Decode AndroidManifest.xml

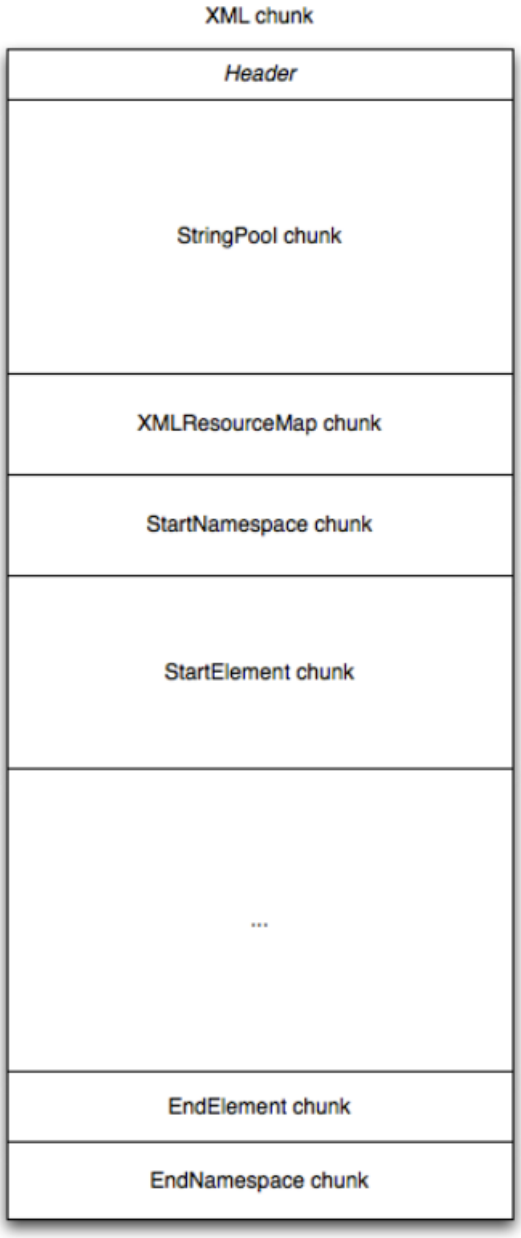
모바일응용 보안 및 실습 assignment1

사이버보안학과

201620641

유 상 정

1. AndroidManifest,xml 구조



AndroidManifest.xml은 위와 같은 구조로 되어있으며, 전체 chunk의 header부터 string pool chunk, namespace chunk, element chunk로 이루어져 있다.

1.1. Chunk type and byte order

androidManifest.xml은 각 chunk마다 header type으로 구분된다. 해당 type에 대한 내용은 다음과 같다.

```
RES_NULL_TYPE               = 0x0000,
RES_STRING_POOL_TYPE        = 0x0001,
RES_TABLE_TYPE              = 0x0002,
RES_XML_TYPE                = 0x0003,

// Chunk types in RES_XML_TYPE
RES_XML_FIRST_CHUNK_TYPE    = 0x0100,
RES_XML_START_NAMESPACE_TYPE= 0x0100,
RES_XML_END_NAMESPACE_TYPE  = 0x0101,
RES_XML_START_ELEMENT_TYPE  = 0x0102,
RES_XML_END_ELEMENT_TYPE    = 0x0103,
RES_XML_CDATA_TYPE          = 0x0104,
RES_XML_LAST_CHUNK_TYPE     = 0x017f,
// This contains a uint32_t array mapping strings in the string
// pool back to resource identifiers.  It is optional.
RES_XML_RESOURCE_MAP_TYPE   = 0x0180,

// Chunk types in RES_TABLE_TYPE
RES_TABLE_PACKAGE_TYPE      = 0x0200,
RES_TABLE_TYPE_TYPE         = 0x0201,
RES_TABLE_TYPE_SPEC_TYPE    = 0x0202
```

androidManifest.xml은 위의 hex값을 통해서 각 chunk를 구분한다. 또한 Chunk는 little-endian byte order를 따른다.

1.2. XML Chunk header

Chunk header의 구조는 아래와 같다.

```
// Type identifier for this chunk.  The meaning of this value depends
// on the containing chunk.
uint16_t type;

// Size of the chunk header (in bytes).  Adding this value to
// the address of the chunk allows you to find its associated data
// (if any).
uint16_t headerSize;

// Total size of this chunk (in bytes).  This is the chunkSize plus
// the size of any data associated with the chunk.  Adding this value
// to the chunk allows you to completely skip its contents (including
// any child chunks).  If this value is the same as chunkSize, there is
// no data associated with the chunk.
uint32_t size;
```

Chunk header는 AndroidManifest.xml에서 chunk size와 chunk header의 size 그리고, header type을 나타낸다. 또한 Chunk type은 2byte, Header Size는 2byte, Chunk Size는 4byte로 이루어져 있다. 위의 chunk header는 android의 모든 chunk에 포함되어 있다.

1.3. String Pool Chunk

String Pool Chunk는 xml파일에서 사용될 string들이 저장되는 공간이다. String pool의 header는 위의 header와 구조가 조금 다르다. 다만 본인은 프로그래밍에서 chunk를 header와 body로 나눌 때, 개발 편의를 위해 똑같이 8byte로 나누고, body부분에 header 일부분을 포함시켰다.

1.3.1. String pool header

```
// Number of strings in this pool (number of uint32_t indices that foll
// in the data).
uint32_t stringCount;

// Number of style span arrays in the pool (number of uint32_t indices
// follow the string indices).
uint32_t styleCount;

// Flags.
enum {
    // If set, the string index is sorted by the string values (based
    // on strcmp16()).
    SORTED_FLAG = 1<<0,

    // String pool is encoded in UTF-8
    UTF8_FLAG = 1<<8
};
uint32_t flags;

// Index from header of the string data.
uint32_t stringsStart;

// Index from header of the style data.
uint32_t stylesStart;
```

String pool header는 위에서 설명한 xml header내용을 포함하여, string count (4byte), style count (4byte), flag (4byte), string start(4byte), style start(4byte) 내용을 포함한다. 그래서 전체 header size가 0x1C이다. 실제로 hex editor를 통해 확인이 가능했다.

```
00000008  01 00 1C 00  ....
0000000C  24 05 00 00  $...
```

각 요소를 설명하면,

String count는 String Pool안의 string 개수이다.

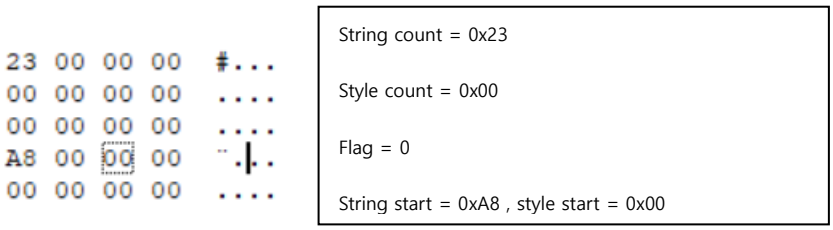
Style Count는 String pool의 Chunk body안의 style과 연관된 string의 수이다. 이 값은 실제로 거의 0이다.

Flags는 SORED_FLAG 또는 UTF8_FLAG를 나타낸다.

Strings Start는 String Pool chunk의 시작부터 string data의 시작지점까지의 offset이며, 실제로 이 데이터를 통해, string데이터가 존재하는 offset을 획득하고 string pool에 저장한다.

Styles Start는 style과 관련된 string 데이터가 존재하는 offset이다.

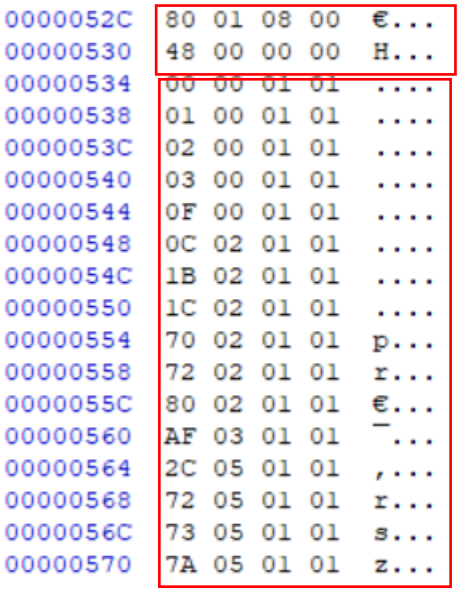
위의 값을 아래사진에서 확인할 수 있다.



String pool body에는 header의 내용을 토대로 각 string 데이터의 offset 및 string 길이와 string data가 저장되어 있다.

1.4. XML ResourceMap Chunk

ResouceMap chunk는 Header가 XML chunk header와 동일하며, Body에는 Resouce ID가 4byte씩 저장되어 있다. Resource ID의 개수는 (Chunk size – header size) / 4 개이다.



위는 header와 body의 구분이다.

1.5. NameSpace Chunk

NameSpace chunk는 Start와 End 두가지가 존재한다. 각 구분은 header type으로 구분되며, start 이후에 end가 나타나게 된다. 열고 닫힘으로 이해할 수 있다.

NameSpace header는 xml header내용을 포함하고 추가적으로 아래 내용을 가지게 된다.

```
// Line number in original source file at which this element appeared.
uint32_t lineNumber;

// Optional XML comment that was associated with this element; -1 if none.
struct ResStringPool_ref comment;
```

lineNumber는 실제 source에서의 line번호를 나타낸다. 다음으로 comment는 element에 대한 comment로서, 없다면 0xffffffff이다.

NaneSpace의 Body는 다음 구조와 같다.

```
// The prefix of the namespace.
struct ResStringPool_ref prefix;

// The URI of the namespace.
struct ResStringPool_ref uri;
```

여기서 prefix는 String Pool의 index로서, Namespace prefix이며, uri는 String Pool의 index로서, namespace URI이다. 실제 xml파일에서의 구성은 prefix="uri" 형식으로 나타낼 수 있다.

1.6. XML Element Chunk

Element Chunk도 namespace와 마찬가지로 start와 end로 나뉘고, start와 end는 짝을 이루게 된다. 또한 header 내용은 namespace와 동일하다. Element start, end는 실제 source code에서 "<" "</>" 이를 나타낸다. 즉, tag의 열림과 닫힘이다. 다만, namespace와 다르게, body이 start와 end가 다르다. 아래는 body 내용이다.

```
// String of the full namespace of this element.
struct ResStringPool_ref ns;

// String name of this node if it is an ELEMENT; the raw
// character data if this is a CDATA node.
struct ResStringPool_ref name;

// Byte offset from the start of this structure where the attributes start.
uint16_t attributeStart;

// Size of the ResXMLTree_attribute structures that follow.
uint16_t attributeSize;

// Number of attributes associated with an ELEMENT.  These are
// available as an array of ResXMLTree_attribute structures
// immediately following this node.
uint16_t attributeCount;

// Index (1-based) of the "id" attribute. 0 if none.
uint16_t idIndex;

// Index (1-based) of the "class" attribute. 0 if none.
uint16_t classIndex;

// Index (1-based) of the "style" attribute. 0 if none.
uint16_t styleIndex;
```

각 요소를 설명하자면,

ns는 포함된 XML chunk의 String Pool에 대한 index로 이 element의 namespace 이름 (namespace uri)을 지정한다. Namespace 이름이 없는 경우 -1이다.

Name은 element의 지역이름으로, 포함된 XML chunk의 string pool에 대한 index이다.

End element Chunk에는 Body에 위의 두 내용만 존재한다.

Attribute Start는 chunk body의 시작지점으로부터의 offset으로써, chunk body내의 attribute data가 저장된 위치의 시작지점이다. 또한 이 값은 고정으로 항상 0x0014이다. 그리고 이 값을 통해 attribute 참조를 시작한다.

Attribute size는 각 attribute에 사용되는 size이며, 항상 0x0014이다. 이 size를 통해 각 attribute를 구분한다.

Attribute count는 attribute의 개수로서, attribute를 체크한다.

Id_index는 original XML에서 id 속성을 나타내는 경우 지정한다.

Class_index는 original XML에서 class 속성을 나타내는 경우 지정한다.

Style_index는 original XML에서 style 속성을 나타내는 경우 지정한다.

추가적으로 BODY는 위의 내용 다음으로, 각 attribute 내용 자체가 저장된다. 이 때, attribute도 하나의 구조를 이룬다. 그 구조는 다음과 같다.

```
// Namespace of this attribute.
struct ResStringPool_ref ns;

// Name of this attribute.
struct ResStringPool_ref name;

// The original raw string value of this attribute.
struct ResStringPool_ref rawValue;

// Processesd typed value of this attribute.
struct Res_value typedValue;
```

ns는 포함된 XML chunk의 String Pool에 대한 index로 이 Attribute의 namespace 이름 (namespace uri)을 지정한다.

Name은 Attribute의 local이름으로, 포함된 XML chunk의 string pool에 대한 index이다.

rawValue는 String Pool의 index로 나타나는 Attribute의 value를 지정한다. 만약 original value를 이용할 수 없다면 -1 이다.

Typed Value는 Res_value 구조체의 instance다. Res_value 구조체는 아래와 같다.

```
// Number of bytes in this structure.
uint16_t size;

// Always set to 0.
uint8_t res0;

// Type of the data value.

// ... elided

uint8_t dataType;

// ... elided

// The data for this item, as interpreted according to dataType
uint32_t data;

// ... elided
```

각 value에 대한 size와 data type, 그리고 실제 data를 저장한다.

2. Decoding

AndroidManifest.xml의 구조에 따라, StringPool Chunk로부터, 사용되는 string을 정리하고, 각 namespace에서 prefix, url 또한 각 element에서 ns와 name을 통해 string pool의 string 모음으로부터 string을 참조하여, xml의 tag형식으로 element를 출력해주면 AndoroidManifest.xml 파일의 디코딩이 가능하다.

2.1. Decoding Code

아래는 python을 이용한 AndroidManifest.xml Decoding Source code이다.

```python
from struct import unpack

total_chunk_size = 0x00
string_pool = []
xml_resource_map_list = []
namespace = {}

tab = 0
element_count = 0

file = open("AndroidManifest.xml","rb")

class HEADER_DATA:
```

```
  def __init__(self, Header_Type,Header_Size,Chunk_Size):
    self.Header_Type =Header_Type
    self.Header_Size = Header_Size
    self.Chunk_Size = Chunk_Size

  def getHeaderType(self):
    return self.Header_Type
  def getHeaderSize(self):
    return self.Header_Size
  def getChunkSize(self):
    return self.Chunk_Size
  def print(self):
    print(hex(self.Header_Type),"\t" +"HEADER_TYPE")
    print(hex(self.Header_Size),"\t" + "HEADER_SIZE")
    print(hex(self.Chunk_Size) , "\t" + "CHUNK_SIZE")

  def getBodySize(self):
    return self.Chunk_Size - 8


#STRING_POOL DATA 뽑기
def getStringBodyData(STRING_POOL_BODY):
  stringCount = int(unpack("<L",STRING_POOL_BODY[0:4])[0])
  styleCount = int(unpack("<L",STRING_POOL_BODY[4:8])[0])
  flags = unpack("<L",STRING_POOL_BODY[8:12])[0]
  stringsStart = unpack("<L",STRING_POOL_BODY[12:16])[0] #address
  stylesStart = unpack("<L",STRING_POOL_BODY[16:20])[0] #address
  # print(hex(stringCount),"\t"+"STRING_COUNT")
  # print(hex(styleCount),"\t"+"STYLE_COUNT")
  # print(hex(flags),"\t"+"FLAGS")
  # print(hex(stringsStart),"\t"+"STRINGS_START")
  # print(hex(stylesStart),"\t"+"STYLES_START")
  x = 20 #read 위치 시작
  y = x + 4 # read 위치 끝

  strings_offsets = []
  strings = []
  for i in range(0,stringCount):
    strings_offsets.append(int(unpack("<L",STRING_POOL_BODY[x:y])[0]))
    x = y
    y += 4

  y -= 2 # string은 두글자씩 읽으므로
  for i in range(0,stringCount):
    string = ''
    if i == stringCount -1:  #마지막 string 인 경우
      length = unpack("<H",STRING_POOL_BODY[x:y])[0]
      x += 2
```

```
      last_string = STRING_POOL_BODY[x:(x+length)*2]

      for i in range(0, length):
        data = unpack("<H",last_string[2*i:2*(i+1)])[0] #마지막 스트링 더하기
        string+= chr(data)
      strings.append(string)

    else:
      string_start = strings_offsets[i]
      string_end = strings_offsets[i+1]
      length = unpack("<H",STRING_POOL_BODY[x:y])[0]
      count = 0
      x = y
      y += 2
      for z in range(string_start+2,string_end,2):
        if count == length :
           #offset 건너띄기
          x = y
          y += 2
          continue
        data = unpack("<H",STRING_POOL_BODY[x:y])[0]
        x = y
        y += 2
        string += chr(data)
        count += 1

      strings.append(string)
  # print("--string list--")
  #print(strings)
  return strings


#XML_RESOURCE_MAP DATA 뽑기
def getResourceMapBodyData(XML_RESOURCE_MAP_BODY, size):
  xml_resource_map_list = []
  for i in range(int(size/4)):
    data = unpack("<L",XML_RESOURCE_MAP_BODY[i*4:(i+1)*4])[0]
    xml_resource_map_list.append(data)
  # print(xml_resource_map_list)
  return xml_resource_map_list  #address list

#XML_START_NAMESPACE DATA 뽑기
def getStartNameSpaceData(XML_START_NAMESPACE_BODY):
  line_number = unpack("<L",XML_START_NAMESPACE_BODY[0:4])[0]
  comment = unpack("<L",XML_START_NAMESPACE_BODY[4:8])[0]
  prefix = unpack("<L",XML_START_NAMESPACE_BODY[8:12])[0]
  uri = unpack("<L",XML_START_NAMESPACE_BODY[12:16])[0]
  namespace[string_pool[uri]] = string_pool[prefix]
```

```
#XML_END_NAME_SPACE DATA 뽑기
def getEndNameSpaceData(XML_END_NAMESPACE_BODY):
  line_number = unpack("<L",XML_END_NAMESPACE_BODY[0:4])[0]
  comment = unpack("<L",XML_END_NAMESPACE_BODY[4:8])[0]
  prefix = unpack("<L",XML_END_NAMESPACE_BODY[8:12])[0]
  uri = unpack("<L",XML_END_NAMESPACE_BODY[12:16])[0]

#XML_START_ELEMENT DATA 뽑기
def getStartElementData(XML_START_ELEMENT_BODY):
  #-----element의 정보 알려줌
  line_number = unpack("<L",XML_START_ELEMENT_BODY[0:4])[0]
  comment = unpack("<L",XML_START_ELEMENT_BODY[4:8])[0]
  ns = unpack("<L",XML_START_ELEMENT_BODY[8:12])[0]
  name = unpack("<L",XML_START_ELEMENT_BODY[12:16])[0] # LinearLayout
  attribute_start = unpack("<H",XML_START_ELEMENT_BODY[16:18])[0]
  attribute_size = unpack("<H",XML_START_ELEMENT_BODY[18:20])[0]
  attribute_count = unpack("<H",XML_START_ELEMENT_BODY[20:22])[0]
  id_index = unpack("<H",XML_START_ELEMENT_BODY[22:24])[0]
  class_index = unpack("<H",XML_START_ELEMENT_BODY[24:26])[0]
  style_index = unpack("<H",XML_START_ELEMENT_BODY[26:28])[0]

  #print 요소 추가
  print_element = ((tab -1)*"\t" + "<")
  print_element += string_pool[name] +"\n" + tab*"\t"

  if element_count == 0:
    for string in string_pool:
      if "http://schemas.android.com/apk/res/android" == string:
        print_element += ' xmlns:android="'+string +'"\n'+ tab*"\t"
        break

  attributes = []
  for i in range(attribute_count):
    #attribute는 body 28지점부터 시작 , attribute size는 고정
    start = 28 + i*attribute_size
    end = start + attribute_size
    attribute = XML_START_ELEMENT_BODY[start : end]
    attribute_ns = unpack("<L",attribute[0:4])[0]  #사용될 문자열 string_pool i
ndex
    attribute_name = unpack("<L",attribute[4:8])[0] #사용될 문자열 index
    attribute_raw_value = unpack("<L",attribute[8:12])[0] #원리 XML에 포함된 X
ML chunk의 string_pool의 index로 나타내는 속성의 값 지정
    attribute_data_size = unpack("<H",attribute[12:14])[0]
    #attribute[14] == 0
    attribute_data_type = attribute[15]
    attribute_data = unpack("<L",attribute[16:20])[0]
    print_element += " "
```

```
    if attribute_ns == 0xffffffff: continue # element has no namespace name
    #---attribute print
    print_element += (namespace[string_pool[attribute_ns]] +":")
    print_element +=string_pool[attribute_name]

    #data type 에 따라 로직 처리
    if attribute_data_type == 0x01: #resource
      print_element +=('="@' + hex(attribute_data) + '"')
    elif attribute_data_type == 0x03: #string 값 -> string_pool 에서 가져옴
      print_element += ('="' + string_pool[attribute_raw_value] + '"')
    elif attribute_data_type == 0x10: #int 값
      print_element += ('="' + str(attribute_data) + '"')
    elif attribute_data_type == 0x11: #hex 값
      print_element += ('="' + hex(attribute_data) + '"')
    elif attribute_data_type == 0x12: #boolean 값
      print_element += ('="' + str(bool(attribute_data)) + '"')
    print_element+='\n' + tab*'\t'
    attributes.append(attribute)
  if print_element[len(print_element)-
1] ==' ' : print_element = print_element[:len(print_element) - 2]
  print_element += ">"
  #line 출력
  print(print_element)


#XML_END_ELEMENT DATA 뽑기
def getEndElementData(XML_END_ELEMENT_BODY):
  line_number = unpack("<L",XML_END_ELEMENT_BODY[0:4])[0]
  comment = unpack("<L",XML_END_ELEMENT_BODY[4:8])[0]
  ns = unpack("<L",XML_END_ELEMENT_BODY[8:12])[0]
  name = unpack("<L",XML_END_ELEMENT_BODY[12:16])[0]

  print_element = (tab-1)*"\t" + "</" + string_pool[name] + ">"
  print(print_element)



##아래쪽 MAIN
#--------------------------------Top Header
HEADER = HEADER_DATA(unpack("<H",file.read(2))[0],unpack("<H",file.read(2))[0]
,unpack("<L",file.read(4))[0])

# print("----HEADER----")
# HEADER.print()
total_chunk_size += HEADER.getHeaderSize()
#----- STRING_POOL_CHUNK
STRING_POOL_CHUNK = []
```

```
STRING_POOL_HEADER = HEADER_DATA(unpack("<H",file.read(2))[0],unpack("<H",file
.read(2))[0],unpack("<L",file.read(4))[0])
STRING_POOL_CHUNK.append(STRING_POOL_HEADER)
STRING_POOL_BODY = file.read(STRING_POOL_HEADER.getBodySize())
STRING_POOL_CHUNK.append(STRING_POOL_BODY)

# print("\n----STRING_POOL_CHUNK----")
# STRING_POOL_HEADER.print()
#string 배열 획득
string_pool = getStringBodyData(STRING_POOL_BODY)

total_chunk_size += STRING_POOL_HEADER.getChunkSize()

XML_RESOURCE_MAP_CHUNK = []
XML_RESOURCE_MAP_HEADER = HEADER_DATA(unpack("<H",file.read(2))[0],unpack("<H"
,file.read(2))[0],unpack("<L",file.read(4))[0])
XML_RESOURCE_MAP_CHUNK.append(XML_RESOURCE_MAP_HEADER)
XML_RESOURCE_MAP_BODY = file.read(XML_RESOURCE_MAP_HEADER.getBodySize())
XML_RESOURCE_MAP_CHUNK.append(XML_RESOURCE_MAP_BODY)

#----XML_RESOURCE_MAP -> 맵핑위치라고 생각 됨
# print("\n----XML_RESOURCE_MAP----")
# XML_RESOURCE_MAP_HEADER.print()
xml_resource_map_list = getResourceMapBodyData(XML_RESOURCE_MAP_BODY,XML_RESOU
RCE_MAP_HEADER.getBodySize())

total_chunk_size += XML_RESOURCE_MAP_HEADER.getChunkSize()

print('<?xml version="1.0" encoding="utf-8"?>')
#----- ----namespace, element
while True:
  if total_chunk_size == HEADER.Chunk_Size : break
  XML_CHUNK = []
  XML_HEADER = HEADER_DATA(unpack("<H",file.read(2))[0],unpack("<H",file.read(
2))[0],unpack("<L",file.read(4))[0])
  XML_CHUNK.append(XML_HEADER)
  XML_BODY = file.read(XML_HEADER.getBodySize())
  if(XML_HEADER.getHeaderType() == 0x0100):
    #print("\n----XML_START_NAMESPACE")
    #XML_HEADER.print()
    getStartNameSpaceData(XML_BODY)
  elif(XML_HEADER.getHeaderType() == 0x0101):
    #print("\n----XML_END_NAMESPACE")
    #XML_HEADER.print()
    getEndNameSpaceData(XML_BODY)
  elif(XML_HEADER.getHeaderType() == 0x0102):
    #print("\n----XML_START_ELEMENT")
    #XML_HEADER.print()
```

```
    tab+=1
    getStartElementData(XML_BODY)
    element_count += 1
  elif(XML_HEADER.getHeaderType() == 0x0103):
    #print("\n----XML_END_ELEMENT")
    #XML_HEADER.print()
    getEndElementData(XML_BODY)
    tab-=1
    element_count -= 1
  total_chunk_size += XML_HEADER.getChunkSize()

file.close()
```

2.2. Decoder 결과

아래 결과는 Android Studio에서 empty activity로 생성한 파일을 apk추출하여 구한 AndroidManifest.xml파일의 decoding 결과이다.

```
sangjeong20@DESKTOP-UU9URND:~/mo$ python3 decoder.py
<?xml version="1.0" encoding="utf-8"?>
<manifest
         xmlns:android="http://schemas.android.com/apk/res/android"
         android:versionCode="1"
         android:versionName="1.0"
         android:compileSdkVersion="30"
         android:compileSdkVersionCodename="11"
         >
        <uses-sdk
                 android:minSdkVersion="16"
                 android:targetSdkVersion="30"
                >
        </uses-sdk>
        <application
                 android:theme="@0x7f0f019c"
                 android:label="@0x7f0e001b"
                 android:icon="@0x7f0c0000"
                 android:debuggable="True"
                 android:testOnly="True"
                 android:allowBackup="True"
                 android:supportsRtl="True"
                 android:roundIcon="@0x7f0c0001"
                 android:appComponentFactory="androidx.core.app.CoreComponentFactory"
                >
                <activity
                          android:name="com.example.helloworld.MainActivity"
                         >
                         <intent-filter
                                 >
                                 <action
                                          android:name="android.intent.action.MAIN"
                                         >
                                 </action>
                                 <category
                                          android:name="android.intent.category.LAUNCHER"
                                         >
                                 </category>
                         </intent-filter>
                 </activity>
        </application>
</manifest>
```

실제로 결과를 통해 Decoding에 성공했음을 파악할 수 있다.

3. 참고 사이트

https://justanapplication.wordpress.com/category/android/android-binary-xml/

Android Internals: Binary XML