hexway

Apple bleee. Everyone knows What Happens on Your iPhone

What if	3
Nearby	4
Wi-Fi password sharing	5
Airdrop	9
Others	13
Handoff	13
Airpods	13
Wi-Fi settings	13
Hotspot	13
PoCs	14
Links	14

In this article, we will outline our research process, from our initial ideas to first POCs. Technical specialists may find this interesting.

What if...

Let's take a look at the BLE traffic. For this purpose, we've slightly modified the scripts from py-bluetooth-utils repository'

Unlock the phone and run the BLE sniffer.

```
python ble_adv_search.py -m 54:69:F1:23:2B:47
[54:69:F1:23:2B:47] 0e02011a0aff4c0010050b1c0fc556
. . .
[54:69:F1:23:2B:47] 0e02011a0aff4c0010050b1c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c0010050b1c0fc556
. . .
```

Turn the phone off.

```
...
[54:69:F1:23:2B:47] 0e02011a0aff4c0010050b1c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c0010050b1c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c001005031c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c001005031c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c001005031c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c001005031c0fc556
[54:69:F1:23:2B:47] 0e02011a0aff4c001005031c0fc556
...
```

We can see that only one byte reflects the change in screen status. Apple uses ADV_IND messages to send out current status data.

That's the structure of a typical advertise data packet:



Turning the phone on and off, navigating the menu, changing settings and running different apps (phone, calendar, photos, settings), we've identified the fields responsible for the Wi-Fi and buffer status and several types of BLE messages.

0x05 - Airdrop 0x07 - Airpods 0x10 - Nearby 0x0b - Watch Connection 0x0c - Handoff 0x0d - Wi-Fi Settings 0x0e - Hotspot 0x0f - Wi-Fi Join Network

Nearby

That's a Nearby message:

0	1	2 5	
+	-+	++	
1	1	I I	
status	wifi	other	
1	1		
+	-+	++	

where status can be:

0x0b	-	Home	screen
0x1c	-	Home	screen
0x1b	-	Home	screen
0x11	-	Home	screen
0x03	-	Off	
0x18	-	Off	
0x09	-	Off	
0x13	-	0ff	
0x0a	-	Off	
0x1a	-	Off	
0x01	-	0ff	
0x07	-	Lock	screen
0x17	-	Lock	screen
0x0e	-	Calli	ng
0x5b	-	Home	screen
0x5a	-	Off	

That's enough to write a simple packet analyzer that allows us to get data from all nearby Apple devices in real-time.

Apple devices scann	er					
Мас	State	Device	WI-FI	OS	Phone	Time
50:2D:AC:99:12:94						1563353463
7E:B5:C1:97:E4:C9						1563353463
51:7B:B1:BB:E5:51	Lock screen					1563353463
56:E6:3F:CD:76:86		Watch		WatchOS		1563353453
6B:54:70:E6:25:7D						1563353463
49:5E:D2:98:47:47						1563353463
41:CE:CF:85:21:B8		Watch		WatchOS		1563353463

Video Demo:

https://www.youtube.com/watch?v=Bi602yAIBAw

To analyze the BLE packets further, we chose to use Adafruit Bluefruit LE Sniffer that helps analyze the Wireshark BLE traffic.

Wi-Fi password sharing

Then, we analized how users are identified when two devices interact. This process is used for conencting to Wi-Fi.



While trying to connect to a network, the device sends the following BLE packet:

c03080fc5563125c9d087555a77e3e2005f10020b0c

Trying to connect to various SSID on different devices (yes, we could've just reversed sharingd in IDA/radare/gydra) we found out that this message has the following format:

0	1	2		5		8		12	15	18
 flag	gs type (0x0	 e 08)	auth tag		sha(appleID)		sha(phone_nbr)	 sha(e	email) sha(SSI	D)

As you can see, the device sends the first 3 bytes of the SHA256 hash of phone number/email/appleID sha256(phone_number)[0:3]

Probably, devices hash all contacts and then compare them to the values from BLE messages, and in case there is a match offer to share the Wi-Fi password (if they have it, of course). We'll look into this process in more detail in the following articles. What we tried to do here is to recover a phone number from those 3 bytes of the hash.

First, we have to understand phone number formats. E.164 is a recommendation describing various formats.

- A phone number can have a maximum of 15 digits
- And it can be divided into:

- 1. Country code (1-3 digits)
- 2. National destination code (NDC)
- 3. Subscriber number

Of course, formats vary with country, but the idea is the same. So, we can calculate the values of SHA256 for the numbers of a particular city.

<sha256[0:3]>:<phone_number>

We made a table of phone numbers for a city with a population of about 5 000 000. Considering the large number of subscribers, collisions are inevitable. In average, we have a collision of 10-15 numbers for 3 bytes of hash.

We made an API for quick requests to the table to get a number from a hash:



There are two approaches to ensuring the accuracy of identification:

1. HLR (Home Location Register) Lookup that allows identifying inactive subscribers and subscribers from other regions

2. A number must be associated with an AppleID, so we can identify valid numbers by checking if iMessage is available for a certain number (we'll talk about this approach in more details in the future articles).

Combining these two approaches, we can accurately identify a phone number in almost 100% of cases.

Can we activate the Wi-Fi password sharing popup on a device? That's an open issue.



Thus, we have a script that identifies the users connected to Wi-Fi.

https://www.youtube.com/watch?v=kTtNX5Tmk3Q

Moreover, we can send BLE requests for a Wi-Fi password hoping that the victim will provide us, for example, with a corporate network password. We'll talk about this vector more in our other articles as well. By the way, you can use an Android app (like nRF Connect) to clone and repeat different BLE messages.



Airdrop

Let's see if this mechanism of identifying users for the purpose of sharing Wi-Fi passwords is universal. Apple Airdrop is one of the points we can investigate, as it has 3 privacy options:

- 1. Receiving Off
- 2. Contacts Only
- 3. Everyone

So, how do devices identify each other?



We run Airdrop and follow the BLE sniffer:

0000000000000001123412341234123400

0	8	9	11	13	15	17	18
zeros	 st(0x01) 	 sha(AppleID) 	 sha(phone) 	 sha(email) 	 sha(email2) 	 zero	1

As you can see, in this case, devices sends only 2 bytes of the SHA256 hash, which is enough to identify the phone number. Traffic analysis showed that BLE is only used to initiate AirDrop transfer. The transfer itself happens via Wi-Fi using the AWDL technology that establishes a peer2peer connection between the server (receiver) and the client (sender). To analyze the AWDL packets, we can use Wireshark and the awd10 interface available on all devices.

rface		Device All advertising devices OPasskey / O	DOB key	Adv Hop		— н	lelp Defaults
	Time	Source	Destination	Proto	col Length	uTF8String	
1	0.000000	fe80::5403:92ff:feb1:858d	ff02::fb	MDNS		75	
2	0.538514	fe80::b077:d4ff:feca:b3f0	ff02::fb	MDNS	; 1	.84	
3	0.538522	fe80::b077:d4ff:feca:b3f0	ff02::fb	MDNS	1	.39	
4	0.538525	fe80::b077:d4ff:feca:b3f0	ff02::fb	MDNS	1	.78	
5	0.538527	fe80::b077:d4ff:feca:b3f0	ff02::fb	MDNS	1	.28	
6	0.546228	fe80::8827:23ff:fe4f:7a0c	ff02::fb	MDNS	1	.79	
7	0.546243	fe80::8827:23ff:fe4f:7a0c	ff02::fb	MDNS	1	.39	
8	0.546247	fe80::8827:23ff:fe4f:7a0c	ff02::fb	MDNS	: 1	.73	
9	0.546251	fe80::8827:23ff:fe4f:7a0c	ff02::fb	MDNS	; 1	.28	
10	0.587139	fe80::145b:2aff:feb8:1929	fe80::5403:92ff:feb1:858d	TCP		98	
11	0.588149	fe80::5403:92ff:feb1:858d	fe80::145b:2aff:feb8:1929	TCP		98	
12	0.644979	fe80::145b:2aff:feb8:1929	fe80::5403:92ff:feb1:858d	TCP		86	
13	0.645015	fe80::145b:2aff:feb8:1929	fe80::5403:92ff:feb1:858d	TLSV	1 3	29	
14	0.645303	fe80::5403:92ff:feb1:858d	fe80::145b:2aff:feb8:1929	TCP		86	
15	0.645305	fe80::5403:92ff:feb1:858d	fe80::145b:2aff:feb8:1929	TCP		86	
16	1.001762	fe80::5403:92ff:feb1:858d	ff02::fb	MDNS	1	75	

0000	33	33	00	00	00	fb	56	03	92	b1	85	8d	86	dd	60	0e	33 · · · · V · · · · · · · · · ·
0010	92	e7	00	dd	11	ff	fe	80	00	00	00	00	00	00	54	03	· · · · · · · · · · · · · · · · · · ·
0020	92	ff	fe	b1	85	8d	ff	02	00	00	00	00	00	00	00	00	
0030	00	00	00	00	00	fb	14	e9	14	e9	00	dd	4b	f9	00	00	· · · · · · · · · · · · K · · ·
0040	84	00	00	00	00	04	00	00	00	03	0c	32	34	34	30	31	
0050	32	65	31	39	65	39	62	08	5 f	61	69	72	64	72	6f	70	2e19e9b∙ _airdrop
0060	04	5f	74	63	70	05	6c	6f	63	61	6c	00	00	21	80	01	·_tcp·lo cal··!··
0070	00	00	00	78	00	0f	00	00	00	00	22	42	06	63	68	69	···x····"B·chi
0080	70	69	6b	с0	27	с0	0c	00	10	80	01	00	00	11	94	00	pik·'···
0090	0a	09	66	6c	61	67	73	3d	35	30	37	09	5f	73	65	72	<pre> • flags= 507 · ser </pre>

As you see, after sending a few MDNS packets, the devices exchange their certificates and then use the secure TLS connection for data transfer. We were about to begin the reverse engineering of the sharingd app, which is responsible for AirDrop, when people from Technische Universität Darmstadt released a white papper A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link about AirDrop (sadface.png). This white paper beautifully describes the functioning of AirDrop. Now, we'll take a brief look at the AirDrop protocol workflow:



During authentication, the device sends its identification data (sender's record data), that contains the **full** SHA256 hash of the phone number. Thus, if we answer all Airdrop BLE requests, we'll get the sender's contact data, including phone number hash.

We've slightly modified the opendrop utility to do that.

Here are the results:

https://www.youtube.com/watch?v=mREIeH_s3z8

Others

Below, you can see various BLE message formats that we encountered during our research.

Handoff

Airpods

0	3	4	5	6	7	9		25
 some 	 state1 	 state2 	 data1 	 data2 	 data3 	 	data4	+
+	+	+	+	+	+	_+		also a

By sending this message, we can make Apple devices display AirPods inforamtion as if they were connected. Just watch this funny video:

https://www.youtube.com/watch?v=HoSuLUtrkXo

Wi-Fi settings



Hotspot

0	2	3	4	5 6
+	-+	-+	-+	-++
1	1	1	1	1
data1	battery	data2	cell serv	cell bars
1	1	1	1	1 1
+	-+	-+	-+	-++



You can find all scripts in our GitHub repository: Apple bleee

Links

- https://github.com/hexway/apple_bleee
- https://hexway.io/blog/apple-bleee/
- https://arxiv.org/pdf/1904.10600.pdf
- https://www.usenix.org/system/files/sec19fall_stute_prepub.pdf
- https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf

Contact us:

contact@hexway.io

https://hexway.io

PoCs