Automatic Fingerprinting Of Vulnerable BLE IoT Devices With Static UUIDs From Mobile Apps

Chaoshun Zuo, **Haohuang Wen**, Zhiqiang Lin, and Yinqian Zhang

Department of Computer Science and Engineering
The Ohio State University

CCS 2019

Bluetooth Low Energy and IoT

Bluetooth™ 4.0

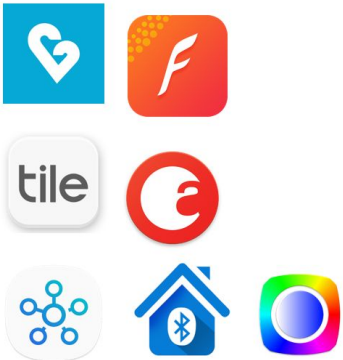
Low Energy

BLE IoT Devices and Companion Apps

BLE IoT Devices

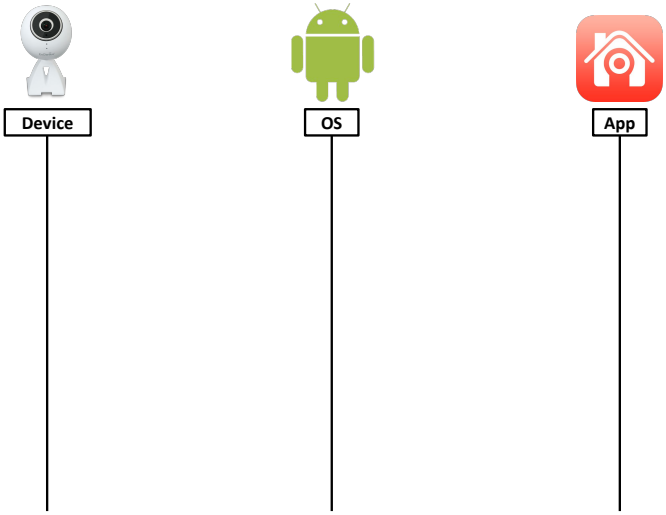
BLE IoT Devices and Companion Apps

BLE IoT Devices

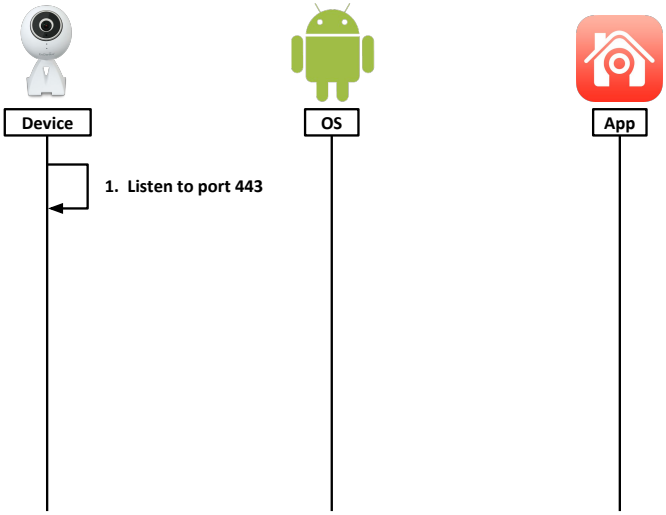


Companion Mobile Apps

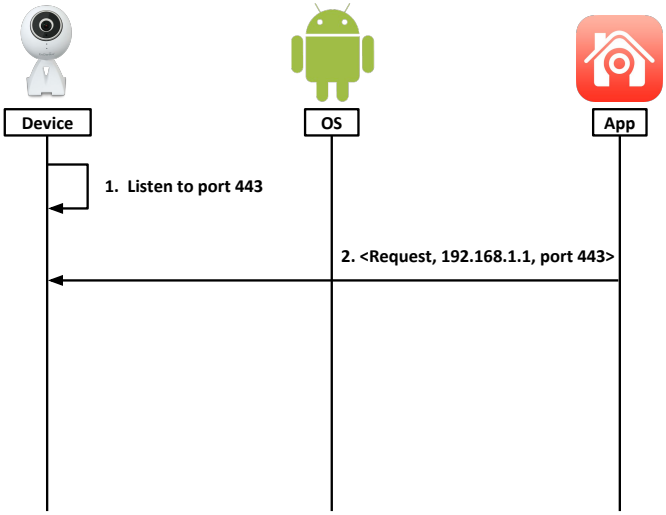
General Workflow of Device Communication in TCP/IP Setting



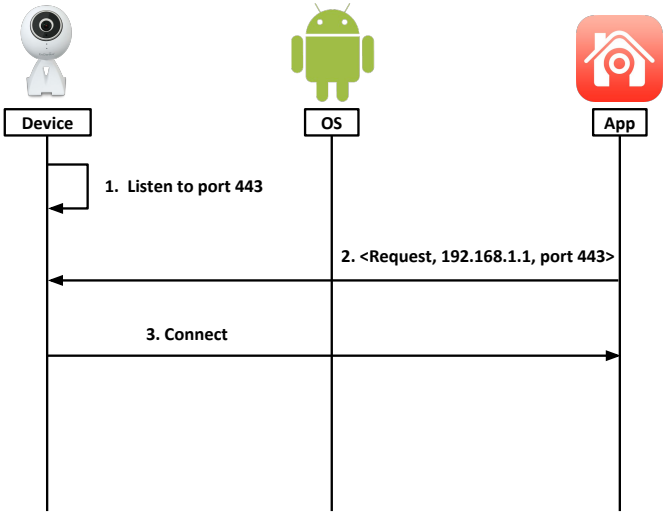
General Workflow of Device Communication in TCP/IP Setting



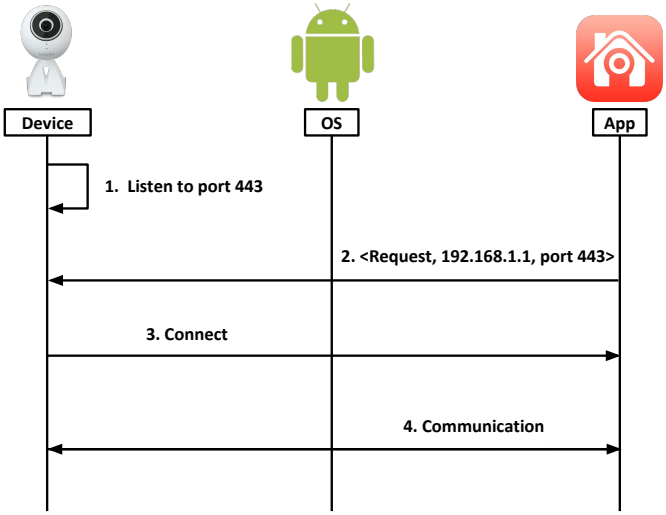
General Workflow of Device Communication in TCP/IP Setting



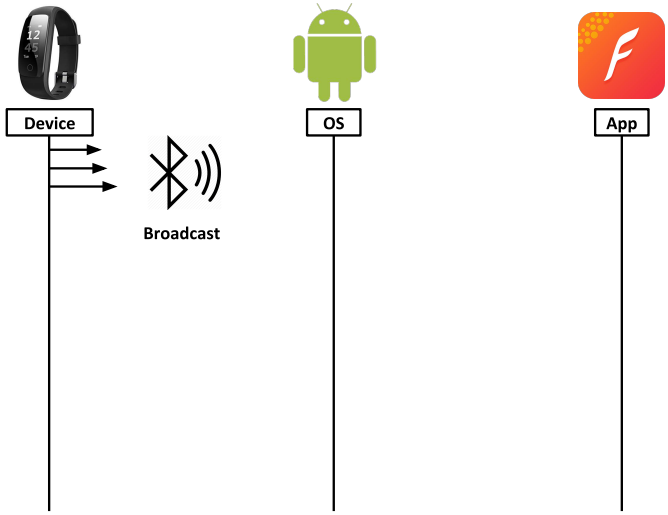
General Workflow of Device Communication in TCP/IP Setting



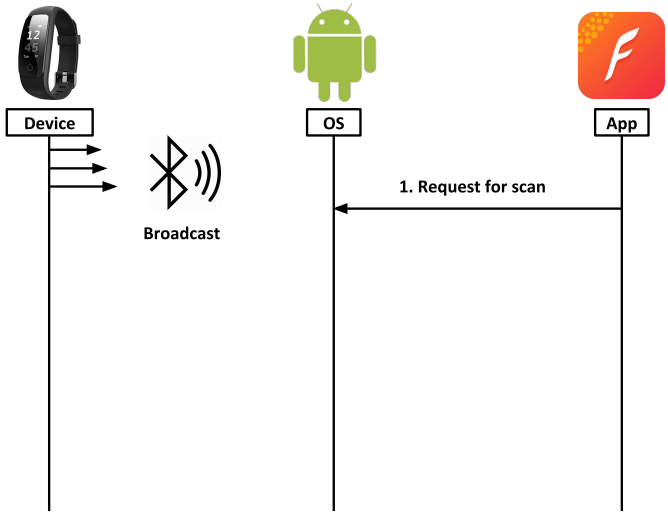
General Workflow of Device Communication in TCP/IP Setting



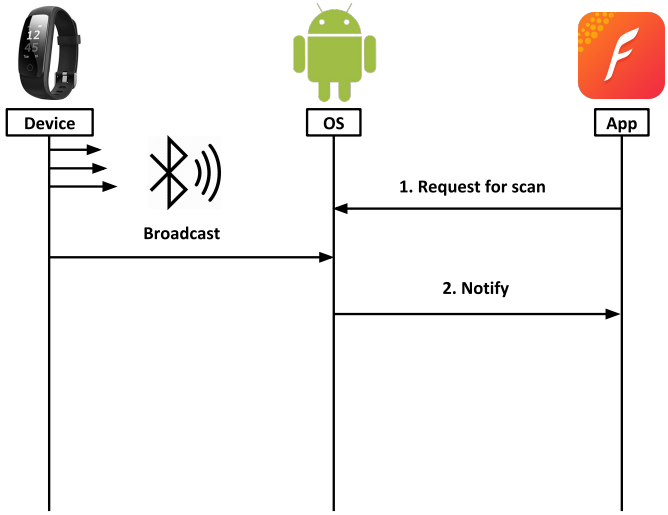
General Workflow of BLE IoT Devices and Companion Apps



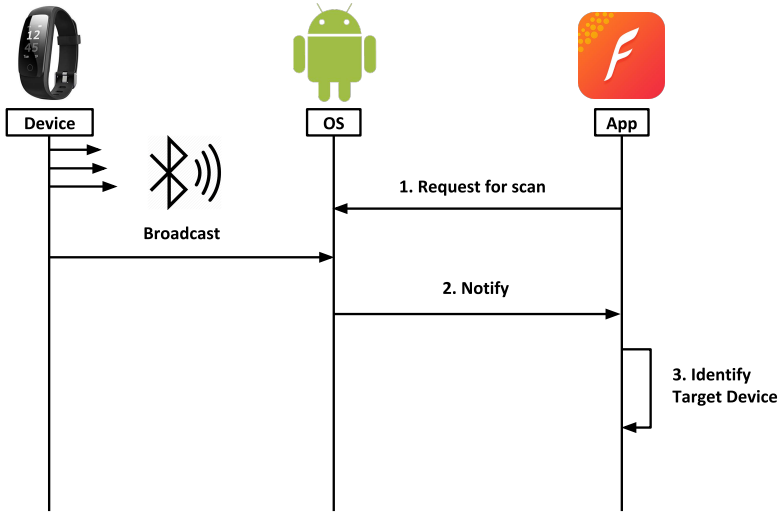
General Workflow of BLE IoT Devices and Companion Apps



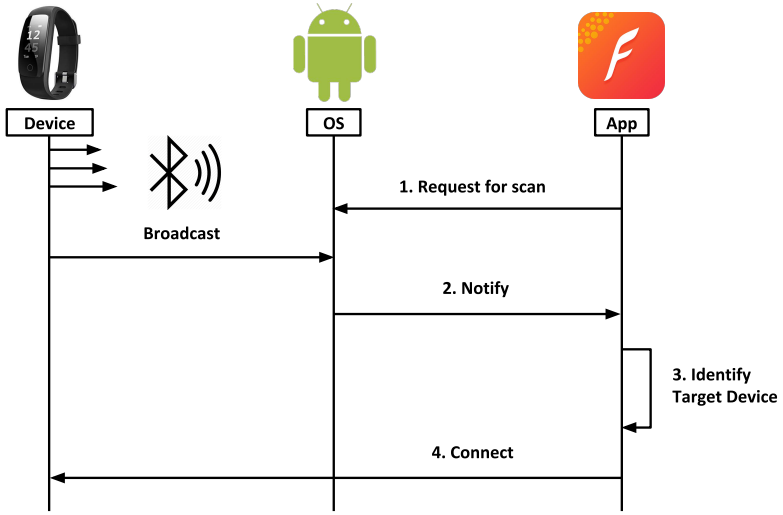
General Workflow of BLE IoT Devices and Companion Apps



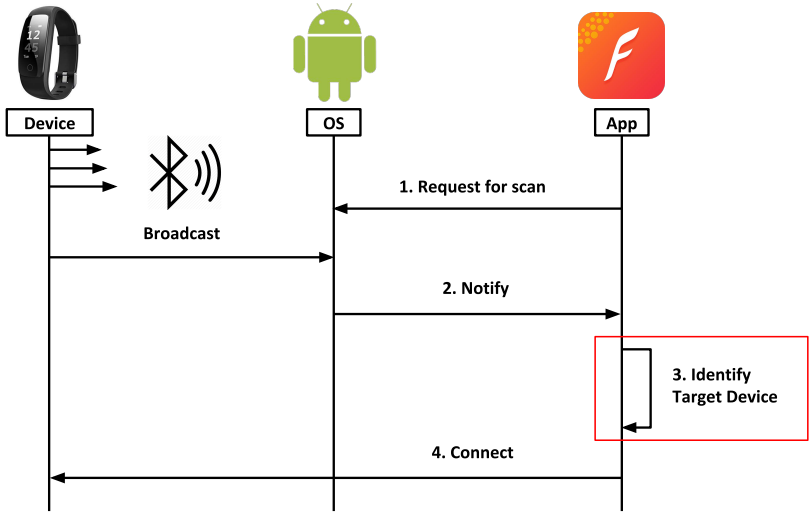
General Workflow of BLE IoT Devices and Companion Apps



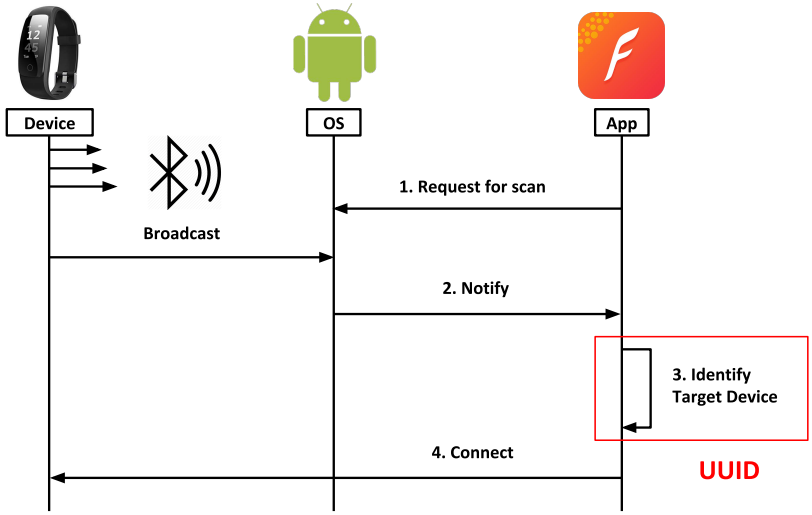
General Workflow of BLE IoT Devices and Companion Apps



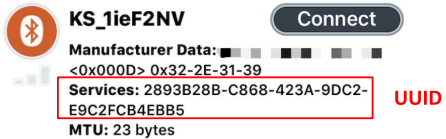
General Workflow of BLE IoT Devices and Companion Apps



General Workflow of BLE IoT Devices and Companion Apps

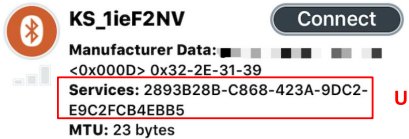


Our Observations



A BLE Broadcast Packet

Our Observations



A BLE Broadcast Packet

```
public class TemperatureService {
    public static final UUID EVENT_CHAR_UUID;
    public static final UUID PAIR_STATUS_CHAR_UUID;
    public static final UUID REQUEST_CHAR_UUID;
    public static final UUID RESPONSE_CHAR_UUID;
    public static final ParcelUuid SERVICE_PARCEL_UUID;
    public static final UUID SERVICE_UUID;

    static {
        TemperatureService.SERVICE_UUID = UUID.fromString("2893B28B-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.SERVICE_PARCEL_UUID = new ParcelUuid(TemperatureService.SERVICE_UUID);
        TemperatureService.REQUEST_CHAR_UUID = UUID.fromString("28930000-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.RESPONSE_CHAR_UUID = UUID.fromString("28930001-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.EVENT_CHAR_UUID = UUID.fromString("28930002-C868-423A-9DC2-E9C2FCB4EBB5");
        TemperatureService.PAIR_STATUS_CHAR_UUID = UUID.fromString("28930003-C868-423A-9DC2-E9C2FCB4EBB5");
    }
```

UUID

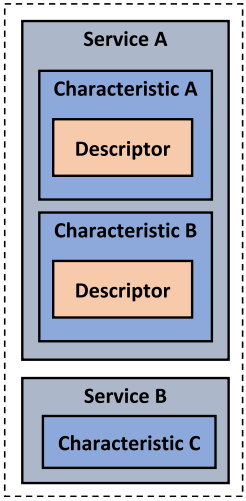
Decompiled Code in a Companion App

Our Observations

Key Insights

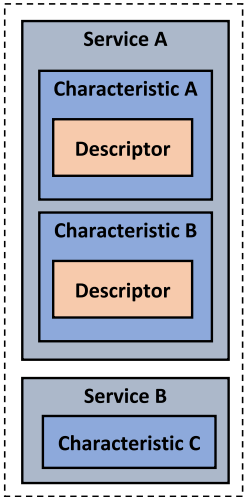
1. UUIDs are broadcasted by BLE IoT devices to nearby smartphones.
2. UUIDs are static.
3. Mobile apps contain UUIDs.
4. Mobile apps identify target BLE IoT devices based on their broadcast UUIDs.

Hierarchy of UUIDs



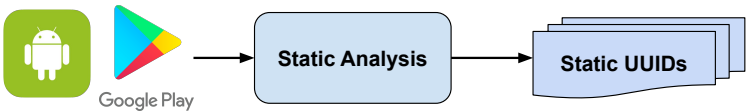

```
Service
    name: KINSA_SERVICE
    uuid: 00000000-006a-746c-6165…
    characteristics:
              name: REQUEST_CHARACTERISTIC
              uuid: 00000004-006a-746c-6165…
              descriptors: […]

              name: RESPONSE_CHARACTERISTIC
              uuid: 00000002-006a-746c-6165…
              descriptors: […]

Service
     name: BATTERY_SERVICE
     uuid: 180F
     characteristics: […]
      …
```

Hierarchy of UUIDs

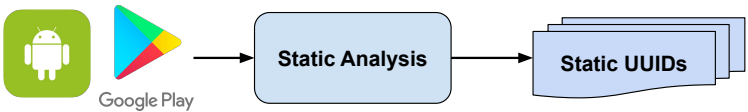



```
Service
    name: KINSA_SERVICE
    uuid: 00000000-006a-746c-6165…
    characteristics:
            name: REQUEST_CHARACTERISTIC
            uuid: 00000004-006a-746c-6165…
            descriptors: […]

            name: RESPONSE_CHARACTERISTIC
            uuid: 00000002-006a-746c-6165…
            descriptors: […]

Service
    name: BATTERY_SERVICE
    uuid: 180F
    characteristics: […]
     …
```

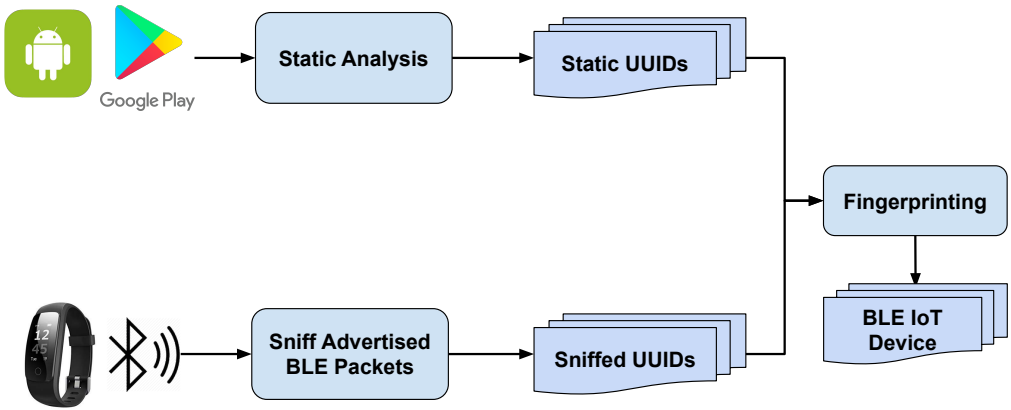
How to Fingerprint a BLE IoT Device with Static UUIDs



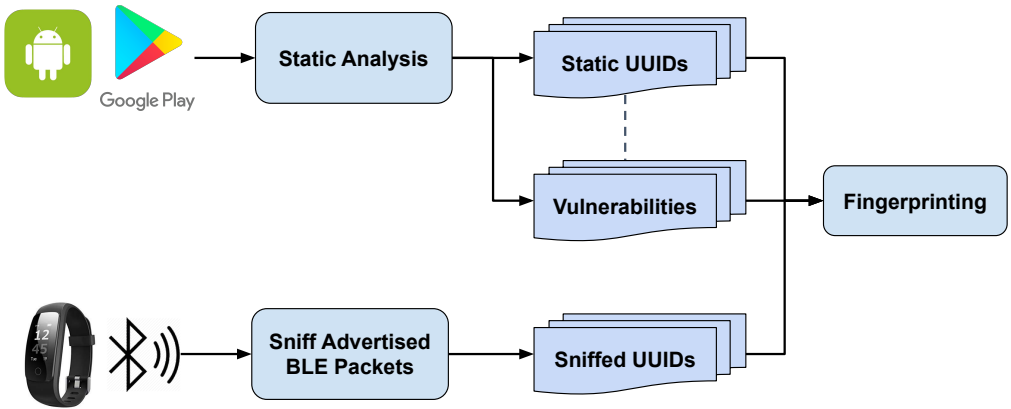
How to Fingerprint a BLE IoT Device with Static UUIDs



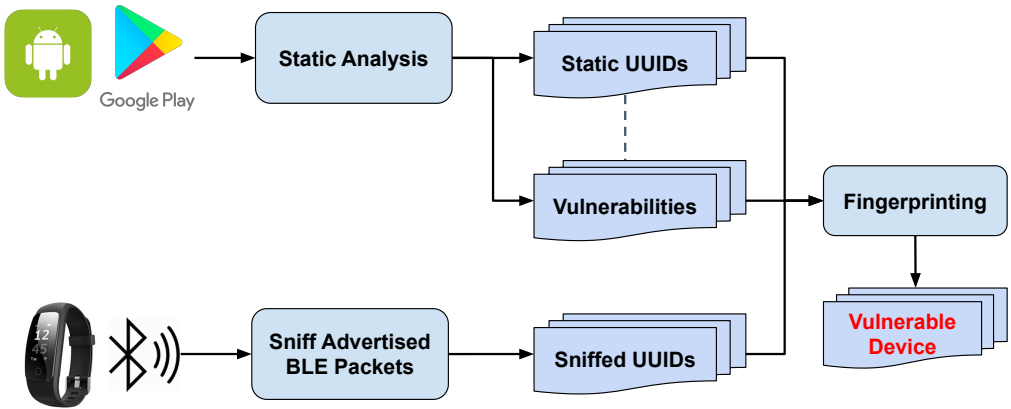
How to Fingerprint a BLE IoT Device with Static UUIDs



Application of BLE IoT Device Fingerprinting



Application of BLE IoT Device Fingerprinting

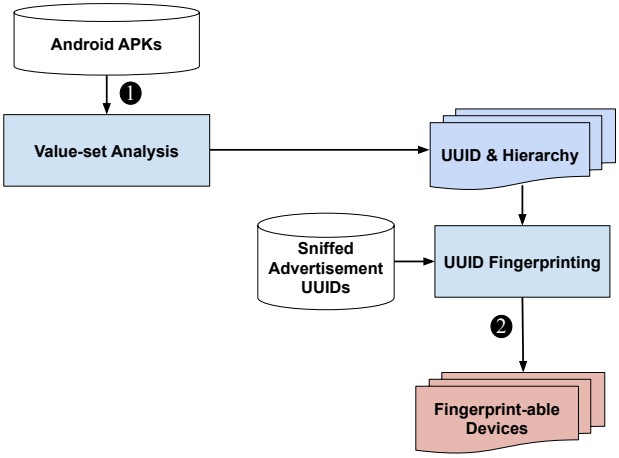


Our Contributions

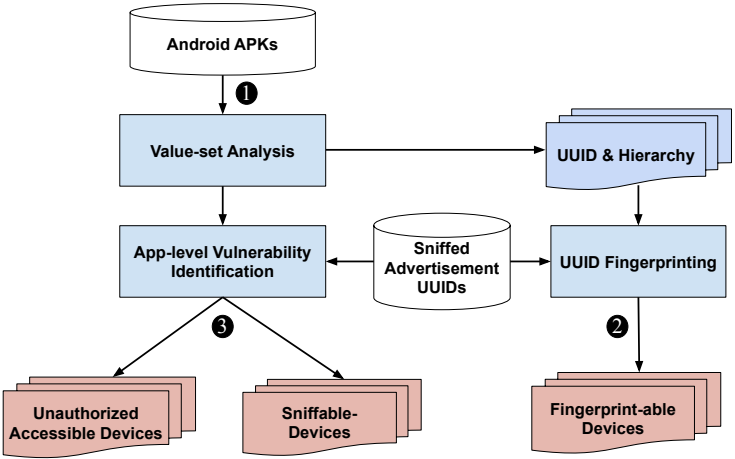
1. **Novel Discovery**. We are the *first* to discover BLE IoT devices can be fingerprinted with static UUIDs.
2. **Effective Techniques**. We have implemented an automatic tool BLEScope to harvest UUIDs and detect vulnerabilities from mobile apps.
3. **Evaluation**. We have tested our tool with $18,166$ BLE mobile apps from Google Play store, and found $168,093$ UUIDs and $1,757$ vulnerable BLE IoT apps.
4. **Countermeasures.** We present channel-level protection, app-level protection, and protocol-level protection (with dynamic UUID generation).

Overview of BLEScope

Overview of BLEScope



Overview of BLEScope



Challenges and Insights

Challenges

1. How to extract UUIDs from mobile apps
2. How to reconstruct UUID hierarchy
3. How to identify flawed authentication vulnerability

Challenges and Insights

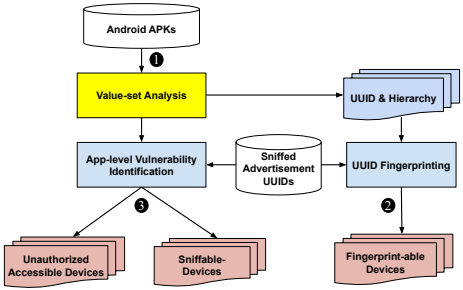
Challenges

1. How to extract UUIDs from mobile apps
2. How to reconstruct UUID hierarchy
3. How to identify flawed authentication vulnerability

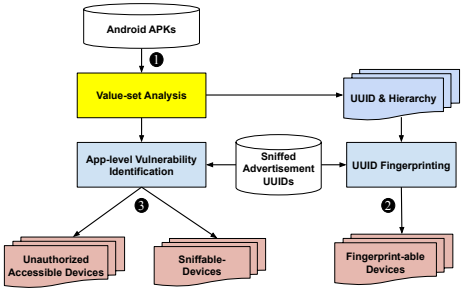
Solutions

1. Resolving UUIDs using context and **value-set analysis**
2. Reconstructing UUID hierarchy with **control dependence**
3. Identifying flawed authentication with **data dependence**

Value Set Analysis



Value Set Analysis



Category	API Name
UUID	BluetoothGatt: BluetoothGattService getService BluetoothGattService: BluetoothGattCharacteristic getCharacteristic BluetoothGattCharacteristic: BluetoothGattDescriptor getDescriptor ScanFilter.Builder: ScanFilter.Builder setServiceUuid ScanFilter.Builder: ScanFilter.Builder setServiceUuid ScanFilter.Builder: ScanFilter.Builder setServiceData ScanFilter.Builder: ScanFilter.Builder setServiceData

Table: APIs for UUID extraction and hierarchy reconstruction

UUID Extraction

```
1  public class KelvinDeviceProfile
2     private KelvinDeviceProfile(BlueToothLeGatt arg3)
3       super();
4       BluetoothGattService v0 = arg3.getService(KelvinGatt.KINSA_SERVICE);
5       if(v0!=null)
6         this.request = v0.getCharacteristic(KelvinGatt.REQUEST_CHARACTERISTICS);
7         this.response = v0.getCharacteristic(KelvinGatt.RESPONSE_CHARACTERISTICS);
8
9
10      BluetoothGattService v3 = arg3.getService(KelvinGatt.BATTERY_SERVICE_UUID);
11      if(v3!=null)
12        this.batterylevel = v3.getCharacteristic(KelvinGatt.BATTERY_VALUE_CHAR_UUID);
13
14
15
16
17 public class KelvinGatt
18     public UUID KINSA_SERVICE = UUID.fromString(00000000-006a-746c-6165-4861736e694b);
19     public UUID REQUEST_CHARACTERISTICS = UUID.fromString(00000004-006a-746c-6165-4861736e694b);
20     public UUID RESPONSE_CHARACTERISTICS = UUID.fromString(00000002-006a-746c-6165-4861736e694b);
21     public UUID BATTERY_SERVICE_UUID = UUID.fromString(0000180F-0000-1000-8000-00805f9b34fb);
22     public UUID BATTERY_VALUE_CHAR_UUID = UUID.fromString(00002A19-0000-1000-8000-00805f9b34fb);
23
```

UUID Extraction

```
1  public class KelvinDeviceProfile
2     private KelvinDeviceProfile(BlueToothLeGatt arg3)
3       super();
4       BluetoothGattService v0 = arg3.getService(KelvinGatt.KINSA_SERVICE);
5       if(v0!=null)
6         this.request = v0.getCharacteristic(KelvinGatt.REQUEST_CHARACTERISTICS);
7         this.response = v0.getCharacteristic(KelvinGatt.RESPONSE_CHARACTERISTICS);
8
9
10      BluetoothGattService v3 = arg3.getService(KelvinGatt.BATTERY_SERVICE_UUID);
11      if(v3!=null)
12        this.batterylevel = v3.getCharacteristic(KelvinGatt.BATTERY_VALUE_CHAR_UUID);
13
14
15
16
17 public class KelvinGatt
18    public UUID KINSA_SERVICE = UUID.fromString(00000000-006a-746c-6165-4861736e694b);
19    public UUID REQUEST_CHARACTERISTICS = UUID.fromString(00000004-006a-746c-6165-4861736e694b);
20    public UUID RESPONSE_CHARACTERISTICS = UUID.fromString(00000002-006a-746c-6165-4861736e694b);
21    public UUID BATTERY_SERVICE_UUID = UUID.fromString(0000180F-0000-1000-8000-00805f9b34fb);
22    public UUID BATTERY_VALUE_CHAR_UUID = UUID.fromString(00002A19-0000-1000-8000-00805f9b34fb);
23
```

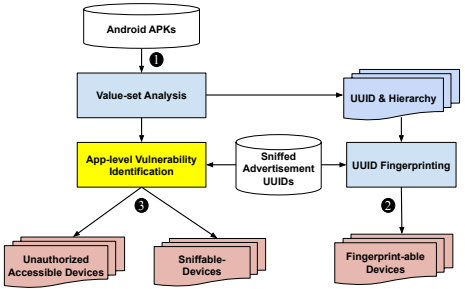
UUID Hierarchy Reconstruction

```
1  public class KelvinDeviceProfile
2     private KelvinDeviceProfile(BlueToothLeGatt arg3)
3       super();
4       BluetoothGattService v0 = arg3.getService(KelvinGatt.KINSA_SERVICE);
5       if(v0!=null)
6         this.request = v0.getCharacteristic(KelvinGatt.REQUEST_CHARACTERISTICS);
7         this.response = v0.getCharacteristic(KelvinGatt.RESPONSE_CHARACTERISTICS);
8
9
10      BluetoothGattService v3 = arg3.getService(KelvinGatt.BATTERY_SERVICE_UUID);
11      if(v3!=null)
12        this.batterylevel = v3.getCharacteristic(KelvinGatt.BATTERY_VALUE_CHAR_UUID);
13
14
15
16
17 public class KelvinGatt
18     public UUID KINSA_SERVICE = UUID.fromString(00000000-006a-746c-6165-4861736e694b);
19     public UUID REQUEST_CHARACTERISTICS = UUID.fromString(00000004-006a-746c-6165-4861736e694b);
20     public UUID RESPONSE_CHARACTERISTICS = UUID.fromString(00000002-006a-746c-6165-4861736e694b);
21     public UUID BATTERY_SERVICE_UUID = UUID.fromString(0000180F-0000-1000-8000-00805f9b34fb);
22     public UUID BATTERY_VALUE_CHAR_UUID = UUID.fromString(00002A19-0000-1000-8000-00805f9b34fb);
23
```

UUID Hierarchy Reconstruction

```
1  public class KelvinDeviceProfile
2     private KelvinDeviceProfile(BlueToothLeGatt arg3)
3       super();
4       BluetoothGattService v0 = arg3.getService(KelvinGatt.KINSA_SERVICE);
5       if(v0!=null)
6         this.request = v0.getCharacteristic(KelvinGatt.REQUEST_CHARACTERISTICS);
7         this.response = v0.getCharacteristic(KelvinGatt.RESPONSE_CHARACTERISTICS);
8
9
10      BluetoothGattService v3 = arg3.getService(KelvinGatt.BATTERY_SERVICE_UUID);
11      if(v3!=null)
12        this.batterylevel = v3.getCharacteristic(KelvinGatt.BATTERY_VALUE_CHAR_UUID);
13
14
15
16
17 public class KelvinGatt
18     public UUID KINSA_SERVICE = UUID.fromString(00000000-006a-746c-6165-4861736e694b);
19     public UUID REQUEST_CHARACTERISTICS = UUID.fromString(00000004-006a-746c-6165-4861736e694b);
20     public UUID RESPONSE_CHARACTERISTICS = UUID.fromString(00000002-006a-746c-6165-4861736e694b);
21     public UUID BATTERY_SERVICE_UUID = UUID.fromString(0000180F-0000-1000-8000-00805f9b34fb);
22     public UUID BATTERY_VALUE_CHAR_UUID = UUID.fromString(00002A19-0000-1000-8000-00805f9b34fb);
23
```

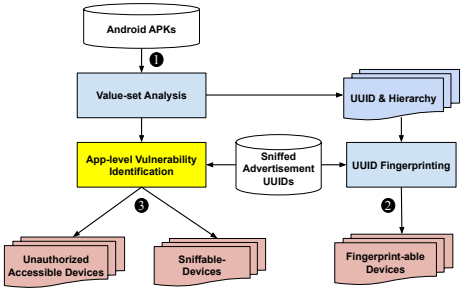
App-level Vulnerability Identification



Category	API Name
"Just Works"	BluetoothDevice: boolean createBond() BluetoothDevice.ACTION_BOND_STATE_CHANGED
Authentication	BluetoothGattCharacteristic: boolean setValue(String) BluetoothGattCharacteristic: boolean setValue(int,int,int) BluetoothGattCharacteristic: boolean setValue(byte[]) BluetoothGattCharacteristic: boolean setValue(int,int,int,int)
Cryptography	Cipher: byte[] doFinal(byte[]) Mac: byte[] doFinal(byte[]) MessageDigest: byte[] digest(byte[])

Table: APIs for app-level vulnerability identification

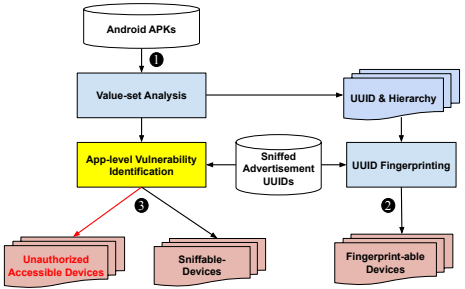
App-level Vulnerability Identification



Category	API Name
"Just Works"	BluetoothDevice: boolean createBond() BluetoothDevice.ACTION_BOND_STATE_CHANGED
Authentication	BluetoothGattCharacteristic: boolean setValue(String) BluetoothGattCharacteristic: boolean setValue(int,int,int) BluetoothGattCharacteristic: boolean setValue(byte[]) BluetoothGattCharacteristic: boolean setValue(int,int,int,int)
Cryptography	Cipher: byte[] doFinal(byte[]) Mac: byte[] doFinal(byte[]) MessageDigest: byte[] digest(byte[])

Table: APIs for app-level vulnerability identification

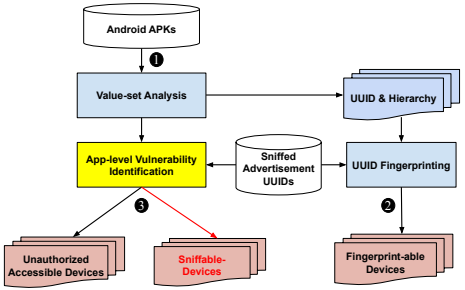
App-level Vulnerability Identification



Category	API Name
"Just Works"	BluetoothDevice: boolean createBond() BluetoothDevice.ACTION_BOND_STATE_CHANGED
Authentication	BluetoothGattCharacteristic: boolean setValue(String) BluetoothGattCharacteristic: boolean setValue(int,int,int) BluetoothGattCharacteristic: boolean setValue(byte[]) BluetoothGattCharacteristic: boolean setValue(int,int,int,int)
Cryptography	Cipher: byte[] doFinal(byte[]) Mac: byte[] doFinal(byte[]) MessageDigest: byte[] digest(byte[])

Table: APIs for app-level vulnerability identification

App-level Vulnerability Identification



Category	API Name
"Just Works"	BluetoothDevice: boolean createBond() BluetoothDevice.ACTION_BOND_STATE_CHANGED
Authentication	BluetoothGattCharacteristic: boolean setValue(String) BluetoothGattCharacteristic: boolean setValue(int,int,int) BluetoothGattCharacteristic: boolean setValue(byte[]) BluetoothGattCharacteristic: boolean setValue(int,int,int,int)
Cryptography	Cipher: byte[] doFinal(byte[]) Mac: byte[] doFinal(byte[]) MessageDigest: byte[] digest(byte[])

Table: APIs for app-level vulnerability identification

Companion Mobile App Collection

1. We downloaded 2 million mobile apps from Google Play as of April 2019.
2. We identified BLE IoT apps by searching for after-connection BLE APIs.
3. 18,166 BLE IoT apps are found for our analysis

Companion Mobile App Collection

1. We downloaded 2 million mobile apps from Google Play as of April 2019.
2. We identified BLE IoT apps by searching for after-connection BLE APIs.
3. 18,166 BLE IoT apps are found for our analysis

API Name
BluetoothGatt: List getServices
BluetoothGatt: BluetoothGattService getService
BluetoothGattService: UUID getUuid
BluetoothGattService: BluetoothGattCharacteristic getCharacteristic
BluetoothGattCharacteristic: UUID getUuid

Table: APIs used to identify the BLE related IoT apps

Result of UUID Extraction and Hierarchy Reconstruction

Item	Value	%
# Apps Collected	18,166	
# UUID Identified	168,093	
# Unique UUID Identified	13,566	
# UUID Hierarchy Edges	540,797	100.0
# UUID Hierarchy Service Edges	316,379	58.5
# UUID Hierarchy Characteristics Edges	224,418	41.5

Table: Experimental result of UUID extraction and hierarchy reconstruction.

Result of UUID Extraction and Hierarchy Reconstruction

opcode	# operations	opcode	# operations
+	79,743	—	1,398
/	9,684	&	1,266
*	5,364	>>>	894
<<	1,860	^	462
-	1,775	>>	17

Table: Operations to resolve UUIDs.

Result of UUID Extraction and Hierarchy Reconstruction

opcode	# operations	opcode	# operations
+	79,743	—	1,398
/	9,684	&	1,266
*	5,364	>>>	894
<<	1,860	^	462
-	1,775	>>	17

Table: Operations to resolve UUIDs.

# Apps Mapped to a Single UUID	Value	%
# 1	8,870	65.4
# 2	1,831	13.5
# 3	688	5.0
# 4	469	3.5
# 5	330	2.4
# ≥ 6	1,378	10.1

Table: Mapping between UUID and apps.

Result of App-level Vulnerability Identification

Item	Value	%
# Apps Support BLE	18,166	100.0
# "Just Works" Pairing	11,141	61.3
# Vulnerable Apps	1,757	15.8
# Absent Cryptographic Usage	1,510	13.6
# Flawed Authentication	1,434	12.9

Table: Insecure app identification result.

Result of App-level Vulnerability Identification

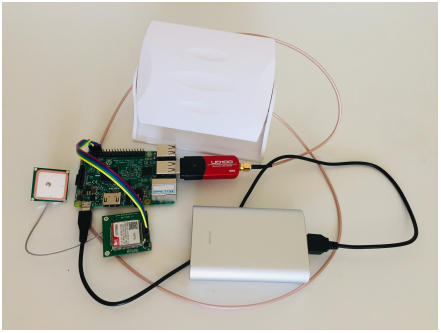
Item	Value	%
# Apps Support BLE	18,166	100.0
# "Just Works" Pairing	11,141	61.3
# Vulnerable Apps	1,757	15.8
# Absent Cryptographic Usage	1,510	13.6
# Flawed Authentication	1,434	12.9

Table: Insecure app identification result.

Category	# App	"Just Works"	Absent Crypto	Flawed Auth.
Health & Fitness	3,849	2,639	221	207
Tools	2,833	1,895	385	362
Lifestyle	2,173	1,081	147	141
Business	1,660	972	90	85
Travel & Local	967	582	90	87

Table: Top 5 category of the IoT apps.

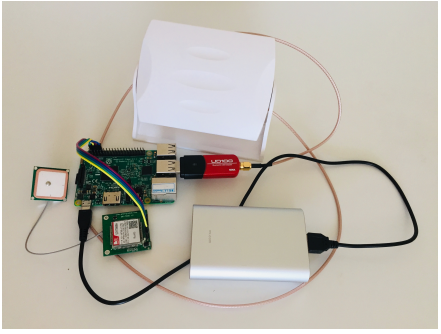
Field Test Environment Setup



BLE Sniffer

- Raspberry-Pi
- Parani-UD100 (Bluetooth adapter)
- Antenna RP-SMA-R/A (1km amplifier)
- SIM7000A GPS module (GPS sensor)

Field Test Environment Setup



Field Test Result

Item	Value	%
# Unique BLE Device	30,862	
# Unique BLE Device w. UUID	5,822	18.9
# Fingerprintable	5,509	94.6
# Vulnerable	431	7.4
# Sniffable	369	6.7
# Unauthorized Accessible	342	6.2

Table: Experimental result of our field test.

Field Test Result

Company Name	# Devices
Google	2,436
Tile, Inc.	441
-	243
-	208
Logitech International SA	131
Nest Labs Inc.	114
Google	92
Hewlett-Packard Company	74
-	46
-	44
-	44

Table: Top 10 devices in the field test.

Field Test Result

Company Name	# Devices
Google	2,436
Tile, Inc.	441
-	243
-	208
Logitech International SA	131
Nest Labs Inc.	114
Google	92
Hewlett-Packard Company	74
-	46
-	44
-	44

Table: Top 10 devices in the field test.

Field Test Result

Device Description	# Device
Digital Thermometer	7
Car Dongle	6
Key Finder A	6
Smart Lamp	5
Key Finder B	5
Smart Toy A	4
Smart VFD	4
Air Condition Sensor	4
Smart Toy B	4
Accessibility Device	4

Table: Top 10 **vulnerable** devices.

Anti-UUID Fingerprinting

Countermeasures

1. **App-level protection**. Use obfuscation [HGM18], encoding, encryption, or cloud to hide UUIDs in mobile apps.
2. **Channel-level protection**. BLE-Guardian [FKS16]

Anti-UUID Fingerprinting

Countermeasures

1. **App-level protection**. Use obfuscation [HGM18], encoding, encryption, or cloud to hide UUIDs in mobile apps.
2. **Channel-level protection**. BLE-Guardian [FKS16]

Drawbacks

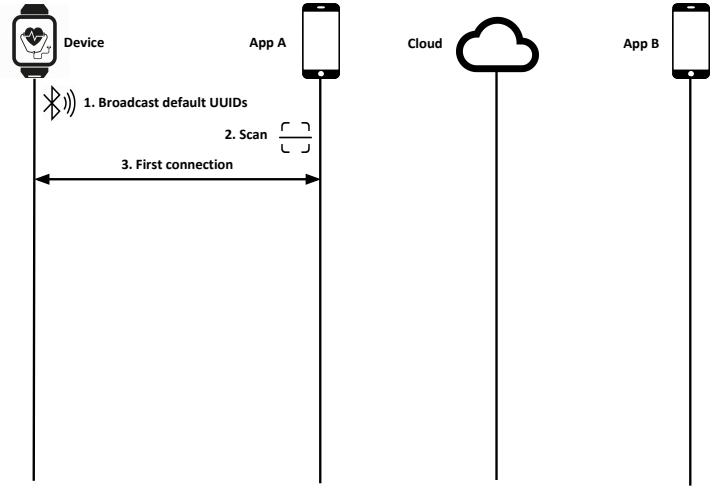
1. UUIDs are statically constructed and can still be retrieved from apps.
2. Additional hardware support is required.
3. Not fundamental solutions.

Anti-UUID Fingerprinting

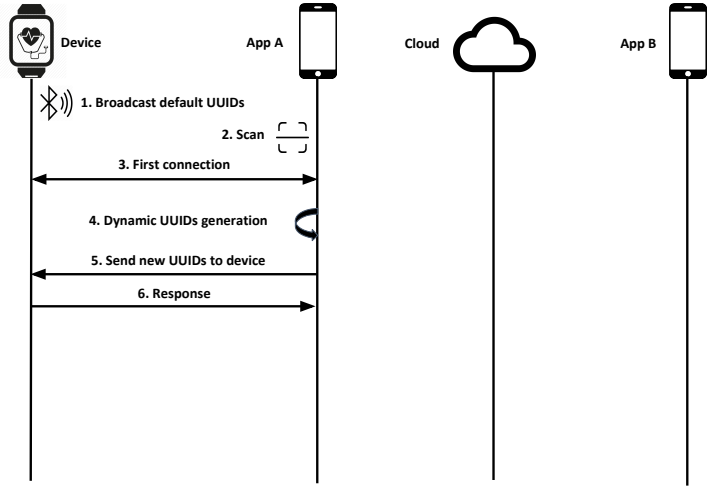
Countermeasures

1. **App-level protection**. Use obfuscation [HGM18], encoding, encryption, or cloud to hide UUIDs in mobile apps.
2. **Channel-level protection**. BLE-Guardian [FKS16]
3. **Protocol-level protection**. Construct one-time dynamic UUIDs for broadcast and communication.

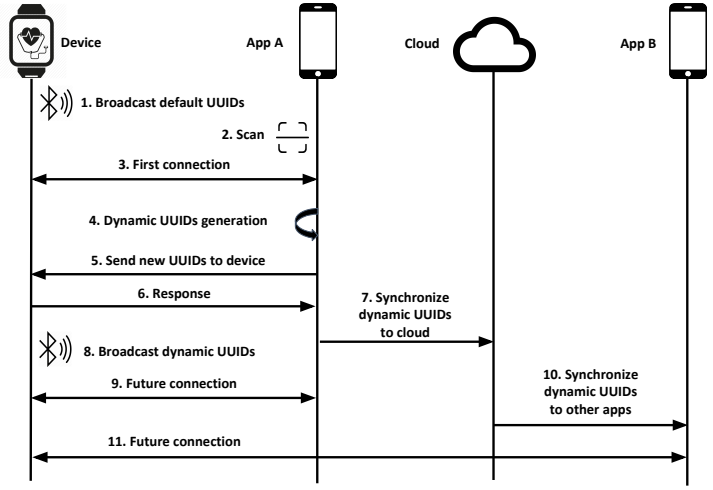
Dynamic UUID Generation



Dynamic UUID Generation



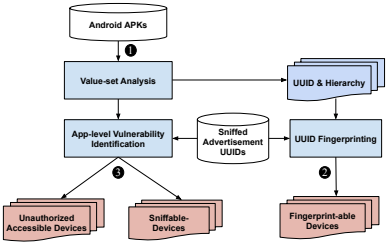
Dynamic UUID Generation

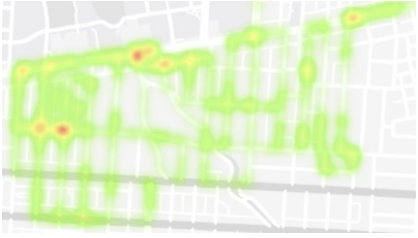


Related Work

1. **IoT Security**.
   - Vulnerability discovery of IoT devices. Credential leakage [CAWM17, CHMS14], unchanged address [BMI08, DPCM16], privilege misconfiguration [FJP16, HLM+16], unencrypted channel [ZL17a] and memory corruption [CDZ+18].
   - Defenses of vulnerabilities [FPR+16, DMK+12, TZL+17, FKS16].
2. **BLE Security**. Insecure pairing protocol and eavesdropping attack [Rya13]. MITM attacks [SBA18, SMS18], and brute force attack to break long term pairing key [Zeg15].
3. **Vulnerability discovery based on mobile apps analysis**.
   - Client Side: FlowDroid [ARF+14], Amandroid [WROR14], TaintDroid [EGC+10], PiOS [EKKV11], CHEX [LLW+12], SMV-Hunter [SSG+14].
   - Server Side: AutoForge [ZWWL16], SmartGen [ZL17b], AuthScope [ZZL17], LeakScope [ZLZ19], WARDroid [MG18].

BLEScope





BLEScope

- Automatic UUID extraction and hierarchy reconstruction from mobile apps
- Identify app-level vulnerabilities by directly analyzing mobile apps

App Analysis and Field Test Result

- We analyzed 18,166 apps and discovered 168,093 UUIDs and 1,757 vulnerable apps
- 5,822 BLE devices were discovered in the field test, and 94.6% can be fingerprinted

Limitations and Future Work

1. **Fingerprinting precision**. We did not use the hierarchy UUIDs to fingerprint the device. This is due to ethical consideration, since it requires to fetch the data from the devices to construct the hierarchy of UUIDs (unauthorized access).

Limitations and Future Work

1. **Fingerprinting precision**. We did not use the hierarchy UUIDs to fingerprint the device. This is due to ethical consideration, since it requires to fetch the data from the devices to construct the hierarchy of UUIDs (unauthorized access).
2. **False negatives**. We applied a strict rule to detect flawed authentication in apps.

Limitations and Future Work

1. **Fingerprinting precision**. We did not use the hierarchy UUIDs to fingerprint the device. This is due to ethical consideration, since it requires to fetch the data from the devices to construct the hierarchy of UUIDs (unauthorized access).
2. **False negatives**. We applied a strict rule to detect flawed authentication in apps.
3. **Branch explosion**. The backward slicing attempts to exhaustively explore all possible branches. We will terminate our analysis for such apps.

Limitations and Future Work

1. **Fingerprinting precision**. We did not use the hierarchy UUIDs to fingerprint the device. This is due to ethical consideration, since it requires to fetch the data from the devices to construct the hierarchy of UUIDs (unauthorized access).
2. **False negatives**. We applied a strict rule to detect flawed authentication in apps.
3. **Branch explosion**. The backward slicing attempts to exhaustively explore all possible branches. We will terminate our analysis for such apps.
4. **Optional UUIDs**. UUIDs do not always exist in BLE broadcast packets [BLS19]. No mobile apps, no need to broadcast UUIDs. (In our field test, we found 25k such BLE devices.)

Thank You

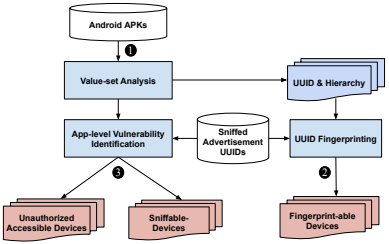
Automatic Fingerprinting Of Vulnerable BLE IoT Devices With Static UUIDs From Mobile Apps

Chaoshun Zuo, **Haohuang Wen**, Zhiqiang Lin, and Yinqian Zhang

Department of Computer Science and Engineering
The Ohio State University

CCS 2019

Takeaway



BLEScope

- Automatic UUID extraction and hierarchy reconstruction from mobile apps
- Identify app-level vulnerabilities by directly analyzing mobile apps

App Analysis and Field Test Result

- We analyzed 18,166 apps and discovered 168,093 UUIDs and 1,757 vulnerable apps
- 5,822 BLE devices were discovered in the field test, and 94.6% can be fingerprinted

References I

Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel, *Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps*, Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (New York, NY, USA), PLDI '14, ACM, 2014, pp. 259–269.

Johannes K Becker, David Li, and David Starobinski, *Tracking anonymized bluetooth devices*, Proceedings on Privacy Enhancing Technologies **2019** (2019), no. 3, 50–65.

Redjem Bouhenguel, Imad Mahgoub, and Mohammad Ilyas, *Bluetooth security in wearable computing applications*, 2008 international symposium on high capacity optical networks and enabling technologies, IEEE, 2008, pp. 182–186.

Brian Cusack, Bryce Antony, Gerard Ward, and Shaunak Mody, *Assessment of security vulnerabilities in wearable devices*.

Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang, *Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing.*, NDSS, 2018.

Britt Cyr, Webb Horn, Daniela Miao, and Michael Specter, *Security analysis of wearable fitness devices (fitbit)*, Massachusetts Institute of Technology **1** (2014).

Charalampos Doukas, Ilias Maglogiannis, Vassiliki Koufi, Flora Malamateniou, and George Vassilacopoulos, *Enabling data protection through pki encryption in iot m-health devices*, 2012 IEEE 12th International Conference on Bioinformatics & Bioengineering (BIBE), IEEE, 2012, pp. 25–29.

References II

Aveek K Das, Parth H Pathak, Chen-Nee Chuah, and Prasant Mohapatra, *Uncovering privacy leakage in ble network traffic of wearable fitness trackers*, Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, ACM, 2016, pp. 99–104.

W. Enck, P. Gilbert, B.G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N. Sheth, *TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones*, OSDI, 2010.

M. Egele, C. Kruegel, E. Kirda, and G. Vigna, *Pios: Detecting privacy leaks in ios applications*, NDSS, 2011.

Earlence Fernandes, Jaeyeon Jung, and Atul Prakash, *Security analysis of emerging smart home applications*, 2016 IEEE Symposium on Security and Privacy (SP), IEEE, 2016, pp. 636–654.

Kassem Fawaz, Kyu-Han Kim, and Kang G Shin, *Protecting privacy of {BLE} device users*, 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 1205–1221.

Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash, *Flowfence: Practical data protection for emerging iot application frameworks*, 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 531–548.

Mahmoud Hammad, Joshua Garcia, and Sam Malek, *A large-scale empirical study on the effects of code obfuscations on android apps and anti-malware products*, Proceedings of the 40th International Conference on Software Engineering, ACM, 2018, pp. 421–431.

Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner, *Smart locks: Lessons for securing commodity internet of things devices*, Proceedings of the 11th ACM on Asia conference on computer and communications security, ACM, 2016, pp. 461–472.

References III

Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang, *Chex: statically vetting android apps for component hijacking vulnerabilities*, Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 229–240.

Abner Mendoza and Guofei Gu, *Mobile application web api reconnaissance: Web-to-mobile inconsistencies & vulnerabilities*, 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 756–769.

Mike Ryan, *Bluetooth: With low energy comes low security*, Presented as part of the 7th {USENIX} Workshop on Offensive Technologies, 2013.

Pallavi Sivakumaran and Jorge Blasco Alis, *A low energy profile: Analysing characteristic security on ble peripherals*, Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, ACM, 2018, pp. 152–154.

Da-Zhi Sun, Yi Mu, and Willy Susilo, *Man-in-the-middle attacks on secure simple pairing in bluetooth standard v5. 0 and its countermeasure*, Personal and Ubiquitous Computing **22** (2018), no. 1, 55–67.

David Sounthiraraj, Justin Sahs, Garrett Greenwood, Zhiqiang Lin, and Latifur Khan, *Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps*, Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14) (San Diego, CA), February 2014.

Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague, *Smartauth: User-centered authorization for the internet of things*, 26th {USENIX} Security Symposium ({USENIX} Security 17), 2017, pp. 361–378.

References IV

Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby, *Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps*, Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA), CCS '14, ACM, 2014, pp. 1329–1341.

Wondimu K Zegeye, *Exploiting bluetooth low energy pairing vulnerability in telemedicine*, International Foundation for Telemetering, 2015.

Qiaoyang Zhang and Zhiyao Liang, *Security analysis of bluetooth low energy based smart wristbands*, 2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST), IEEE, 2017, pp. 421–425.

Chaoshun Zuo and Zhiqiang Lin, *Exposing server urls of mobile apps with selective symbolic execution*, Proceedings of the 26th World Wide Web Conference (Perth, Australia), April 2017.

Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang, *Why does your data leak? uncovering the data leakage in cloud from mobile apps*, Proc. IEEE Symposium on Security and Privacy, 2019.

Chaoshun Zuo, Wubing Wang, Rui Wang, and Zhiqiang Lin, *Automatic forgery of cryptographically consistent messages to identify security vulnerabilities in mobile services*, Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'16) (San Diego, CA), February 2016.

Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin, *Authscope: Towards automatic discovery of vulnerable authorizations in online services*, Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS'17) (Dallas, TX), November 2017.