IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing

Jiongyi Chen¹, Wenrui Diao², Qingchuan Zhao³, Chaoshun Zuo³, Zhiqiang Lin³,⁴, XiaoFeng Wang⁵, Wing Cheong Lau¹, Menghan Sun¹, Rongai Yang¹, and Kehuan Zhang¹

Chinese University of Hong Kong¹, Jinan University², University of Texas at Dallas³, Ohio State University⁴, Indiana University Bloomington⁵

NDSS 2018

Presented By
Md Mahbubur Rahman
Wayne State University
Outline

• IoT Trend
• Motivation
• IoTFuzzer (This paper)
• Challenges
• Architecture: IoTFuzzer
• Implementation and Evaluation
• Conclusion
Internet of Things (IoT) Market

• Applications
  • Smart Home, Smart City, Agricultural IoT, etc.

• Market growth by 2020
  • 20.4 billion IoT devices
  • $3 trillion

• Smart Home
  • $53.45 billion by 2022

Smart Home market value
(Source: Zion Research Analysis 2017)
Is IoT Secure?

• NOT really!

• Attacks: 2014-2016
  • Mo

• Mirai ℓ
  • Onl
  • Disl

• Reaper botnet attack

Firmwares of the IoT devices are not properly implemented & protected!!
What’s Done!

• Few attempts have been made that closely deal with firmwares.
  [Davidson et al. USENIX Sec.’13, Cui et al. NDSS’13, Chen Black Hat’09, Shoshitaishvili et al. NDSS’15]

• Limitations
  • Firmware acquisition: vendors may not make it public
  • Firmware identification & unpacking: unknown architecture, proprietary compression/encryption
  • Executable analysis: requires lots of manual efforts and is not accurate

It is worth looking into the IoT official applications
IoT Official Application

• Controls and manages IoT applications

Contains rich information about the IoT system

Major data input channel of IoT device

Command messages

Protocol specifications & encryption schemes of messages

Courtesy: Authors
IoTFuzzer: A Firmware-free Fuzzing Framework

- Detects memory corruptions in IoT devices
  - Null-pointer exceptions, buffer overflow, out-of-bound accesses, etc.

- Leverages official apps and program logics to create meaningful test messages

- Fuzzes in a protocol-guided way without explicitly reverse engineering the protocols
IoTFuzzer: Challenges

• Diverse data formats and protocols
  • XML, JSON, key-value pairs

• Proprietary cryptographic functions

• Crash monitoring
  • How to determine the real-time status of the device?

TP-Link Kasa Code Snippet

```java
// Message construction
public final ControlResult a(...) {
...
Object localObject = new com/tplink/smarthome/b/e;
((e)localObject).<init>("system");
g localalg = new com/tplink/smarthome/b/g;
localalg.<init>("set_dev_location");
...
localg.a("longitude", localDouble);
localDouble = Double.valueOf(paramDouble1);
localg.a("latitude", localDouble);
...
return (ControlResult)localObject;
}
// Message: {"system":"set_dev_location":"longitude":10.111213141,"latitude ":51.617181920}}

//Message encryption
public static byte[] a(byte[] paramArrayOfByte) {
...
k = paramArrayOfByte[j];
i = (byte)(i ^ k);
paramArrayOfByte[j] = i;
i = paramArrayOfByte[j] = i;
j += 1;
...
return paramArrayOfByte;
}
```
IoTFuzzer: Solutions

• Diverse data formats and protocols
  • Mutate protocol fields before they are constructed as message

• Proprietary cryptographic functions
  • Reuse cryptographic functions in the runtime

• Crash monitoring
  • Insert heartbeat messages
IoTFuzzer: Scope and Assumptions

• Goal: Automatically generate protocol-aware messages to the IoT devices to discover memory corruptions

• Assumptions
  • IoT device under testing are configurable and controllable with mobile apps
  • Wi-Fi communication protocol
  • Android apps
IoTFuzzer: Architecture

• 2-phase architecture

• Phase 1:
  • App analysis
IoTFuzzer: Architecture

- 2-phase architecture

- **Phase 1:**
  - App analysis

- **Phase 2:**
  - Fuzzing
IoTFuzzer: Architecture – Phase 1

- UI Analysis

  - Call Path Construction
    - Identify networking UI elements by constructing call paths from networking APIs to UI event handlers
    - Networking APIs: URL.openConnection(), Socket.getOutputStream(), etc
    - Androguard [1]

  - Activity Transition Graph Construction
    - To trigger networking API events
    - Monkeyrunner [2]

IoTFuzzer: Architecture – Phase 1

• Taint Analysis
  • Identify protocol fields (variables) and functions
  • TaintDroid [W. Enck et al. TOCS’14]

• Taint Sources: strings, system APIs, user inputs

• Taint Sinks: data used at networking APIs and encryption functions

• Cryptographic Function Identification
  • Lots of related work
  • IoTFuzzer employs a lightweight technique
  • Cryptographic functions contain arithmetic operations and called during the message delivery execution
IoTFuzzer: Architecture – Phase 1

Code example

```java
// Message construction
public final ControlResult a(...) {
    ...
    Object localObject = new com/tplink/smarthome/b/e;
    ((e)localObject).<init>("system");
    g localg = new com/tplink/smarthome/b/g;
    localg.<init>("set_dev_location");
    ...
    localg.a("longitude", localDouble);
    localDouble = Double.valueOf(paramDouble1);
    localg.a("latitude", localDouble);
    ...
    return (ControlResult)localObject;
}

// Message: {"system":{"set_dev_location":{"longitude":10.111213141,"latitude ":51.617181920}}}

// Message encryption
public static byte[] a(byte[]
    paramArrayOfByte) {
```
IoTFuzzer: Architecture – Phase 2

- Runtime Mutation
  - Function Hooking
    - Dynamically hooks the recorded functions and mutate the protocol fields at runtime to generate probe messages
    - Xposed [3]
  - Fuzzing Scheduling: to fuzz only a subset of all protocol fields

- Fuzzing Policy:
  - Change the length of the strings to check overflow and out-of-bound access
  - Change integer, double, or float (large values) to check overflow and out-of-bound access
  - Change object types and provide empty values to check misinterpretation and null-pointer exeception

IoTFuzzer: Architecture – Phase 2

- Response monitoring
  - Response Types
    - Expected response
    - Unexpected response
    - No response
    - Disconnection
  
  - Crash Detection
    - TCP-based connection: disconnection
    - UDP-based connection: insert a heartbeat message after every 10 probe messages
Implementation

- Implemented on 17 off-the-shelf IoT devices (apps are available on Google Play)

<table>
<thead>
<tr>
<th>Device Type</th>
<th>Vendor</th>
<th>Device Model</th>
<th>Firmware Version</th>
<th>Protocol and Format (Encrypted: Yes/No)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Camera</td>
<td>D-Link</td>
<td>DCS-5010L</td>
<td>1.13</td>
<td>HTTP, K-V Pairs (N)</td>
</tr>
<tr>
<td>Smart Bulb</td>
<td>TP-Link</td>
<td>LB100</td>
<td>1.1.2</td>
<td>UDP, JSON (Y)</td>
</tr>
<tr>
<td></td>
<td>KONKE</td>
<td>KK-Light</td>
<td>1.1.0</td>
<td>UDP, String (Y)</td>
</tr>
<tr>
<td>Smart Plug</td>
<td>Belkin</td>
<td>Wemo Switch</td>
<td>2.00</td>
<td>HTTP, XML (N)</td>
</tr>
<tr>
<td></td>
<td>TP-Link</td>
<td>HS110</td>
<td>v1_151016</td>
<td>TCP, JSON (Y)</td>
</tr>
<tr>
<td></td>
<td>D-Link</td>
<td>DSP-W215</td>
<td>1.02</td>
<td>HNAP, XML (N)</td>
</tr>
<tr>
<td>Printer</td>
<td>Brother</td>
<td>HL-L5100DN</td>
<td>Ver. E</td>
<td>LPD &amp; HTTP, URI (N)</td>
</tr>
<tr>
<td>NAS</td>
<td>Western Digital</td>
<td>My Passport Pro</td>
<td>1.01.08</td>
<td>HTTP, JSON (N)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>My Cloud</td>
<td>2.21.126</td>
<td>HTTP, JSON (N)</td>
</tr>
<tr>
<td></td>
<td>QNAP</td>
<td>TS-212P</td>
<td>4.2.2</td>
<td>HTTP, K-V Pairs (N)</td>
</tr>
<tr>
<td>IoT Hub</td>
<td>Philips</td>
<td>Hue Bridge</td>
<td>01036659</td>
<td>HTTP, JSON (N)</td>
</tr>
<tr>
<td>Home Router</td>
<td>NETGEAR</td>
<td>N300</td>
<td>1.0.0.34</td>
<td>HTTP, XML (N)</td>
</tr>
<tr>
<td></td>
<td>Linksys</td>
<td>E1200</td>
<td>2.0.7</td>
<td>HNAP, XML (N)</td>
</tr>
<tr>
<td></td>
<td>Xiaomi</td>
<td>Xiaomi Router</td>
<td>2.19.32</td>
<td>HTTP, K-V Pairs (N)</td>
</tr>
<tr>
<td>Story Teller</td>
<td>Xiaomi</td>
<td>C-1</td>
<td>1.2.4_89</td>
<td>UDP, JSON (Y)</td>
</tr>
<tr>
<td>Extension Socket</td>
<td>KONKE</td>
<td>Mini-K Socket</td>
<td>sva.1.4</td>
<td>UDP, String (Y)</td>
</tr>
<tr>
<td>Humidifier</td>
<td>POVOSS</td>
<td>PW103</td>
<td>v2.0.1</td>
<td>UDP, String (Y)</td>
</tr>
</tbody>
</table>
Evaluation

• Testing Environment
  • UI Analysis: Ubuntu 14-04 Intel Core i7 quad-core 2.81 GHz CPU 8GB RAM
  • Taint Tracking: Google’s Nexus 4
  • Network: Fully controlled local Wi-Fi

• 15 memory corruptions were found including 8 previously unknown

<table>
<thead>
<tr>
<th>Device</th>
<th>Vulnerability Type</th>
<th># of Issues</th>
<th>Remotely Exploitable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belkin WeMo (Switch)</td>
<td>Null Pointer Dereference</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>TP-Link HS110 (Plug)</td>
<td>Null Pointer Dereference</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>D-Link DSP-W215 (Plug)</td>
<td>Buffer Overflow (Stack-based)</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>WD My Cloud (NAS)</td>
<td>Buffer Overflow (Stack-based)</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>QNAP TS-212P (NAS)</td>
<td>Buffer Overflow (Heap-based)</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Brother HL-L5100DN (Printer)</td>
<td>Unknown Crash</td>
<td>1</td>
<td>Not determined</td>
</tr>
<tr>
<td>Philips Hue Bridge (Hub)</td>
<td>Unknown Crash</td>
<td>1</td>
<td>Not determined</td>
</tr>
<tr>
<td>WD My Passport Pro (NAS)</td>
<td>Unknown Crash</td>
<td>1</td>
<td>Not determined</td>
</tr>
<tr>
<td>POVOS PW103 (Humidifier)</td>
<td>Unknown Crash</td>
<td>1</td>
<td>Not determined</td>
</tr>
</tbody>
</table>
Evaluation

• Fuzzing accuracy
Conclusion

• IoTfuzzer: Limitations
  • Only support Wi-Fi connections
  • Can only fuzz app-related code in IoT devices
  • Only detects memory related corruptions that lead to crashes
Questions?