifically, each Filedrop app (Filedrop$_A$) probes all IP addresses within the local network using TCP requests directed toward port 52734, on which the app is listening. Once connected to another app (Filedrop$_B$), the sender transmits its randomly-generated ID device$_{id_a}$, user-defined name device$_{name_a}$, an Elliptic Curve Cryptography (ECC) public key (Key$_{pub_a}$) and a set of other meta data. The recipient Filedrop$_B$ then responds with its own set of parameters, device$_{id_b}$, device$_{name_b}$ and Key$_{pub_b}$. After that they generate a pair of secrets r$_a$ and r$_b$ independently and exchange the secrets using the other party's public key. Using the secrets, both Filedrop apps come up with a common session key Key$_{session}$. Then they display each other's device name in the list of available devices nearby. Once the user of one device selects the other to drop a file, a secure channel is then established using Key$_{session}$ for transferring the file. This process (service discovery, address resolving and secure channel establishment) is completely automatic, without any pre-configuration requirement.

With all such protection in place, the mechanism, however, is still weak. Fundamentally, it is the user who decides on which device to connect, based upon its name device$_{name_b}$. The problem is that an attack device can easily take the name of any other device without being noticed. Specifically, in the attack, a malicious device runs Filedrop$_{mal}$ using the target device's ID device$_{id_b}$ and name device$_{name_b}$. Our analysis shows that once Filedrop$_{mal}$ probes Filedrop$_A$ before Filedrop$_B$ does, Filedrop$_A$ keeps record of device$_{id_b}$ and device$_{name_b}$ and will not connect to any other device with the same ID and name. As a result, Filedrop$_B$ cannot talk to Filedrop$_A$, and whenever device$_{name_b}$ is selected by the user, always Filedrop$_{mal}$ will be connected.

As we can see here, the problem comes from the lack of binding between the public key and its device or the device owner. Apparently, this can be addressed using a public key certificate. It turns out, however, that certifying a public key for a device or a user is complicated: every piece of identifiable information can be changed, including the device name, for resolving conflicts on the fly, and the user's identity information may not be known to the party she wants to communicate with. As a result, it becomes extremely difficult to have a trusted third party issue a certificate to tie a public key to any device related information or the user. In Section III-C we show that AirDrop indeed utilizes Apple's public key infrastructure but still fails to protect the service from a MitM attack.

C. Cracking Bonjour Protection

As mentioned earlier, Bonjour is a major ZeroConf mechanism developed by Apple. It supports automatic service discovery and host-name/IP resolution (Section II): for a short summary, in the discovery step, the Bonjour client broadcasts an mDNS query of type PTR to discover services of specific types, e.g., _printer._tcp and the server (e.g., a HP printer) responds with a service instance name such as "HP Printer [928FE5]"; in the resolution step, the client broadcasts mDNS queries of type SRV and A with the instance name and the server replies with the host name, e.g., LaserJet.local, and its IP address.

A problem for this fully automated mechanism is that again, little protection is in place to ensure that parties involved properly authenticate each other. With this weakness, the mechanism is still used in a not-fully-trusted environment, in the absence of additional security measures. Actually, even when people want to protect it, authentication on top of Bonjour, without pre-configuring shared secret, is hard, as we found in our research. Here, we elaborate our findings through two examples, which attack popular Bonjour-capable systems: automatic printer discovery and AirDrop.

Misleading printer discovery. Today, all major printer vendors support Bonjour-based automatic printer discovery. More specifically, whenever the user searches her local network for printers, her Mac runs Bonjour to find printer service instances and let the user choose. A selected printer has its service instance name (e.g., "HP Printer [928FE5]") saved on the Mac, which enables the user to access the printer without going through the service discovery step again. On the other hand, each time the user prints through the service instance name, the target printer's host name and IP address need to be resolved, using the printer's service instance name. In our research, we confirmed that this process can be manipulated to steal the document the user intends to print out.

The attack happens when a malicious host (e.g., a compromised Mac) in the network broadcasts an mDNS response with an existing printer's instance name (e.g., "HP Printer [928FE5]"*). Note that such a response can be completely fake, not responding to any mDNS query. Nevertheless, each device (e.g., printer) observes the response automatically caches it (the mapping between a service type to a service instance) and when a conflict is discovered (i.e., the recipient device finds that the response carries its own instance name), the receiver automatically resolves the conflict by changing its own instance name (e.g., to "HP Printer [928FE5] (2)"). The problem is that the Mac keeping the printer's instance name does not know about that. When the Mac uses the printer, the mDNS request sent out for resolving the printer's host name and IP will *not* be responded by the printer, since the instance name on the request no longer belongs to it. The

*Typically, the service instance name includes a random string.

malicious host, however, will reply with its IP. As a result, the user's document will be sent to the malicious host, which can forward the document to the original printer, silently serving as a man in the middle.

We implemented the attack on a real-world organizational network, using a MacBook Pro (2.6 GHz Intel i5, 8 GB memory, OS X 10.10.4) as the attack device. Our approach successfully intercepted documents to be printed out on the target printer. Note that the problem is not limited to printer discovery: actually most apps and systems utilize Bonjour do not have protection at all and therefore are equally vulnerable to such a MitM attack. An example is the popular PhotoSync app, whose communication between a Mac and an iPhone (for synchronizing photos) is exploited by our MitM attack for stealing the photos exchanged across the devices.

Hacking AirDrop. A unique feature of Bonjour is that all the identifiers of a device using the mechanism, including its service instance name, host name, IP address, are generated dynamically and can be changed at any time. This feature enables automatic configuration of an ad-hoc network through which devices easily discovers each other and establishes communication channels among them. On the other hand, it also makes authentication of the devices involved difficult. A prominent example here is Apple AirDrop, an ad-hoc service that supports short-range exchange of documents between OS X and iOS devices. The service is built on top of Bonjour, enhancing the ZeroConf mechanism with TLS-based security protection. Below, we introduce how the service works.

We revealed the whole AirDrop process through reverse engineering and inspecting the binary code of `sharingd` (at /usr/libexec/sharingd), a system component for Airdrop. It turns out that, after the Bonjour discovery and resolving steps (using a PTR for the service type _airdrop._tcp.local), the AirDrop sender running on iOS or OS X discovers the service instance name, IP and port of another device supporting AirDrop (the server). Then, the sender establishes a TLS connection with the server to collect its device name (a name for the user to recognize the party she wants to talk to, such as `Jeff's iPhone`), Apple account information, etc. The name and the information are used to build up a list of discovered devices from which the user chooses one to drop documents. Such a file transfer happens through an HTTPS, with the AirDrop client (i.e., sender) connecting to the server through the URLs such as https://Jeffs-iPhone.local/Ask and https://Jeffs-iPhone.local/Upload, where Jeffs-iPhone.local is the server's host name. Once the file transfer is complete, the server sends back an HTTP status code 200 to confirm the success of the transfer.

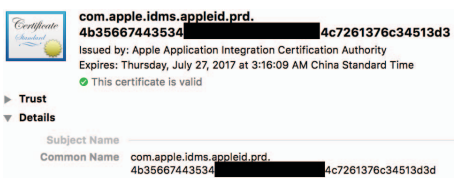


Figure 3: Apple certificate issued to an Apple account

With the TLS protection, it is less clear how the sender verifies the server's TLS certificate, which belongs to the device owner's Apple account (Apple ID). Since none of the server device's identifiers (service instance name, device name, IP, etc.) is meant for a long-term use, they can be changed on the fly and therefore cannot be bound to the user's TLS certificate. Unlike a website, whose certificate uses the site's host name (e.g., apple.com) as its *common name* that needs to be checked during an HTTPS connection, the Apple account of one individual does not have such identity information that other people can easily verify. Actually, as we find out, what is bound to one's Apple certificate (used for the TLS connection) is actually a random string prefixed with com.apple.idms.appleid.prd (Figure 3), which is supposed to be related to her Apple ID. This random string is hard to use by others for a manual check on whether it indeed belongs to her.

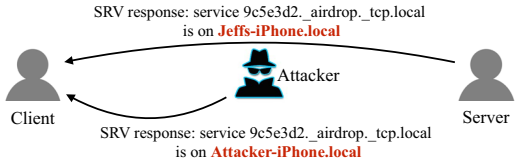
Figure 4: The process of checking contacts in Airdrop

Fundamentally, linking a human to her certificate is complicated, due to the challenge in finding any identifiable information both well-known (no privacy implication) and unique: e.g., name can be duplicate; date of birth, social security number are confidential, which people may not share lightly with the party they just want to drop a file. What Apple does, as we discovered in our research, is to bind an individual's Apple ID (denoted by an email address) to the aforementioned random string in her certificate. However, this binding relation is not public by Apple's design. Specifically, through the TLS connection, the server sends to the client the hash value of its owner's Apple ID (the email address), which is signed by Apple together with the random string on her certificate. The hash value is utilized by the client to search her contacts: only if it matches a saved email address, the client identifies who the TLS server indeed is, as the saved picture of the contact is then displayed to the user. Otherwise (the email hash is unknown to the client), the aforementioned binding relation is unknown to the client: the displayed device name of the server is arbitrarily claimed by the server in the TLS communication. This process is illustrated in Figure 4.

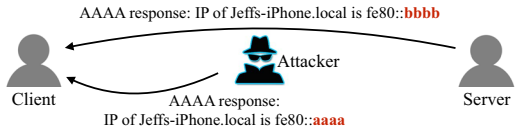
In our research, we reveal that Apple's design is not secure in practice: oftentimes, Apple users don't save known people's Apple IDs into contacts. Specifically, people might only save others' phone numbers, not email addresses. Also, one person tends to have multiple emails. The chance is that many of her contacts do not share with her their Apple IDs

(email addresses). For example, what you know about your colleagues are most likely their working email addresses. Indeed, in our measurement study, we checked all 1,230 contacts saved on 9 individuals' iPhones. It turns out that only 119 contacts (9.7%) out of 1,230 are saved with their Apple IDs. Specifically, 240 contacts (19.5%) out of 1,230 are saved with their email addresses (the remaining 1,007 contacts, i.e., 81.9%, only have phone numbers or personal webpage links) and more than half of these saved email addresses are not Apple IDs. Actually, Apple apparently treats the Apple ID as private information and only sends out its hash value for certificate check. Further, the approach does not work on those still not included on one's contact list, who AirDrop is also supposed to serve.

Given the fact that highly likely this identity check (linking one's certificate to her identity known to the user initiating the AirDrop process) fails in practice, Apple still shows to the user the list of device names, even when the certificates involved cannot be bound to any known contacts through the Apple IDs (email addresses). Once the user chooses a device (through the device's name like `Jeff's iPhone`), her documents will be transferred through the AirDrop mechanism, even when the validity of the server's certificate cannot be fully verified.



(a). Attack by faking SRV response



(b). Attack by faking AAAA response

Figure 5: Attack on Airdrop

Exploiting this weakness, we successfully attacked AirDrop in our research. Specifically, the attack happens when the attack device sends an mDNS packet of type SRV to bind the service instance name of the real AirDrop server to its own host name (Figure 5(a)). Note that this mDNS response can be unicast to the victim, i.e., the AirDrop client (the party that initiates the AirDrop communication), to avoid being found out by the server. After that, the TLS connection (towards the server's instance name) initiated by the client will go to the attack device's host name. Alternatively, the attacker can send to the AirDrop client an mDNS packet of type AAAA, binding the server's host name to the attacker's IP address (Figure 5(b)). Again, this packet can be delivered through a unicast channel, without exposing to the server. This, again, will cause the TLS request from the client to go to the attack device. In either case, the attack device finally impersonates the server to the client, and then connects to the server to act as a man in the middle. Since the user during the process has no way to find out whether she is talking to the right person, she may choose the wrong device on the list and send her documents to the attacker. A demo is online at [22]. The problem has been recognized by Apple, which is working with us to find a solution.

D. Measurement

To find out the scope and magnitude of the security weaknesses in ZeroConf systems, we performed a measurement study, analyzing 61 popular Mac and iOS apps designed to operate without configurations. Our findings demonstrate the significance of the issue: a vast majority (88.5%) of the apps we analyzed turn out to be unprotected at all, even though the environment they work in cannot be fully trusted. Examples of such apps and our findings are summarized in Table I. The complete list is in Table II.

ZeroConf Channels	Vulnerable/ Sampled	Sensitive Information Leaked	App Examples
BLE	10/13	user name and password for Mac OS X	Near Lock
MC	24/24	files and photos transferred, instant message	Bluetooth U, Photo Transfer, Airdates
Bonjour	18/22	files, directories and clipboard synced, documents printed, instant message	Copybin, Printer Pro Lite
Homegrown	2/2	remote keyboard input and files transferred	Remote Mouse, SHAREit

Table I: Summary of vulnerable Apps

ZeroConf Channel	App
BLE	Near Lock, Tether, MacID, Proximity Lock, Bluetooth Lock, Bluetooth Lock Pro, Knock, Bluelock, BT Msgr, Sochat
MC	BluetoothPhotoShareExpert, Photo Transfer LITE, Beam It!, Bluetooth U, Bluetooth Transfer Free, Bluetooth Transfer, Multipeer, FileTransfer iFamily, ZombieChat,SocialCard, Wave Off The Grid Chat, HyperConnect, Meshwork, chatty, WVLT, PeerTy, TABI, PubChat, BluetoothVideoTransfer, nocknock, LocalTalk, Tabitop, AirDates, ChatUp, Probity
Bonjour	Photo Transfer App, Flowr-Photo Journal, RemoteSnap, PDF Printer Lite, ClipAgent, Flick, Sync Photos to Storage, DropCopy, Pasteasy, SyncBook, Mobile Mouse Server, Print Pro Lite, Air Printer Lite, Remote App Launcher, Clippy, Air Media Server, Schick, AirBeam, ShutterSnitch, AirBridge, Photo Sync, copybin
Homegrown	Remote Mouse, SHAREit

Table II: Full List of Vulnerable Apps that We Found

Settings. To understand the pervasiveness of the security weaknesses in ZeroConf apps and their implications, we searched on Apple's Mac and iOS App Stores for the apps using the aforementioned ZeroConf channels, e.g., Bluetooth low energy (BLE), Multipeer Connectivity (MC), Bonjour and homegrown ZeroConf. The search was based on a set of keywords, as illustrated below. On the apps discovered, we ran Hopper [8] to dissemble those with OS X into Intel x64 instructions and those with iOS into ARMv7 instructions. Note that before the dissembling operations, the iOS apps were first decrypted using Clutch [4]. From the sections _objc_methname and _objc_selrefs within the dissembled code, we utilized a python script to find out

whether channel-related APIs are called in the code. The apps not including the APIs were filter out, since apparently they are not implemented with any ZeroConf mechanisms. From what remained, we randomly sampled 10 to 20 apps for each ZeroConf channel and inspected their code statically and dynamically to find out whether they are vulnerable to the impersonation and MitM attacks. For those confirmed to be vulnerable (without proper authentication of the peers the device talks to), shown in Table II, we further studied what sensitive information could be leaked to the adversary capable of exploiting the aforementioned vulnerabilities.

BLE. For apps using the BLE channel, we searched on the Apple App Store using keywords *Bluetooth, BLE, Bluetooth low energy*, etc. From all the apps returned, those not calling the BLE API `scanForPeripheralsWithServices:options:` and `startAdvertising:` were automatically removed. Among what were left, we randomly selected 13 apps and confirmed that 10 of them are vulnerable to our MitM attack (see Table II). The other 3 apps require pre-shared secrets, e.g., a shared bar-code or a piece of code distributed by a remote server, which need to be jointly configured by the parties involved in the communication.

We further looked into what sensitive information these 10 vulnerable apps could disclose to the adversary. It turns out that some of them (e.g., BT Msgr, Sochat) use the BLE channel for instance messaging (IM) with other devices nearby, which completely avoids the need for Internet connection and accounts registration. However, such apps are subject to our MitM attacks (see Section III-A) due to their lack of authentication over BLE: we found that our attack device easily intercepted the messages exchanged between these devices by claiming their device names or service instance names.

As another example, a popular app `Near Lock` on both iOS and Mac leverages a user's iPhone to automatically lock and unlock her Mac. Specifically, when the user (with her iPhone) walks away from the Mac, the Mac app locks OS X automatically. Once she comes back, the app automatically unlocks her OS X. This happens because the iOS app transfers the user's login credentials (user name and password) to its OS X counterpart through BLE. Since this ZeroConf channel does not properly authenticate the parties connecting to the device and the laptop, the app is completely vulnerable to the MitM attack similar to what is described in Section III-A. Actually, `Near Lock` already puts some protection in place, encrypting the data transferred. However, the secret key is hard-coded within its binary and was recovered in our analysis. Through our MitM attack, we were able to obtain the user name and the password for the app user's Macbook.

MC. Also, we analyzed the Apple apps discovered using the key words *Multipeer, Multipeer Connectivity* etc. Again, only those containing specific APIs, such as `initWithPeer:discoveryInfo:serviceType:`, `initWithPeer:serviceType:`, `initWithServiceType:discoveryInfo:session:` were selected. From the apps, we further randomly sampled 24 in two sub-categories, those directly using the `peerID` provided by the MC framework (Section III-B) to identify peers and those using their own identifiers, like QQ (Section III-B). We manually confirmed that all these 24 apps are vulnerable.

From all the apps we studied, we found that the `MC` framework is frequently utilized to quickly transfer photos and files to peer devices nearby. This design takes advantage of the high bandwidth of local networks, without resorting to the relatively slow Internet connection. A prominent example is the popular app `Bluetooth U`, which transfers files between the iOS devices standing close to each other. The app implements a callback of MC, `session:didReceiveCertificate:fromPeer:certificateHandler:`, which receives a certificate from its peer before establishing a secure channel between them. It turned out that, however, just like the certificate dilemma in AirDrop (Section III-C), without a pre-configuration to bind the peer device to its certificate, the app cannot properly verify the certificate, linking it to its owner. This allows an MitM attack to succeed, leaking out all sensitive files transferred to a man in the middle. Some other apps use `MC` to deliver one's photos to the peer devices, e.g., `Photo Transfer`, which were found to be equally vulnerable to our MitM attacks, disclosing all photos to the adversary.

Interestingly, we also find that `MC` is taken advantage of by dating apps for instant messaging with people nearby even when Internet access is not available, e.g., waiting to get on board or even in flight. `Airdates` is such an example, which is designed to work even in Flight Mode. This app utilizes `MC` to automatically find peers devices nearby, which allows the user to continue his/her chat privately with people on the same aircraft even after boarding and in flight. With the MC vulnerability (Section III-B), the MitM attacker can intercept any message transferred between victims.

Bonjour. We further studied Bonjour-based ZeroConf apps, which were collected from the App Store using the keywords like *photo transfer*, *file transfer*, *server, bonjour*, *printer*, *remote control*, and *local media*, etc. From the collection, we randomly chose 22 of them, among which 18 were manually confirmed to be vulnerable. These apps are not only used for local transfer of files and photos, but also designed to share clipboard across peer devices, print from one device to another, and even turn one device into a remote for its other devices (see Table I). Those we still cannot exploit utilize pre-shared secret (e.g., manually entering a PIN code) to authenticate the peer devices during the communication and therefore require joint effort from the peers to configure their systems.

An example of the vulnerable Bonjour apps is `Copybin`, a popular program that synchronizes data (files, directories and clipboard) across multiple devices. When only two devices running the apps within the same network, they connect to each other automatically. If there are multiple ones, the user needs to choose from a list of device names the party she wants to talk to. The problem of the app is that it does not authenticate the peer device when connecting to it. This allows impersonation and MitM attacks to succeed. Another example is `Printer Pro Lite`, an app that enables the user to print documents from her iPhone to her Mac. Again, we found that the communication between devices is not authenticated when using the app and therefore can be exploited to steal the document transferred between them.

Homegrown ZeroConf. In addition to those running existing ZeroConf mechanisms, there are apps that communicate through their own configuration-free techniques. To find out these apps, we retrieved from the App Stores a set of apps using the aforementioned keywords and filtered those using known ZeroConf channels, e.g. Bonjour. We manually execute such apps to find out whether they are trying to automatically discover peer devices in the neighborhood. If so, we collected their traffic using WireShark and inspected their binary code. Every possible problem discovered from the traffic and binary code was further evaluated by running our attack device to perform an impersonation or MitM attack. Here we elaborate two popular apps that are problematic.

The first example in this category is `Remote Mouse`, which is the 13rd most popular app in Utilities of the Mac App Store. Once installed on the Mac OS, the app allows the user to control the Mac OS through her iPhone using the iOS version of this app (e.g., remote keyboard input, mouse control, shutdown/restart/sleep/logoff or app launch). More specifically, we found that the server running on the OS X automatically broadcasts UDP packets to the local network using destination port 2007 and 2008. The data field of the UDP packets are the device name and IP address, etc. After receiving the broadcasted UDP packet, the `Remote Mouse` client on the phone allows the user to choose a target server and establish TCP connection with the Mac using the received information, such as IP, device name, etc. This app, However, turns out to have a similar problem as Filedrop (Section III-B) whose device name could be mimicked. Consequently, an MitM adversary exploiting the problem could steal sensitive information transferred from iOS to Mac, such as remote keyboard input, which could be passwords or any other sensitive data.

As another example, an app called `SHAREit` by Lenovo, the world's largest PC vendor [11], also built its own ZeroConf mechanism. This is a cross-platform app on iOS, Android, Windows, with over 50 million downloads, which is designed to transfer files to peer devices in the same local area network. When operating in "receiver" mode, it sends UDP packets to multicast address 255.255.255.255, including a "from" field (used as device identifier), IP address, a "nickname" field (used to display the target name by the "sender" device). When another device running in the "sender" mode receives such a UDP multicast packet, the sender automatically sends back a unicast UDP packet to the receiver, exchanging device identifier, IP address, nickname and more. Next, the sender and receiver establish a TCP connection, through which the sender could send files. These ZeroConf steps proceeds automatically, without any pre-configurations on any of the devices. This usability oriented app, again, is susceptible to our MitM attack, when the device identifier and nickname could be hijacked. As a result, any files transferred in the local network, through wireless or wired connection, are completely exposed to the adversary.

IV. Protecting Apple ZeroConf

Our security analysis shows that there is a significant misalignment between the usability-oriented design that characterizes existing Apple ZeroConf systems, and the security threats they face in practice. Although it is tempting to think about falling back from the whole idea of ZeroConf, and instead ask the users to jointly configure their individual systems (e.g., sharing secrets) before the communication happens, such configuration steps can indeed become cumbersome and in some cases complicated. As an example, consider the situation that a different secret need to be distributed to every pair of Mac desktops within an organization for secure file transfer, or between a device and all the printers it connects to, for protecting the documents it prints. Even when it comes to the configuration between two mobile devices, a simple approach like scanning QR code was found to be inconvenient and not preferred by mobile users according to a prior survey [40], which has been further confirmed in our human subject study.

To better protect the ZeroConf system without undermining its usability, we developed a suite of novel techniques in our research. More specifically, we first consider an *optimistic* approach in which the device considers its operating environment safe if the necessary condition of an impersonation or MitM attack is not satisfied. A more generic solution is to leverage Apple's PKI to authenticate the parties involved in ZeroConf operations. Our research brings to light where the existing PKI fails and how to bridge the gap and make it work on today's ZeroConf systems.

A. Conflict Detection

A key observation from our security analysis of ZeroConf systems is that any attempt to impersonate an existing device must hijack that device's service instance name or host name, which will cause a conflict observable to the party searching for the victim (the device being impersonated) when the victim also responds to the party's request and the adversary

cannot interfere with their communication (Figure 6(a)). In practice, disrupting two parties' communication through a WiFi direct link or a local network is difficult for the adversary without access to the routing infrastructure of the network. Further, in all the scenarios we analyzed, it is reasonable to consider that each party involved in the communication knows it happening and her device is on, ready for receiving messages. Since identity hijacking is a necessary condition for the impersonation and MitM attacks and the conflict is *inevitable* assuming no disruption and the victim always on, we can conclude that a ZeroConf network is attack-free, under the assumptions, if no conflict is observed.

This observation leads to a conflict-detection design and its implementations on OS X and iOS, which defeats our attacks and also fully preserves the usability of the existing ZeroConf systems. Here we elaborate our design and implementation.

Design. In a ZeroConf network, whenever the sender broadcasts a service discovery or name resolving message, the hosts that meet the requirements set by the sender (e.g., a given type of services) respond with a set of identity-related attributes, e.g., service instance name, device name, host name, IP address, etc., notifying the sender of their existence. Also every member in the network caches the response (e.g., the IP address of a host) for the follow-up communication. In the presence of an impersonation attack, the adversary needs to either reply to the sender's request with the service name or the host name of the victim (the rightful owner of the attributes), which causes a conflict the sender sees immediately, or later communicate with the sender to update its cache. In the former case, the sender detects the conflict and decides that the current environment may not be attack-free. In the case of the cache updates (for resolving service name to host name, or host name to IP), the sender can broadcast the updates it receives and wait for a conflict complaint, which the victim is supposed to send in the presence of an attack; if no complaint arrives within a timeout window, the updates are considered legitimate.

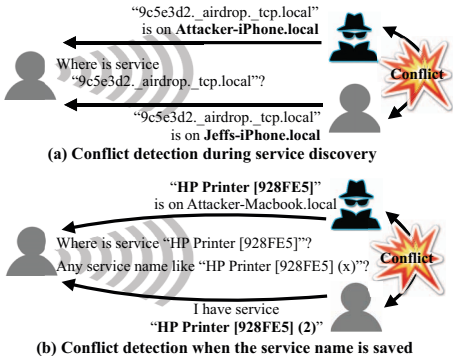


Figure 6: The Design of Conflict Detection

In practice, the situation can be a bit more complicated. For the ZeroConf system like Bonjour, the sender can save the service instance name (e.g., for printer) to skip the service discovery stage next time and directly resolve the service name to the host name and the IP address. The trouble is that such a system includes an inherent conflict resolving mechanism: the attack device can directly notify the victim (through unicast) that it is claiming the same service name, forcing the victim to switch to a different name, as described in the attack on Bonjour printer discovery (Section III-C); when this happens, the sender does not get information about this change and therefore will follow the service name to communicate with a wrong host. To address this issue, we leverage the unique way Bonjour resolves such a conflict, in which the victim automatically takes a new name with a fixed format: e.g., from "HP Printer [928FE5]" to "HP Printer [928FE5] (2)". This allows the sender to attempt to resolve both service names before the communication and detect a conflict if the second name also gets a response. Figure 6(b) illustrates the design.

Once a conflict is found, the sender needs to decide how to move forward. Depending on the specific application scenario, it can either resolve the conflict automatically or ask for manual intervention. Specifically, for AirDrop and other file transfer apps, the sender can solicit the SPYC vouches from the parties it will establish a TLS channel with. In this case, the conflicting parties' voice recordings will be both played to the user to decide who to connect (for more details, see Section IV). For printer discovery, what the sender can do is to send an error report to the system administrator, asking for an investigation on the the conflict. Note that although not all conflicts are caused by malicious acts, in practice, service instance names and host names are generated in a way that conflicts can be avoided (e.g., appending the names with random strings) and therefore innocent conflicts are rare.

Implementation. With its relatively simple design, implementation of the technique on OS X and iOS, however, is by no means trivial. Due to the closed nature of the OSes, direct changes to its system service like AirDrop is off the table. In our research, we built an app to detect the conflict on Bonjour-based systems, a first step toward securing a typical ZeroConf system. Even such a third-party solution relies on in-depth understanding about how the systems work. For instance, we found that Bonjour running on iPhone and Macbook transfers data through *Apple Wireless Direct Link (AWDL)*, a low latency and high speed WiFi peer-to-peer connection that operates on a dedicated network interface, typically `awdl0`. Apple does not let a third-party app access this "Apple-private" interface, since they are all supposed to use the common interface `en0` for wireless communication. This makes conflict detection impossible because both the attacker and the victim's mDNS traffic cannot be observed by our protection app. To find a solution, we reverse-engineered `mDNSResponder`, a dedicated Apple system process that sends and receives mDNS packets, and maintains mDNS

caches. More specifically, we ran Hopper, a disassemble tool, to dissemble `mDNSResponder`. From its binary code, our analysis shows that the program configures a listening socket through the system call `setsockopt()`. Interestingly, we found that when the call's *option_name* parameter is set to *0x1104*, a private value never made public, the socket can work on `awdl0`, which allows a third-party app that establishes the socket connection through the system process to send and receive data from this private network interface.

Access to `awdl0` is only the first step. To monitor the Bonjour traffic, we need to monitor the port 5353, where mDNS packets go. This is challenging as `mDNSResponder` has already occupied this port ever since the OS starts. On Mac OS X, our solution is to share the port 5353 with `mDNSResponder`, which we found can be done by setting the `SO_REUSEADDR` flag in the system call `setsockopt()`. In this way, a third-party native app we built (with the root privilege) successfully acquires the right to listen on the port and monitor all the Bonjour traffic. This approach, however, cannot work on iOS without jailbreaking the device to gain the root privilege. Therefore we need to find another way to implement our conflict-detection app.

On iOS, what we can leverage is an interface that allows a third-party app to query `mDNSResponder` for the mDNS traffic it processes. Specifically, an app can call `DNSServiceBrowse` to talk to `mDNSResponder` through inter-process communication. From this channel, the app can obtain such information as the service name, host and IP addresses of the party communicating with the system services such as AirDrop. A problem here is that the mechanism is *not* designed for conflict detection: when more than one device claim the same service name or host name, only one will be returned by `mDNSResponder`. This makes sense when considering the nature of ZeroConf, which allows a device to change its host name, IP, etc. on the fly (e.g., a user may alter her device name in the `Settings` of her device). However, the treatment also denies our protection app any chance to find conflicts. Our solution is to discover such service parameters through a different channel and then compare the findings with what is reported by `mDNSResponder`. Specifically, consider that all the devices in the communication run our protection app. Whenever the user invokes AirDrop, our app first broadcasts a service discovery request through a different port (10011 used in our implementation) to other devices in the ad-hoc network to collect their `(serviceName, hostName, IP)` from the protection apps listening on the port on these devices. The list of triplets received from the devices are then compared with the outcomes of the query on the AirDrop traffic managed by `mDNSResponder` to detect any conflicting identity attributes from different devices. Once the conflict is found, the system resorts to other means, such as the SPYC vouch (Section IV-B) to determine which party to communicate with.

B. Speaking out Your Certificate

The conflict detection techniques work completely automatically, fully preserving the zero-configuration property of existing systems. However, in the presence of a conflict, it cannot help the user identify the trusted party to connect to. Also a more fundamental solution against the impersonation and MitM attacks should rely on authentication of the parties involved in communication. In the absence of a shared secret (which needs to be configured across multiple devices), apparently the best solution is to leverage Apple's PKI, using each party's Apple certificate to establish a secure channel between authenticated peers. This treatment, however, turns out to be more complicated than it appears to be: in all the data-sharing cases mentioned above, only Handoff can be potentially secured by authenticating two apps (across iPhone and Mac) with their app signatures; all other cases involve the certificate of a user's Apple account. The challenge here is how to properly verify one's ownership of a certificate, which has not been addressed by existing techniques.

Personal certificate. To link a certificate to a user, we need to attach to the certificate some identifiable but nonsensitive information from the user, which also needs to be well known to her contacts. Our idea is to use her voice biometrics to tie her certificate to her identity, assuming that the party verifying the certificate knows her voice. Specifically, we developed a technique that enables the user to speak out her certificate and use the voice recording to vouch for the relation between the certificate and her identity. To verify the certificate, one first needs to check whether the voice indeed belongs to the person she knows and also whether the certificate content has been correctly spoken. The logic here is that to impersonate someone else, the attacker needs the victim's cooperation to speak out the attacker's certificate, which is not supposed to happen. As a result, the attempt to deceive other parties into using the attacker's certificate as the victim's will fail if those parties know the victim's voice.

Making the idea work in practice, however, is much more complicated: after all, we cannot ask the user to read through all the content on her certificate, which is long and contains mostly numbers, both hard to read and hard to verify (by checking whether both voice and the content read are accurate). Our solution is a new technique, SPYC (Speak out Your Certificate) vouch, which converts the certificate into a few pronounceable but rare or even fake words (for preventing a synthesis attack using the records of one's daily conversation) that uniquely summarize the content of the whole certificate. By voicing these words, one generates a vouch for her certificate, which ties the certificate to her identity, whose correctness can be conveniently verified by humans (Section V).

The use of the SPYC vouch avoids the need to jointly configure secrets between the parties in communication: that is, no longer do we need two users to work together to

configure their systems before they can securely talk to each other. On the other hand, it still needs the user to record her voice the first time when she uses the system. This can be part of the certificate exchange step and therefore does not need to be done through the configuration of the system. Also, during certificate verification, one still need to listen to the record and confirm that it is indeed from the right party who says the right words. This step is very much in line with the current use of ZeroConf systems, which often need the user's intervention to choose the peer they should contact (e.g., `Jeff's iPhone`).

SPYC vouch. Consider a user A with a certificate C issued by a certificate authority (CA), which is used to communicate with another user B. Here is how the SPYC vouch of the certificate is created and verified.

• *Certificate mapping*. The CA first computes a hash value $h = H(C, s)$, where H is a cryptographic hash function and s is a random string. The random string can be part of C (e.g., a string attached to its common name), and otherwise, it needs to be signed by the CA together with C. Then, the CA extracts σ, a string with the nk most significant bits from h, and partitions it into k n-bit segments: $\sigma = \sigma_1 \| \cdots \| \sigma_k$ (where '$\|$' is concatenation). During this process, s is selected in a way that $\sigma_1 \cdots \sigma_k$ are all different. These segments are used to look up a dictionary with 2^n words: from the locations $\sigma_1 \cdots \sigma_k$, the CA picks out k words $W = \langle w_1, \cdots, w_k \rangle$ and sends (C, s, W) and the signature on C and s to A.

• *Vouch generation*. The first time the user A sends out the certificate (e.g., through AirDrop), her device requires her to speak out all the words in W. The recording of her speech, R, together with (C, s, W), forms her SPYC vouch V_A.

• *Verification*. Before establishing a secure channel with B, A first sends to B V_A. The user B's device checks the signature on C and s, and computes $H(C, s)$ and uses the same dictionary to verify W. If correct, it displays W to B and plays R. The user B verifies that indeed it is A's voice and indeed all words $\langle w_1, \cdots, w_k \rangle$ have been spoken. Once this is confirmed, the device automatically binds C to A and stores this relation.

Intuitively, our approach maps a user's certificate to a set of words using the truncated hash value of the certificate and a *public* dictionary. Each segment is interpreted as a position to locate a word in the dictionary. Verifying the certificate requires a joint effort from the device and its human user: the former confirms that the words match the certificate's hash and the latter ensures that the words are spoken with the right person's voice. Our implementation truncates the output of SHA-256, using the first 102 bits to locate 6 words (17 bits each) from a dictionary of 222,595 words.

Center to the approach is the dictionary, which needs to be chosen carefully. It is expected to contain a large number of words that are both pronounceable and rare. The first property is for usability and the second is for security: those words should be very unlikely to be used in daily life and sound different from common words, which makes one's voice samples for the words hard to obtain. This is important for defeating a speech synthesis attack. In our research, we looked into different options, including fake word generators [34], make-up word list [49] and different kinds of dictionaries. We found that although a fake-word generator can easily produce a large number of words never used in real life, those words are often difficult to pronounce and in some cases, sound similarly. The make-up word list includes the fake words of better quality, though still not extremely easy to speak. By comparison, most of the dictionary words are easily pronounceable, even for those rarely used. In our research, we removed from Google's Trillion Word Corpus[1] 10000 most common words and the remaining words have only less than 2% of chance to be used in daily life, according to a study [43]. From these words, we further dropped 1154 words with similar pronunciations, based upon the list provided by [26], [7]. The new dictionary built in this way contains 222595 words. Table III shows examples of the words from the dictionary.

Words List from	Words Examples
Fake Word Generator	onviscei, ushfur, leontyl, dedebuturav, epreansean, stioncisi
Make-up Word List	phrintce, ghighgns, scrawgued, quawgn, snyscks, dwoarched
Rare Dictionary Words	ablegate, elvish, patronship, satisfice, tagatose, unnerved
	automorphism, choregic, miliarensis, hariolate, porterage, lagomorph
	spiritdom, mischievously, noonstead, antiquation, gigsman, lusciously

Table III: Words examples of different words lists

Security analysis. The security guarantee of the SPYC vouch is based upon the challenges the adversary faces to obtain the victim's voice for the adversary's certificate, a necessary step to impersonate the victim to other parties. This is considered to be extremely difficult. Specifically, hoping that Apple (the CA) accidentally issues two certificates with the same word sequence is unrealistic, which is on the order of 2^{-nk}, in our implementation, 2^{-102}. The main avenue left here for the adversary is synthesizing the victim's speech on the adversary's certificate words. Prior research [50] shows that the existing voice-based authentication over VoIP is vulnerable to the *voice reordering attack*, in which the adversary reorders previously eavesdropped words and sentences spoken by the victim to produce the short speech for authentication, and the *voice morphing attack*, where a few sentences spoken by the victim are used to convert another person's speech into hers. Our design ensures that the chance for a successful reordering attack is negligible. Specifically, all the words in the SPYC vouch are distinct and rare, even never used in daily life. Therefore, collecting appropriate voice samples for those words from one's daily life or public audio materials (e.g., Youtube video) is hard. If the adversary expects his certificate to contain all the words in

the victim's certificate, though in a different order, the odds he is facing is as follows: $\frac{k}{2^n} \times \frac{k-1}{2^n-1} \times \cdots \frac{1}{2^n-k+1} < \frac{k!}{(2^n-k+1)^k}$, which is well below 2^{-90} with the parameters chosen in our implementation ($n = 17$ and $k = 6$). Note that the CA is not under the adversary's control and trying out new certificates issued by Apple, through registering new accounts, needs to go through CAPTCHA and other steps and therefore cannot be done efficiently.

When it comes to the voice morphing attack, we analyzed the state-of-the-art voice transformation technique, CMU's Festvox [5], which has been used in the prior research to show the attack on voice-based authentication [50], [47]. Our study shows that unlike the authentication performed over the noisy and stream-based VoIP channel [27], the SPYC vouch is pre-recorded (once for all) and entirely delivered (through TCP) to the recipient before played, and therefore the SPYC vouch has a much better quality. On the other hand, the current voice morphing technique is still distance away from producing perfect sound, as acknowledged to us by the member on the Festvox project. Specifically, we found that even in the ideal situation, using noise-free voice samples from the victim, still the voice created sounds robotic and in the most cases is quite far away from the voice of the target individual being mimicked: in one of our human subject studies (Section V-B), all 20 participants easily identified the unique features of the synthesized speech and when comparing the speech with that from the victim, which is exactly the situation when the user has to compare two vouches to resolve a conflict, all of them chose the right one with high confidence. The findings make us believe that the threat posed by today's voice synthesis techniques to voice authentication is likely to be still limited.

On the other hand, we acknowledge that our technique is not designed to defend against a dedicated, targeted attack, in which the adversary may spend a long time to collect a large amount of the victim's voice samples or even actively lure her into uttering the words or the sound similar to what the adversary looks for.

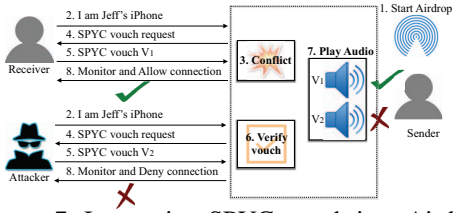


Figure 7: Integrating SPYC vouch into Airdrop

AirDrop integration. To understand whether the SPYC vouch can be easily used in a real-world ZeroConf system, we studied how to integrate the technique into AirDrop. Ideally, the integration can make SPYC part of AirDrop (Figure 7). As mentioned earlier (Section III-C), during the service discovery stage, the AirDrop sender establishes a TLS connection to every service instance discovered and then utilizes the hash value of the Apple ID email acquired from the server to locate the contact on the sender device. When this attempt fails, what we can do is to let the sender request the server's certificate vouch. The content of the vouch (delivered through the TLS channel) is first checked by the sender device to ensure the consistency between the certificate and the words. Then, the user on the sender end listens to the audio recording to authenticate the owner of the certificate. A verified certificate is then kept by the sender, together with its owner.

This design, however, cannot be built into OS X and iOS without Apple's help, because AirDrop is a system service and just like other system components on the Apple platform, it is closed source. What we implemented in our research is an integration that utilizes a third-party app (called *monitor*) to control the AirDrop communication. Specifically, on OS X, the monitor, with the native privilege, runs `netstat` to keep track of the parties the AirDrop sender is interacting with. Once the sender is found to establish a TLS connection with an IP address, the monitor immediately connects to its counterpart (another monitor app) on the service device with that IP to acquire the device user's SPYC vouch. The user on the sender side then verifies the vouch through the monitor before deciding on whether to drop a file to the server (by clicking on the service instance name displayed through the AirDrop interface). We also built an enforcement mechanism through which the monitor can configure `pfctl` (similar to iptables [10] on Linux), a firewall working on the UNIX layer of the OS X, to prevent AirDrop from communicating with other IP addresses before the current AirDrop transaction is done. Our implementation on iOS works in a similar way except that it cannot rely on `pfctl`, a component not available on iOS. Therefore, the iOS monitor only detects the potential problem (e.g., when its counterpart on the observed IP address does not exist or when the SPYC vouch is not correct) and report to the user and alert her to what can go wrong. A demo of our system is posted online [22].

V. Evaluation

In this section, we report the evaluation of our protection techniques (Section IV) through two human subject studies involving 60 participants and a performance valuation.

A. Usability And Effectiveness

The usability of our protection (Section IV) was evaluated through a user study approved by our organization's IRB. For the study, we recruited 40 participants. All of them utilized the implementation of our monitor app (Section IV-B) to authenticate his/her friend or acquaintances in the presence of an MitM adversary. Through a post-test questionnaire, all participants agreed that our monitor app is convenient and comfortable to use. Regarding effectiveness, all participants were able to easily distinguish their friend or acquaintance's voice from that of the adversary.

Recruitment. Our study (Study 1) was advertised as an investigation that "evaluates the usability of voice based authentication" (from our advertisement). The recruitment effort resulted in 40 participates from an educational institution, all coming in pairs. Each member in a pair claimed to know the other's voice since this is a requirement for participating in our study (also the assumption for the SPYC). The demographic information of the participants is presented in Table IV (under column *Study 1*). We informed the participants that this is an anonymous study that does not collect any personal or identifiable information. We offered each of them a compensation of 10 USDs.

	Study 1	**Study 2**
Number of Participants	40	20
Gender		
Male	55%	55%
Female	45%	45%
Age		
18-20		35%
21-25	62%	15%
26-30	33%	35%
31-35	5%	
50+		15%
Education		
High school diploma	3%	30%
Some college	3%	10%
Bachelor's degree	40%	
Master's degree	47%	55%
MBA	7%	
English as Primary Language		
Yes	28%	50%
No	72%	50%

Table IV: Demographic Information of Participants

Experiment. The experiment took place in two different rooms within our organization. It took each pair about 35 minutes to complete the task and the experiment was conducted one pair at a time. In the experiment, the participants were first asked to fill a pre-test questionnaire designed to collect demographic information, as well as other situations that might affect the effects of the study. For example, we asked their native languages and in what language they usually speak to their partners (the other member in the same pair). We also collected the confidence level about their familiarity with the other's voice, which was chosen from (*definitely, probably, note sure, probably not, definitely not*).

Following the questionnaire, we explained to them how our SPYC system works. We clearly told the participants that they need to evaluate the usability of the system, i.e., whether it is convenient and comfortable to use. We assigned the pair of participants different roles, one as the "sender" of a document through Airdrop, the other as the "receiver". Then the two participants were placed in different rooms, not being able to see or hear each other. They were each given an iPhone (running iOS 8.4) on which they could run our monitor app. The first time a participate launched the monitor, he/she followed the on-screen instructions to record his/her voice of the six words (in English) from our dictionary that describes a certificate (Section IV-B). They were aware that this recording only needs to be done once for all. Also nearby operated by our researcher were two devices acting as the MitM adversary. On one of the devices, an adversary already spoke out his certificate (recording 6 words). On the other, we asked the receiver to record 3,4 or 5 words they spoke (for the certificate) while the adversary spoke the remaining words (totally 6 words) which was also recorded. Till this point, the experiment setup was done and we then asked the sender to authenticate the three receivers, the real one together with the two from the adversary devices.

The sender, right before sending a file to the real receiver using AirDrop, was reported by the monitor three conflicts (the real receiver, the two adversaries, one using his/her own voices and the other using the recording of the mixture of the real receiver and the adversary's voice). He/she then listened each receiver's SPYC vouch and indicated which belongs to his/her partner through our app. We then asked the sender and the real receiver to switch roles and repeat the experiment above. This allowed each of them to evaluate the usability of the system at both the "sender" and the "receiver" roles. The pair were also asked to operate on the Mac version of our monitor app to find out its usability, since document delivery through AirDrop also happens between Mac OSes and between iOS and Mac OS.

At the end of the experiment, each participant was asked to finish a post-test questionnaire. We asked whether they thought our app was convenient to identify their partner and comfortable to use. We also asked them to compare our voice based authentication with two other authentication mechanisms. The first one is `Physical Method`, through which two participants must walk to each other, face-to-face, and share a pre-configured secret between their devices (iPhone or Macbook). We let the participants aware that the WiFi Direct (on which AirDrop is built) supported up to 200 meters in distance [9]. The second mechanism is `Out-of-Band Method`, through which the participants are supposed to share a pre-configured secret through an out-of-band channel, such as email, text messages, phone call or social networks, etc. By answering our post-test questionnaire, all participants compared our SPYC mechanism with the two alternative methods. The results are reported below.

Results. All participants thought that SPYC was convenient and comfortable to use. In the direct comparison with `Out-of-Band Method`, 39 participants out of 40 agreed that our voice method was more convenient, and 1 preferred the alternative approach. In the direct comparison with `Physical Method`, 36 participants out of 40 agreed that our voice method was more convenient, 2 neither agreed nor disagreed, and 2 participants thought that the alternative (`Physical Method`) was more convenient.

Besides, as indicated in our pre-test questionnaire, 12 out of all 40 participants (30%) reported to regularly talk to

his/her partner (of our study) in English. Also, 12 participants (30%) reported that "definitely" sure about his/her partner's voice and 26 (40%) reported "probably" and 2 (5%) reported "not sure". Even though some pairs of participants did not regularly talk to each other in English or were even not sure that they were able to recognize their partner's voice, all participants in our study successfully distinguished the real receiver's voice from that of the adversary and the mixture one in our experiment. The result provides evidence that the SPYC vouch is both convenient and reliable to use.

B. Resilient to the Voice Morphing Attack

A recent study shows that existing voice-based authentication over VoIP is vulnerable to voice synthesis attacks [50], including a re-ordering attack in which the adversary reorders and pastes the victim's voice unit (words or sentences) together to form a speech for authentication, and a voice morphing attack in which the adversary's voice is converted to that of the victim's in speaking authentication terms, using Festvox [5], a state-of-the-art voice transformation tool developed by Carnegie Mellon University. The design of the SPYC vouch ensures that only distinct and rare or complete fake words will be spoken, which defeats the re-ordering attack. Our design of using pre-recorded words instead of speaking over the noisy VoIP channel could also potentially make the attack less effective, particularly given that the current transformation technique still cannot fully achieve the quality that makes the synthetic and the real voice indistinguishable. Further with the help of the conflict detection, authenticating SPYC vouches mostly happen when the real user's voice is present together with that from the impersonator, which could make the attack even harder to succeed. Saying that, an experimental study is important to understanding whether indeed our SPYC mechanism is capable of withstanding the morphing attack. To this end, we performed the second user study (Study 2).

In our study, we ran the most up-to-date version of Festvox [5] (the same system used for the attack in the prior work [50]) to create voice morphing attacks, when the participants were using our system. Specifically, we transformed the voice of an "attacker" into each participant's voice using Festvox, and then used the transformed voice to mislead the participants.

Recruitment. We recruited 20 participates from the same educational institution. Again, all participants came in pairs, claiming to know each other's voice. Due to the complexity of the task, each participant was paid 20 USDs for completing the tasks. Demographic information of the participants is given in Table IV (under column *Study 2*). We also recruited a male and a female "attackers". Each spoke 50 sentences, which were recorded to train Festvox [5], as happened in the prior study [50]. Each of them also recorded his/her voice for three SPYC vouches. The objective of the attack was to transform the attacker's voice into the victim's when speaking the vouch.

Experiment The experiment took each pair of participants about 45 minutes in separated private rooms. Each pair were first asked to fill a pre-test questionnaire similar to the one described above. Then, each participant spoke exactly the same 50 sentences spoken by the attacker based on gender. This is necessary for Festvox to train a model that transforms the attacker's voice to the participant's. Again, the size of the training set is exactly what was used in the prior study [50]. Besides, each participant also spoke 3 SPYC vouches which were recorded and used for comparison later.

After the voice recording, we waited 10 minutes for Festvox to learn the voice features of the two participants (denoted by A and B) and the attacker (Mal). Then Festvox transformed the three SPYC vouches spoken by the attacker into A's voice (denoted by $\text{Voice}_{Mal\_to\_A}$) and B's. After that, we conducted four tests on both A and B separately. In particular, in the test 1, the participant A listened to the first transformed vouch (mimicking B) and indicated whether it indeed came from B. In the test 2, we first let the participant know that voice might be synthesized, not spoken by the original person. Then during the experiment, we randomly chose between the synthesized vouch or the authenticate one (note the content was completely different from that used in the test 1) and let A decide whether he/she heard the authentic vouch from B. In the test 3, the participant A listened to both the synthesized voice (mimicking B) and the original SPYC voice spoken by B (different content) to tell which one came from B. This test evaluated the effectiveness of our technique under the scenario where our monitor detects the conflict and plays the SPYC vouches from the related parties to the user for conflict resolution. Also in each test, the participant rated the quality of each piece of voice with (`poor, fair, good, very clear`). In the test 4, we repeated what was done in the test 2 (using different voice), except that the participant was told before the test that the authentic vouch was expected to be of good quality under our design. Participant B did the same set of tests as A completed, except that the fake vouches now were based upon mimicking A's voice.

Result The results of all four tests indicate that our protection was successful in defeating the voice morphing attack. Specifically, in the test 1, when the participant was not aware of the possibility of the attack, 17 out of the 20 participants did not believe the authenticity of the vouch while the remaining 3 did. However, in the test 2, when they knew voice could be synthesized and listened to only one piece of voice (randomly chosen between the authentic and synthesized vouches), all 20 participants correctly identified the original voice and morphed voice. In the test 3, when morphed and original voice were put together for a comparison (mimicking the situation of conflict resolving), all 20 participants easily made the right choice. In the test 4, when the quality of the

voice was considered, all 20 participants easily differentiated the fake and real vouches.

The outcomes of our study show that even with the progress made by the state-of-the-art voice transformation techniques, voice-based authentication can still provide effective protection, particularly when the system has been carefully designed to exploit the weaknesses of the current voice-synthesizing technique (e.g., difficulty in producing the high-quality voice, as acknowledged by a member on the Festvox team). Also note the cost of such voice morphing attacks is nontrivial: the attacker needs to record the victim's voice for 50 sentences of high quality (as did in our research) and speaks exactly the same 50 sentences before the voice transformation can be done.

C. Performance

We further evaluated the performance of our monitor app on two Macbook Pro (Mid 2014 model, 2.6 GHz Intel i5, 8 GB memory, SSD) and two iPhone 5. On the Macbooks, we found that throughout our study, the CPU usage of our monitor is 1.5% and the memory usage was 28 MB on average. On iOS, the average CPU and memory usage are below 6.5% and 9 MB. Further in our study, we measured the delay introduced when running our monitor app to protect AirDrop. The delay here comes from three sources. The first is the time required for speaking out a certificate (SPYC, Section IV-B). As stated before (Section IV-B), this only needs to be done once for all. We asked 10 users to speak out their randomly-generated certificates three times each and found that the time one took to speak her certificate ranges from 5.2 to 7.4 seconds, with the average 5.98 seconds. The second source is the delay in listening to the SPYC vouch and identifying the person in communication. In our research, we measured this delay on both the iPhone and the Macbook, based upon 10 users' experiences with each of them repeating the experiment for 10 times. It turns out that the authentication on the iPhone took 9.02 seconds on average while on Macbook it is 8.65 seconds. The third source is the time our app spends on monitoring the whole file-transfer process through AirDrop, which was evaluated by comparing the delays observed with and without our protection. Specifically, we measured the time AirDrop took to deliver the files of different sizes (50, 100, 200, 400, 600, 800 MB and 1 GB), first under the protection of the monitor and then not. Each file was transferred 10 times on both iPhone and Macbook and the total delay was averaged over all the measurements of the time consumed. This study demonstrates that the overheads of our approach are completely negligible (Table V), which were found to be overshadowed by the variations of the file transfer times.

	50MB	100MB	200MB	400MB	600MB	800MB	1GB
iOS	1.33%	0.71%	-1.67%	-0.12%	-1.3%	0.82%	-0.78%
Mac OS X	2.22%	-1.61%	1.05%	-1.24%	0.78%	1.69%	-0.27%

Table V: Performance overhead when transferring files of different sizes

VI. Discussion

Security of SPYC. Our experiment findings show that the threat of the state-of-the-art voice transformation techniques to our SPYC mechanism is limited. This conclusion, apparently, is in conflict with the findings made in the prior study [50], which shows that the voice morphing could cause the voice-based authentication to fail in nearly 50% of the cases. This discrepancy is mostly introduced by the differences in the experiment settings and application domains. Specifically, the prior research considers the individuals who are not very familiar with how the victim speaks (the participants only learned the victim's voice during the study), and the authentication string is supposed to be spoken over the noisy VoIP channel (different noise profiles used in the study). Under this setting, the study shows that even in the absence of the attack, almost 50% of times the participants could not determine the authenticity of the voices [50]. By comparison, SPYC is used between those who are familiar with each other's voice and also our design requires the vouch (the recording of the user's voice) to be downloaded for verification, which minimizes the impacts of the channel noise. Most importantly, we show that by carefully designing the system and providing a bit more background information to the user (e.g., expected voice quality, possible presence of synthetic voices), voice-based authentication can still offer effective protection against impersonation and MitM attacks.

Also it is important to note that the voice-morphing attack is essentially a type of targeted attacks. As mentioned earlier (Section V-B), the cost of the attack is substantial. Particularly the high-quality voice samples are needed for 50 sentences. Therefore we believe that the SPYC mechanism significantly raises the bar to the attack on ZeroConf systems.

Limitations and future research. On the other hand, further research is needed to improve our current design and implementation. For example, the dictionary used in our study has been carefully chosen to ensure that the words it includes are both rare and distinct. However, they are real words and there is a chance that people still say them. Alternatively we could look at fake words with distinct pronunciations. How to generate and select these words needs further investigation. Also, a non-biometric solution to the certificate verification problem should also be studied. One possibility is to let the Apple user choose her own publishable identifiers, e.g., Facebook profile, personal website, etc., and include it as part of her account information. During email communication with her contacts, such information can be automatically exchanged across different Apple devices. Further effort is needed to find out how to make the approach work. Finally, we strongly believe that given the fact that ZeroConf systems today tend to be deployed in an untrusted environment, guidelines should be in place to help the developers build

such systems with proper protection, in line with the threat it is facing.

VII. Related Work

ZeroConf security. Security threats to Link-Local Multicast Name Resolution (LLMNR), a ZeroConf protocol used in Microsoft Windows, have been mentioned in technical blogs [14], [15], [13], [19] and an IETF documentation [16]. Unlike Bonjour, LLMNR is not designed for automatic service discovery, a common feature of a ZeroConf system. Instead, it just supports name resolving [12], which is found to be vulnerable to a DNS spoofing attack. By comparison, our study on Bonjour also focuses on its service discovery stage, particularly its automatic mechanism for conflict resolving, which can be exploited by the adversary to silently hijack the victim's service name, and the fundamental challenge in protecting it with TLS (Section III-C).

Bluetooth security. Related to Bluetooth ZeroConf is the work [37], [36] on the security of the devices without input capabilities (e.g., no keyboard, no display). Different from the studies, our research is the first that investigated how the *Just Works* and *OOB* (Section II) pairing modes [21] are supported on Apple's Core Bluetooth framework. Under the framework, the actual pairing operations are transparent to the app developers. However, we found that the insecure *Just Works* mode is the default setting. This usability-orient design (easing paring process) also makes the developer more likely to choose the insecure mode within her apps (Section III-A). Further we show that even under the *OOB* mode, a malicious app on Macbook can still gain unauthorized access to the service on an iPhone.

Authentication between devices. Techniques for authentication between devices have been studied recently [41], [46], [45], [32]: e.g., a two-factor authentication mechanism based on ambient sound [41], barcode scanning [46], shaking devices together [45], sharing radio environment [32], etc. These approaches are not suitable for authenticating the devices transferring files over distance, in the presence of the adversary.

Most related to our work is the technique that enables the user to authenticate herself by speaking binary code or PGP words over the VoIP channel [24], [17], [39], [38]. It is used in Zfone [25], a secure VoIP software. This type of voice-based authentication [35] is found to be vulnerable to the voice reordering attacks and the voice morphing attack [50], [47]. By comparison, SPYC (Section IV-B) is designed to address such threats: it utilizes rare or fake words (rather than the common PGP words and binary) to defeat the reordering attack and requires the user to download the vouch (voice recording) to authenticate the certificate owner, which avoids the noise introduced by the channel. This approach, together with our unique conflict detection approach (Section IV-A), is found to be resilient to those attacks. Also importantly, our study shows that a carefully-designed voice based approach can still provide effective protection in practice.

TLS security. Prior research [44], [51], [48], [30], [33], [31] demonstrates the cryptographic or implementation flaws within TLS systems: e.g., the server's certificate is not verified properly by the apps [33] or browsers [31]. However, our work is the first that demonstrates the difficulty in linking a human to her certificate (Section III-C), which makes a certificate hard to verify. This is found to be fundamental for authenticating devices under the ZeroConf setting.

VIII. Conclusion

In this paper, we describe the first systematic study on the security protection of Apple ZeroConf systems, which reveals that the security protection within these systems is either not in place at all or ineffective, allowing the adversary to get access to sensitive user data. Addressing such security risks is nontrivial, due to the challenge in binding a human individual to her certificate. Our solution includes a conflict detection technique and SPYC, a voice-based approach designed to be convenient to use and effective against speech synthesis attacks. We implemented our techniques in AirDrop and evaluated its usability and effectiveness through user studies. Our research shows that a well-designed voice authentication can still offer effective protection, in spite of the progress made on speech synthesis. Moving forward, we believe that our findings and techniques will contribute to better designs of ZeroConf systems, enhancing their security protection while keeping them zero-configured.

Acknowledgment

We thank anonymous reviewers for their comments. The IU authors are supported in part by the NSF CNS-1223477, 1223495 and 1527141. Authors from Tsinghua University are supported in part by Research Grant of Beijing Higher Institution Engineering Research Center, and Tsinghua University Initiative Scientific Research Program.

References

[1] "10,000 most common english words," https://github.com/first20hours/google-10000-english.
[2] "About Multipeer Connectivity," https://developer.apple.com/library/ios/documentation/MultipeerConnectivity/Reference/MultipeerConnectivityFramework/.
[3] "Bonjour," https://www.apple.com/hk/en/support/bonjour/.
[4] "Clutch," https://github.com/KJCracks/Clutch.
[5] "FestVox," http://festvox.org.
[6] "Filedrop," http://www.filedropme.com.
[7] "Homophones," http://www.allaboutlearningpress.com/homophones/.
[8] "Hopper v3," http://www.hopperapp.com.
[9] "How far does a Wi-Fi Direct connection travel?" hhttp://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel.
[10] "iptables," http://www.netfilter.org/projects/iptables/.
[11] "Lenovo," www.lenovo.com/.

[12] "LLMNR, mDNS and mDNS Responders in Windows," http://lists.apple.com/archives/rendezvous-dev/2004/Apr/msg00031.html.

[13] "LLMNR Spoofer," http://www.rapid7.com/db/modules/auxiliary/spoof/llmnr/llmnr_response.

[14] "Local Network Attacks: LLMNR and NBT-NS Poisoning." https://www.sternsecurity.com/blog/local-network-attacks-llmnr-and-nbt-ns-poisoning.

[15] "Local Network Vulnerabilities - LLMNR and NBT-NS Poisoning," http://www.surecloud.com/newsletter/local-network-vulnerabilities-llmnr-and-nbt-ns-poisoning.

[16] "Multicast DNS (mDNS) Threat Model and Security Consideration," http://tools.ietf.org/html/draft-rafiee-dnssd-mdns-threatmodel-00.

[17] "PGP word list," https://en.wikipedia.org/wiki/PGP_word_list.

[18] "QQ," http://imqq.com.

[19] "Responder," https://github.com/SpiderLabs/Responder.

[20] "Scribe," http://usescribe.com.

[21] "Security, Bluetooth Smart (Low Energy)," https://developer.bluetooth.org/TechnologyOverview/Pages/LE-Security.aspx.

[22] "Supporting materials," https://sites.google.com/site/applezeroconf/.

[23] "Tencent QQ," https://en.wikipedia.org/wiki/Tencent_QQ.

[24] "The International PGP Home Page," http://www.pgpi.org.

[25] "The Zfone(TM) Project," http://zfoneproject.com.

[26] "Triple/quadruple/quintuple/sextuple homonyms," http://people.sc.fsu.edu/~jburkardt/fun/wordplay/multinyms.html.

[27] "Voice over IP," https://en.wikipedia.org/wiki/Voice_over_IP.

[28] "Zero Configuration Networking (Zeroconf)," http://www.zeroconf.org.

[29] "ZeroConf-style Bluetooth," http://www.theregister.co.uk/2003/06/18/sony_preps_zeroconfstyle_bluetooth_tech/.

[30] N. Al Fardan and K. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *Security and Privacy (SP), 2013 IEEE Symposium on*, May 2013, pp. 526–540.

[31] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 114–129.

[32] E. de Lara, "Amigo: Proximity-based authentication of mobile devices," Google Tech Talk, Mountain View, CA, July 2007.

[33] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in) security," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 50–61.

[34] M. Gasser, "A random word generator for pronounceable passwords," DTIC Document, Tech. Rep., 1975.

[35] M. Goodrich, M. Sirivianos, J. Solis, G. Tsudik, and E. Uzun, "Loud and clear: Human-verifiable authentication based on audio," in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, 2006, pp. 10–10.

[36] K. M. Haataja and K. Hyppönen, "Man-in-the-middle attacks on bluetooth: a comparative analysis, a novel attack, and countermeasures," in *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*. IEEE, 2008, pp. 1096–1102.

[37] K. Hyppönen and K. M. Haataja, "Nino man-in-the-middle attack on bluetooth secure simple pairing," in *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*. IEEE, 2007, pp. 1–5.

[38] P. Juola, "Isolated-word confusion metrics and the pgpfone alphabet," *arXiv preprint cmp-lg/9608021*, 1996.

[39] P. Juola, "Whole-word phonetic distances and the pgpfone alphabet," in *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, vol. 1. IEEE, 1996, pp. 98–101.

[40] R. Kainda, I. Flechais, and A. Roscoe, "Usability and security of out-of-band channels in secure device pairing protocols," in *Proceedings of the 5th Symposium on Usable Privacy and Security*. ACM, 2009, p. 11.

[41] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, "Sound-proof: Usable two-factor authentication based on ambient sound," in *24rd USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015, pp. 483–498.

[42] A. Kobsa, R. Sonawalla, G. Tsudik, E. Uzun, and Y. Wang, "Serial hook-ups: a comparative usability study of secure device pairing methods," in *Proceedings of the 5th Symposium on Usable Privacy and Security*. ACM, 2009, p. 10.

[43] B. Laufer and G. C. Ravenhorst-Kalovski, "Lexical threshold revisited: Lexical text coverage, learners' vocabulary size and reading comprehension." *Reading in a foreign language*, vol. 22, no. 1, pp. 15–30, 2010.

[44] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the tls protocol," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 62–72.

[45] R. Mayrhofer and H. Gellersen, "Shake well before use: Authentication based on accelerometer data," in *Proceedings of the 5th International Conference on Pervasive Computing*, ser. PERVASIVE'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 144–161.

[46] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *Security and privacy, 2005 IEEE symposium on*. IEEE, 2005, pp. 110–124.

[47] D. Mukhopadhyay, M. Shirvanian, and N. Saxena, "All your voices are belong to us: Stealing voices to fool humans and machines," in *Computer Security–ESORICS 2015*. Springer, 2015, pp. 599–621.

[48] K. G. Paterson and N. J. AlFardan, "Plaintext-recovery attacks against datagram TLS," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[49] K. Rastle, J. Harrington, and M. Coltheart, "358,534 nonwords: The arc nonword database," *The Quarterly Journal of Experimental Psychology: Section A*, vol. 55, no. 4, pp. 1339–1362, 2002.

[50] M. Shirvanian and N. Saxena, "Wiretapping via mimicry: Short voice imitation man-in-the-middle attacks on crypto phones," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 868–879.

[51] M. Vanhoef and F. Piessens, "All your biases belong to us: Breaking rc4 in wpa-tkip and tls," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 97–112.