Hacking Tizen: The OS of Everything

Ajin Abraham
ajin25@gmail.com
http://opensecurity.in

Abstract

Tizen is a lightweight operating system, which is built to run on various kinds of devices. Samsung's first Tizen based devices were launched in India on Jan 2015. This paper talks about the security analysis done on Tizen OS and explains about Tizen architecture, security model, application sandboxing, and resource access control powered by SMACK. It also explains the vulnerabilities identified in Tizen OS during the research which are responsibly disclosed to Tizen community.

Introduction

Tizen is an open and flexible operating system built from the ground up to address the needs of all stakeholders of the mobile and connected device ecosystem, including device, application developers, manufacturers, mobile operators, and independent software vendors (ISVs). Tizen is developed by a community of developers, under open source governance, and is open to all members who wish to participate.[1]

– *Tizen.org*

The Tizen operating system comes in different flavours including Tizen IVI (in-vehicle infotainment), Tizen Mobile, Tizen TV, and Tizen Wearable. Like Android, Tizen is an operating system built on top of the Linux kernel and is supported by The Linux Foundation.

It brings up the concept of Internet of Things (IoT) or Smart Home and is buzzed as *The OS of Everything.* This new operating system is powered by Intel and Samsung with support of the major technology, hardware and mobile network companies like LG, Fijitsu, Intel, vodafone, docomo, kt, SK telecom etc and this list keeps changing. This paper describes the research done on Tizen operating system 2.2 and Tizen IVI 3.0.

Types of Tizen Applications

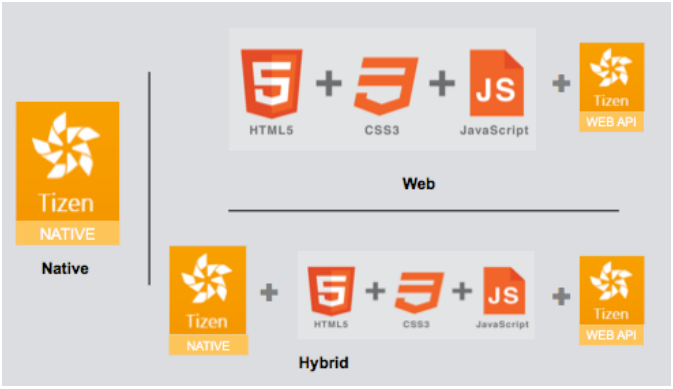

Fig 1: Types of Tizen Application

Applications in Tizen can be written with native code using C/C++ or HTML5/JavaScript/CSS. Like other mobile platforms, Tizen supports three kinds of applications.

- Native Applications
- Web Applications
- Hybrid Applications

Native Application – These applications are written in C/C++

Web Application – These are applications written on HTML5/JavaScript/CSS

Hybrid Application – Hybrid Applications are those having web component as well as a native component.

Tizen Architecture

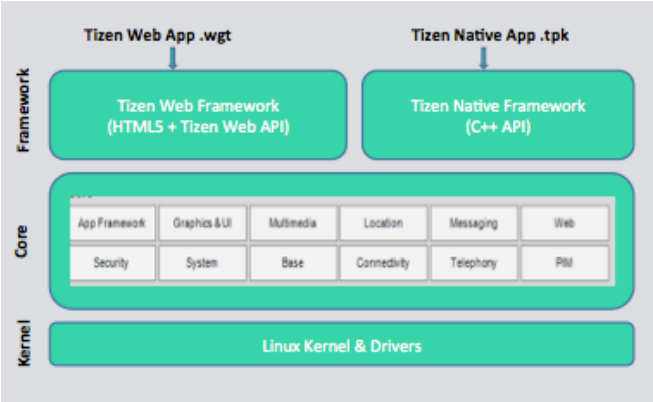

Fig 2: Tizen Architecture

The Tizen architecture as shown in Fig 2 consist of three layers. At the bottom we have the Linux Kernel & Drivers. On top of that, we have the Tizen Core layer which act as an interface between Application Framework layer and the Kernel layer. It facilitates access to device hardware and other features [2].

The Application Framework in the Core layer, contains all the middleware, hardware-related services and provides the set of APIs needed for developing Native, Web or Hybrid Apps.

The Framework includes Tizen Native Framework that facilitate the running of Native and Hybrid Applications and the Tizen Web Framework which provides the Web Runtime (WRT) where the Web Applications run. The Web Apps make use of Web API which consist of HTML5 API's as well as a set of Device APIs provided by Tizen which is protected by Content Security Policy (CSP) and Privileges.

Fig 3: Tizen Web API

The Web API also includes Miscellaneous APIs like WebGL, Vewport Metatag, Typed Array etc.

Tizen Application Structure

As Tizen Applications are available in three different format, they follow diiferent directory structures. All the user developed Apps are installed under **/opt/usr/apps/**

Native Application Structure

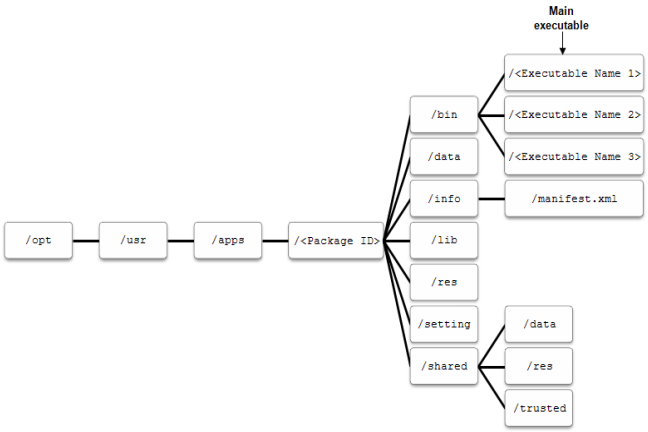

Fig 4: Tizen Native Application Structure

The Tizen Native Application is having .tpk extention which is a zipped package with different directories like bin, which contains the executable and the manifest file where privileges, and other information about the app are defined. It also contains other required resources under res and libraries under lib etc. On installation of the TPK, the contents of the package are extracted to "**/opt/usr/apps/<pkg_id>/**" where pkg_id is a

unique identifier of length 10 containing alphabets and numbers called AppId/AppName.

Web Application Structure

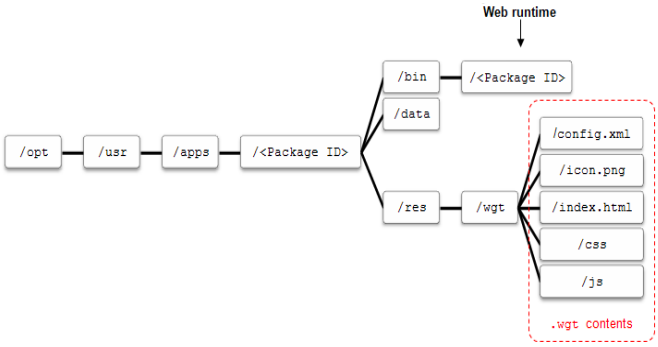

Fig 5: Tizen Web Application Structure

The Web Application is having the extension .wgt and is a zipped package containing the HTML, CSS, JavaScript and config.xml under the wgt directory. On installation of the wgt file, the web applications are extracted into "**/opt/usr/apps/<pkg_id>/res/wgt/**"

Hybrid Application Structure

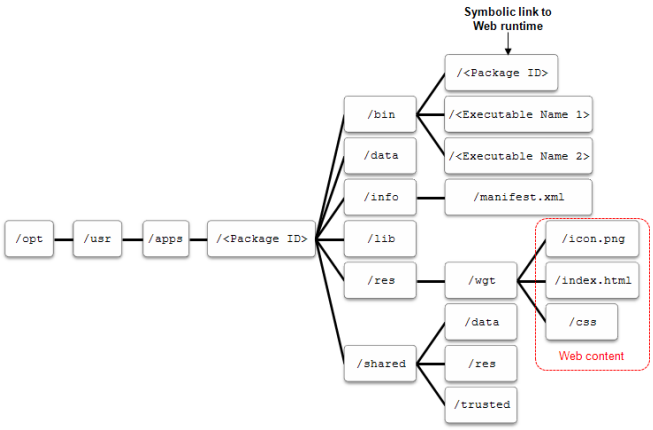

Fig 6: Tizen Hybrid Application Structure

Hybrid Applications contains both binaries as well as the Web content and comes with TPK extension. It is also a zipped package containing both the directories in a Native App and a Web App.

Tizen Security Model

Like permissions in Android, Tizen uses privileges to enforce a least privilege model. In addition to that Tizen make use of application signing, and sandboxed running of process using its sandbox called SMACK [3].

The main principles of this Security Model are:

- **Non root applications**
    - All applications run under same non-root user ID.
    - Most of the middleware and daemons will run as non-root user.
- **Application Sandboxing**
    - All applications are sandboxed by SMACK.
    - An application is allowed to read/write files in it's home directory and shared media directory (**/opt/usr/media**)
    - Each application is unable to send IPC and sockets, r/w other application files.
- **Content Security Framework (CSF)**
    - Set of APIs/hooks used to create security-related services.
    - These are intented for AV Solutions.
    - Two types of engines: Scan Engine and Site Engine.
    - Scan Engine scans Data and Application for malicious behaviour.
    - Site Engine scans URLs and blocks malicious URLs.
- **Application Sigining**
    - Application can be signed by Authors as well as Distributors.

- **Permission Model/Least privilege**
    - All applications will have manifest/config file describing privileges.
    - Native apps use manifest.xml
    - Web apps use config.xml
    - Manifest file describes SMACK labels and rule as well.
- **Content Security Policy for Web Apps**
    - For Web Applications, Policy or Content Security Policy is defined in the config.xml file.
- **Encrypt HTML/JS/CSS stored in Device**
    - Encrypts at Install time and decrypts at runtime.

SMACK: Simplified Mandatory Access Control Kernel

Tizen's sandbox is caled as Simplified Mandatory Access Control Kernel (SMACK). The basic rule of application sandbox is "**what's mine is mine; what's yours is yours.**" SMACK allows you to add controlled exception to this basic rule. SMACK is a kernel level Linux security module that determines how processes interact each other. In Tizen, every application has its own SMACK label. These labels identify the application and provide access controls [4].

SMACK Terms:

- **Subject** → Any Running Process (Have Smack Label)
- **Object** → File, IPC, Sockets, Process
- **Access** → Read (r), Write (w), Execute (e), Append (a), Lock (l), Transmute (t)

In SMACK, the subject can only access an object if the labels match or if there exist a permission that grant access to the requested resource. A subject is an active entity while an object is a passive entity, which include files, directories, IPC, sockets and process.

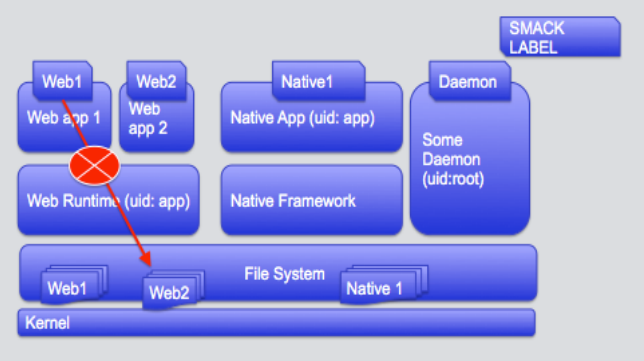

Fig 7: SMACK Sandbox

SMACK ensures that applications are sandboxed and one application cannot access the files and data of other application and vice versa. However SMACK allows controlled exceptions to this.

The interesting thing about SMACK rules is that Tizen got about **41,000** SMACK rules in Tizen version 2.2.1. The number of SMACK rules is so huge that, there is a high chance that developers may mess up. So in Tizen 3 onwards they will introduce Cynara [5] and a Smack Three domain Model.

Privileges

For Tizen applications, privileges are like permissions for Android applications. In order to use different APIs, appropriate privileges should be defined.

Fig 8: manifest.xml in Native Applications

For native applications, privileges are defined in the manifest.xml file

Fig 9: config.xml in Web Applications

For web applications, privileges are defined in config.xml file.

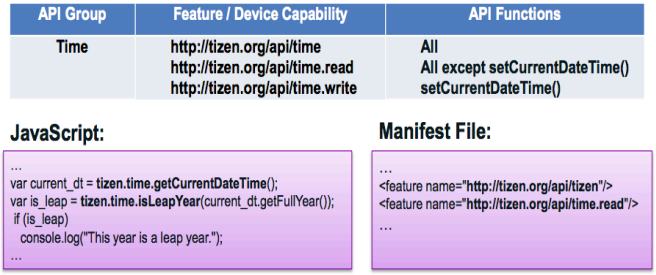

Fig 10: API-Privilege Requirement in Tizen

For example, In order to invoke a device Web API, the necessary privileges should be declared in the config.xml file [6]. A hybrid application contains both manifest.xml and config.xml file.

There are three levels of privilege:

- **Public privileges:** These privileges are open to Tizen application developers.
- **Partner privileges:** These privileges can only be used by developers that are registered as partners on the Tizen Store.
- **Platform privileges:** These privileges are used by system APIs and are accessible only to a specific set of Tizen developers.

Webkit2 on Tizen

Tizen Web Runtime (WRT) is based on WebKit2 on top of which, the web apps run [7]. WebKit2 is a new API layer over WebKit. It supports split process model like the Google Chrome tabs. In Chrome, each tab is a separate independent process, similarly in Tizen WRT, every web application run as a separate process. So if anything goes wrong in one process, it should not affect other running processes.

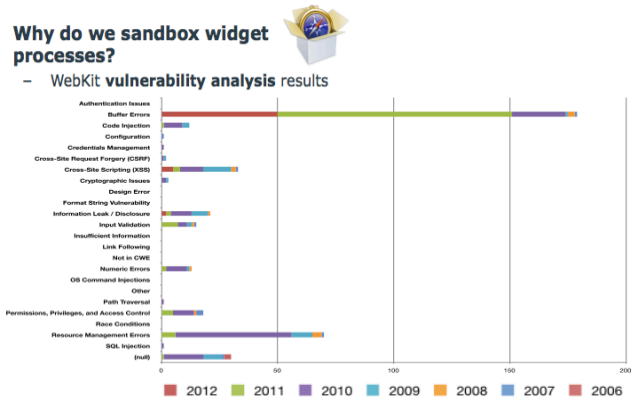

Fig 11: WebKit Vulnerability Analysis

In WRT, every web app runs inside a sandbox. One of the main reasons is that WebKit is known to have a lot of vulnerabilities. A lot of them are still being reported and coming up. To prevent the impact of these, Tizen make use of the split process model and the application sandbox.

Comparison with Android and iOS

Android	Tizen	iOS
1. Apps identified by UID 2. Permission Model: AndroidManifest.xml 3. IPC: Using Binder 4. Sandbox: SELinux 5. Signed: Signed by Developer	1. Users identified by UID (app) 2. Permission Model: manifest.xml & config.xml 3. IPC: MessagePort IPC using socket. 4. Sandbox: SMACK & CSP 5. Signed: Signed by Developer & Distributor	1. All Apps run under user "mobile". 2. No permission model. Ask for Permissions at Runtime. 3. IPC: URL Schemes, x-callback URL, Extension, XPC based IPC. 4. Sandbox: Powerbox, Seatbelt 5. Signed: Signed by Distributor

Security Issues in Tizen

This section covers the security issues identified in Tizen during the research.

Attack Surface of over privileged Apps: Android Web Apps vs. Tizen Web Apps

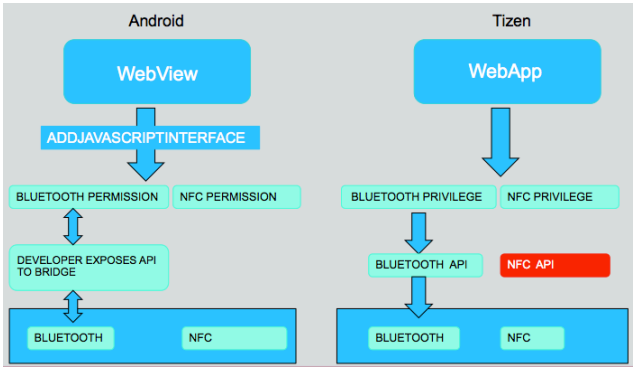

Fig 12: Android Web Apps vs. Tizen Web Apps

Consider an over privileged Android web app and a Tizen web app. In Android, web applications leverage device functionalities by making use of the JavaScript bridge (*addjavascriptinterface*) where developer has to expose device functionalities to the bridge by exposing the API that corresponds the device functionality. The required permissions should be described In the AndroidManifest.xml for the exposed device API.

In Tizen the difference is that the developer doesn't have to write or expose the API to the bridge. The device APIs are already defined and implicitly exposed to the bridge. To access these APIs, proper privileges should be defined. Also they have CSP to restrict out bound access.

Now say if the application is over privileged, for example, the NFC permission is an unwanted permission or privilege declared both in Android and Tizen Application. Say suppose an XSS occurs in the context of the web app or else an ad page that works inside the context, then attack surface on Tizen is greater than that in Android.

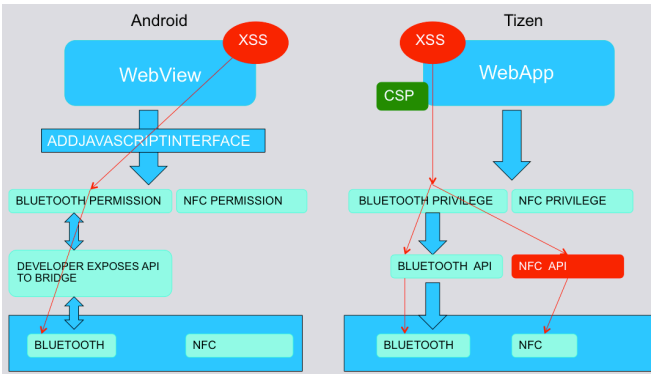

Fig 13: Attack Surface: Android Web Apps vs. Tizen Web Apps

In Android, the vulnerability can take advantage of Bluetooth in the device as the permission is there and the developer exposes the API to bridge, but couldn't access NFC even though the permission is defined. The reason for this is that the API is not explicitly exposed to the bridge by the developer.

But in the case of Tizen, all the Device APIs are already available and are implicitly exposed. So since the Bluetooth and NFC privileges are defined, such vulnerability can access both Bluetooth and NFC.

Issues in OS Memory Protection: DEP and ASLR

Data Execution Prevention (**DEP**) is a security feature that marks the stack as non executable. So when malicious code is on the stack, it won't get executed due to DEP. During the analysis of the OS memory protection mechanisms, it was found that DEP is not seen/working in Tizen.

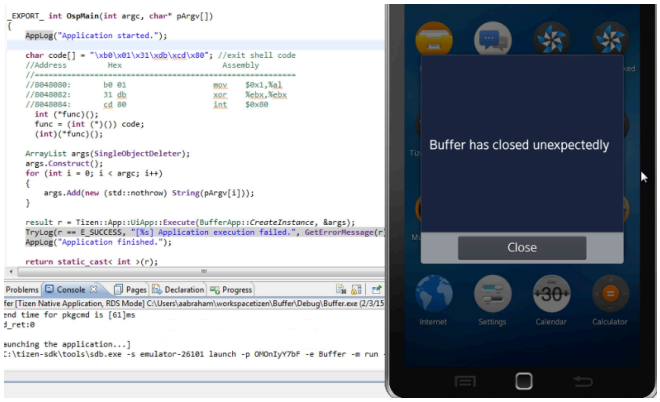

Fig 14: Exit Shellcode in Action

From the above figure, it is clear that the exit shellcode is loaded into the stack got executed. This verifies that Data Execution Prevention is either not there or not working as expected.

Address Space Layout Randomization (ASLR) is another memory protection technique that prevents reliable exploitation by rebasing the ASLR enabled modules on each run. As per the documentation, it says ASLR is fully implemented in Tizen 2.1 itself. It was found that ASLR was broken in Tizen 2.1 by a researcher named Shuichiro Suzuki [8]. He found out that even though ASLR was enabled as per **/proc/sys/kernel/randomize_va_space** which is set to **2,** but the personality value in **/proc/self/personality** is set to **00040000** which corresponds to **(ADDR_NO_RANDOMIZE)** and disables ASLR. This issue was patched in Tizen 2.2. During the analysis, it was found that **/proc/self/personality** is now set to **0000000,** which was nowhere documented as such. To verify whether ASLR works, a sample application was compiled with **–fPIE** flag which corresponds to Position Independent Executable. This flag marks the executable as ASLR enabled. To test for the validity of ASLR, the application was run twice and the consecutive memory maps available under **/proc/pid/maps** were observed.

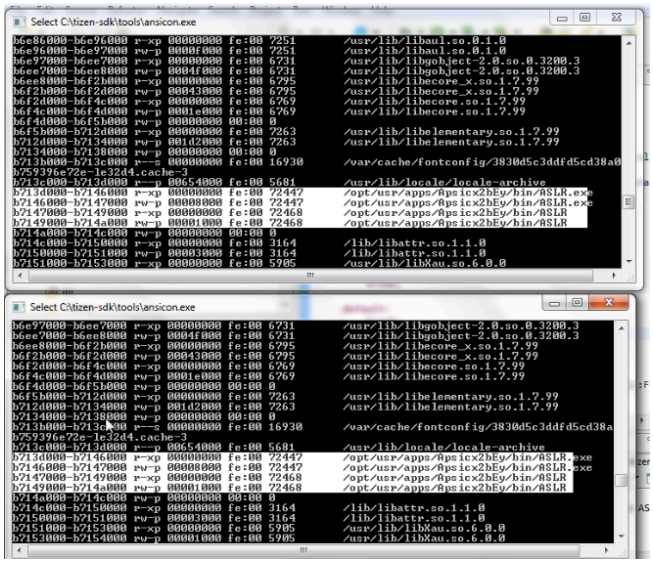

Fig 15: Memory Maps of the same application run twice.

As shown in the above figure, it was found that all the address of heap, stack and main modules remains the same which means ASLR is not seen in action.

Issues in the Stock Tizen Browser

A security analysis of Tizen's Stock browser, which is based on WebKit, was done and couple of issues was identified.

URL Spoofing and Content Injection

A WebKit bug was identified on the stock Tizen 2.2 browser that allows attacker to open a new tab with any URL and inject arbitrary data into the tab. The PoC code that triggers the bug is shown below.

```
<script>
w=window.open('https://facebook.com/');
w.document.write("<h1>You 've been
Hacked</h1>"); w.focus();
</script>
```

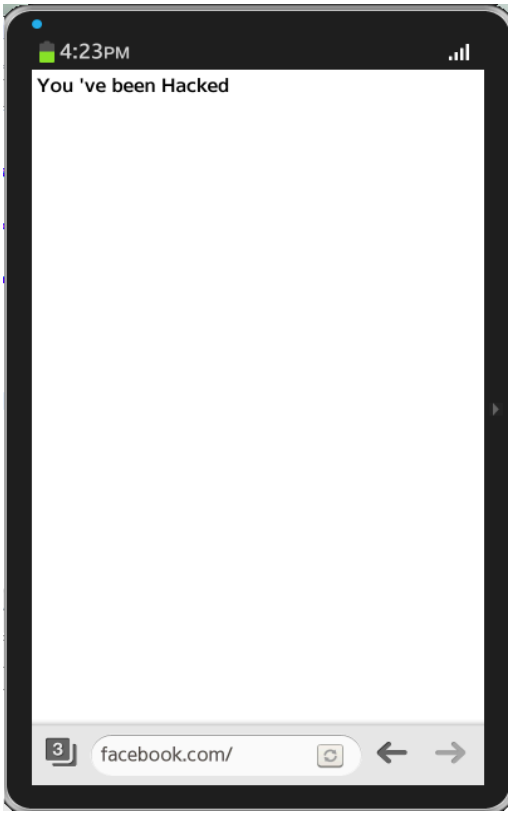

Fig 16: PoC of URL Spoofing and Content Injection

URL Spoofing and Content Injection are shown in the above figure.

Bypassing Content Security Policy

Another WebKit bug was identified on Tizen Stock browser that bypasses Content Security Policy (CSP).

We create a page with the CSP ***Content-Security-Policy: default-src 'self'; script-src 'self'*** which means load everything from same domain and load scripts from same domain.

Following is the PoC code for CSP Bypass

```
<script>
a=document.createElement('script');a.id='x';
a.src='\u0000https://rawgit.com/ajinabraham/PoC/master/script.js';
document.body.appendChild(a);
</script>
```

Here we create a script tag with JavaScript null byte prepended to the URL in the script src. This tricks the browser's JavaScript parser and load the script from a different domain (rawgit.com) and CSP get bypassed. The JavaScript file https://rawgit.com/ajinabraham/PoC/master/script.js contains the code that pop up an alert dialog.

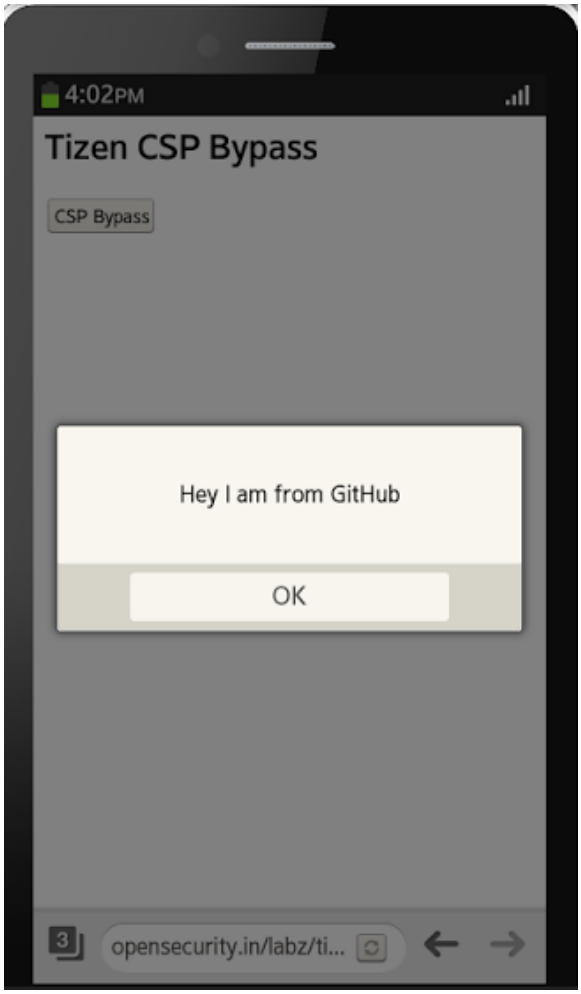

Fig 16: PoC of CSP Bypass

The above figure shows that CSP is bypassed.

Pentesting Methodologies

Penesting of Tizen Application includes the following methodologies

- **Whitebox**: Access to source code and knowledge about the application.
- **Blackbox**: No access to source code and no idea about the application.

We again classify this process into

- **Static Analysis**
- **Dynamic Analysis**
- **Network Analysis**

Static Analysis

The Static Analysis starts with the TPK or WGT file. As mentioned before both are zipped packages. Inside TPK you will find the executable under bin directory. It is compiled using LLVM-3.1, which makes use of clang or clang++.

Certificate Analysis

The certificate information is available in two files.

- **author-signature.xml** – This file contains the developer signature.
- **signature1.xml** – This file contains the distributor signature.

The signature information can be extracted using tizen_certificate_parser.py [9].

Manifest Analysis

The manifest/config file can be found at <unzipped_package>/info/manifest.xml for TPK and <unzipped_package>/config.xml for WGT.

During manifest analysis, proper checks should be made for over privileges, improper CSP/Policy and SMACK rules. Code Review is the most important part of Static Analysis. In a whitebox approach it's pretty straight forward as the source code is available. But if it's a blackbox approach then the situation changes with a TPK as it contains clang/clang++ compiled binaries. Futher analysis is possible only after decompiling the binary. The decompilation of the binary into C and ASM can be done using retdec API[10]. To decompile the binary, use the python script ,tizen_tpk_decompiler.py [11].

Following are some of the things that you should check while doing a code review.

- Weak Encryption/Crypto
- Hardcoded plaintext information
- Logging sensitive information
- SSL overriding/bypass
- Possible client side SQLi, XSS etc.

Most of the issues can be categorized under OWASP Mobile Top 10 [12].

Dynamic Analysis

For Dynamic Analysis, we can run the App in Tizen VM, Web Simulator or Tizen Device. Tizen SDK comes with various tools to make our job easier. It contains tools like Secure Debug Bridge (SDB) which is similar to Android's Android Debug Bridge and highly useful tool named Dynamic Analyzer which show us information about the process, file operations, thread information, UI flow etc. For logging, Tizen uses **dlog**, which is similar to Android's **logcat**. Once the application is installed on the device, file analysis can be done which includes check for uncessary permissions on the application directories and files.

Network Analysis

To perform Network Analysis packet capturing or traffic interception has to be done. To analyze HTTPS traffic we need to configure a proxy through *Settings -> WiFi* and configure the Proxy.

Proxy certificate can be installed from SD card through *Settings -> About device -> Manage certificates -> User certificates -> Install*. The certificate is installed as a user certificate.

```
in-mac-02:tools aabraham$ openssl x509 -in /Users/aabraham/Desktop/burp_ca.der -inform DER -out /Users/aabraham/Desktop/burp_ca.pem -outform PEM
in-mac-02:tools aabraham$ ./sdb push /Users/aabraham/Desktop/burp_ca.pem /tmp/
pushed          burp_ca.pem   100%       1021 B
1 file(s) pushed. 0 file(s) skipped.
/Users/aabraham/Desktop/burp_ca.pem   30 KB/s (1021 bytes in 0.033s)
in-mac-02:tools aabraham$ ./sdb shell
sh-4.1$ su
sh-4.1# mv /tmp/burp_ca.pem /etc/ssl/certs/aaaaaaaa.0
sh-4.1# ls /etc/ssl/certs/
00673b5b.0  2e4eed3c.0  578d5c04.0  7d5a75e4.0  add67345.0  d537fba6.0
02265526.0  2e5ac55d.0  57e0f75e.0  812e17de.0  ae8153b9.0  d59297b8.0
024dc131.0  2fa87019.0  57bbd831.0  8160b96c.0  aeb67534.0  d64f06f3.0
039c618a.0  2fb1850a.0  57bcb2da.0  81b9768f.0  aee5f10d.0  d777342d.0
03e16f6c.0  33815e15.0  58a44af1.0  8470719d.0  b0f3e76e.0  d7e0dc79.0
03f0efa4.0  343eb6cb.0  594f1775.0  84cba82f.0  b1159c4c.0  d8274e24.0
062cdee6.0  349f2832.0  5a3f0ff8.0  85cde254.0  b13cc6df.0  d957f522.0
080911ac.0  3513523f.0  5a5372fc.0  86212b19.0  b1b8a7f3.0  d9d12c58.0
0810ba98.0  381ce4dd.0  5adba5d6.0  87753b0d.0  b204d74a.0  dbc54cab.0
08aef7bb.0  399e7759.0  5c44d531.0  882de061.0  b42ff584.0  ddc328ff.0
09789157.0  3a3b02ce.0  5cf9d536.0  8867006a.0  b66938e9.0  e113c810.0
0996ae1d.0  3ad48a91.0  5e4e69e7.0  88f89ea7.0  b6c5745d.0  e2799e36.0
```

Fig 17: Install certificates into Trusted Certificate Store

You can install the certificate into the trusted certificate store as well. The trusted certificates are stored under **/etc/ssl/certs/** with filename in the format <**8HEXChars.0**>. The certificate is in PEM format. To install a certificate into trusted certificate store, we need root access. Copy the CA certificate to **/etc/ssl/certs/** directory to make it trusted. Once we are able to intercept the traffic, we can rely on OWASP Top 10 Web Risks [13] for testing the web service and the server.

Security Concerns in Tizen

Some of the security concerns of Tizen are given below

- Tizen uses WebKit. A lot of security bugs are uncovered in WebKit and that will affect the future of Tizen as well. The use of WebKit2 with split process model will provide some level of sandbox on the process which is a reasonably way of protecting from the possible issues.

- Unlike Android where developer explicitly needs to expose the device APIs to the JavaScript Bridge, the device Web APIs are implicitly defined in Tizen and this will increase the attack surface. So compared to Android an unused or extra privilege and an improper CSP can cause a comparatively higher threat in Tizen if XSS/third-party script executes in the App context.

- Another issue with Tizen 2.2 is there is too much of SMACK rules. Hence there is high chance that developers may mess up and this will not give the right result as expected.

Conclusion

Tizen OS had put a lot of efforts in creating a highly efficient Security Model/Architecture compared to other similar lightweight operating systems. But at some point, they made it so complex that people can easily mess up, like SMACK rules. And couple of implementation issues that are identified during the research exists in Tizen. But the platform looks promising, as they have given high priority to Security.

Reference

[1] Tizen. https://www.tizen.org/, April 2014.

[2] https://developer.tizen.org/dev-guide/2.2.0/org.tizen.gettingstarted/html/tizen_overview/tizen_architecture.htm

[3] https://wiki.tizen.org/wiki/Security/Tizen_2.X_Security_Model

[4] https://wiki.tizen.org/wiki/Security:Smack

[5] https://wiki.tizen.org/wiki/Security:Cynara

[6] https://wiki.tizen.org/wiki/Security/Tizen_2.X_Privileges

[7] http://download.tizen.org/misc/media/conference2

012/wednesday/ballroom-c/2012-05-09-1100-1140-webkit-efl_and_webkit2-efl.pdf

[8] Shuichiro Suzuki, www.ffri.jp/assets/files/monthly.../MR201305_Tizen_Security_ENG.pdf

[9] https://github.com/ajinabraham/tizen-security/blob/master/tizen_certificate_parser.py

[10] https://retdec.com/api/

[11] https://github.com/ajinabraham/tizen-security/blob/master/tizen_tpk_decompiler.py

[12] https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

[13] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project