Unlocking the Potential of Domain Aware Binary Analysis in the Era of IoT

Zhiqiang Lin

zlin@cse.ohio-state.edu

April 18th, 2023
History of Computing (Since 1980s)
History of Computing (Since 1980s)

- PC 1981
- TCP/IP & Internet 1982
- WWW 1989
- Netscape 1994
- Windows 1995
History of Computing (Since 1980s)

- PC 1981
- TCP/IP & Internet 1982
- WWW 1989
- Netscape 1994
- Windows 1995
- WiFi 1999
- 3G/4G 2000s
- Android 2005
- iPhone 2007
- Internet Mobile
- TCP/IP & Internet 1982
- WWW 1989
- 3G/4G 2000s
- Netscape 1994
- Windows 1995
- WiFi 1999
- Android 2005
- iPhone 2007
History of Computing (Since 1980s)

- **PC**: 1981
- **TCP/IP & Internet**: 1982
- **WWW**: 1989
- **Netscape**: 1994
- **Windows**: 1995
- **WiFi**: 1999
- **3G/4G 2000s**: 2000s
- **Android**: 2005
- **iPhone**: 2007
- **iPhone 2010s**: 2010s
- **Nest Thermostat**: 2011
- **Tesla Model S**: 2012
- **Amazon Alexa**: 2014
- **TCP/IP & Internet**: 1982
- **WWW**: 1989
- **IoT**: 2010s
- **Nest Thermostat**: 2011
- **Tesla Model S**: 2012
- **Amazon Alexa**: 2014
History of Computing (Since 1980s)

- **Internet**
  - PC 1981
  - TCP/IP & Internet 1982
  - WWW 1989
  - Netscape 1994
  - Windows 1995
  - WiFi 1999
  - 3G/4G 2000s
  - Android 2005
  - iPhone 2007
  - IoT 2010s
  - Nest Thermostat 2011
  - Tesla Model S 2012
  - Amazon Alexa 2014

- **Mobile**
  - Edge Computing 2015
  - 5G 2019
  - COVID-19 Pandemic 2020

- **IoT**
  - Alexa 2014
  - PC 1981
  - TCP/IP & Internet 1982
  - WWW 1989
  - Netscape 1994
  - Windows 1995
  - WiFi 1999
  - 3G/4G 2000s
  - Android 2005
  - iPhone 2007
  - IoT 2010s
  - Nest Thermostat 2011
  - Tesla Model S 2012
  - Amazon Alexa 2014
  - Edge Computing 2015
  - 5G 2019
  - COVID-19 Pandemic 2020

- **TCP/IP & Internet**
  - 1982

- **WWW**
  - 1989

- **Mobile**
  - 2007

- **Android**
  - 2005

- **PC**
  - 1981

- **Netscape**
  - 1994

- **Windows**
  - 1995

- **WiFi**
  - 1999

- **3G/4G**
  - 2000s

- **Nest Thermostat**
  - 2011

- **Tesla Model S**
  - 2012

- **Amazon Alexa**
  - 2014

- **Edge Computing**
  - 2015

- **5G**
  - 2019

- **COVID-19 Pandemic**
  - 2020
History of Computing (Since 1980s)

- **Internet**
  - PC: 1981
  - TCP/IP & Internet: 1982
  - WWW: 1989
  - Netscape: 1994
  - Windows: 1995
  - Wi-Fi: 1999
  - 3G/4G: 2000s
  - Android: 2005
  - iPhone: 2007
  - IoT: 2010s
  - Nest Thermostat: 2011
  - Tesla Model S: 2012
  - Amazon Alexa: 2014
  - Edge Computing: 2015

- **Mobile**
  - PC: 1981
  - 3G/4G: 2000s
  - Netscape: 1994

- **IoT**
  - PC: 1981
  - 5G: 2019
  - Edge Computing: 2015

Data Source: statista
History of Computing (Since 1980s)

- PC 1981
- TCP/IP & Internet 1982
- WWW 1989
- Netscape 1994
- Windows 1995
- WiFi 1999
- 3G/4G 2000s
- Android 2005
- iPhone 2007
- IoT 2010s
- Nest Thermostat 2011
- Tesla Model 3 2012
- Amazon Alexa 2014
- Edge Computing 2015
- 5G 2019
- COVID-19 Pandemic 2020
- Stuxnet 2010
- TRENDSnet Webcam Hack 2012
- Jeep Cherokee hack 2015
- Mirai Botnet 2016
- Owlet Heart Monitor Hack 2016
- Tesla Free-Fall Hack 2017
- Verkada Camera Hack 2021
Modern IoT Architecture

An Embedded IoT Device

Application
- App1
- App2
- App3
- App4
- ...

Middleware
- APIs
- Modules
- Services
- SDKs
- Libraries
- Framework

OS / Bare-Metal
- HAL

Hardware
- Processor
- RAM
- Storage
- I/O

Peripherals
- Sensor
- Actuator
- USB
- Camera
- Clock
- Microphone
- Keyboard
Modern IoT Architecture

An Embedded IoT Device

Application
- App1
- App2
- App3
- App4
- ...

Middleware
- APIs
- SDKs
- Libraries
- Framework

OS / Bare-Metal

Hardware
- Processor
- RAM
- Storage
- I/O

Peripherals
- Sensor
- Actuator
- USB
- Camera
- Clock
- Microphone
- Keyboard

An Embedded IoT Device

Cloud

Router

User
Modern IoT Architecture

An Embedded IoT Device

Application
- App1
- App2
- App3
- App4
- ...

Middleware
- APIs
- Modules
- Services
- SDKs
- Libraries
- Framework

OS / Bare-Metal

HAL

Hardware
- Processor
- RAM
- Storage
- I/O

Peripherals
- Sensor
- Actuator
- USB
- Camera
- Clock
- Microphone
- Keyboard

Motivations

QtRE

FirmXRay

AutoMap

Takeaway

References
Modern IoT Architecture

An Embedded IoT Device

Application
- App1
- App2
- App3
- App4
- ...

Middleware
- APIs
- Modules
- Services
- SDKs
- Libraries
- Framework

OS / Bare-Metal

HAL

Hardware
- Processor
- RAM
- Storage
- I/O

Peripherals
- Sensor
- Actuator
- USB
- Camera
- Clock
- Microphone
- Keyboard
Modern IoT Architecture

An Embedded IoT Device

Application
- App1
- App2
- App3
- App4
- ...

Middleware
- APIs
- Modules
- Services
- SDKs
- Libraries
- Framework

OS / Bare-Metal

HAL

Hardware
- Processor
- RAM
- Storage
- I/O

Peripherals
- Sensor
- Actuator
- USB
- Camera
- Clock
- Microphone
- Keyboard

RTOS

AWS IoT

Azure IoT Hub

Qt

MQTT

Watson IoT

Intel Atom
Modern IoT Architecture

Application

- App1
- App2
- App3
- App4
- ...

Middleware

- APIs
- Modules
- Services
- SDKs
- Libraries
- Framework

OS / Bare-Metal

HAL

Hardware

- Processor
- RAM
- Storage
- I/O

Peripherals

- Sensor
- Actuator
- USB
- Camera
- Clock
- Microphone
- Keyboard

An Embedded IoT Device
Domain-Aware Binary Analysis

Binary code analysis is *challenging*

- Control flow recovery, semantic understanding, vulnerability detection, root-cause analysis...
Domain-Aware Binary Analysis

Binary code analysis is *challenging*

- Control flow recovery, semantic understanding, vulnerability detection, root-cause analysis...

Why Domain-Aware

*One size does not fit all*

- Heterogeneous architecture, OS, APIs of different IoT vendors
- Domain-specific challenges
Domain-Aware Binary Analysis

Binary code analysis is *challenging*

- Control flow recovery, semantic understanding, vulnerability detection, root-cause analysis...

Why Domain-Aware

1. *One size does not fit all*
   - Heterogeneous architecture, OS, APIs of different IoT vendors
   - Domain-specific challenges

2. *Learn from the domain*
   - Unique domain insights for binary analysis
   - Novel techniques and methodology
   - Transition to other domains
Our Recent Works on (IoT) Binary Analysis

Our Recent Works on (IoT) Binary Analysis

2. Understanding IoT Security from a Market-Scale Perspective. In CCS 2022
Our Recent Works on (IoT) Binary Analysis

2. Understanding IoT Security from a Market-Scale Perspective. In CCS 2022
Our Recent Works on (IoT) Binary Analysis

2. Understanding IoT Security from a Market-Scale Perspective. In CCS 2022
Our Recent Works on (IoT) Binary Analysis

2. Understanding IoT Security from a Market-Scale Perspective. In CCS 2022
5. FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities from Bare-Metal Firmware. In CCS 2020
Our Recent Works on (IoT) Binary Analysis

2. Understanding IoT Security from a Market-Scale Perspective. In CCS 2022
5. FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities from Bare-Metal Firmware. In CCS 2020
6. Plug-N-Pwned: Comprehensive Vulnerability Analysis of OBD-II Dongles as A New Over-the-Air Attack Surface in Automotive IoT. In USENIX Security 2020
Our Recent Works on (IoT) Binary Analysis

2. Understanding IoT Security from a Market-Scale Perspective. In CCS 2022
5. FirmXRay: Detecting Bluetooth Link Layer Vulnerabilities from Bare-Metal Firmware. In CCS 2020
6. Plug-N-Pwned: Comprehensive Vulnerability Analysis of OBD-II Dongles as A New Over-the-Air Attack Surface in Automotive IoT. In USENIX Security 2020
Background
Background
Background
Background

Across 70+ industries!

Motivations
QtRE
FirmXRay
AutoMap
Takeaway
References
Tesla’s Infotainment System
Tesla’s Infotainment System
Tesla’s Infotainment System
Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
- Existing C++ binary analysis tools can be applied [ghi, SWS+16]
Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
- Existing C++ binary analysis tools can be applied [ghi, SWS+16]

Binary RE Challenges

- Control Flow Graph (CFG) Recovery. Indirect control flow transfers such as callbacks and indirect calls [PCvdV+17, VDVGC+16]
Motivation

Enabling Security Analysis of Qt Programs

- Reverse engineering (RE) is one of the keys to vet Qt binaries
- Existing C++ binary analysis tools can be applied [ghi, SWS+16]

Binary RE Challenges

- **Control Flow Graph (CFG) Recovery.** Indirect control flow transfers such as callbacks and indirect calls [PCvdV+17, VDVGC+16]
- **Symbol Recovery** (e.g., names/types of functions/variables). Code stripping during binary compilation [TTN+19, SCD+18]
Key Insights

Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - We instead leverage it to identify Qt-specific function callbacks

2. Qt’s Dynamic Introspection
   - Originally designed for run-time class member query and update
   - We repurpose it to recover rich semantic symbols from the binary program
Key Insights

Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*
Key Insights

Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*

2. Qt’s Dynamic Introspection
Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*

2. Qt’s Dynamic Introspection
   - Originally designed for run-time class member query and update
Unique Insights from Qt’s Mechanisms

1. Qt’s Signal and Slot
   - Originally designed for efficient function callback implementation among GUIs
   - *We instead leverage it to identify Qt-specific function callbacks*

2. Qt’s Dynamic Introspection
   - Originally designed for run-time class member query and update
   - *We repurpose it to recover rich semantic symbols from the binary program*
Qt's Signal and Slot Mechanism

```cpp
MainWindow::MainWindow() {
  
  // Create lineEdit instance
  v0 = operator.new(0x30)

  QLineEdit(v0)

  *(this + 0x30) = v0

  ...
}
```
Qt’s Signal and Slot Mechanism

```cpp
MainWindow::MainWindow() {
    ... // Create lineEdit instance
    v0 = operator.new(0x30)
    QLineEdit(v0)
    *(this + 0x30) = v0
    ... // Register callbacks
    connect(*(this+0x30),"2textChanged(QString)", this, "1updateText(QString)", 0)
    connect(*(this+0x30),"2editingFinished()", this, "1handleInput()", 0)
    ...}
```
Qt’s Signal and Slot Mechanism

```cpp
MainWindow::MainWindow() {
    ...
    // Create lineEdit instance
    v0 = operator.new(0x30)
    QLineEdit(v0)
    *(this + 0x30) = v0
    ...
    // Register callbacks
    connect(*(this+0x30), "2textChanged(QString)", this, "1updateText(QString)", 0)
    connect(*(this+0x30), "2editingFinished()", this, "1handleInput()", 0)
    ...
}
```

**Signal**: When the lineEdit instance is changed, the `2textChanged` slot is called.

**Slot**: When the editing finished signal is emitted, the `1handleInput` function is called.
Qt's Signal and Slot Mechanism

```cpp
MainWindow::MainWindow() {
    ...
    // Create lineEdit instance
    v0 = operator.new(0x30)
    QLineEdit(v0)
    *(this + 0x30) = v0
    ...
    // Register callbacks
    connect(*(this + 0x30), "2textChanged(QString)", this, "1updateText(QString)", 0)
    ...
    connect(*(this + 0x30), "2editingFinished()", this, "1handleInput()", 0)
    ...
}
```

```cpp
MainWindow::MainWindow() {
    ...
    // Create lineEdit instance
    v0 = operator.new(0x30)
    QLineEdit(v0)
    *(this + 0x30) = v0
    ...
    // Register callbacks
    connect(*(this + 0x30), "2textChanged(QString)", this, "1updateText(QString)", 0)
    ...
    connect(*(this + 0x30), "2editingFinished()", this, "1handleInput()", 0)
    ...
}```
Qt’s Signal and Slot Mechanism

```cpp
MainWindow::MainWindow() {
    ... // Create lineEdit instance
    v0 = operator.new(0x30)
    QLineEdit(v0)
    *(this + 0x30) = v0
    ... // Register callbacks
    connect(*(this+0x30), "2textChanged(QString)" , this, "updateText(QString)", 0)
    connect(*(this+0x30), "2editingFinished()" , this, "handleInput()", 0)
    ... }
}

MainWindow::updateText(QString v1) {
    // Slot
    if (v1 != null)
        *(this + 0x48) = v1 // this->text
}
```
Qt’s Dynamic Introspection

```
MainWindow::handleInput() {
    // Slot
    editingFinished
    window.text = "secret"
}
```
Qt’s Dynamic Introspection

```cpp
MainWindow::handleInput() {
    // Slot
    v1 = *(this + 0x48)    // this->text
    if (v1 == "secret") {
        // Dynamic introspection
    }
}
```

window.text = “secret”
Qt’s Dynamic Introspection

```cpp
MainWindow::handleInput() {
    // Slot
    v1 = *(this + 0x48) // this->text
    if (v1 == "secret") {
        // Dynamic introspection
        this->setProperty("text", "test")
        qDebug() << v1 // Will print out "test"
    }
}
```

window.text = “secret”

window.text = “test”
Qt’s Dynamic Introspection

```cpp
21 MainWindow::handleInput() {
22     // Slot
23     v1 = *(this + 0x48)  // this->text
24     if (v1 == "secret") {
25         // Dynamic introspection
26         this->setProperty("text", "test")
27         qDebug() << v1  // Will print out “test”
28     }
29 }
```

window.text = “secret”

window.text = “test”
Qt's Dynamic Introspection

```cpp
MainWindow::handleInput() {
    // Slot
    v1 = *(this + 0x48) // this->text
    if (v1 == "secret") {
        // Dynamic introspection
        this->setProperty("text", "test")
        qDebug() << v1 // Will print out "test"
    }
}
```

```
MainWindow::qt_metacall(... int v1, void** v2) {
    ...
}
```

**Metadata**

**Property Table**

<table>
<thead>
<tr>
<th>Index</th>
<th>Name Index</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>QString</td>
</tr>
</tbody>
</table>

**String Table**

<table>
<thead>
<tr>
<th>Index</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>text</td>
</tr>
</tbody>
</table>

Please enter Access Code

Please enter text here.

window.text = “secret”

window.text = “test”
Qt's Dynamic Introspection

```cpp
MainWindow::handleInput() {
    // Slot
    v1 = *(this + 0x48) // this->text
    if (v1 == "secret") {
        // Dynamic introspection
        this->setProperty("text", "test")
        qDebug() << v1 // Will print out "test"
    }
}
```

```
MainWindow::qt_metacall(int v1, void** v2) {
    ... if (v1 == 0) {
        // Set property value by index
        *(this + 0x48) = (QString) v2
    }
}
```

```plaintext
window.text = "secret"
window.text = "test"
```

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Query index</th>
</tr>
</thead>
<tbody>
<tr>
<td>window.text = &quot;secret&quot;</td>
<td></td>
</tr>
<tr>
<td>window.text = &quot;test&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
Qt's Dynamic Introspection

```cpp
MainWindow::handleInput() {
    // Slot
    v1 = *(this + 0x48) // this->text
    if (v1 == "secret") {
        // Dynamic introspection
        this->setProperty("text", "test")
        qDebug() << v1 // Will print out "test"
    }
}

MainWindow::qt_metacall(... int v1, void** v2) {
    ... 
    if (v1 == 0) {
        // Set property value by index
        *(this + 0x48) = (QString) v2
    }
}
```
Qt’s Dynamic Introspection

```cpp
MainWindow::handleInput() {
    // Slot
    v1 = *(this + 0x48) // this->text
    if (v1 == "secret") {
        // Dynamic introspection
        this->setProperty("text", "test")
        qDebug() << v1 // Will print out "test"
    }
}

MainWindow::qt_metacall(...) int v1, void** v2) {
    ... if (v1 == 0) {
    // Set property value by index
    *(this + 0x48) = (QString) v2
    }
}
```

**Property Table**

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>text</td>
<td>QString</td>
</tr>
</tbody>
</table>

**String Table**

<table>
<thead>
<tr>
<th>Index</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>text</td>
</tr>
</tbody>
</table>

**Metadata**

- *(this + 0x48)*
  - Type: QString
  - Name: text
Application: Egg Hunt in Tesla Infotainment

Tesla Back to the Future Easter Egg
December 1, 2020
To activate this easter egg the vehicle needs to be at exactly 121 miles (or 121 km) of range. Then simply touch the ba...

Tesla: Mario Kart’s Rainbow Road / SNL Easter Egg
December 7, 2020
The on screen animation of your car’s environment will change. The computer vision generated road that the car is driv...

Easter eggs in Tesla vehicles
Application: Egg Hunt in Tesla Infotainment

- Do they raise security concerns?
- How to systematically identify them?
Application: Egg Hunt in Tesla Infotainment

- Do they raise security concerns?
- How to systematically identify them?
  - Coverage-based fuzzing (emulation required)
  - Input validation analysis on Qt binaries
Application: Egg Hunt in Tesla Infotainment

Experiment Setup

- Use input validation analysis to extract hidden commands from Tesla firmware
- Identify user input variables from the recovered Qt symbols
- Analyze the recovered Qt control flow

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Var./Func. Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLineEdit</td>
<td>text()</td>
</tr>
<tr>
<td>QLineEdit</td>
<td>text</td>
</tr>
<tr>
<td>QAbstractSpinBox</td>
<td>text</td>
</tr>
<tr>
<td>QDoubleSpinBox</td>
<td>text</td>
</tr>
<tr>
<td>QSpinBox</td>
<td>text</td>
</tr>
<tr>
<td>QDateTimeEdit</td>
<td>text</td>
</tr>
<tr>
<td>TextField</td>
<td>text</td>
</tr>
<tr>
<td>PasswordTextField</td>
<td>text</td>
</tr>
<tr>
<td>WebEntryField</td>
<td>text</td>
</tr>
<tr>
<td>NavigationSearchBox</td>
<td>text</td>
</tr>
<tr>
<td>CompleterTextField</td>
<td>text</td>
</tr>
<tr>
<td>ExtEntryField</td>
<td>text</td>
</tr>
</tbody>
</table>

Table: Identified user input variables.
## Application: Egg Hunt in Tesla Infotainment

<table>
<thead>
<tr>
<th>Category</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easter Egg</td>
<td>“007”</td>
<td>Submarine Easter egg</td>
</tr>
<tr>
<td></td>
<td>“modelxmas”</td>
<td>Show holiday lights</td>
</tr>
<tr>
<td></td>
<td>“42”</td>
<td>Change car name</td>
</tr>
<tr>
<td></td>
<td>“mars”</td>
<td>Turn map into Mars surface</td>
</tr>
<tr>
<td></td>
<td>“transport”</td>
<td>Transport mode</td>
</tr>
<tr>
<td></td>
<td>“performance”</td>
<td>Performance mode</td>
</tr>
<tr>
<td></td>
<td>“showroom”</td>
<td>Showroom mode</td>
</tr>
<tr>
<td>Access Token</td>
<td>SecurityToken1</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>SecurityToken2</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x18e5a977</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x73bbee22</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td>Master Pwd</td>
<td>“3500”</td>
<td>Exit valet mode</td>
</tr>
</tbody>
</table>

Table: Hidden commands from Tesla firmware.
### Application: Egg Hunt in Tesla Infotainment

<table>
<thead>
<tr>
<th>Category</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easter Egg</td>
<td>“007”</td>
<td>Submarine Easter egg</td>
</tr>
<tr>
<td></td>
<td>“modelxmas”</td>
<td>Show holiday lights</td>
</tr>
<tr>
<td></td>
<td>“42”</td>
<td>Change car name</td>
</tr>
<tr>
<td></td>
<td>“mars”</td>
<td>Turn map into Mars surface</td>
</tr>
<tr>
<td></td>
<td>“transport”</td>
<td>Transport mode</td>
</tr>
<tr>
<td></td>
<td>“performance”</td>
<td>Performance mode</td>
</tr>
<tr>
<td></td>
<td>“showroom”</td>
<td>Showroom mode</td>
</tr>
<tr>
<td>Access</td>
<td>SecurityToken1</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td>Token</td>
<td>SecurityToken2</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x18e5a977</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x73bbee22</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td>Master Pwd</td>
<td>“3500”</td>
<td>Exit valet mode</td>
</tr>
</tbody>
</table>

Table: Hidden commands from Tesla firmware.
# Application: Egg Hunt in Tesla Infotainment

<table>
<thead>
<tr>
<th>Category</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easter Egg</td>
<td>“007”</td>
<td>Submarine Easter egg</td>
</tr>
<tr>
<td></td>
<td>“modelxmas”</td>
<td>Show holiday lights</td>
</tr>
<tr>
<td></td>
<td>“42”</td>
<td>Change car name</td>
</tr>
<tr>
<td></td>
<td>“mars”</td>
<td>Turn map into Mars surface</td>
</tr>
<tr>
<td></td>
<td>“transport”</td>
<td>Transport mode</td>
</tr>
<tr>
<td></td>
<td>“performance”</td>
<td>Performance mode</td>
</tr>
<tr>
<td></td>
<td>“showroom”</td>
<td>Showroom mode</td>
</tr>
<tr>
<td>Access Token</td>
<td>SecurityToken1</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>SecurityToken2</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x18e5a977</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x73bbee22</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td>Master Pwd</td>
<td>“3500”</td>
<td>Exit valet mode</td>
</tr>
</tbody>
</table>

Table: Hidden commands from Tesla firmware.
## Application: Egg Hunt in Tesla Infotainment

<table>
<thead>
<tr>
<th>Category</th>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easter Egg</td>
<td>“007”</td>
<td>Submarine Easter egg</td>
</tr>
<tr>
<td></td>
<td>“modelxmas”</td>
<td>Show holiday lights</td>
</tr>
<tr>
<td></td>
<td>“42”</td>
<td>Change car name</td>
</tr>
<tr>
<td></td>
<td>“mars”</td>
<td>Turn map into Mars surface</td>
</tr>
<tr>
<td></td>
<td>“transport”</td>
<td>Transport mode</td>
</tr>
<tr>
<td></td>
<td>“performance”</td>
<td>Performance mode</td>
</tr>
<tr>
<td></td>
<td>“showroom”</td>
<td>Showroom mode</td>
</tr>
<tr>
<td>Access Token</td>
<td>SecurityToken1</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>SecurityToken2</td>
<td>Enable diagnostic mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x18e5a977</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td></td>
<td>crc(token)==0x73bbee22</td>
<td>Enable developer mode</td>
</tr>
<tr>
<td>Master Pwd</td>
<td>“3500”</td>
<td>Exit valet mode</td>
</tr>
</tbody>
</table>

Table: Hidden commands from Tesla firmware.

## Disclosure

The Tesla security team acknowledged our findings in 2022/4 and have eliminated the feasible paths for exploiting these hidden commands in the latest firmware.
QtRE [USENIX Security’23]

- A static analysis tool that leverages Qt’s unique insights for function callback and symbol recovery
QtRE [USENIX Security’23]

- A static analysis tool that leverages Qt’s unique insights for function callback and symbol recovery
- It additionally recovered (based on GHIDRA) 10,867 callbacks and 24,973 symbols among 123 binaries
QtRE [USENIX Security’23]

- A static analysis tool that leverages Qt’s unique insights for function callback and symbol recovery
- It additionally recovered (based on Ghidra) 10,867 callbacks and 24,973 symbols among 123 binaries
- We demonstrate an application of input validation analysis with QtRE, and extracted 12 unique hidden commands five new to the public.

The source code will be released at https://github.com/OSUSecLab/QtRE.
Bluetooth Low Energy

Bluetooth 4.0

Low Energy
Low Technical Barrier for IoT Development
Low Technical Barrier for IoT Development
Low Technical Barrier for IoT Development
Low Technical Barrier for IoT Development
Low Technical Barrier for IoT Development

Are they secure?
BLE Link Layer Vulnerabilities

Vulnerabilities

1. **Identity Tracking.** Configure static MAC address during broadcast [DPCM16].

---

**Motivations**

**FirmXRay**

**AutoMap**

**Takeaway**

**References**
BLE Link Layer Vulnerabilities

**Vulnerabilities**

1. **Identity Tracking.** Configure static MAC address during broadcast [DPCM16].
2. **Active MITM.** Just Works is adopted as the pairing method.

---

**Diagram: BLE Link Layer**

- **Peripheral**
  - 1. Broadcast
  - 3. Connection Request
  - 4. Connection Established

- **Central**
  - 2. Scan

**II) Pairing and Bonding**

- 5. Pairing Feature Exchange
  - 6. STK/LTK Generation (Legacy/LESC Pairing)
  - 7. Transport Specific Key Distribution

**III) Data Transmission**

- 8. Read/Write Data
BLE Link Layer Vulnerabilities

Vulnerabilities

1. **Identity Tracking**. Configure static MAC address during broadcast [DPCM16].
2. **Active MITM**. Just Works is adopted as the pairing method.
3. **Passive MITM**. Legacy pairing is used during key exchange [ble14].
BLE Link Layer Vulnerabilities

Vulnerabilities

1. **Identity Tracking.** Configure static MAC address during broadcast [DPCM16].
2. **Active MITM.** Just Works is adopted as the pairing method.
3. **Passive MITM.** Legacy pairing is used during key exchange [ble14].

Identification

1. Traffic analysis
2. Mobile app analysis
**BLE Link Layer Vulnerabilities**

### Broadcast and Connection

1. **Broadcast**
2. **Scan**
3. **Connection Request**
4. **Connection Established**

### Pairing and Bonding

5. **Pairing Feature Exchange**
6. **STK/LTK Generation (Legacy/LESC Pairing)**
7. **Transport Specific Key Distribution**

### Data Transmission

8. **Read/Write Data**

---

**Vulnerabilities**

1. **Identity Tracking.** Configure static MAC address during broadcast [DPCM16].
2. **Active MITM.** Just Works is adopted as the pairing method.
3. **Passive MITM.** Legacy pairing is used during key exchange [ble14].

**Identification**

1. Traffic analysis
2. Mobile app analysis
3. Firmware analysis
An Example of a Just Works Pairing Vulnerability

Read Only Memory

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Instruction</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x243a8</td>
<td>mov r2, #0x0</td>
<td>r2 = 0x0</td>
</tr>
<tr>
<td>2</td>
<td>0x243aa</td>
<td>orr r2, #0x1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x243ac</td>
<td>and r2, #0xe1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x243ae</td>
<td>add r2, #0xc</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0x243b0</td>
<td>and r2, #0xdf</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0x243b2</td>
<td>ldr r1, [0x260c8]</td>
<td>r1 = 0x0</td>
</tr>
<tr>
<td>7</td>
<td>0x243b4</td>
<td>str r2, [r1, #0x0]</td>
<td>r2 = 0x0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0x25f44</td>
<td>ldr r2, [0x260c8]</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0x25f46</td>
<td>mov r1, #0x0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0x25f48</td>
<td>svc 0x7f</td>
<td>SD_BLE_GAP_SEC_PARAMS_REPLY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0x260c8</td>
<td>0x20003268</td>
<td>ble_gap_secParms_t*</td>
</tr>
</tbody>
</table>

Register Values

r1 = 0x0
r2 = 0x0
An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
1  243a8   mov    r2, #0x0
2  243aa   orr r2, #0x1
3  243ac   and r2, #0xe1
4  243ae   add r2, #0xc
5  243b0   and r2, #0xdf
6  243b2   ldr r1, [0x260c8]
7  243b4   str r2, [r1, #0x0]
...
8  25f44   ldr r2, [0x260c8]
9  25f46   mov r1, #0x0
10  25f48   svc 0x7f
   // SD_BLE_GAP_SEC_PARAMS_REPLY
...
11  260c8  0x20003268
   // ble_gap_sec_parms_t*
```

Register Values

```
r1 = 0x0
r2 = 0xD
```
An Example of a Just Works Pairing Vulnerability

Read Only Memory

```
1 243a8   mov    r2, #0x0
2 243aa   orr r2, #0x1
3 243ac   and    r2, #0xe1
4 243ae   add r2, #0xc
5 243b0   and    r2, #0xdf
6 243b2   ldr r1, [0x260c8]
7 243b4   str r2, [r1, #0x0]
...```

Random Access Memory

```
Struct ble_gap_sec_params_t

20003268 uint8 pairing_feature
20003269 uint8 min_key_size
20003270 uint8 max_key_size
20003271 ble_gap_sec_kdist_t kdist_own
20003275 ble_gap_sec_kdist_t kdist_peer

Register Values

r1 = 0x20003268
r2 = 0xD
```

Read Only Memory

```
1 243a8   mov    r2, #0x0
2 243aa   orr r2, #0x1
3 243ac   and    r2, #0xe1
4 243ae   add r2, #0xc
5 243b0   and    r2, #0xdf
6 243b2   ldr r1, [0x260c8]
7 243b4   str r2, [r1, #0x0]
...```

Random Access Memory

```
Struct ble_gap_sec_params_t

20003268 uint8 pairing_feature
20003269 uint8 min_key_size
20003270 uint8 max_key_size
20003271 ble_gap_sec_kdist_t kdist_own
20003275 ble_gap_sec_kdist_t kdist_peer

Register Values

r1 = 0x20003268
r2 = 0xD
```
An Example of a Just Works Pairing Vulnerability

Read Only Memory

1  243a8  mov    r2, #0x0
2  243aa  orr r2, #0x1
3  243ac  and r2, #0xe1
4  243ae  add r2, #0xc
5  243b0  and r2, #0xdf
6  243b2  ldr r1, [0x260c8]
7  243b4  str r2, [r1, #0x0]
...  
8  25f44  ldr r2, [0x260c8]
9  25f46  mov r1, #0x0
10  25f48  svc 0x7f
// SD_BLE_GAP_SEC_PARAMS_REPLY
...
11  260c8  0x20003268
// ble_gap_sec_parms_t*

Random Access Memory

Struct ble_gap_sec_params_t

20003268  uint8 pairing_feature = 0xD

20003269  uint8 min_key_size
20003270  uint8 max_key_size
20003271  ble_gap_sec_kdist_t kdist_own
20003275  ble_gap_sec_kdist_t kdist_peer

Register Values

r1 = 0x20003268
r2 = 0xD
An Example of a Just Works Pairing Vulnerability

Read Only Memory

1 243a8 mov r2, #0x0
2 243aa orr r2, #0x1
3 243ac and r2, #0xe1
4 243ae add r2, #0xc
5 243b0 and r2, #0xdf
6 243b2 ldr r1, [0x260c8]
7 243b4 str r2, [r1, #0x0]
... 8 25f44 ldr r2, [0x260c8]
9 25f46 mov r1, #0x0
10 25f48 svc 0x7f

// SD_BLE_GAP_SEC_PARAMS_REPLY
...
11 260c8 0x20003268
   // ble_gap_sec_params_t

Random Access Memory

Struct ble_gap_sec_params_t

20003268 uint8 pairing_feature = 0xD
20003269 uint8 min_key_size
20003270 uint8 max_key_size
20003271 ble_gap_sec_kdist_t kdist_own
20003275 ble_gap_sec_kdist_t kdist_peer

Register Values

r1 = 0x0
r2 = 0x20003268
An Example of a Just Works Pairing Vulnerability

<table>
<thead>
<tr>
<th>Read Only Memory</th>
<th>Random Access Memory</th>
<th>Register Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  243a8   mov    r2, #0x0</td>
<td>Struct ble_gap_sec_params_t</td>
<td>r1 = 0x0</td>
</tr>
<tr>
<td>2 243aa   orr r2, #0x1</td>
<td>uint8 pairing_feature = 0x0D</td>
<td></td>
</tr>
<tr>
<td>3  243ac   and r2, #0xe1</td>
<td>BOND</td>
<td>MITM</td>
</tr>
<tr>
<td>4  243ae add r2, #0xc</td>
<td>20003268</td>
<td>uint8 min_key_size</td>
</tr>
<tr>
<td>5 243b0   and r2, #0xdf</td>
<td>20003269</td>
<td>uint8 max_key_size</td>
</tr>
<tr>
<td>6 243b2   ldr r1, [0x260c8]</td>
<td>ble_gap_sec_kdist_t kdist_own</td>
<td></td>
</tr>
<tr>
<td>7  243b4 str r2, [r1,#0x0]</td>
<td>ble_gap_sec_kdist_t kdist_peer</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>r2 = 0x20003268</td>
</tr>
<tr>
<td>8 25f44   ldr r2, [0x260c8]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9  25f46 mov r1, #0x0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 25f48 svc 0x7f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>// SD_BLE_GAP_SEC_PARAMS_REPLY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 260c8 0x20003268</td>
<td></td>
<td></td>
</tr>
<tr>
<td>// ble_gap_sec_params_t*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling

Read Only Memory

```
1 243a8 mov r2, #0x0
2 243aa orr r2, #0x1
3 243ac and r2, #0xe1
4 ... ble_gap_sec_params_t
5 243b0 and r2, #0xdf
6 243b2 ldr r1, [0x260c8]
7 243b4 str r2, [r1, #0x0]
```

Random Access Memory

```
Struct ble_gap_sec_params_t

20003268 uint8 pairing_feature = 0xD

<table>
<thead>
<tr>
<th>BOND</th>
<th>MITM</th>
<th>IO</th>
<th>OOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
// BOND = 1, MITM = 0
// IO = 3, OOB = 0

20003269 uint8 min_key_size
20003270 uint8 max_key_size
20003271 ble_gap_sec_kdist_t kdist_own
20003275 ble_gap_sec_kdist_t kdist_peer

Register Values

r1 = 0x0
r2 = 0x20003268
```

// SD_BLE_GAP_SEC_PARAMS_REPLY

...
An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling

Read Only Memory

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x00243a8</td>
<td>r2, #0x0</td>
<td>mov</td>
</tr>
<tr>
<td>2</td>
<td>0x00243aa</td>
<td>r2, #0x1</td>
<td>orr</td>
</tr>
<tr>
<td>3</td>
<td>0x00243ac</td>
<td>r2, #0xe1</td>
<td>and</td>
</tr>
<tr>
<td>4</td>
<td>0x00243ae</td>
<td>r2, #0xc</td>
<td>add</td>
</tr>
<tr>
<td>5</td>
<td>0x00243b0</td>
<td>r2, #0xdf</td>
<td>and</td>
</tr>
<tr>
<td>6</td>
<td>0x00243b2</td>
<td>[0x260c8]</td>
<td>ldr</td>
</tr>
<tr>
<td>7</td>
<td>0x00243b4</td>
<td>[r1, #0x0]</td>
<td>str</td>
</tr>
<tr>
<td>8</td>
<td>0x0025f44</td>
<td>r2, [0x260c8]</td>
<td>ldr</td>
</tr>
<tr>
<td>9</td>
<td>0x0025f46</td>
<td>r1, #0x0</td>
<td>mov</td>
</tr>
<tr>
<td>10</td>
<td>0x0025f48</td>
<td>0x7f</td>
<td>svc</td>
</tr>
</tbody>
</table>

// SD_BLE_GAP_SEC_PARAMS_REPLY
...

11 0x00260c8 | 0x20003268 | ble_gap_sec_params_t |

Recognize data structures

Random Access Memory

Struct ble_gap_sec_params_t

- uint8 pairing_feature = 0xD

<table>
<thead>
<tr>
<th>BOND</th>
<th>MITM</th>
<th>IO</th>
<th>OOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20003268</td>
<td>r2 = 0x0020003268</td>
<td>pairing_feature = 0xD</td>
</tr>
<tr>
<td>0x20003269</td>
<td>r1 = 0x003269</td>
<td>min_key_size</td>
</tr>
<tr>
<td>0x20003270</td>
<td>r1 = 0x003270</td>
<td>max_key_size</td>
</tr>
<tr>
<td>0x20003271</td>
<td>kdist_own</td>
<td>ble_gap_sec_kdist_t</td>
</tr>
<tr>
<td>0x20003275</td>
<td>kdist_peer</td>
<td>ble_gap_sec_kdist_t</td>
</tr>
</tbody>
</table>

Register Values

r1 = 0x0
r2 = 0x20003268
An Example of a Just Works Pairing Vulnerability

Correct Firmware Disassembling

Read Only Memory

1 243a8  mov  r2, #0x0
2 243aa  orr r2, #0x1
3  243ac  and r2, #0xe1
4  ... Values
r1 = 0x0
r2 = 0x20003268

Value computation

Register Values

r1 = 0x0
r2 = 0x20003268

Recognize data structures

Random Access Memory

Struct ble_gap_sec_params_t

20003268  uint8 pairing_feature = 0xD

<table>
<thead>
<tr>
<th>BOND</th>
<th>MITM</th>
<th>IO</th>
<th>OOB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
// BOND = 1, MITM = 0
// IO   = 3, OOB  = 0

20003269  uint8 min_key_size
20003270  uint8 max_key_size
20003271  ble_gap_sec_kdist_t kdist_own
20003275  ble_gap_sec_kdist_t kdist_peer

Value computation
Firmware Collection
Firmware Collection

2M Free Apps
Firmware Collection

2M Free Apps

Filter

13K BLE Apps
Firmware Collection

2M Free Apps

13K BLE Apps

793 Firmware

Filter

Unpack

Extract
Firmware Collection

- 2M Free Apps
- 13K BLE Apps
- 793 Firmware

- Filter
- Unpack
- Extract

- 768 Nordic
- 25 TI
Identity Tracking Vulnerability Identification

Among the 538 devices, nearly all of them (98.1%) have configured random static addresses that do not change periodically.
Identity Tracking Vulnerability Identification

Among the 538 devices, nearly all of them (98.1%) have configured random static addresses that do not change periodically.

<table>
<thead>
<tr>
<th>Firmware Name</th>
<th>Mobile App</th>
<th>Category</th>
<th># Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>cogobeacon</td>
<td>com.aegismobility.guardian</td>
<td>Car Accessory</td>
<td>4</td>
</tr>
<tr>
<td>sd_bl</td>
<td>fr.solem.solemwf</td>
<td>Agricultural Equip.</td>
<td>2</td>
</tr>
<tr>
<td>LRFL_nRF52</td>
<td>fr.solem.solemwf</td>
<td>Agricultural Equip.</td>
<td>2</td>
</tr>
<tr>
<td>orb</td>
<td>one.shade.app</td>
<td>Smart Light</td>
<td>1</td>
</tr>
<tr>
<td>sd_bl</td>
<td>com.rainbird</td>
<td>Agricultural Equip.</td>
<td>1</td>
</tr>
</tbody>
</table>

Table: Firmware using private MAC address.
Experiment Results

**Active MITM Vulnerability Identification**

385 (71.5%) devices use Just Works pairing, which essentially does not provide any protection against active MITM attacks at the BLE link layer.
### Experiment Results

#### Active MITM Vulnerability Identification

385 (71.5%) devices use Just Works pairing, which essentially does not provide any protection against active MITM attacks at the BLE link layer.

<table>
<thead>
<tr>
<th>Item</th>
<th>N</th>
<th>T</th>
<th>Total</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td># Total Device</td>
<td>513</td>
<td>25</td>
<td>538</td>
<td>100</td>
</tr>
<tr>
<td># Device w/ active MITM vulnerability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Device w/ Just Works pairing only</td>
<td>317</td>
<td>1</td>
<td>318</td>
<td>59.1</td>
</tr>
<tr>
<td># Device w/ flawed Passkey implementation</td>
<td>37</td>
<td>0</td>
<td>37</td>
<td>6.9</td>
</tr>
<tr>
<td># Device w/ flawed OOB implementation</td>
<td>30</td>
<td>0</td>
<td>30</td>
<td>5.6</td>
</tr>
<tr>
<td># Device w/ secure pairing</td>
<td>6</td>
<td>24</td>
<td>30</td>
<td>3.8</td>
</tr>
<tr>
<td># Device w/ correct Passkey implementation</td>
<td>3</td>
<td>24</td>
<td>27</td>
<td>3.4</td>
</tr>
<tr>
<td># Device w/ correct OOB implementation</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table: Pairing configurations of devices (N:Nordic, T:TI).
Passive MITM Vulnerability Identification

98.5% of the devices fail to enforce LESC pairing, and thus they can be vulnerable to passive MITM attacks if there is no application-layer encryption.
Passive MITM Vulnerability Identification

98.5% of the devices fail to enforce LESC pairing, and thus they can be vulnerable to passive MITM attacks if there is no application-layer encryption.

<table>
<thead>
<tr>
<th>Firmware Name</th>
<th>Mobile App</th>
<th>Category</th>
<th>#</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>DogBodyBoard</td>
<td>com.wowwee.chip</td>
<td>Robot</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>BW_Pro</td>
<td>com.ecomm.smart_panel</td>
<td>Tag</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Smart_Handle</td>
<td>com.exitec.smartlock</td>
<td>Smart Lock</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Sma05</td>
<td>com.smalife.watch</td>
<td>Wearable</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CPRmeter</td>
<td>com.laerdal.cprmeter2</td>
<td>Medical Device</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>WiJumpLE</td>
<td>com.wesssrl.wijumple</td>
<td>Sensor</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>nRF Beacon</td>
<td>no.nordicsemi.android.nrfbeacon</td>
<td>Beacon</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Hoot Bank</td>
<td>com.qvivr.hoot</td>
<td>Debit Card</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table: Firmware that enforce LESC pairing.
FirmXRay [CCS’20]

FirmXRay is a static analysis tool based on Ghidra for detecting BLE link layer vulnerabilities from bare-metal firmware. It offers a scalable approach to efficiently collect bare-metal firmware images from only mobile apps. Vulnerability discovery and attack case studies are also provided.

The source code is available at https://github.com/OSUSecLab/FirmXRay.
Microcontroller Unit (MCU)

- The chip inside the board
- Ubiquitous (e.g., drone, smart light bulb)
Microcontroller Unit (MCU)

- Peripherals are inside the provided board
- Firmware controls peripherals through peripheral registers
- Peripheral executes firmware through the corresponding interrupt
Microcontroller Unit (MCU)

MCU Firmware Vulnerabilities

1. Memory corruption
2. Privacy leakage
3. Peripheral malfunctioning
## Microcontroller Unit (MCU)

### MCU Firmware Vulnerabilities

1. Memory corruption
2. Privacy leakage
3. Peripheral malfunctioning

### Firmware Analysis

1. **Hardware-in-the-loop.** Testing firmware with hardware
2. **Re-hosting.** Emulating firmware without hardware
Microcontroller Unit (MCU)

MCU Firmware Vulnerabilities

1. Memory corruption
2. Privacy leakage
3. Peripheral malfunctioning

Firmware Analysis

1. **Hardware-in-the-loop.** Testing firmware with hardware
2. **Re-hosting.** Emulating firmware without hardware

Common Challenge

Modeling Peripheral Processing
An Example of Processing a Peripheral Register

**Execution just based on the firmware code**

```c
1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) { // check 25-bit
4:    return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;
```

[REG_CLOCK] 0x40023800 = <uninitialized>
An Example of Processing a Peripheral Register

Execution just based on the **firmware code**

```
1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) { // check 25-bit
4:   return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;
```

```
[REG_CLOCK] 0x40023800 = 0x1000000
```
An Example of Processing a Peripheral Register

Execution just based on the firmware code

1: \texttt{REG\_CLOCK} = 0x40023800;
2: \texttt{*REG\_CLOCK} = 0x1000000; // set 24-bit
3: if (*\texttt{REG\_CLOCK} \& 0x2000000) == 0) { // check 25-bit
4: \texttt{return HAL\_ERROR;}
5: }
6: \texttt{Freq = HAL\_RCC\_GetSysClockFreq();}
7: \texttt{return HAL\_OK;}

\texttt{[REG\_CLOCK]} 0x40023800 = 0x1000000
An Example of Processing a Peripheral Register

Execution just based on the firmware code

1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) { // check 25-bit
4:     return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;

[REG_CLOCK] 0x40023800 = 0x1000000
An Example of Processing a Peripheral Register

Execution on **real MCU hardware**

```
1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) { // check 25-bit
    return HAL_ERROR;
3: }
4: Freq = HAL_RCC_GetSysClockFreq();
5: return HAL_OK;
```

[REG_CLOCK] 0x40023800 = 0x3000000
An Example of Processing a Peripheral Register

Execution on real MCU hardware

```c
1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) { // check 25-bit
4:    return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;
```

[REG_CLOCK] 0x40023800 = 0x3000000
An Example of Processing a Peripheral Register

Execution on real MCU hardware

```c
1:    REG_CLOCK = 0x40023800;
2:    *REG_CLOCK = 0x1000000;  // set 24-bit
3:    if (*REG_CLOCK & 0x2000000) == 0) {  // check 25-bit
4:        return HAL_ERROR;
5:    }
6:    Freq = HAL_RCC_GetSysClockFreq();
7:    return HAL_OK;
```

[REG_CLOCK] 0x40023800 = 0x3000000
Peripheral register bits get simultaneously updated by the MCU hardware. As some bits are semantically relevant (e.g., clock status), memory mapping is crucial. Example memory mapping:

```
31:26  25   24   23:0
[REG_CLOCK] 0x40023800
```

- **24 bit - Clock enable**: 0 OFF, 1 ON
- **25 bit - Clock ready flag**: 0 Unlocked, 1 Locked
Hidden Memory Mapping

Peripheral register bits get simultaneously updated by the MCU hardware
As some bits are semantically relevant (e.g., clock status)

1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) {
4:  return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;
Hidden Memory Mapping

Peripheral register bits get simultaneously updated by the MCU hardware
As some bits are semantically relevant (e.g., clock status)

```c
1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) {
4:    return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;
```
Hidden Memory Mapping

Peripheral register bits get simultaneously updated by the MCU hardware. As some bits are semantically relevant (e.g., clock status)

```
1: REG_CLOCK = 0x40023800;
2: *REG_CLOCK = 0x1000000; // set 24-bit
3: if (*REG_CLOCK & 0x2000000) == 0) {
4:     return HAL_ERROR;
5: }
6: Freq = HAL_RCC_GetSysClockFreq();
7: return HAL_OK;
```
Hidden Memory Mapping

Root cause: Autonomous Peripheral Operation

Hardware feature in microcontroller architectures. The peripheral performs its operation without CPU intervention to save energy.
Hidden Memory Mapping

Root cause: Autonomous Peripheral Operation

Hardware feature in microcontroller architectures. The peripheral performs its operation without CPU intervention to save energy.

Bit 24 PLLRDY: Main PLL (PLL) clock ready flag
Set by hardware to indicate that PLL is locked.
0: PLL unlocked
1: PLL locked

Bit 1 SBF: Standby flag
This bit is set by hardware and cleared only by a POR/PDR (power-on reset/power-down reset) or by setting the CSBF bit in the PWR_CR register
0: Device has not been in Standby mode
1: Device has been in Standby mode
AutoMap Overview

Challenges

1. Nearly infinite number of possible writes to peripheral registers
2. Cannot infer memory mappings from code-level
3. Dependency of peripheral register writes
# AutoMap Overview

## Challenges

1. Nearly infinite number of possible writes to peripheral registers
2. Cannot infer memory mappings from code-level
3. Dependency of peripheral register writes

## Solutions

1. On-demand memory mapping inference
2. Differential memory introspection through hardware-in-the-loop
3. Memory context preparation by executing previous peripheral registers write intrusions
Experiment Setup

- Three MCUs
  - Nordic NRF52832
    - 41 example firmware included in SDK
  - STMicroelectronics STM32F103
    - 5 real-world firmware from $\mu$EMU [ZGLZ21]
  - STMicroelectronics STM32F429
    - 4 real-world firmware from $\mu$EMU [ZGLZ21]
Identity Memory Mapping in Example Firmware

At least one memory mapping is discovered in every firmware. Even single register write can affect multiple other registers.
Experiment Results

Identity Memory Mapping in Example Firmware

At least one memory mapping is discovered in every firmware. Even single register write can affect multiple other registers.

<table>
<thead>
<tr>
<th>MCU</th>
<th>Firmware</th>
<th># of Writes Causing M.M</th>
<th>Max # of M.M by single write</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF52832</td>
<td>bk_freertos</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>bk</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>bk_rtc</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>bk_systick</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>bsp</td>
<td>35</td>
<td>11</td>
</tr>
</tbody>
</table>

Table: Memory mapping result on example firmware of NRF52832
## Experiment Results

Integrating AutoMap with $\mu$EMU

<table>
<thead>
<tr>
<th></th>
<th>AutoMap</th>
<th>$\mu$EMU</th>
<th># %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drone</td>
<td>1,413</td>
<td>1,410</td>
<td>5</td>
</tr>
<tr>
<td>Gateway</td>
<td>1,385</td>
<td>1,248</td>
<td>216</td>
</tr>
<tr>
<td>Steering Iron</td>
<td>1,402</td>
<td>1,289</td>
<td>116</td>
</tr>
<tr>
<td>Reflow Oven</td>
<td>845</td>
<td>830</td>
<td>17</td>
</tr>
<tr>
<td>Robot</td>
<td>1,035</td>
<td>964</td>
<td>77</td>
</tr>
</tbody>
</table>

Table: Fuzzing result comparison between $\mu$EMU and both AutoMap and $\mu$EMU.

AutoMap with $\mu$EMU can cover at most 15.59% more basic blocks than $\mu$EMU.
### Integrating AutoMap with $\mu$EMU

AutoMap with $\mu$EMU can cover at most 15.59% more basic blocks than $\mu$EMU.

<table>
<thead>
<tr>
<th>Firmware</th>
<th># executed BBs</th>
<th>BBs portion of</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AutoMap</td>
<td>$\mu$EMU</td>
<td>#</td>
<td>%</td>
</tr>
<tr>
<td>Drone</td>
<td>1,413</td>
<td>1,410</td>
<td>5</td>
<td>0.35%</td>
</tr>
<tr>
<td>Gateway</td>
<td>1,385</td>
<td>1,248</td>
<td>216</td>
<td>15.59%</td>
</tr>
<tr>
<td>Steering_Iron</td>
<td>1,402</td>
<td>1,289</td>
<td>116</td>
<td>8.27%</td>
</tr>
<tr>
<td>Reflow_Oven</td>
<td>845</td>
<td>830</td>
<td>17</td>
<td>2.01%</td>
</tr>
<tr>
<td>Robot</td>
<td>1,035</td>
<td>964</td>
<td>77</td>
<td>7.43%</td>
</tr>
</tbody>
</table>

Table: Fuzzing result comparison between $\mu$EMU and both AutoMap and $\mu$EMU.
Discover memory mapping in peripheral registers.

Propose AutoMap to discover memory mappings systematically.

Emulate firmware properly with memory mappings and execute more basic blocks when AutoMap integrates with \(\mu\)EMU.

The source code is available at https://github.com/OSUSecLab/AutoMap.
The need to analyze new domains for heterogeneous IoT binary analysis

New domains (mechanisms, architecture, API...) lead to new insights and techniques
The Potentials of Domain-Aware Analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th># Repository</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qt</td>
<td>Framework</td>
<td>45,635</td>
<td>35.70%</td>
</tr>
<tr>
<td>ROS</td>
<td>Robotics</td>
<td>16,796</td>
<td>13.14%</td>
</tr>
<tr>
<td>Boost</td>
<td>Framework</td>
<td>6,205</td>
<td>4.85%</td>
</tr>
<tr>
<td>MFC</td>
<td>Framework</td>
<td>4,409</td>
<td>3.45%</td>
</tr>
<tr>
<td>Cocos2d</td>
<td>Game Engine</td>
<td>3,587</td>
<td>2.81%</td>
</tr>
<tr>
<td>OpenFrameworks</td>
<td>Framework</td>
<td>3,264</td>
<td>2.55%</td>
</tr>
<tr>
<td>JUCE</td>
<td>Framework</td>
<td>2,204</td>
<td>1.72%</td>
</tr>
<tr>
<td>PCL</td>
<td>Robotics</td>
<td>1,719</td>
<td>1.34%</td>
</tr>
<tr>
<td>imgui</td>
<td>GUI</td>
<td>1,557</td>
<td>1.22%</td>
</tr>
<tr>
<td>wxWidgets</td>
<td>GUI</td>
<td>1,076</td>
<td>0.84%</td>
</tr>
<tr>
<td>Cinder</td>
<td>Framework</td>
<td>1,042</td>
<td>0.82%</td>
</tr>
<tr>
<td>Allegro</td>
<td>Game Engine</td>
<td>958</td>
<td>0.75%</td>
</tr>
<tr>
<td>Godot</td>
<td>Game Engine</td>
<td>682</td>
<td>0.53%</td>
</tr>
<tr>
<td>Godot</td>
<td>Game Engine</td>
<td>561</td>
<td>0.44%</td>
</tr>
<tr>
<td>GamePlay</td>
<td>Game Engine</td>
<td>547</td>
<td>0.43%</td>
</tr>
<tr>
<td>dlib</td>
<td>Framework</td>
<td>518</td>
<td>0.41%</td>
</tr>
<tr>
<td>FLTK</td>
<td>GUI</td>
<td>436</td>
<td>0.34%</td>
</tr>
<tr>
<td>GTK++</td>
<td>GUI</td>
<td>425</td>
<td>0.33%</td>
</tr>
<tr>
<td>LibU</td>
<td>Framework</td>
<td>376</td>
<td>0.29%</td>
</tr>
<tr>
<td>raylib</td>
<td>Game Engine</td>
<td>349</td>
<td>0.27%</td>
</tr>
<tr>
<td>gtkmm</td>
<td>GUI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Top C++ frameworks for software development.

Bluetooth Low Energy end-product certifications*

Data Source: Nordic Quarterly Presentation Q4 2019
The Potentials of Domain-Aware Analysis

1. Systematically vetting domain-specific applications
The Potentials of Domain-Aware Analysis

1. Systematically vetting domain-specific applications
2. Extension to other IoT domains, architectures, frameworks...
The Potentials of Domain-Aware Analysis

1. Systematically vetting domain-specific applications
2. Extension to other IoT domains, architectures, frameworks...
3. Support various security applications (e.g., Qt-Fuzz, Automap-Fuzz)
The Potentials of Domain-Aware Analysis

1. Systematically vetting domain-specific applications
2. Extension to other IoT domains, architectures, frameworks...
3. Support various security applications (e.g., Qt-Fuzz, Automap-Fuzz)
4. Generalize methodology and insights to other similar domains
Unlocking the Potential of Domain Aware Binary Analysis in the Era of IoT

Zhiqiang Lin

zlin@cse.ohio-state.edu

April 18th, 2023


Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N Nguyen, Recovering variable names for minified code with usage contexts, 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 1165–1175.
