

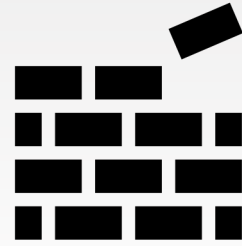
Reading Between the Lines: An Extensive Evaluation of the Security and Privacy Implications of EPUB Reading Systems

Gertjan Franken, Tom Van Goethem, Wouter Joosen

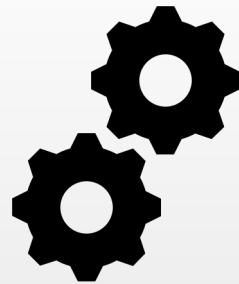




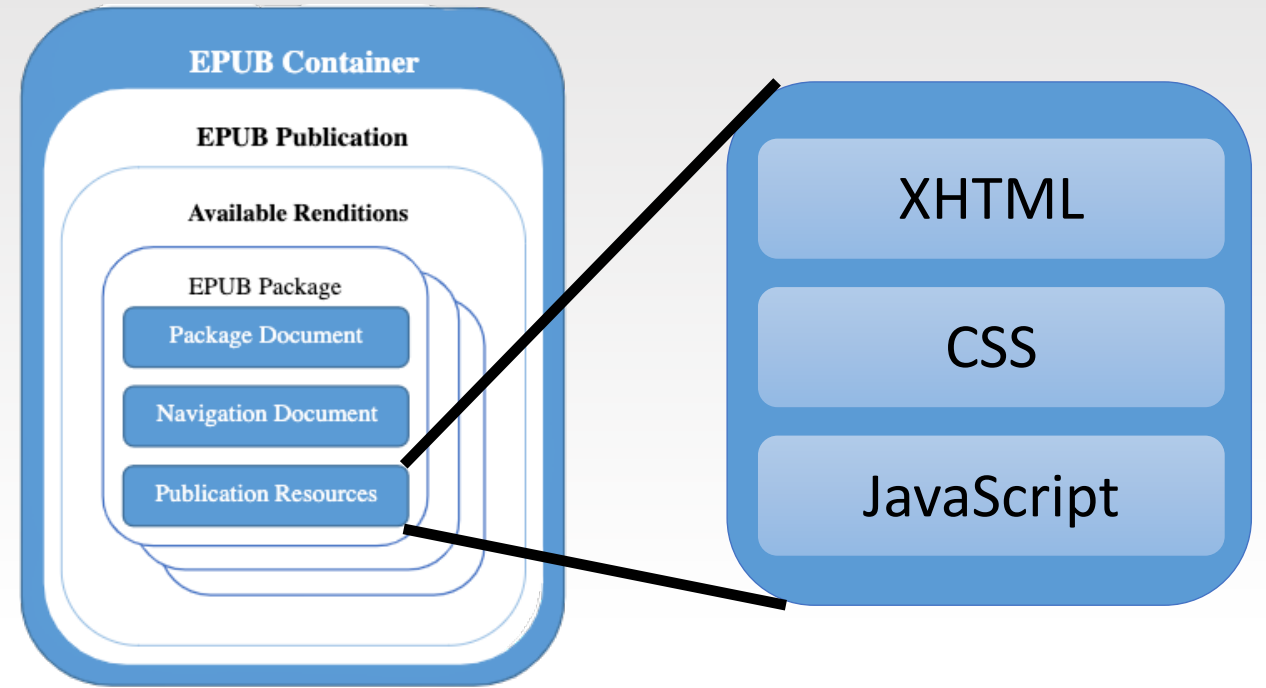
.EPUB (.ZIP)



EPUB file



EPUB
reading system





ePUB 3.2

**Remote
resources**

“ A resource that is located outside of the EPUB Container, typically, but not necessarily, online. ”



- Security considerations
 - User consent or notification for network activity
 - Only rendering, no content access (SOP)

Research questions



- What is the state of freely available EPUB reading systems?



Granted capabilities



Security considerations

- Are these capabilities being (ab)used in the wild?



Malicious EPUBs



Tracking EPUBs

Research questions



- What is the state of freely available EPUB reading systems?



Granted capabilities



Security considerations

- Are these capabilities being (ab)used in the wild?



Malicious EPUBs



Tracking EPUBs

97 reading systems |



15

macOS

9



ubuntu

3

Desktop apps (27)



android

35

iOS

20

Mobile apps (55)



5



5

Browser extensions (10)

amazon kindle

tolino 

 ONYX

Physical e-readers (5)

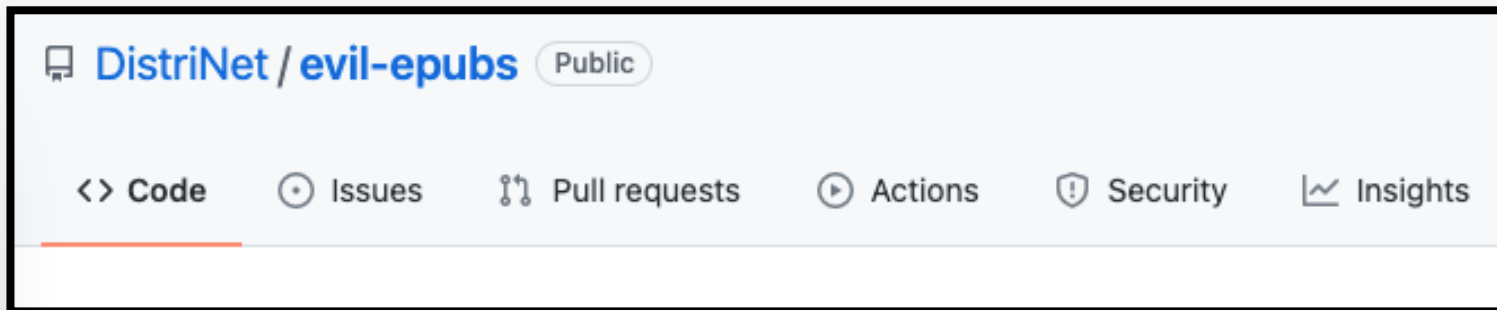


PocketBook

Semi-automated black-box evaluation



GitHub repo: <https://github.com/DistriNet/evil-epubs>



JavaScript execution

Remote communication

Local file system access

Persistent storage

Feature access

URI schemes

Web engine evaluation

1) JavaScript support

Inline

```
<html>
  <p id='msg'>No JS execution</p>
  ...
  <script>
    document.getElementById('msg').innerHTML = "JS was executed!!!";
  </script>
  ...
</html>
```

Backward compability

- E.g. ECMAScript 5 instead of 6

External

```
<html>
  <p id='msg'>No JS execution</p>
  ...
  <script src='../js/change_p.js'></script>
  ...
</html>
```

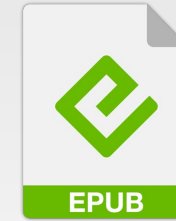
2) Remote communication

- HTTPLeaks [1]
 - Comprehensive set of HTML tags to initiate requests
 - Server-side check
- Consent flow / notification
 - Client-side check



3) Local file system access

- Inference of file existence
 - Rendering
 - <iframe>, , <audio>, <video>, etc.
 - onLoad event
 - Timing attack
 - XMLHttpRequest, Fetch,
 - onError event
 - File System in Userspace (FUSE)
- Leak file contents
 - XMLHttpRequest, Fetch
 - <canvas> to convert to base64
 - <iframe> contentWindow attribute



[file:///](#) protocol

- Direct link
- Symbolic file link (UNIX)
- Symbolic folder link (UNIX)



.html
.txt
.log
.bogus

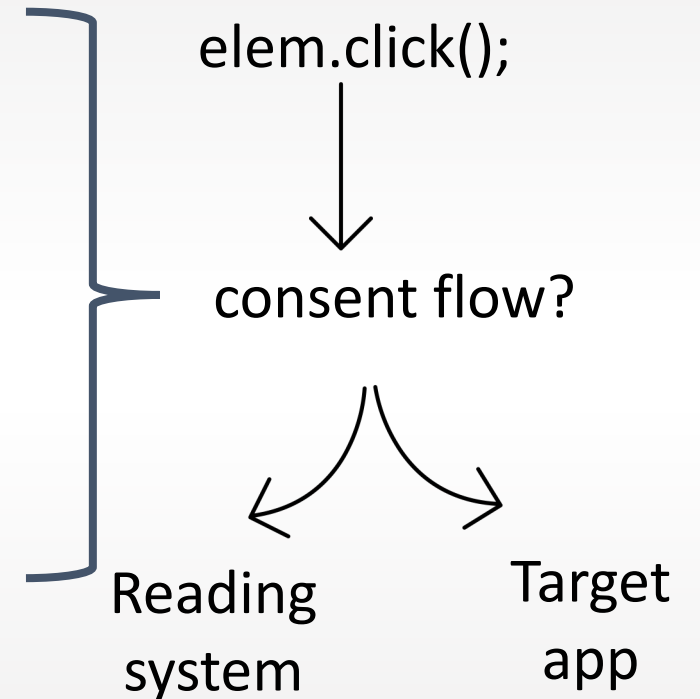
.png
.jpg

.ttf

.mp3
.mp4

4) URI schemes (not included in EPUB 3.2 spec)

- Official URI schemes [1]
 - <mailto:gertjan.franken@kuleuven.be>
 - <tel:+32XXXXXXXX>
- Custom URI schemes
 - `twitter://status?status_id=XXXXXXXXXX`
 - `ms-word://distrinet.cs.kuleuven.be`



[1] Internet Assigned Numbers Authority (IANA). Uniform resource identifier (uri) schemes.
<https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml>

5) Web engine evaluation

- ~~User agent string~~

- Engine fingerprinting based on MDN dataset [1]

- Supported HTML tags and attributes
- Supported JavaScript API

- Match known browser engine with unknown browser engine based on hamming distance between two fingerprints

+/- 3000



<a>	1
<abbr>	0
<acronym>	0
<address>	0
<applet>	1
<area>	0
<article>	1
<aside>	1
<audio>	0
	1
<base>	0
<basefont>	0
<bdi>	1
<bdo>	1

...

[1] MDN's browser compatibility dataset. <https://github.com/mdn/browser-compat-data>.

Results (1)

	Desktop	Smartphone	Browser	E-reader	Total
JavaScript execution	13 (48%)	22 (40%)	3 (30%)*	1 (20%)	39 (40%)
Remote comm.**	15 (56%)	20 (36%)	10 (100%)	1 (20%)	46 (47%)

* Prevented by Content Security Policy

** Only 1 application requires user consent (Apple Books on iOS)

Results (2)

	Desktop	Smartphone *	Browser **	E-reader	Total
Infer existence of local files	10 (37%)	6 (11%)	0	0	16 (16%)
Read content of local files	5 (19%)	3 (5%)	0	0	8 (8%)

* Thanks to iOS design, no applications leaked local file system information

** SOP prevented access to local file system

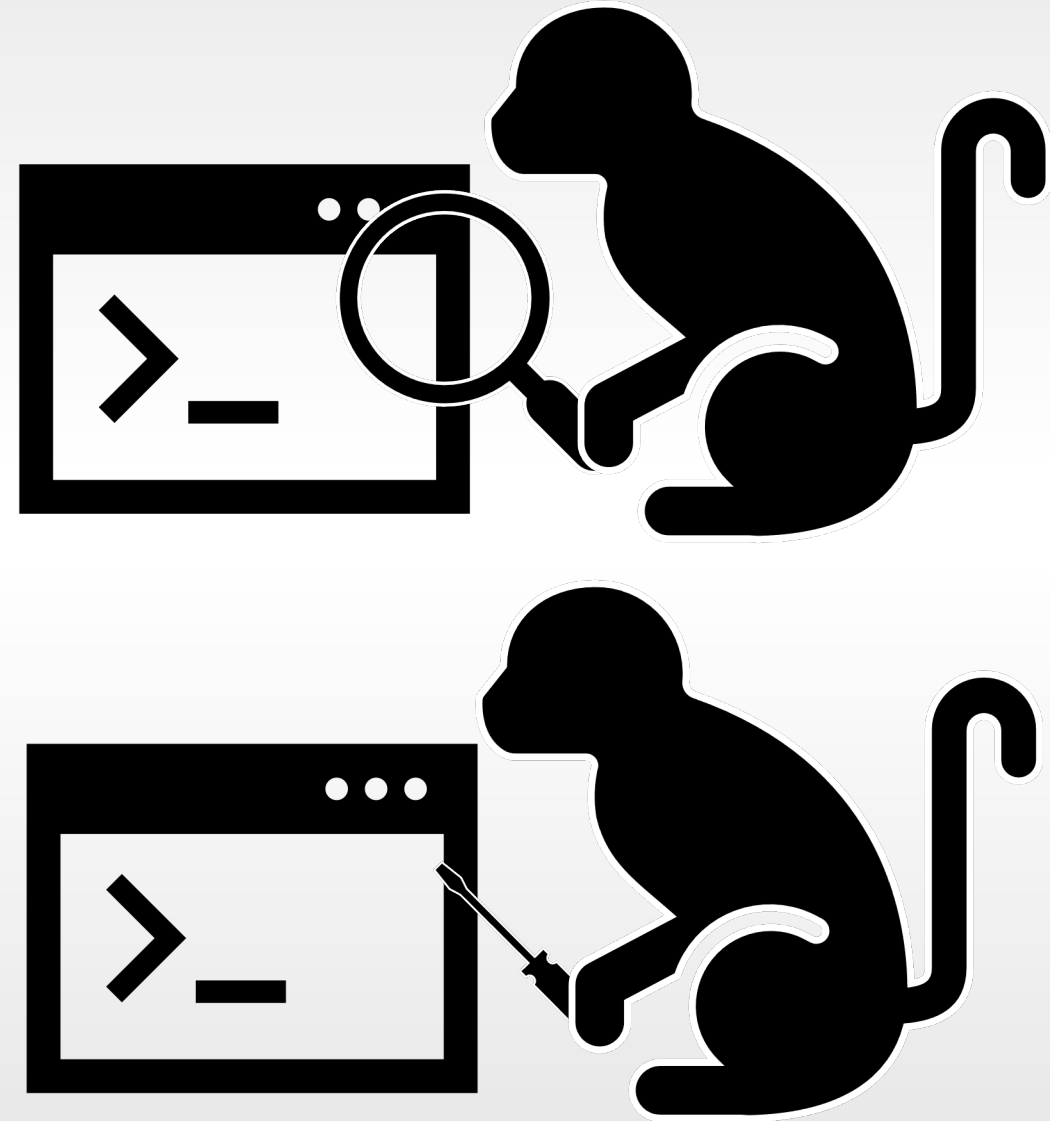
Results (3)

	Desktop	Smartphone	Browser	E-reader	Total
URI handles	4 (15%)	10 (18%)	10 (100%)	0	24 (25%)
Insecure web engine	2 (7%)	0 *	0 *	1 (20%)	3 (3%)

* Embedded web engine updated automatically

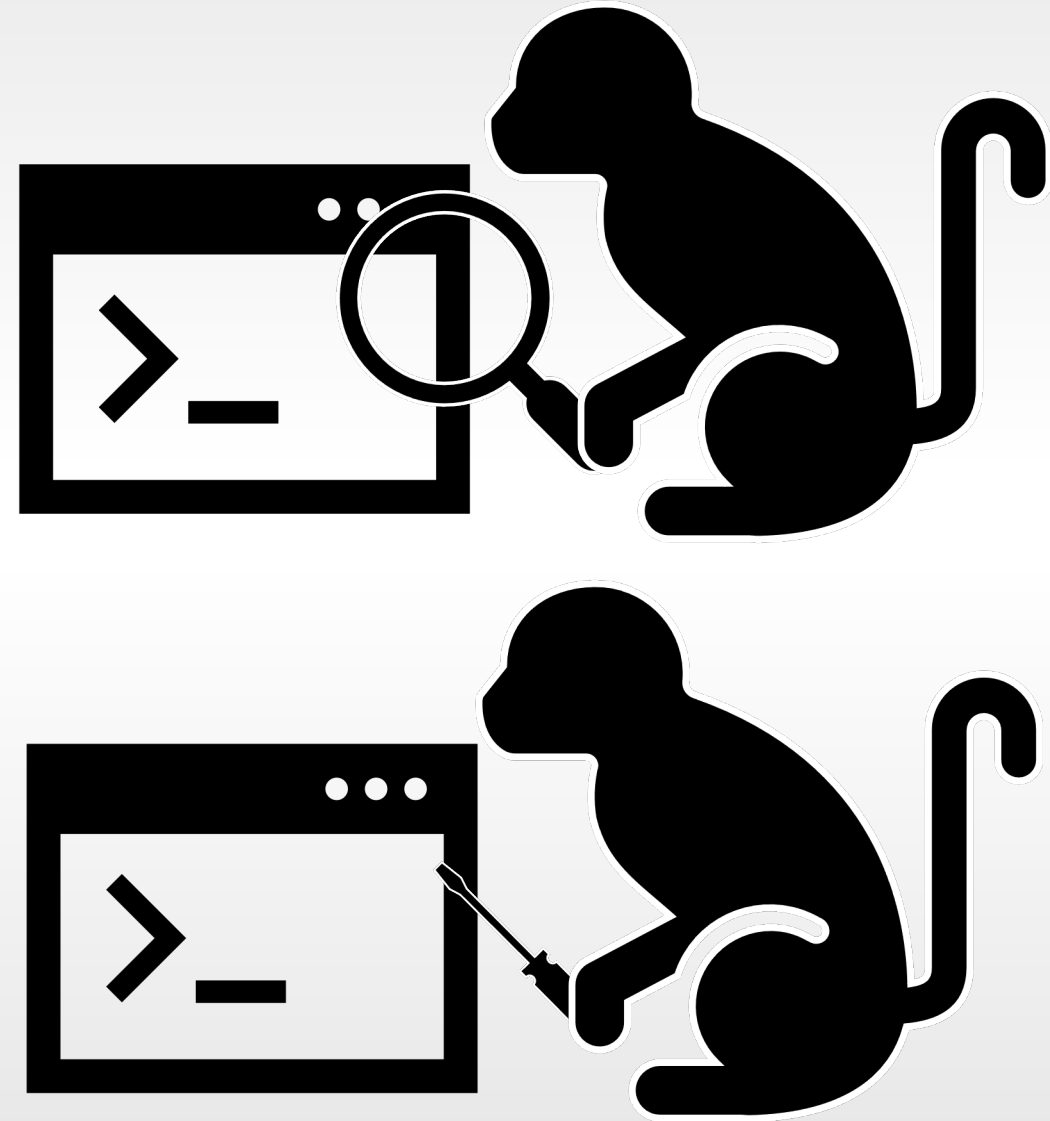
Case studies

- Apple Books
- EPUBReader (Chrome and Firefox extension)
- Amazon Kindle



Case studies

- Apple Books
 - Sym link validation issue
 - persistent DOS
 - user information disclosure
- EPUBReader (Chrome and Firefox extension)
 - CSP circumvention + `<all_urls>` permission
 - universal XSS
- Amazon Kindle
 - Input validation issue + publicly known vulnerability (10 year old WebKit)
 - information leaking



Research questions



- What is the state of freely available EPUB reading systems?



Granted capabilities



Security considerations

- Are these capabilities being (ab)used in the wild?



Malicious EPUBs



Tracking EPUBs

Capability (ab)use in the wild

amazon books



Google Play
Books

kobo



Project
Gutenberg

eBooks.com

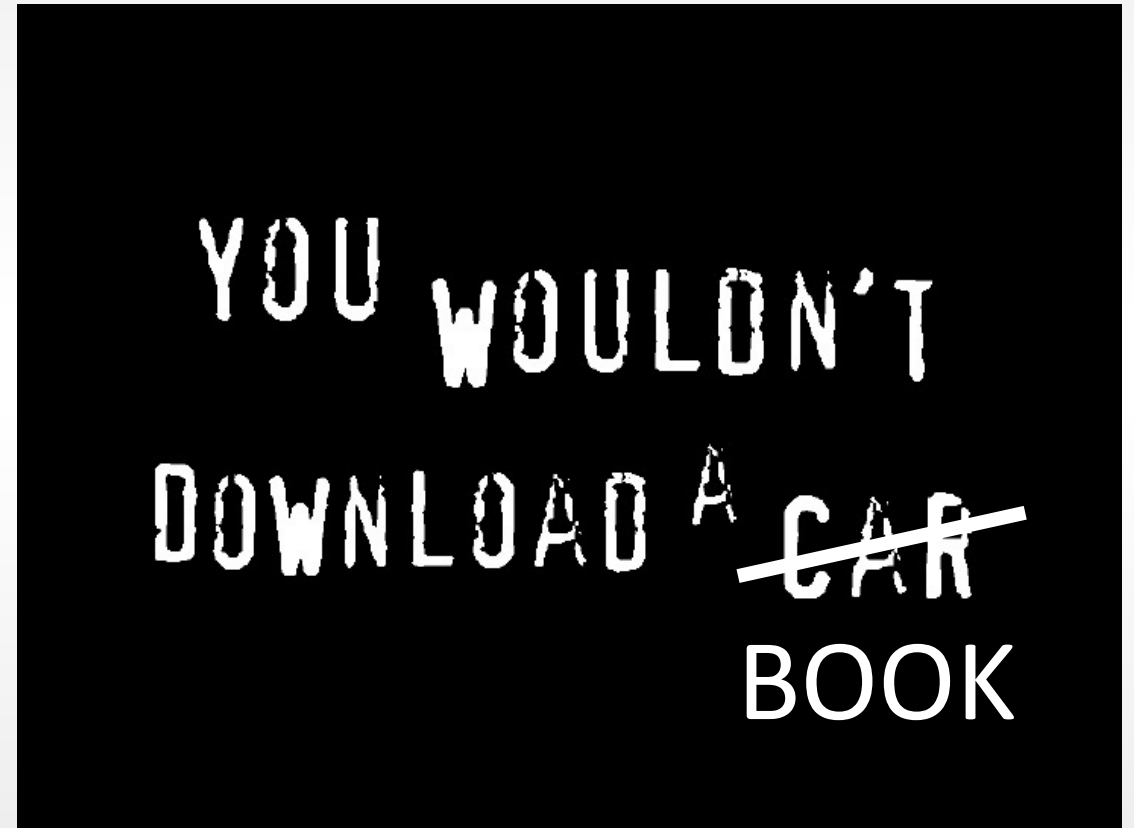
YOU WOULDN'T
DOWNLOAD A ~~CAR~~
BOOK

Capability (ab)use in the wild

- Malicious EPUBs distributed through illegal channels
 - The Pirate Bay, 4shared
 - +/- 9,000 EPUBs



< 1% contained JavaScript (all benign)



Capability (ab)use in the wild

amazon books



Google Play
Books

kobo

Project
Gutenberg

eBooks.com

- Tracking EPUBs distributed through legal channels
 - Free e-books from the most popular EPUB vendors



No indications of tracking

Feasibility of e-book distribution

Self-publishing
services!



4 shared



Distributing your malicious e-book
through **file sharing platforms**



Distributing your malicious e-book
through **official e-book vendors**

amazon books

Get it on
Apple Books

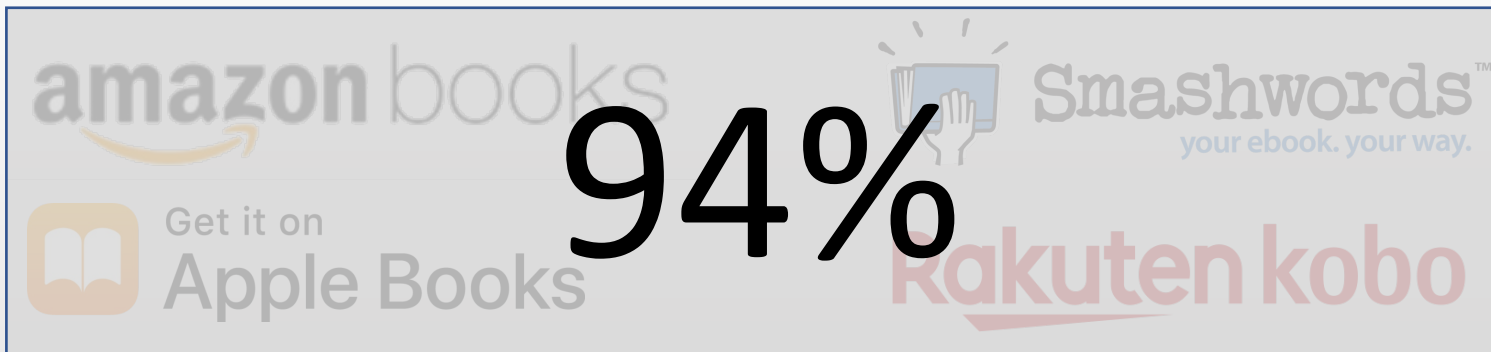
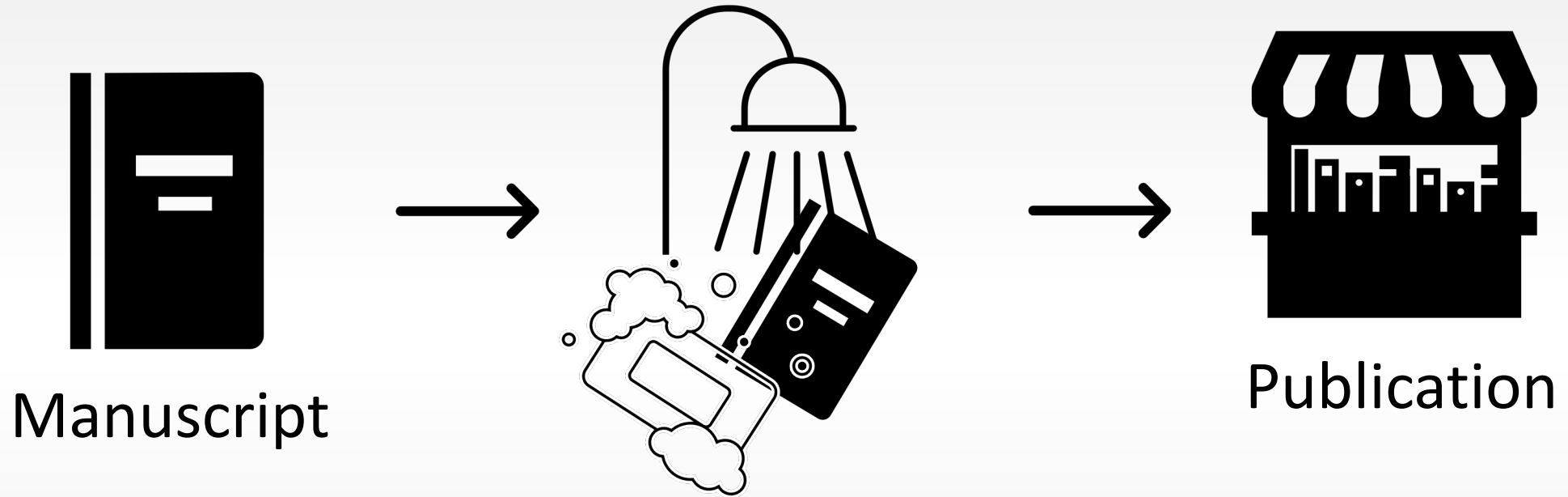
Google Play
Books

BARNES & NOBLE

Smashwords™
your ebook. your way.

Rakuten kobo

Are self-published EPUBs sufficiently sanitized?



BARNES & NOBLE

Key takeaways

- Almost none of the JS-supporting reading systems adhere to security recommendations
 - Significant part does not sufficiently isolate local file system
 - Responsible disclosure: developers of 37 reading systems contacted
- No abuse in the wild detected (as of yet)
 - Although very possible, even through legitimate channels
- Evaluation testbed is open-source [1]
 - Assist EPUB reading system developers
 - Provide transparency to users

EPUB 3.2 concerns (1)

- JavaScript execution:
 - Our real-world study showed minor usage -> limited usability impact
 - Greatly increases attack surface -> huge security impact
 - Prohibit JavaScript execution?
 - Could be recommended as default setting → modifiable by user?
 - User consent requirement?
- Remote resources
 - Local file system -> implies ability to read user file system
 - Huge security impact, limited usability impact?
 - What are the use-cases?
 - Online -> implies ability to leak collected information
 - Huge security impact
 - What are the use-cases?

EPUB 3.2 concerns (2)

- URI schemes
 - Performing malicious actions via installed applications
 - E.g. Skype 4 Business on MacOS + <tel:xxxxxxxx> → initiate call without user interaction

EPUB 3.2 concerns

- Embedded web engine configuration
 - Awareness of unintentionally inherited engine functionality
 - E.g. URI schemes, GeoLocation, MediaDevices
 - Overrides of security defaults
 - E.g. --allow-file-access-from-files, --disable-web-security
 - Updating frequently

EPUB 3.2 concerns (3)

- Hard / strict requirements instead of recommendations
 - At the time of the evaluation, only a few reading systems adhere to the recommendations
- Creating awareness among users and developers
 - Compliance checker?
 - Practical developer guidelines?
 - E.g. how to correctly configure embedded web engine