Parsing CBOR Is A Minefield

A Study of CBOR Parser behavior

Whoami

- Jakob Pachmann
- Offensive Security (OSCP, BSCP)
- Friction Points with CBOR so far: None
- Currently finishing my Masters - which is where this topic comes from



jakob.pachmann@proton.me
https://www.jpac.at
Github: v0xyc0n

What this talk is about

- A parser/decoder analyzes data according to a format's syntax rules and identifies its structure to produce values the application can use, such as integers, strings, or maps.
- Automated comparison of 11 parsers
  - Crash / Hang / Timeout?
  - Single / Illegal Acceptance?
- Combination of probing methods to increase testing surface: Manual, Fuzzing, Differential testing, CVE Analysis
- Introduction to difficulties and learnings from building the test suite

What this talk is **not** about

- Comprehensive analysis of all cbor parsers out there (cbor.io: 54)
- Analysis of all configurations of a single parser
- Test across different hardware configurations
- Analysis of (correctness of) output
- Analysis of higher-level issues (COSE, CWT, CDDL, WebAuthn…)
- All results preliminary: ongoing thesis work, results open to change

What this talk is *actually* about

- The scientific method
- Trying to take shortcuts and getting stuck
- Sharing my struggles
- Memes maybe



0x6443424F52

CBOR

RFC 8949

WTF is CBOR

CBOR

RFC 8949 Concise Binary Object Representation

"The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation."

WTF is CBOR

JSON data model

CBOR is based on the wildly successful JSON data model: numbers, strings, arrays, maps (called objects in JSON), and a few values such as false, true, and null.

No Schema needed

One of the major practical wins of JSON is that successful data interchange is possible without casting a schema in concrete. This works much better in a world where both ends of a communication relationship may be evolving at high speed.

Embracing binary

Some applications that would like to use JSON need to transport binary data, such as encryption keys, graphic data, or sensor values. In JSON, these data need to be encoded (usually in base64 format), adding complexity and bulk.

Concise encoding

Some applications also benefit from CBOR itself being encoded in binary. This saves bulk and allows faster processing. One of the major motivators for the development of CBOR was the Internet of Things, which will include very simple, inexpensive nodes where this counts.

Stable format

CBOR is defined in an Internet Standard, RFC 8949. The format has been designed to be stable for decades.

Extensible

To be able to grow with its applications and to incorporate future developments, a format specification needs to be extensible. CBOR defines **tags** as a mechanism to identify data that warrants additional information beyond the basic data model. Both future RFCs and third parties can define tags, so innovation is “permissionless” but can still be coordinated.

CBOR is… complicated

- Derived from MessagePack, adds extensibility via tags and formalization as standard
- Prominent in authentication (Fido, Passkeys, Webauthn), and IoT (CoAP).
- Apparently, part of quite a few packages:

```
┌──(jakob㉿darth-kali)-[~/Documents/Masterarbeit]
└─$ sudo apt remove libcbor-dev libcbor0.10
[sudo] password for jakob:
The following packages were automatically installed and are no longer required:
  libsimdjson30  libwtmpdb0  openjdk-21-jre  openjdk-21-jre-headless
Use 'sudo apt autoremove' to remove them.

REMOVING:
  kali-linux-core     kali-linux-headless  kali-system-core  libcbor-dev  libfido2-1      openssh-client-gssapi  openssh-sftp-server
  kali-linux-default  kali-system-cli      kali-system-gui   libcbor0.10  openssh-client  openssh-server         snapd

Summary:
  Upgrading: 0, Installing: 0, Removing: 14, Not Upgrading: 1602
  Freed space: 79.5 MB
```

CBOR is… complicated – Payload

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

Type

Major Type	Meaning	Content
0	unsigned integer N	-
1	negative integer -1-N	-
2	byte string	N bytes
3	text string	N bytes (UTF-8 text)
4	array	N data items (elements)
5	map	2N data items (key/value pairs)
6	tag of number N	1 data item
7	simple/float	-

Table 1: Overview over the Definite-Length Use of CBOR Major Types (N = Argument)

Additional Info

- <24 → contains “N” itself
- 24/25/26/27 → look 1/2/4/8 bytes ahead
- 31 → indefinite length till stop code (0xFF)

CBOR is… complicated – Tags & Simple vals

- **Simple Values** (Type 7): false, true, null, undefined (f4-f7), e.g.:
  111 10100 --> false

CBOR is… complicated – Tags & Simple vals

- **Simple Values** (Type 7): false, true, null, undefined (f4-f7), e.g.: 111 10100 --> false
- ~300 **Tags** registered (June 2026) at IANA
  - Most of these just add semantic meaning → Test for “invalid”
  - Notable exceptions:

Tag	Meaning
25	reference the nth previously seen string
29	reference nth marked [by Tag 28] value

Tag	Data Item	Semantics
0	text string	Standard date/time string; see Section 3.4.1
1	integer or float	Epoch-based date/time; see Section 3.4.2
2	byte string	Unsigned bignum; see Section 3.4.3
3	byte string	Negative bignum; see Section 3.4.3
4	array	Decimal fraction; see Section 3.4.4
5	array	Bigfloat; see Section 3.4.4
21	(any)	Expected conversion to base64url encoding; see Section 3.4.5.2
22	(any)	Expected conversion to base64 encoding; see Section 3.4.5.2
23	(any)	Expected conversion to base16 encoding; see Section 3.4.5.2
24	byte string	Encoded CBOR data item; see Section 3.4.5.1
32	text string	URI; see Section 3.4.5.3
33	text string	base64url; see Section 3.4.5.3
34	text string	base64; see Section 3.4.5.3
36	text string	MIME message; see Section 3.4.5.3
55799	(any)	Self-described CBOR; see Section 3.4.6

Table 5: Tag Numbers Defined in RFC 7049

Some examples

Hex

00

20

61 61

82 01 02

9f 01 02 ff

a1 01 02

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

0xFF → Stop

0	1	2	3	4	5	6	7
uint	Neg. int: -1 - N	Byte string	Text string	array	map	Tag	Simple/float

Some examples

Hex	Binary				
00	000 00000				
20	001 00000				
61 61	011 00001	011 00001			
82 01 02	100 00010	000 00001	000 00010		
9f 01 02 ff	100 11111	000 00001	000 00010	111 11111	
a1 01 02	101 00001	000 00001	000 00010		

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

0xFF → Stop

0	1	2	3	4	5	6	7
uint	Neg. int: -1 - N	Byte string	Text string	array	map	Tag	Simple/float

Some examples

Hex	Binary				Decoded
00	000 00000				0
20	001 00000				-1
61 61	011 00001	011 00001			“a”
82 01 02	100 00010	000 00001	000 00010		[1, 2]
9f 01 02 ff	100 11111	000 00001	000 00010	111 11111	[1, 2]
a1 01 02	101 00001	000 00001	000 00010		{1: 2}

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

0xFF → Stop

0	1	2	3	4	5	6	7
uint	Neg. int: -1 - N	Byte string	Text string	array	map	Tag	Simple/float

CBOR is… complicated – Payload

A2646E616D65654A616B6F6263616765181D

CBOR is… complicated – Payload

A2646E616D65654A616B6F6263616765181D

```
{
" name": "Jakob",
"age": 29
}
```

```
A2                  # map(2)
  64                # text(4)
    6E616D65        # "name"
  65                # text(5)
    4A616B6F62      # "Jakob"
  63                # text(3)
    616765          # "age"
  18 1D             # unsigned(29)
```

Want to try for yourself?

https://cbor.me

CBOR is… complicated - Parser Variations

- Stream parsers vs. One-Shot Parsers – What do we do with leftover data?
- Rejection vs replacement of invalid data with stream parsers
- Generic vs Extended Data Model (Tags, mostly)
- Strictness differs widely and is at times highly configurable
- → Often, finding the ”base” implementation is pretty hard™

Parser selection

- Criteria:
  - Widely used
  - Open source
  - Different languages
  - … In hindsight, that might have been an issue
  - Past CVEs?
- Parsers:

tinycbor	libcbor	cbor2 (py)
cbor2 (js)	cborg	node-cbor
PeterO.Cbor	ciborium	fxamacker/cbor
com.upokecenter.cbor	jackson-dataformat-cbor	




yubico


spring
Framework


aws


KEYCLOAK


Qt


redisson


STEAMOS™


tailscale

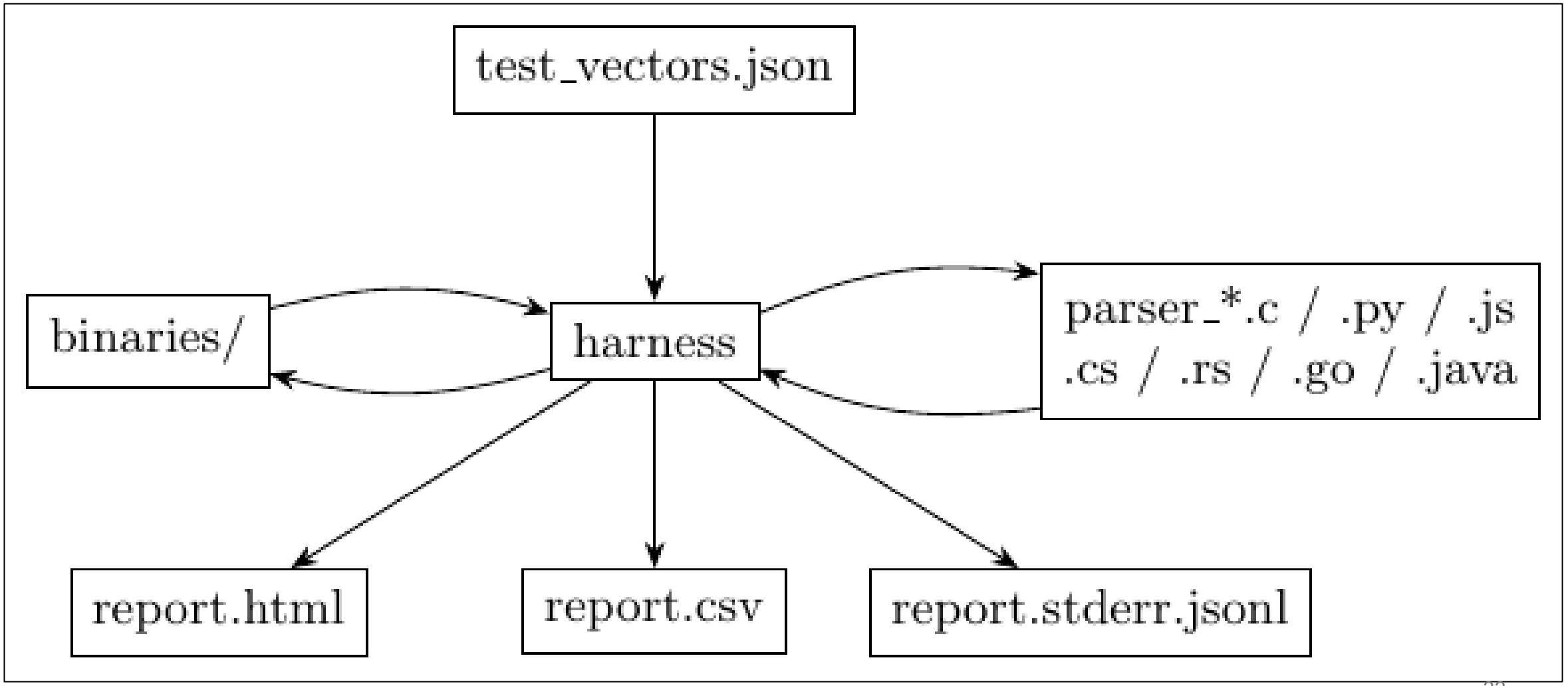

EMU


Bluesky

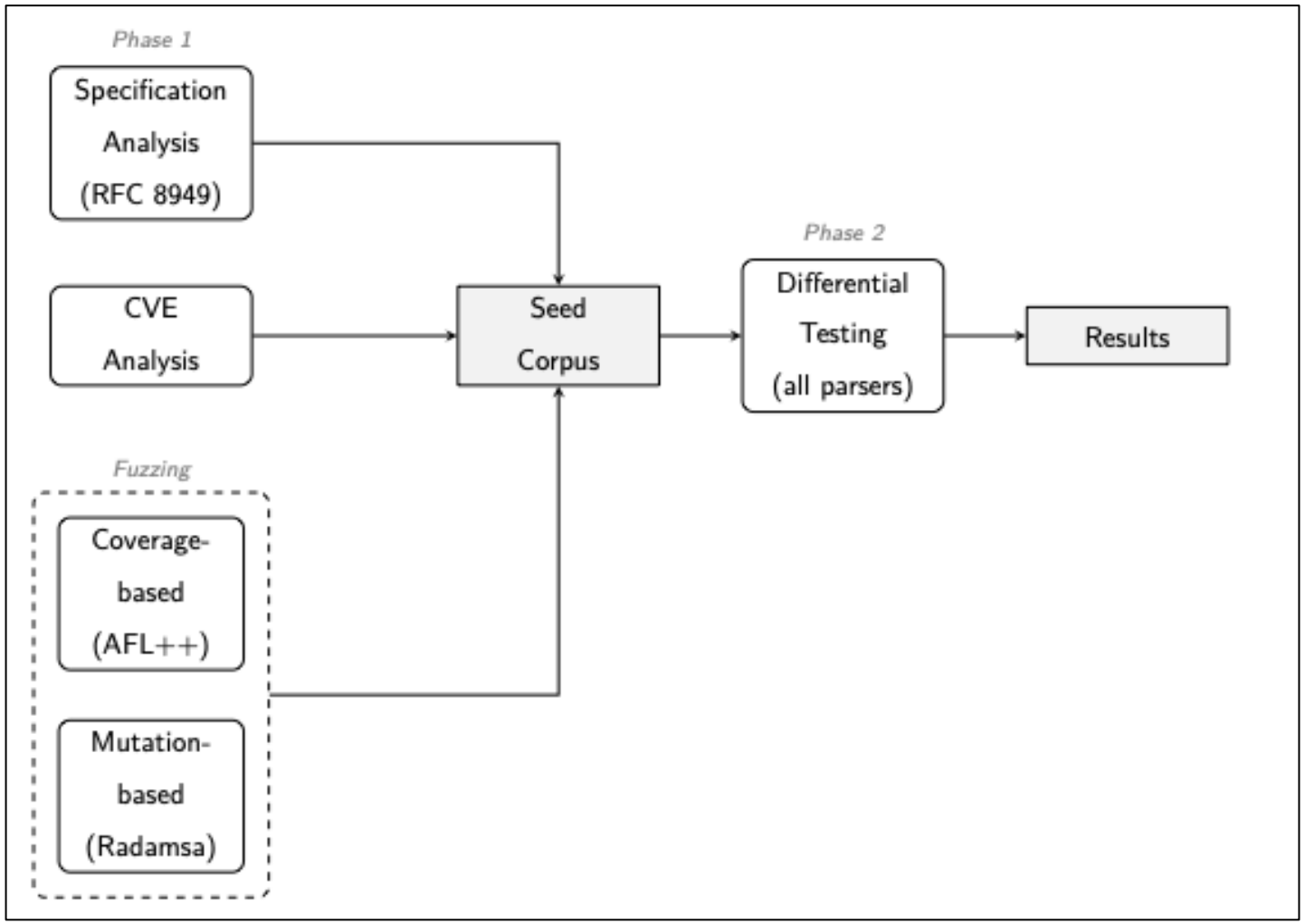

Apache
Solr


elasticsearch

Differential Testing Harness



Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results

Specification Analysis

1. Read the RFC (64 Pages of riveting prose)

Status:	Internet Standard
Obsoletes:	7049
See Also:	STD 94
More info:	Errata exist | Datatracker | IPR | Info page

Stream:	Internet Engineering Task Force (IETF)
RFC:	8949
STD:	94
Obsoletes:	7049
Category:	Standards Track
Published:	December 2020
ISSN:	2070-1721
Authors:	C. Bormann, *Universität Bremen TZI*; P. Hoffman, *ICANN*

RFC 8949
Concise Binary Object Representation (CBOR)

Abstract

The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation. These design goals make it different from earlier binary serializations such as ASN.1 and MessagePack.

This document obsoletes RFC 7049, providing editorial improvements, new details, and errata fixes while keeping full compatibility with the interchange format of RFC 7049. It does not create a new version of the format.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8949.

Specification Analysis

1. Read the RFC (64 Pages of riveting prose)
2. Make notes of edge-cases

Thoughts About... CBOR_RFC8949_Security_Aspects

Summary

Reading the RFC to find edge cases for my corpus.

Content

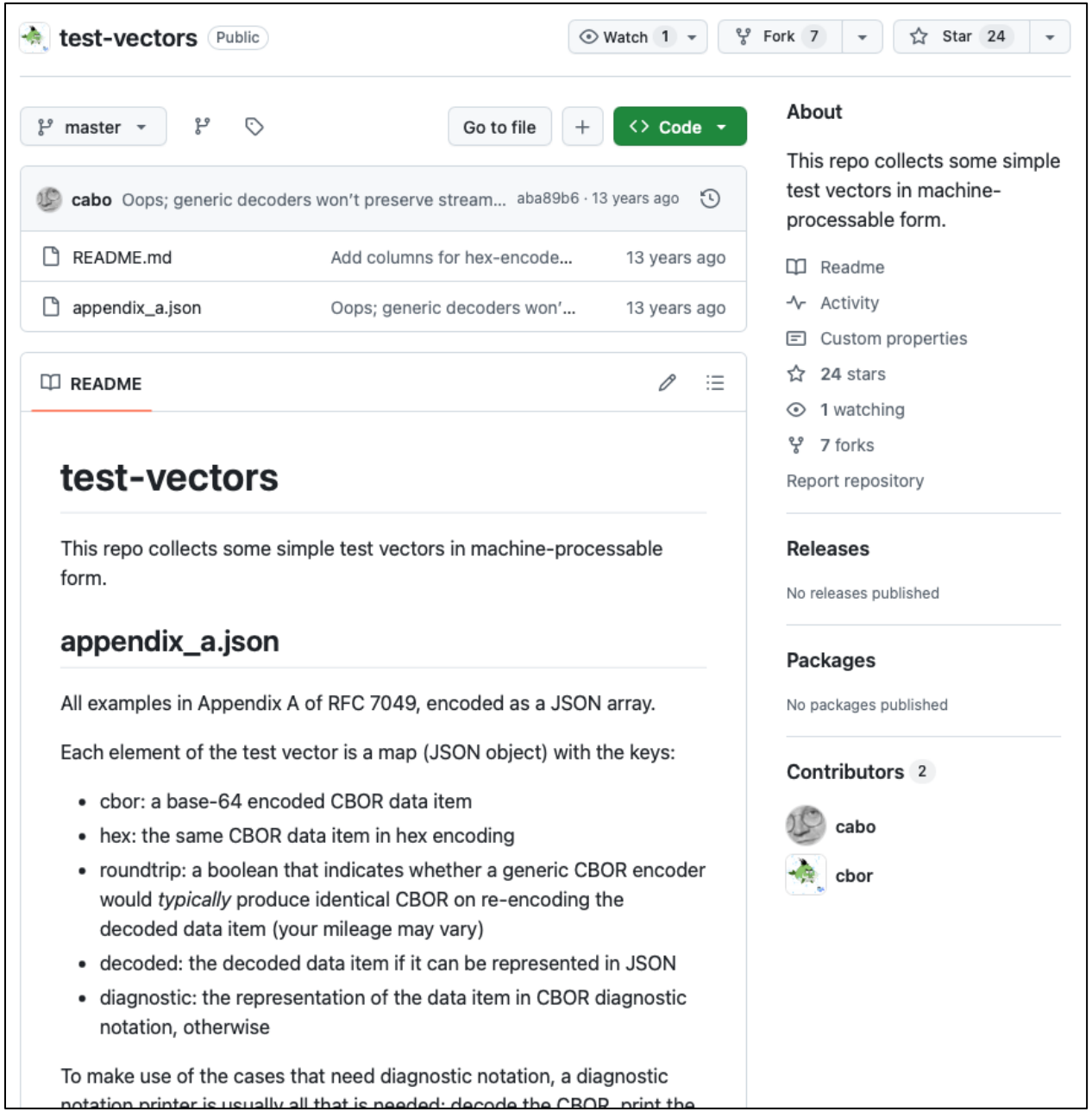
- Data model - what abstract values can be expressed (e.g. "String"), vs
- Syntax/encoding - the concrete bytes used to represent them (e.g. `{"key":"value"}`)
- CBOR uses JSONs syntax and extends its data model
- Data formats:
  - There is no requirement that all data formats be uniquely encoded; that is, it is acceptable that the number "7" might be encoded in multiple different ways.
  - The structures supported are limited to arrays and trees;
- Encoders
  - An encoder and a decoder need to be implementable in a very small amount of code (for example, in class 1 constrained nodes --- This means 10KB RAM, 100KB disk size).
  - The format should use contemporary machine representations of data (for example, not requiring binary-to-decimal conversion).
- A decoder MUST NOT return a decoded data item when it encounters input that is not a well-formed encoded CBOR data item [...] --> Interesting

Testcases

- Check for feeding Data Streams to the parsers, I think not all of them know how to deal with that; --> Check for all parsers if they have "stream decoders"
- In the basic (unextended) generic data model --> what is the extended data model, should I test that too?
  - `false`, `true`, `null`, and `undefined` (simple values identified by 20..23, Section 3.3)

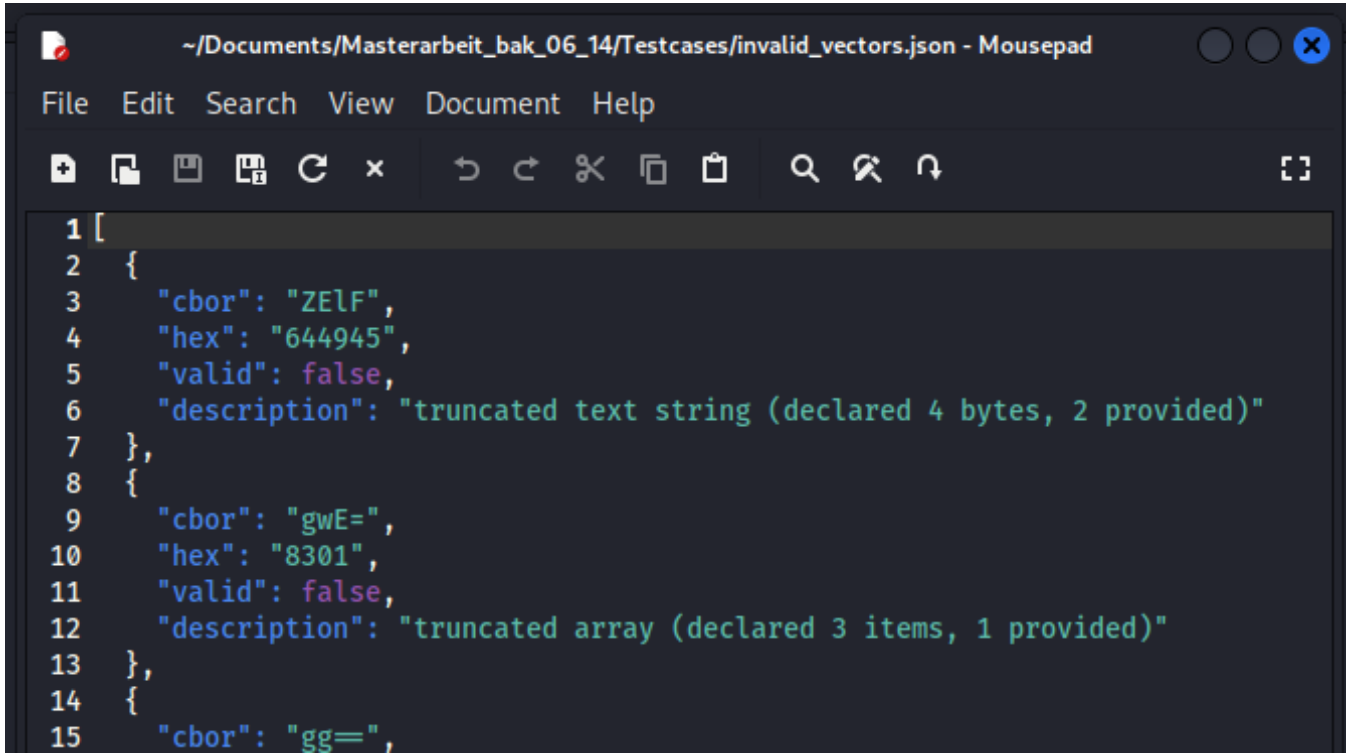
Specification Analysis

1. Read the RFC (64 Pages of riveting prose)
2. Make notes of edge-cases
3. Find a random repo with testcases



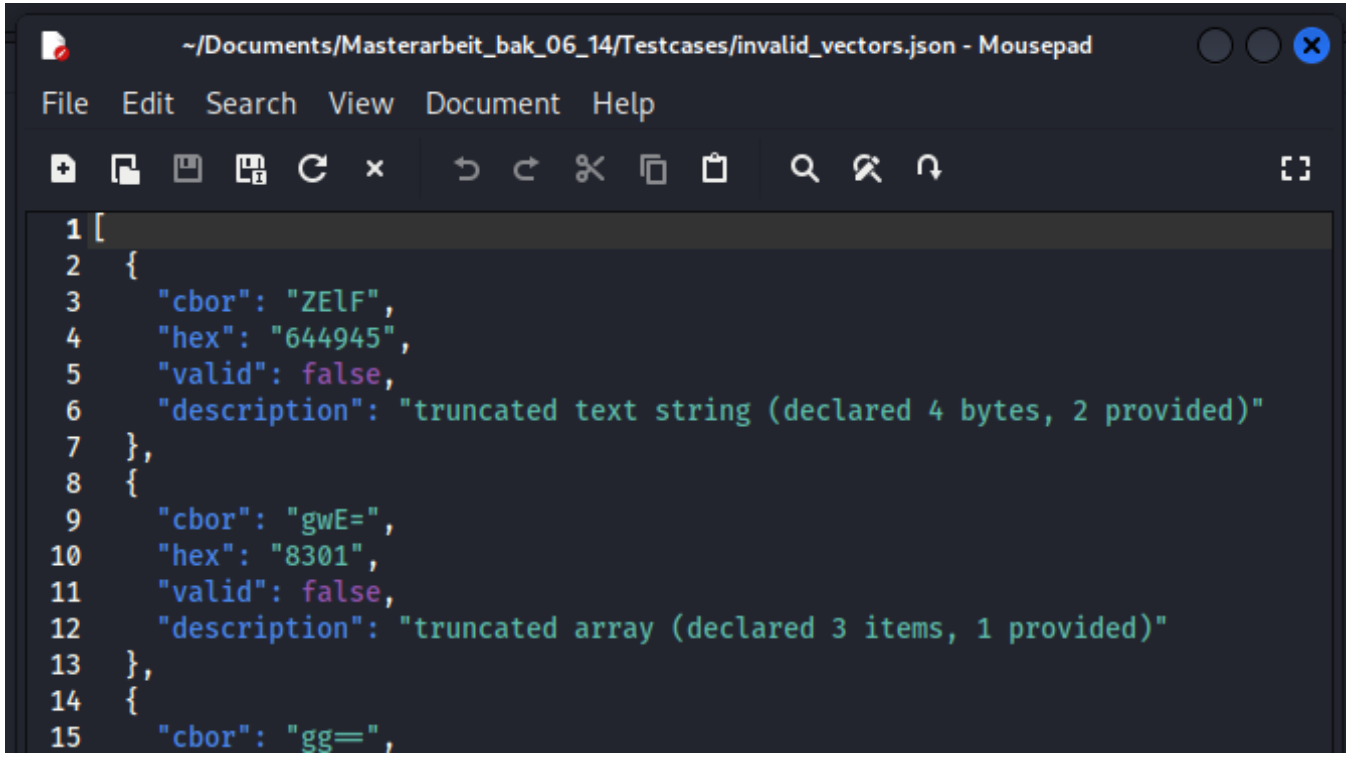
Specification Analysis

1. Read the RFC (64 Pages of riveting prose)
2. Make notes of edge-cases
3. Find a random repo with testcases
4. Use testcases and notes to *come up* with ~30 testcases



Specification Analysis

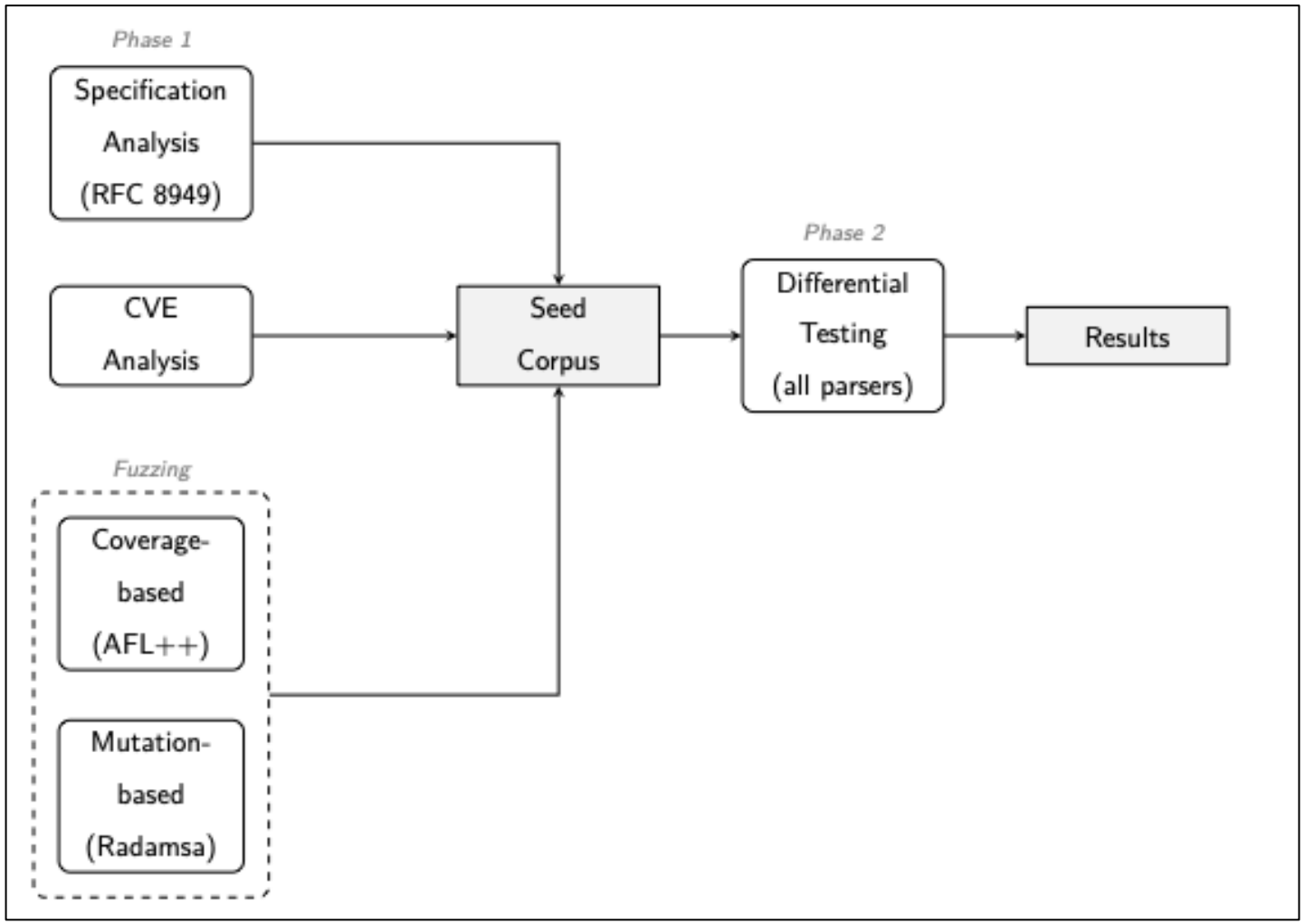
1. Read the RFC (64 Pages of riveting prose)
2. Make notes of edge-cases
3. Find a random repo with testcases
4. Use testcases and notes to *come up* with ~30 testcases
5. Let’s get testing!



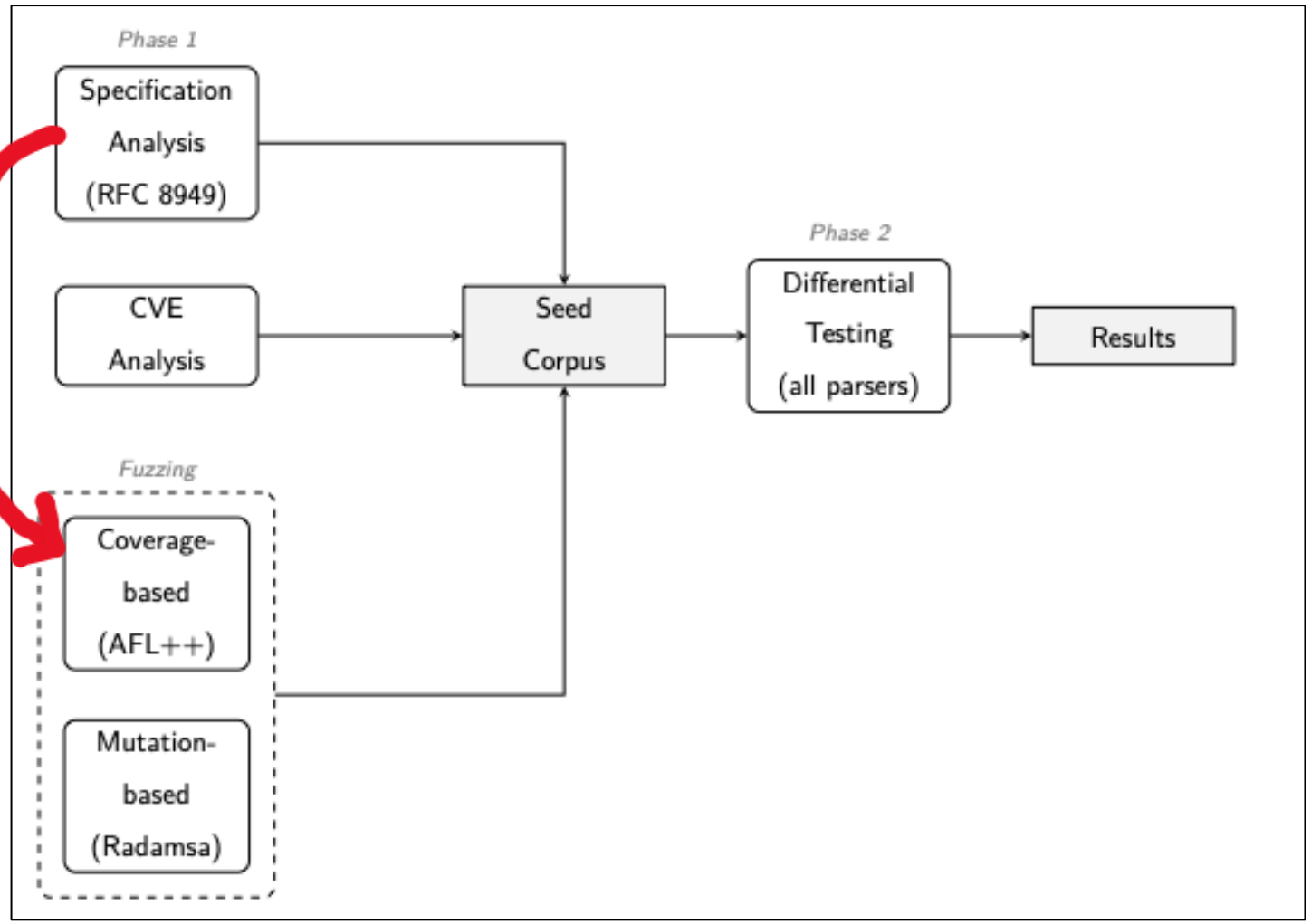
Fuzzing

- Increase the test-surface: introduce fuzzing
- 7 different languages; that is an issue
- I know about afl++, lets go with that and see what happens
- It needs a corpus of base payloads to start with; luckily I have just the thing!

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results

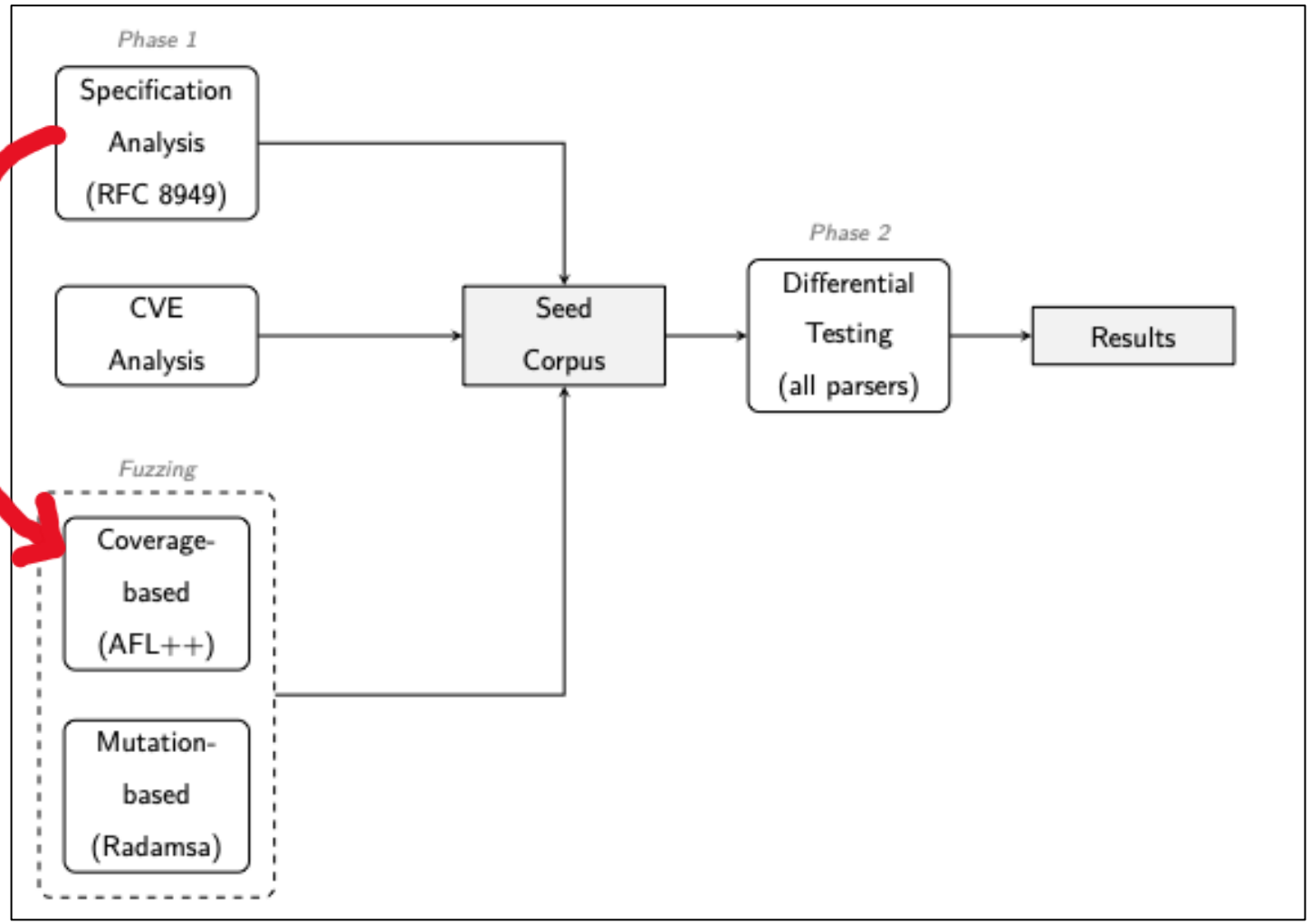
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results

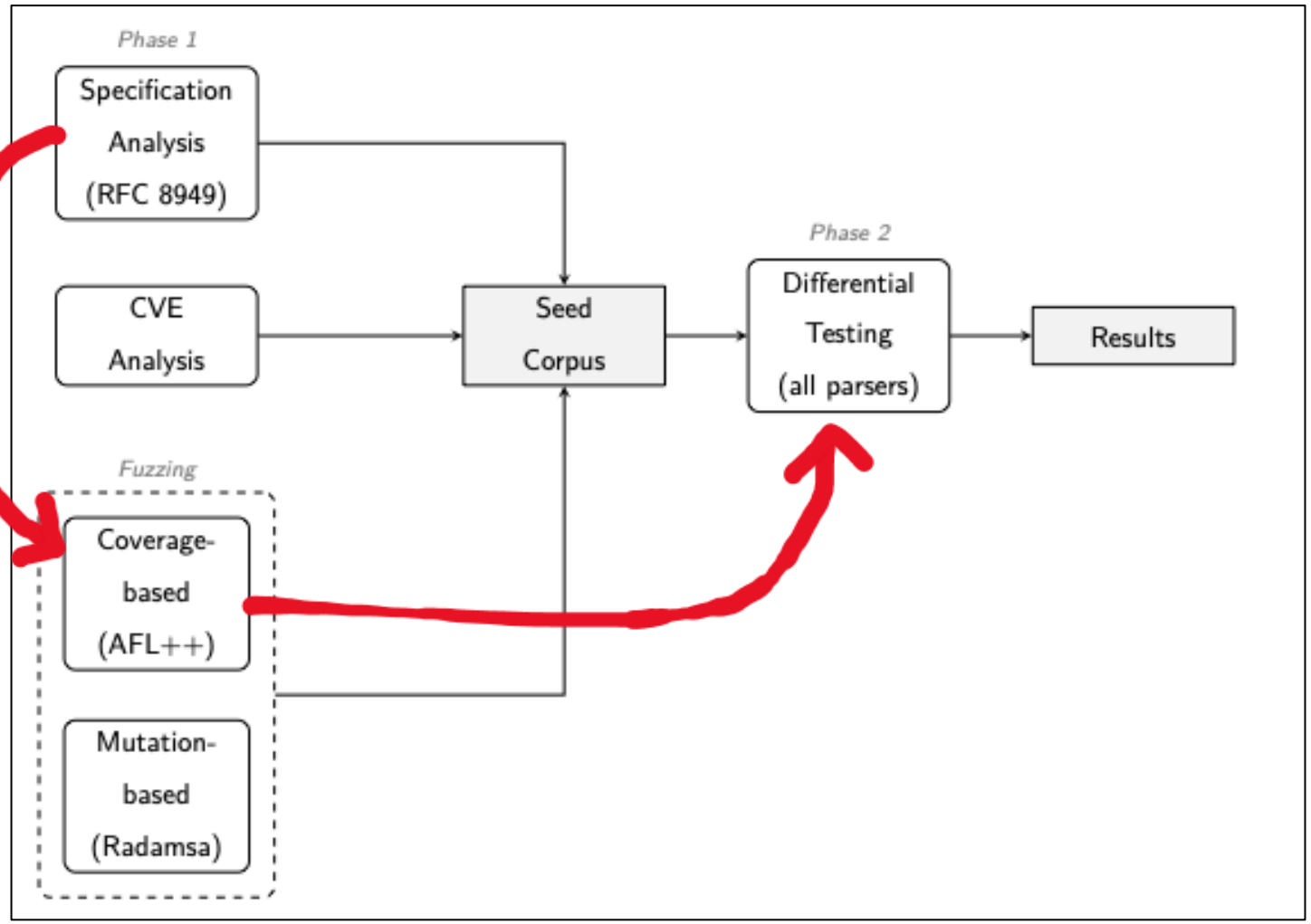
Fuzzing

- Increase the test-surface: introduce fuzzing
- 7 different languages; that is an issue
- I know about afl++, lets go with that and see what happens
- It needs a corpus of base payloads to start with; luckily I have just the thing!
- Wow, results! Let’s immediately try them against all parsers

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results

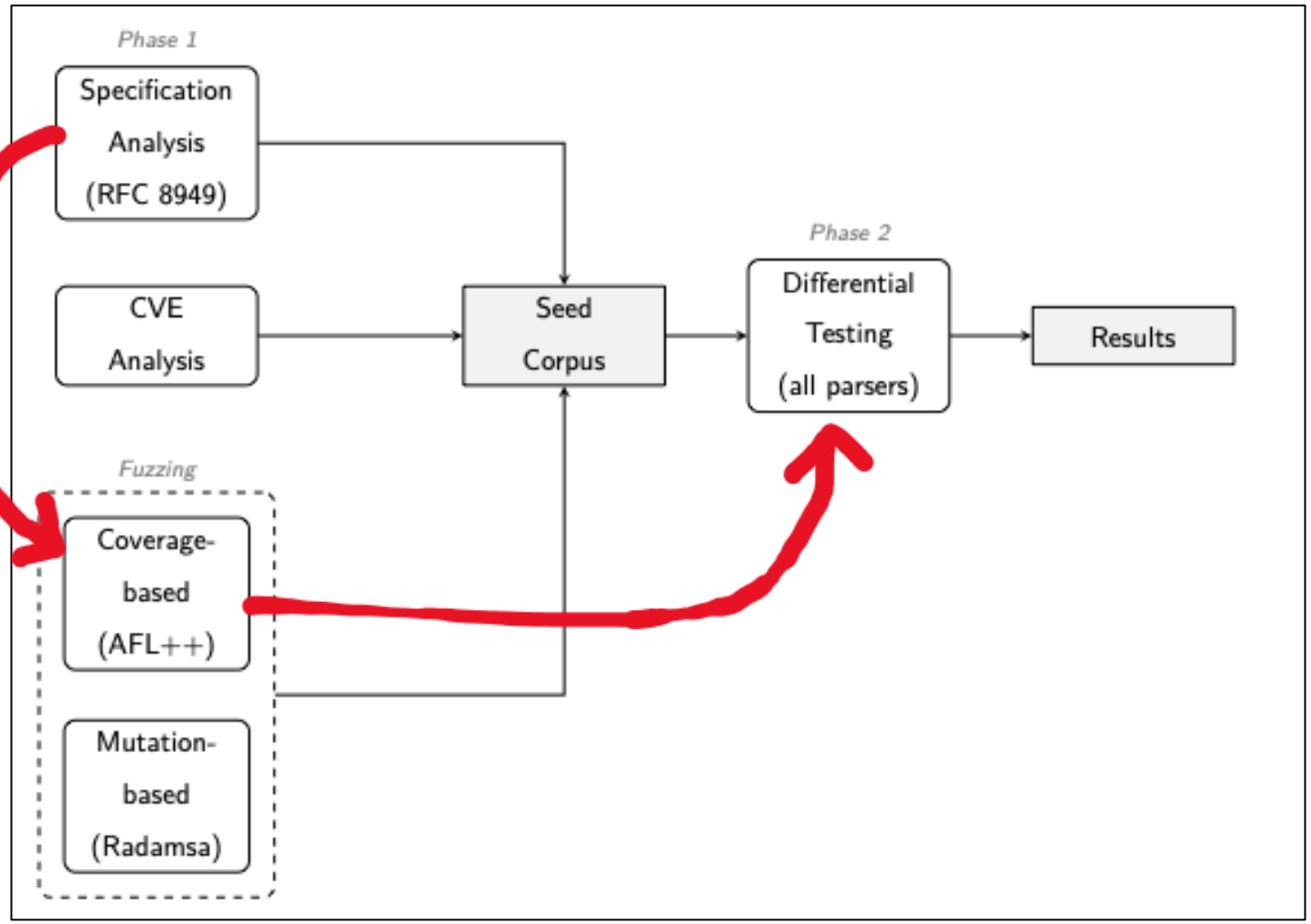
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)

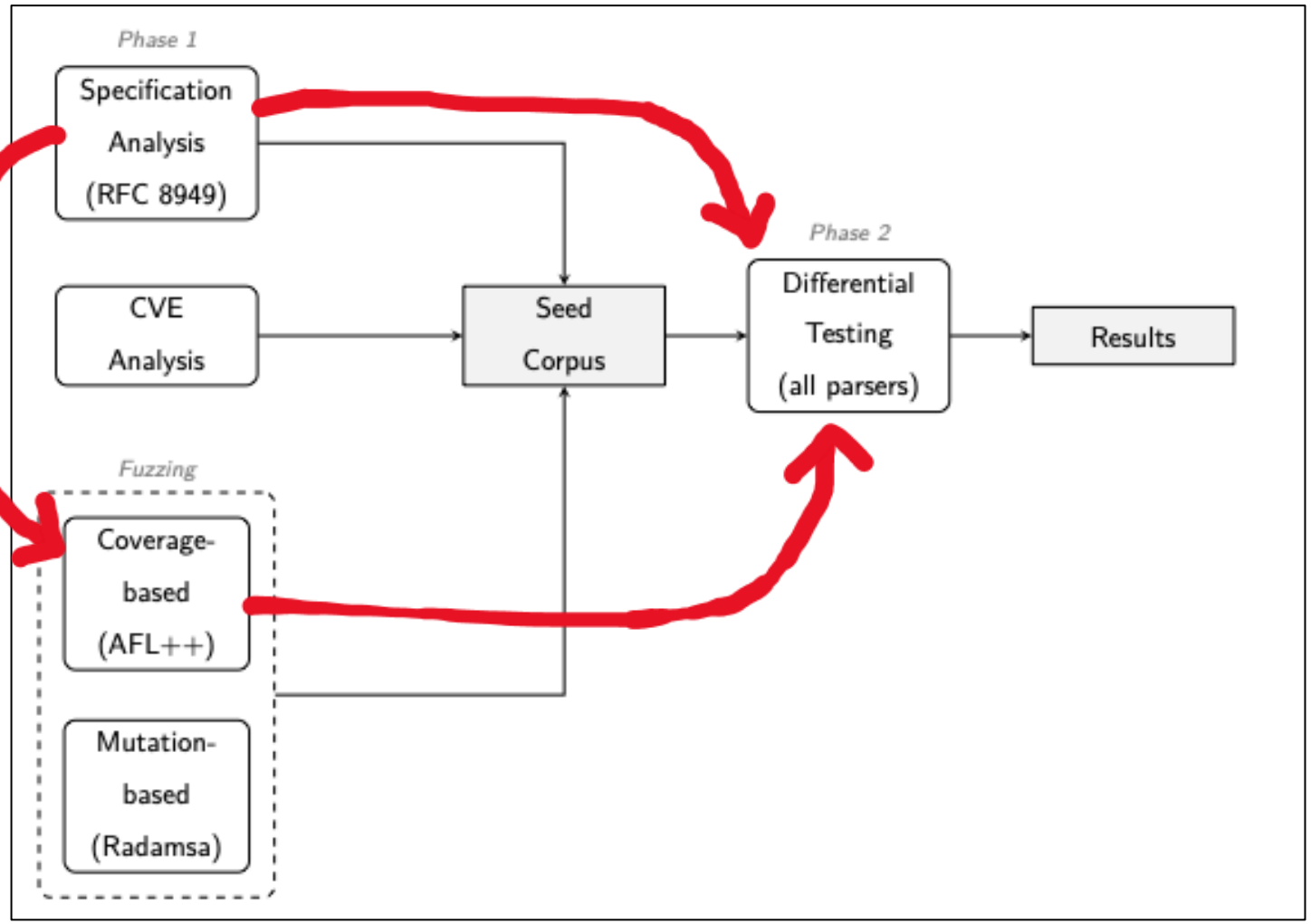
Fuzzing

- Increase the test-surface: introduce fuzzing
- 7 different languages; that is an issue
- I know about afl++, lets go with that and see what happens
- It needs a corpus of base payloads to start with; luckily I have just the thing!
- Wow, results! Let’s immediately try them against all parsers
- The differential testing harness also worked quite well

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results

index	hex	cbor2	libcbor	tinycbor	jackson	upokecenter	fxamacker	ciborium	petero	node-cbor	cborg	divergence
0	9a303a30343a3(	rejected	timeout	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	yes
1	644945	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
2	8301	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
3	82	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
4	a1	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
5	19	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
6	1a0001	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
7	1b00000000	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
8	ff	accepted	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	yes
9	fe	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
10	fc	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
11	62fffe	rejected	rejected	accepted	rejected	rejected	rejected	rejected	rejected	rejected	accepted	yes
12	a101	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
13	9fffff	accepted	accepted	accepted	accepted	accepted	rejected	accepted	rejected	rejected	rejected	yes
14	5f410160ff	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
15		rejected	rejected	rejected	accepted	rejected	rejected	rejected	rejected	accepted	rejected	yes
16	0000	accepted	accepted	accepted	accepted	accepted	rejected	accepted	rejected	rejected	rejected	yes
17	8181818181818:	rejected	accepted	accepted	accepted	accepted	rejected	rejected	accepted	accepted	accepted	yes
18	8181818181818:	rejected	accepted	accepted	accepted	rejected	rejected	rejected	rejected	accepted	accepted	yes
19	fa3f800000	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	
20	fb3ff00000000(	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	
21	1800	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	
22	190000	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	accepted	
23	5affffffff	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
24	7affffffff	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
25	9affffffff	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
26	baffffffff	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	
27	5bfffffffffff1	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	

Fuzzing

- Increase the test-surface: introduce fuzzing
- 7 different languages; that is an issue
- I know about afl++, lets go with that and see what happens
- It needs a corpus of base payloads to start with; luckily I have just the thing!
- Wow, results! Let’s immediately try them against all parsers
- The differential testing harness also worked quite well
- At this point I thought I was pretty much done

New parser, more & structured payloads

- One parser is EoL and recommends switch → No problem, my approach is extensible

New parser, more & structured payloads

- One parser is EoL and recommends switch → No problem, my approach is extensible
- Restructure and recreate the manually created dataset: Valid/invalid is not enough; there is also malformed data → 68 Vectors

New parser, more & structured payloads

- One parser is EoL and recommends switch → No problem, my approach is extensible
- Restructure and recreate the manually created dataset: Valid/invalid is not enough; there is also malformed data → 68 Vectors
- More payloads (spoiler: >200 from afl++): Random permutations with radamsa

New parser, more & structured payloads

- One parser is EoL (node-cbor) and recommends switch to cbor2
  → No problem, my approach is extensible
- Restructure and recreate the manually created dataset:
  Valid/invalid is not enough; there is also malformed data → 68 Vectors
- More payloads (spoiler: >200 from afl++): Random permutations with radamsa

```
1105  mkdir fuzzing/corpus_radamsa_5/
1106  radamsa -o fuzzing/corpus_radamsa_5/radamsa_%n.cbor -n 1000 cbor_binary/*.cbor
1107  python3 cbor_to_vectors.py fuzzing/corpus_radamsa_5 > Testcases/radamsa_1000_5.json
1108  python3 harness.py --vectors Testcases/radamsa_1000_5.json --out ./reports/radamsa_1000_5
```

Interjection – Validity and Well-Formedness

- Payloads can be
  - Well Formed: Syntax **correct**
    - …and Valid: Syntax correct, Semantics **correct**
    - …and Invalid: Syntax correct, Semantics **incorrect**
  - Malformed: Syntax **incorrect**
- Syntax: “can I read this?”
- Semantics: “does it (what I just read) make sense?”
- Malformed (Appendix F.): Too much/too little data, Syntax error
- Invalid
  - 5.3.1. Basic validity: Duplicate keys in a map, Invalid UTF-8 string
  - 5.3.2. Tag validity: Inadmissible type or value for tag content

Interjection – Validity and Well-Formedness - Quiz

Hex	Binary	Verdict
01		
9f 01 02		
82 01 02		
c0 01		

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

0xFF → Stop

Tag	Data Item	Semantics
0	text string	Standard date/time string; see Section 3.4.1

0	1	2	3	4	5	6	7
uint	Neg. int: -1 - N	Byte string	Text string	array	map	Tag	Simple/float

Interjection – Validity and Well-Formedness - Quiz

Hex	Binary			Verdict
01	000 00001			
9f 01 02	100 11111	000 00001	000 00010	
82 01 02	100 00010	000 00001	000 00010	
c0 01	110 00000	000 00001		

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

0xFF → Stop

Tag	Data Item	Semantics
0	text string	Standard date/time string; see Section 3.4.1

0	1	2	3	4	5	6	7
uint	Neg. int: -1 - N	Byte string	Text string	array	map	Tag	Simple/float

Interjection – Validity and Well-Formedness - Quiz

Hex	Binary			Verdict
01	000 00001			Valid: 1
9f 01 02	100 11111	000 00001	000 00010	Malformed: [1, 2
82 01 02	100 00010	000 00001	000 00010	Valid: [1, 2]
c0 01	110 00000	000 00001		Invalid: int 1 /= string

[3 bits major **type** (0-7) | 5 bits **additional info** (0-31)]

0xFF → Stop

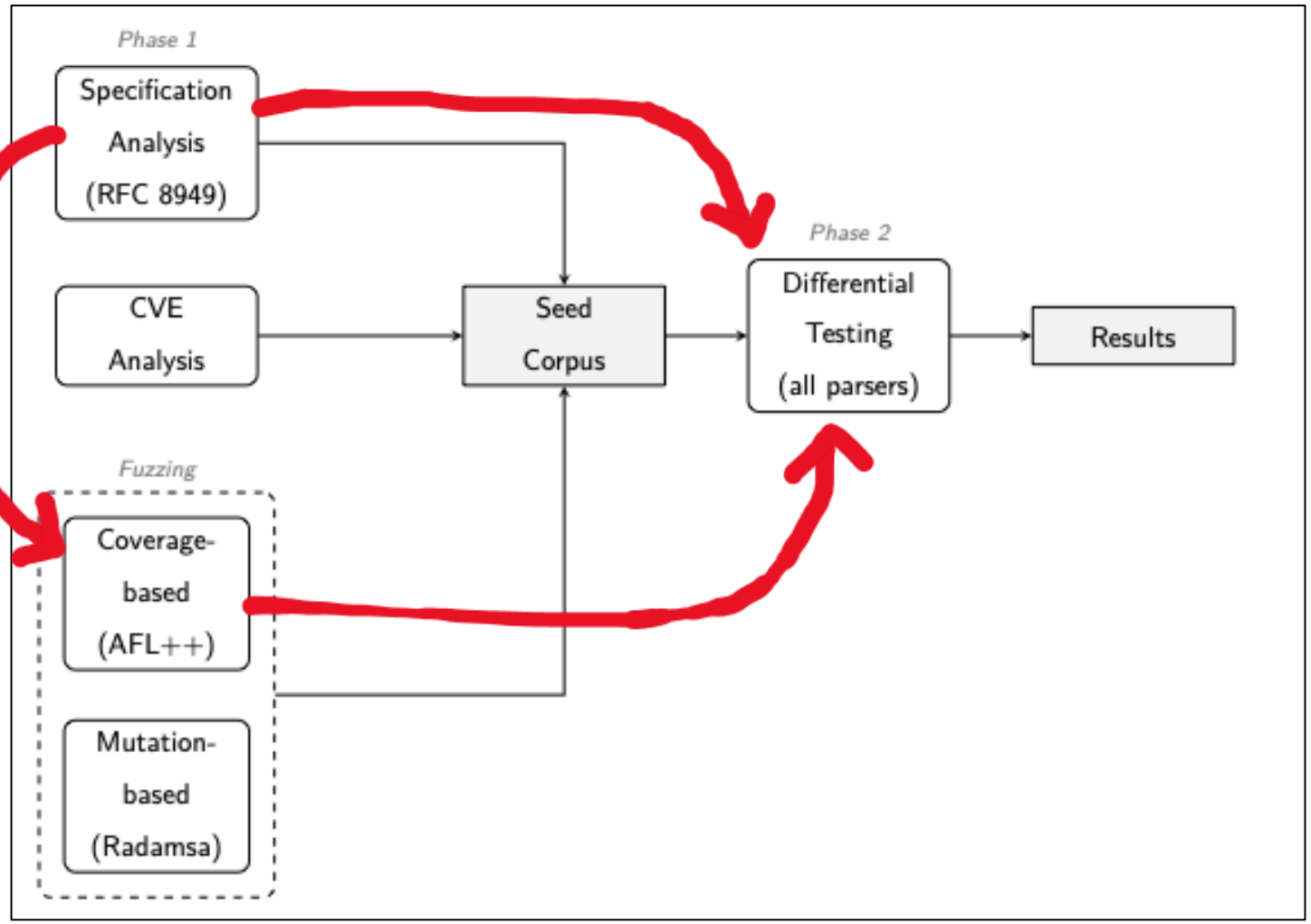
Tag	Data Item	Semantics
0	text string	Standard date/time string; see Section 3.4.1

0	1	2	3	4	5	6	7
uint	Neg. int: -1 - N	Byte string	Text string	array	map	Tag	Simple/float

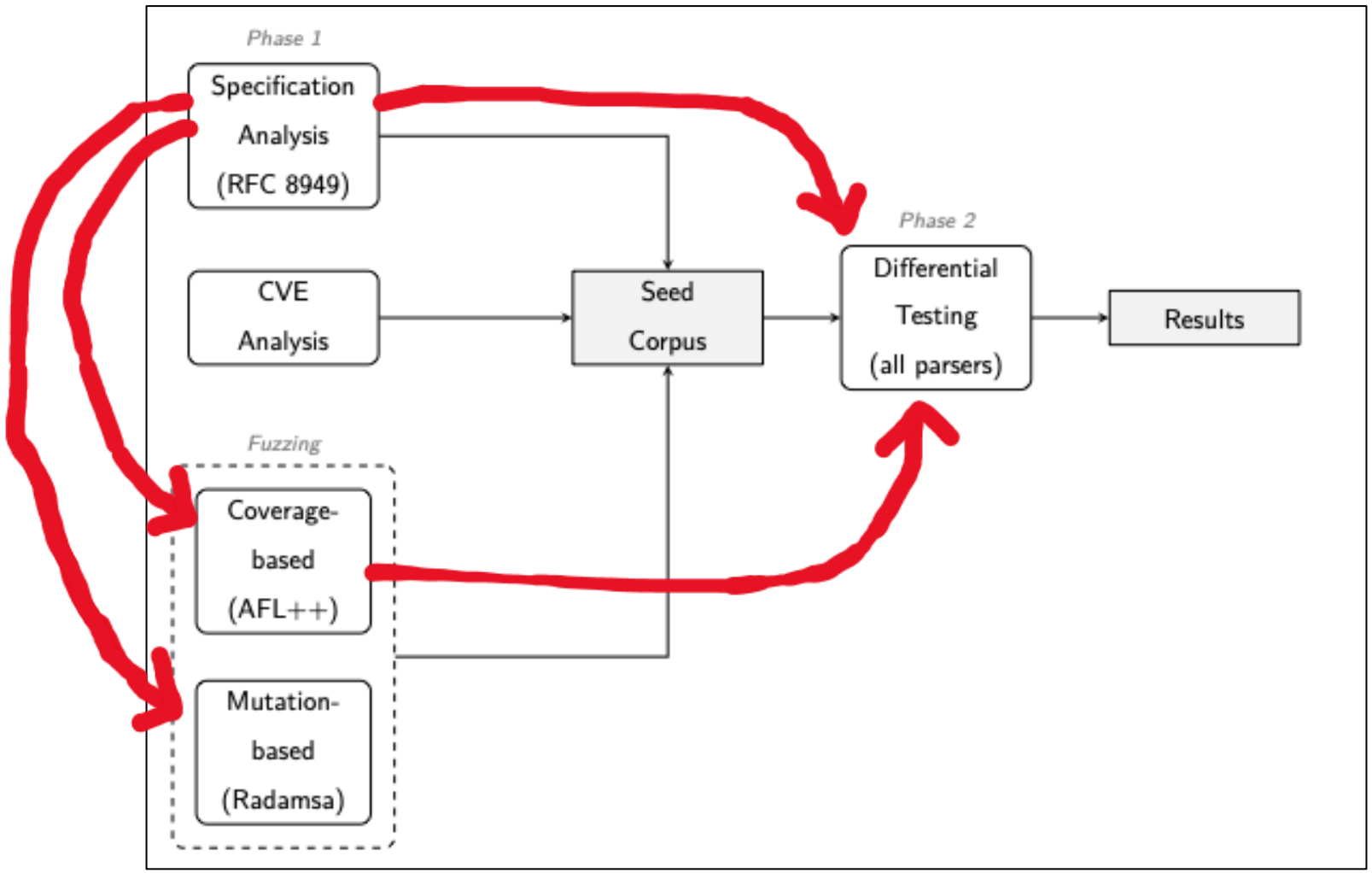
New parser, more & structured payloads

- One parser is EoL and recommends switch → No problem, my approach is extensible
- Restructure and recreate the manually created dataset: Valid/invalid is not enough; there is also malformed data → 68 Vectors
- More payloads (spoiler: >200 from afl++): Random permutations with radamsa
- Input for radamsa should be diverse, and using the observed crashes as input might find more crashes!

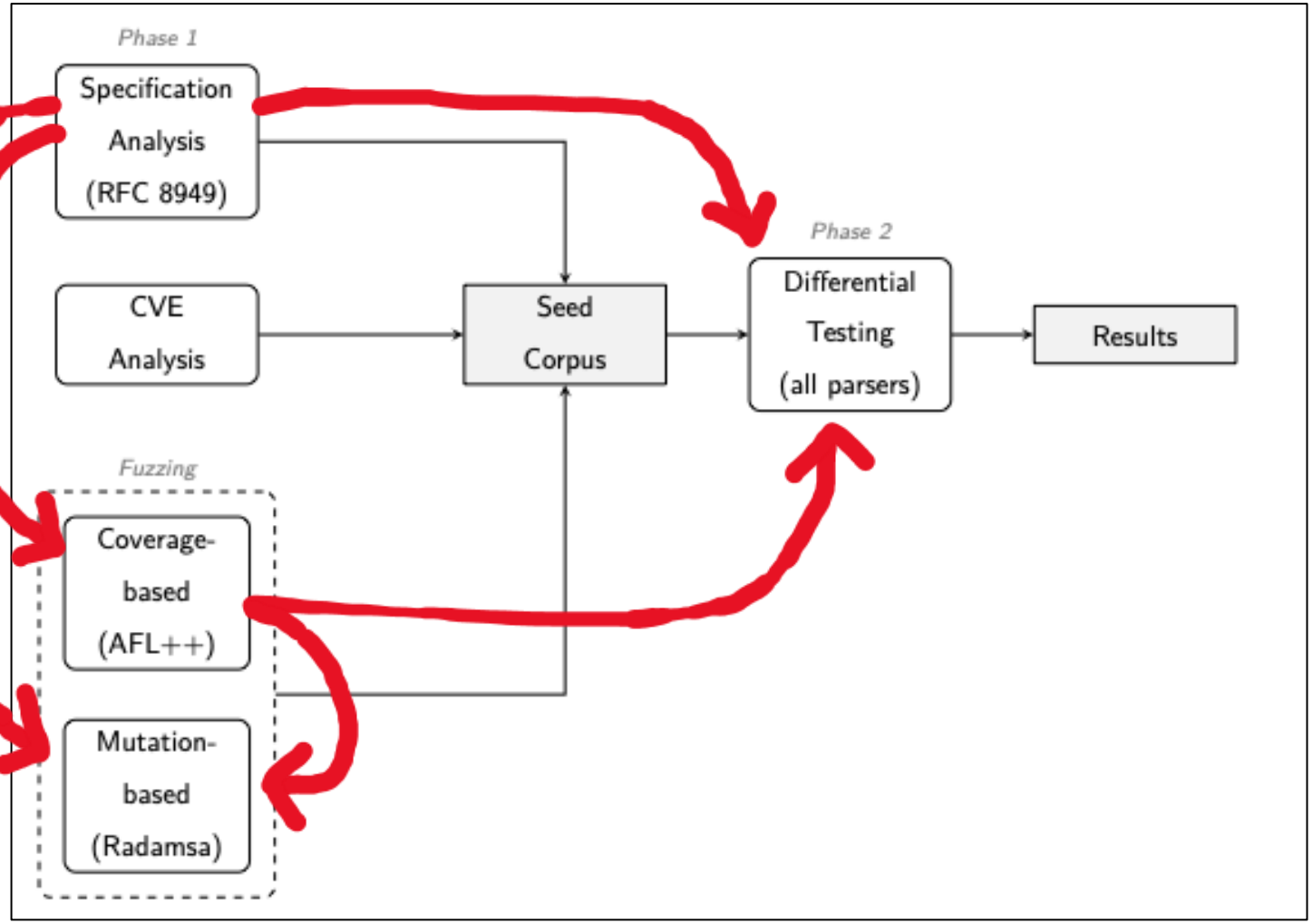
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)

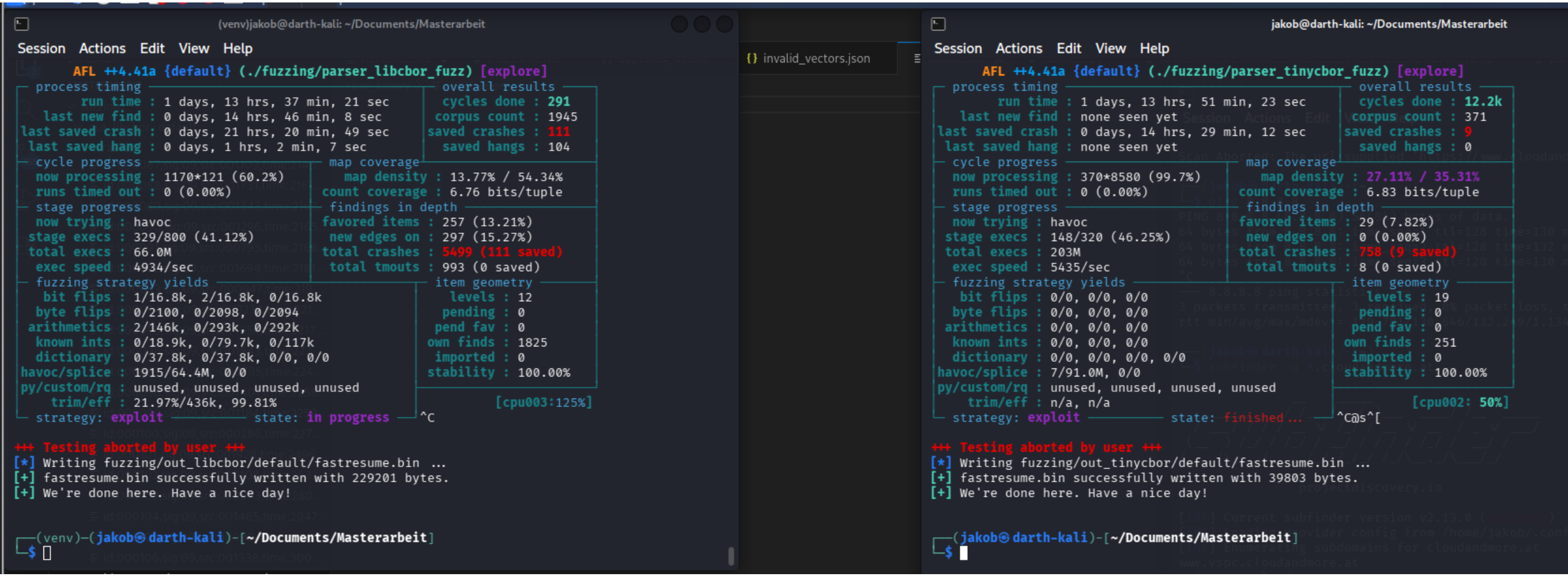
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)

Analysis

- Talk was accepted, so the clock is ticking: Responsible disclosure for the crashes found with libcbor and tinycbor
- So lets dig deeper and earn some CVEs!

Analysis - Fuzzing seems promising...



Analysis - ...until further inspection

libcbor

[Security] Uncontrolled memory allocation in cbor_load() allows DoS via small crafted input #418

New issue

Closed



Benjamin608608 opened on Apr 1

Summary

Uncontrolled memory allocation in `cbor_load()` allows a small crafted CBOR input (as few as 5 bytes) to trigger allocation of gigabytes of memory, causing denial of service.

Assignees

No one assigned

Labels

invalid

Projects



PJK on Apr 4 Owner

This is WAI. The intended mechanism to limit memory allocation is using a custom allocator via cbor_set_allocs, which is more general and also handles e.g. memory bloat via large string headers.

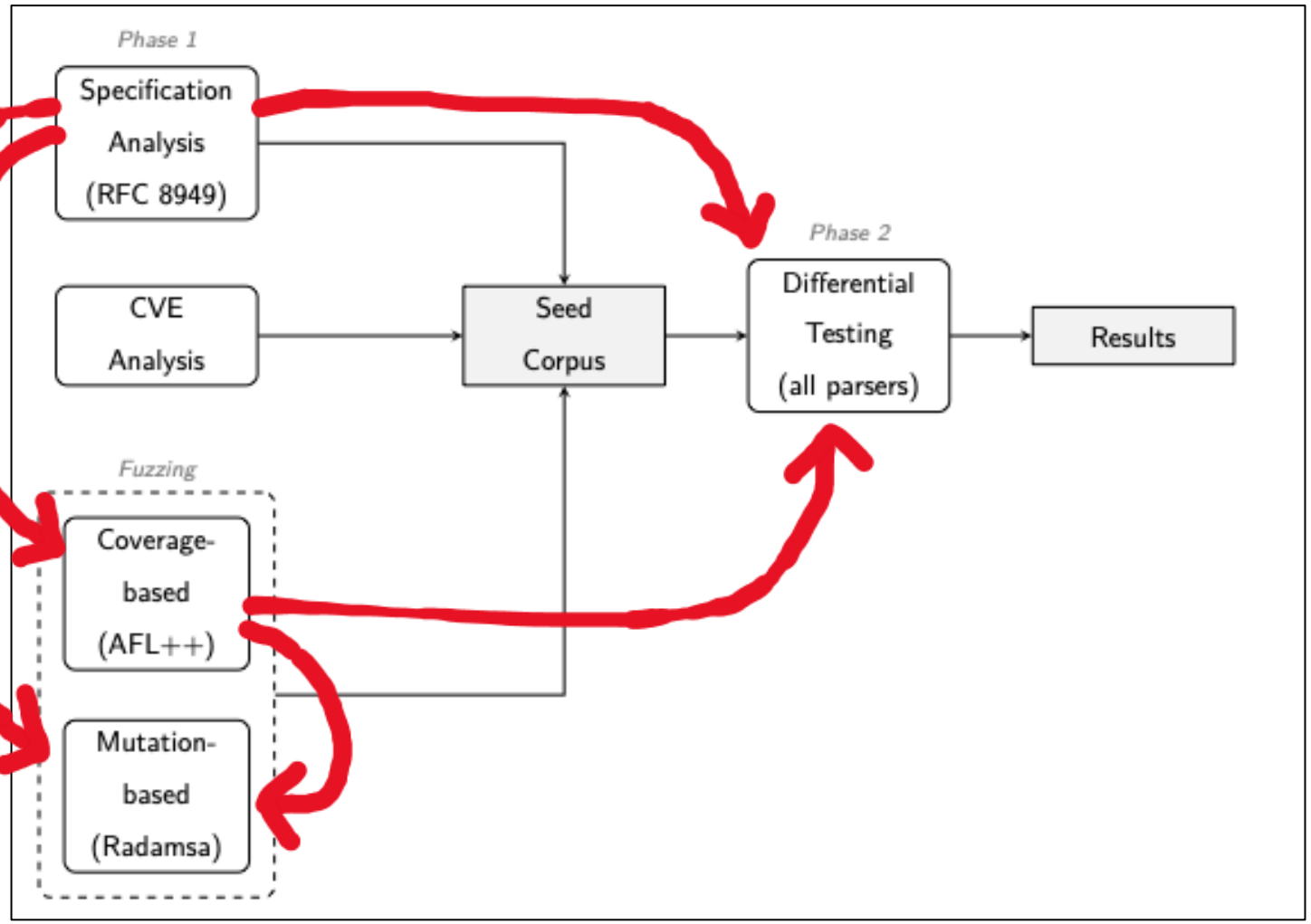
Tinycbor

```c
static CborError consume(CborValue *it) {
    CborError err;
    while (!cbor_value_at_end(it)) {
        if (cbor_value_is_container(it)) {
            CborValue inner;
            err = cbor_value_enter_container(it, &inner);
            if (err) return err;
            err = consume(&inner);
            if (err) return err;
            err = cbor_value_leave_container(it, &inner);
            if (err) return err;
        } else {
            err = cbor_value_advance(it);
            if (err) return err;
        }
    }
    return CborNoError;
}
```

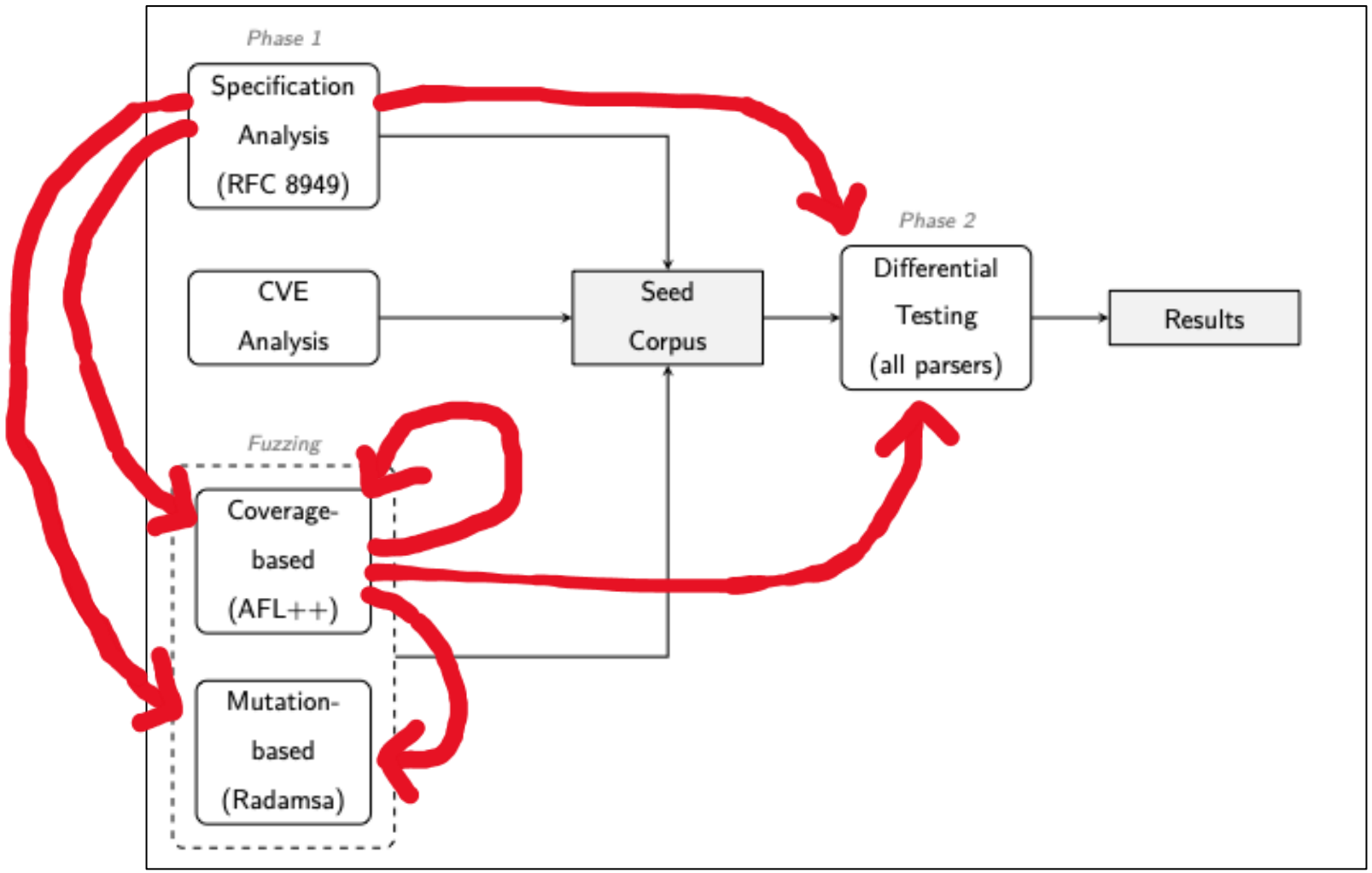
Analysis - What's left?

- Talk was accepted, so the clock is ticking: Responsible disclosure for the crashes found with libcbor and tinycbor
- So lets dig deeper and earn some CVEs!
- …
- Redo the fuzzing run, using the (real) crashes/hangs of libcbor

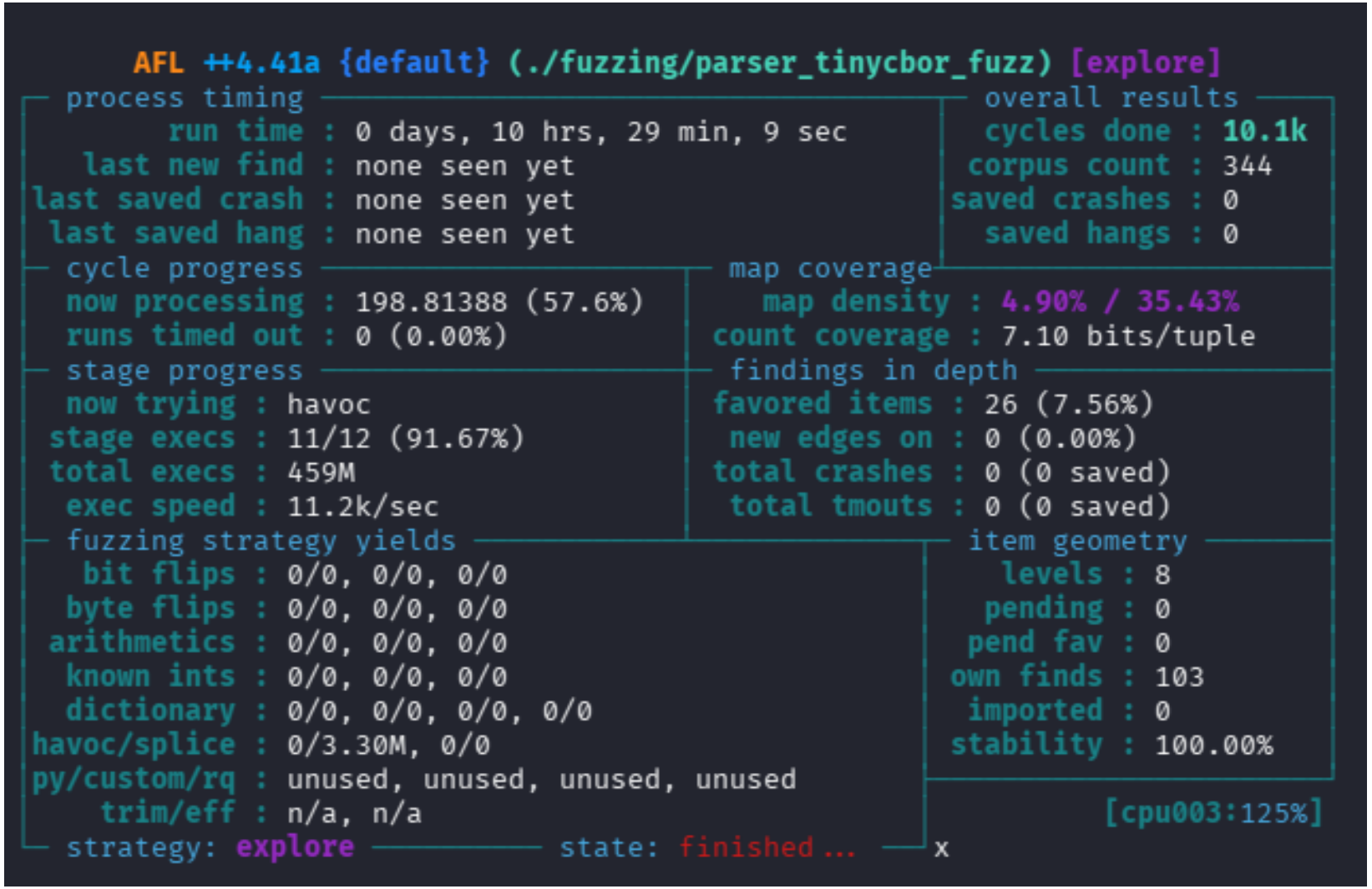
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Fuzzing
Coverage-based (AFL++)
Mutation-based (Radamsa)

... Damn


AFL ++4.41a {default} (./fuzzing/parser_tinycbor_fuzz) [explore]
process timing
run time : 0 days, 10 hrs, 29 min, 9 sec
last new find : none seen yet
last saved crash : none seen yet
last saved hang : none seen yet
overall results
cycles done : 10.1k
corpus count : 344
saved crashes : 0
saved hangs : 0
cycle progress
now processing : 198.81388 (57.6%)
runs timed out : 0 (0.00%)
map coverage
map density : 4.90% / 35.43%
count coverage : 7.10 bits/tuple
stage progress
now trying : havoc
stage execs : 11/12 (91.67%)
total execs : 459M
exec speed : 11.2k/sec
findings in depth
favored items : 26 (7.56%)
new edges on : 0 (0.00%)
total crashes : 0 (0 saved)
total tmouts : 0 (0 saved)
fuzzing strategy yields
bit flips : 0/0, 0/0, 0/0
byte flips : 0/0, 0/0, 0/0
arithmetics : 0/0, 0/0, 0/0
known ints : 0/0, 0/0, 0/0
dictionary : 0/0, 0/0, 0/0, 0/0
havoc/splice : 0/3.30M, 0/0
py/custom/rq : unused, unused, unused, unused
trim/eff : n/a, n/a
item geometry
levels : 8
pending : 0
pend fav : 0
own finds : 103
imported : 0
stability : 100.00%
[cpu003:125%]
strategy: explore
state: finished ...
x

Analysis - What's left?

- Talk was accepted, so the clock is ticking: Responsible disclosure for the crashes found with libcbor and tinycbor
- So lets dig deeper and earn some CVEs!
- …
- Redo the fuzzing run, using the (real) crashes/hangs of libcbor
- Let's redo the payloads again and try to find something worth presenting

Specification Analysis – final_final_2

- [x] Scanning the whole RFC for (encoding / decoding) ✅ 2026-06-21
  - [x] MUST / MUST NOT ✅ 2026-06-21
  - [x] valid ✅ 2026-06-21
  - [x] invalid ✅ 2026-06-21
  - [x] validity ✅ 2026-06-21
  - [x] allow ✅ 2026-06-21
  - [x] example ✅ 2026-06-21
  - [x] well-formed ✅ 2026-06-21
  - [x] malformed ✅ 2026-06-21
- [x] Appendix F ✅ 2026-06-21
- [x] F.1 "Examples of CBOR Data Items That Are Not Well-Formed" ✅ 2026-06-21
- [x] Appendix C --> Not important ✅ 2026-06-21
- [x] `10. Security Considerations` ✅ 2026-06-21
- [x] 6.2 ✅ 2026-06-21
- [x] Scan the doc I already constructed ✅ 2026-06-21
- [x] 5.3.1. Basic validity ✅ 2026-06-21
- [x] 5.3.2. Tag validity ✅ 2026-06-21
- [x] https://github.com/cbor/test-vectors/blob/master/appendix_a.json --> f818 ist malformed (reserved two-byte simple value, F.1 subkind 2) ✅ 2026-06-21

Specification Analysis

- Invalid (edge-cases, specific mentions in RFC)
- Malformed (Mostly Appendix F.1 – great source, lots of examples)
- Valid (edge cases, reference values)


```
{
  "hex": "c000",
  "description": "Inadmissible type for tag content: tag 0 (datetime)
  content MUST be a text string",
  "class": "invalid",
  "category": "§5.3. Validity of Items",
  "subcategory": "$5.3.2. Tag validity",
  "source": "rfc_generated"
},
{
  "hex": "c069796573746572646179",
  "description": "Inadmissible value for tag content: 'yesterday' is
  not valid datetime",
  "class": "invalid",
  "category": "§5.3. Validity of Items",
  "subcategory": "$5.3.2. Tag validity",
  "source": "rfc_verbatim"
},
```


```
(venv)-(jakob㉿darth-kali)-[~/Documents/Masterarbeit]
$ jq '[.[] | select(.class == "invalid")] | length' Testcases/probe_vectors.json
9

(venv)-(jakob㉿darth-kali)-[~/Documents/Masterarbeit]
$ jq '[.[] | select(.class == "valid")] | length' Testcases/probe_vectors.json
96

(venv)-(jakob㉿darth-kali)-[~/Documents/Masterarbeit]
$ jq '[.[] | select(.class == "malformed")] | length' Testcases/probe_vectors.json
94

(venv)-(jakob㉿darth-kali)-[~/Documents/Masterarbeit]
$ date
Fri 02 Oct 2026 14:00:57 IST
```

Analysis - Parsers are acting weird

- There are clear patterns in the outcomes table

index	hex	cbor2	libcbor	tinycbor	jackson	upokecenter	fxamacker	ciborium	petero	node-cbor	cborg	divergence
0	f09080f2a0ca	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
1	f389a081b7bf	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
2	d82076687474	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
3	e819	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
4	f3f3a081ada0	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
5	f9f3e1be82a0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
6	f8f3a080a6fe	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
7	f3a080e0b980	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
8	e0f3a08080a3	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
9	fb7e37e43c88	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
10	f9f3a081aafc	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
11	e28089a08120	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
12	612b2f763332	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
13	f280f15ff35f	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
14	818ca8a0f6	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
15	efbea0f6	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
16	e281a964cab8	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
17	d8207668efbe	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
18	9f018202039f	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
19	9819010203f3	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
20	f3a0e2808880	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
21	9f0102030405	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
22	81bef3e28086	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
23	fb80a7e43c88	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	
24	f9c401	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	

index	hex	cbor2	libcbor	tinycbor	jackson	upokecenter	fxamacker	ciborium	petero	node-cbor	cborg	divergence
0	f09080f2a0ca	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
1	f389a081b7bf	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
2	d82076687474	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
3	e819	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
4	f3f3a081ada0	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
5	f9f3e1be82a0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	yes
6	f8f3a080a6fe	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
7	f3a080e0b980	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
8	e0f3a08080a3	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
9	fb7e37e43c88	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
10	f9f3a081aafc	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
11	e28089a08120	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
12	612b2f763332	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	rejected (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
13	f280f15ff35f	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
14	818ca8a0f6	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
15	efbea0f6	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
16	e281a964cab8	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
17	d8207668efbe	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
18	9f018202039f	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
19	9819010203f3	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
20	f3a0e2808880	accepted (0.0	rejected (0.0	accepted (0.0	accepted (0.0	accepted (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	yes
21	9f0102030405	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
22	81bef3e28086	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.1	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	rejected (0.0	
23	fb80a7e43c88	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	
24	f9c401	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.1	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	accepted (0.0	

Analysis - Parsers are acting weird

- There are clear patterns in the outcomes table
- Oh no
- Is there something wrong with my parsers? Did I introduce this behavior like with tinycbor?

Analysis - Parsers are acting weird

- There are clear patterns in the outcomes table
- Oh no
- Is there something wrong with my parsers? Did I introduce this behavior like with tinycbor?

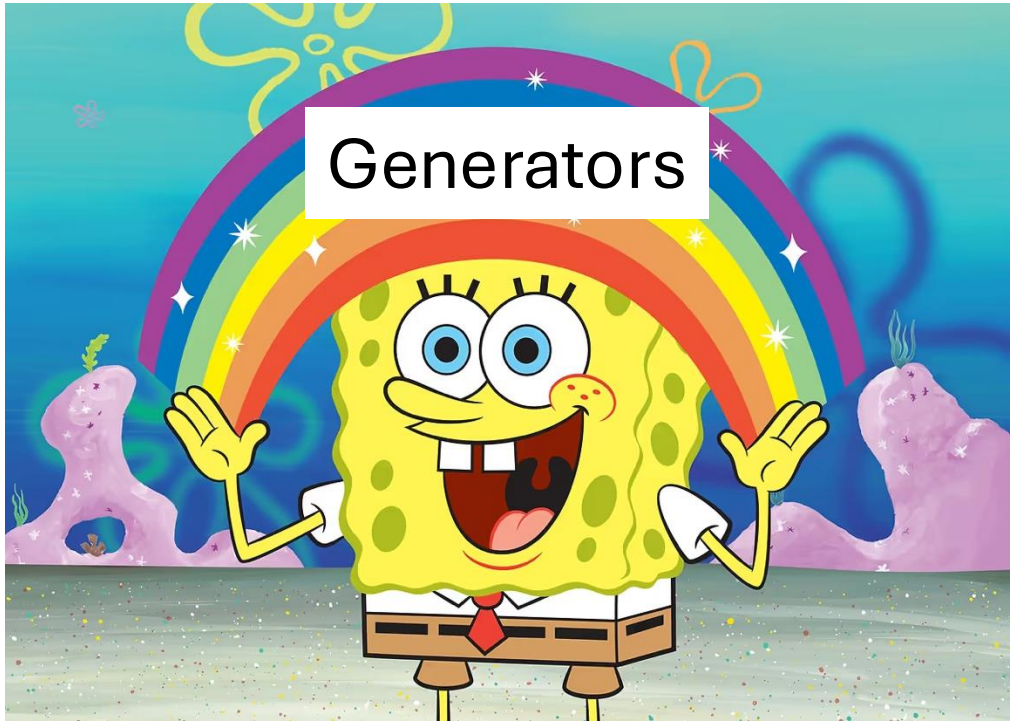
Parser	Implemented: Stream / Item	Supports: Stream / Item	reaction on trailing data	Data Model: Basic / Extended	Strictness / Validation
cbor2 (js)	single-item	single-item, stream	error	Extended	-
ciborium	single-item	single-item	error	Extended	-
fxamacker/cbor	single-item	single-item, stream	error	Extended	built-in (?)
tinycbor	single-item	single-item, stream	ignore	Extended	cbor_value_validate()
libcbor	single-item	single-item, stream	ignore	Extended	cbor_set_allocs(), cbor_stream_decode()
cbor2 (python)	single-item	single-item, stream	ignore	Extended	allow_duplicate_keys, allow_indefinite, max_depth
cborg	single-item	single-item, stream	ignore	only with "cborg/extended"	- ("focused on strictness")
node-cbor	single-item	single-item, stream	ignore	Extended	max_depth, preventDuplicateKeys
PeterO.Cbor	single-item	single-item, stream	ignore	Extended	?
com.upokecenter.cbor	single-item	single-item, stream	ignore	Extended	strict by default (?)
jackson-dataformat-cbor	single-item	single-item, stream	ignore	Extended	strict by default (?)

Let’s push it

Problem

- Manual creation of payloads pushing the limits is tedious / impossible
- Generated Payloads measure in the KB/MB

Solution



```python
# Originals

def nested_array_definite(n):
    return (b"\x81" * n + b"\x00")

def nested_array_indefinite(n):
    return (b"\x9f" * n + b"\x00" + b"\xff" * n)

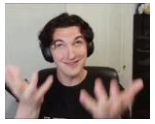
def nested_map_definite(n):
    return (b"\xa1\x01" * n + b"\x00")

def nested_map_indefinite(n):
    return (b"\xbf\x01" * n + b"\x00" + b"\xff" * n)
```

```python
for i, vec in enumerate(vectors):
    if "hex" in vec:
        payload = bytes.fromhex(vec["hex"])
    else:
        name, n = vec["generator"]
        payload = getattr(generators, name)(n)

    path = str(bindir / payload_filename(vec, payload))
    if not Path(path).exists():
        Path(path).write_bytes(payload)
```

```
    "source": "rfc_verbatim"
  },
  {
    "generator": ["nested_map_indefinite", 10000000],
    "description": "10.000.000 nested maps with indefinite length",
    "class": "valid",
    "category": "§10 Security Considerations",
    "subcategory": "Resource exhaustion",
    "source": "rfc_generated"
  },
  {
    "generator": ["nested_map_definite", 10000000],
    "description": "10.000.000 nested maps with fixed length",
    "class": "valid",
    "category": "§10 Security Considerations",
    "subcategory": "Resource exhaustion",
    "source": "rfc_generated"
  },
  {
    "generator": ["nested_array_indefinite", 10000000],
    "description": "10.000.000 nested arrays with indefinite length",
    "class": "valid",
    "category": "§10 Security Considerations",
    "subcategory": "Resource exhaustion",
    "source": "rfc_generated"
  },
  {
    "generator": ["nested_array_definite", 10000000],
    "description": "10.000.000 nested arrays with fixed length",
    "class": "valid",
    "category": "§10 Security Considerations",
    "subcategory": "Resource exhaustion",
    "source": "rfc_generated"
  },
```

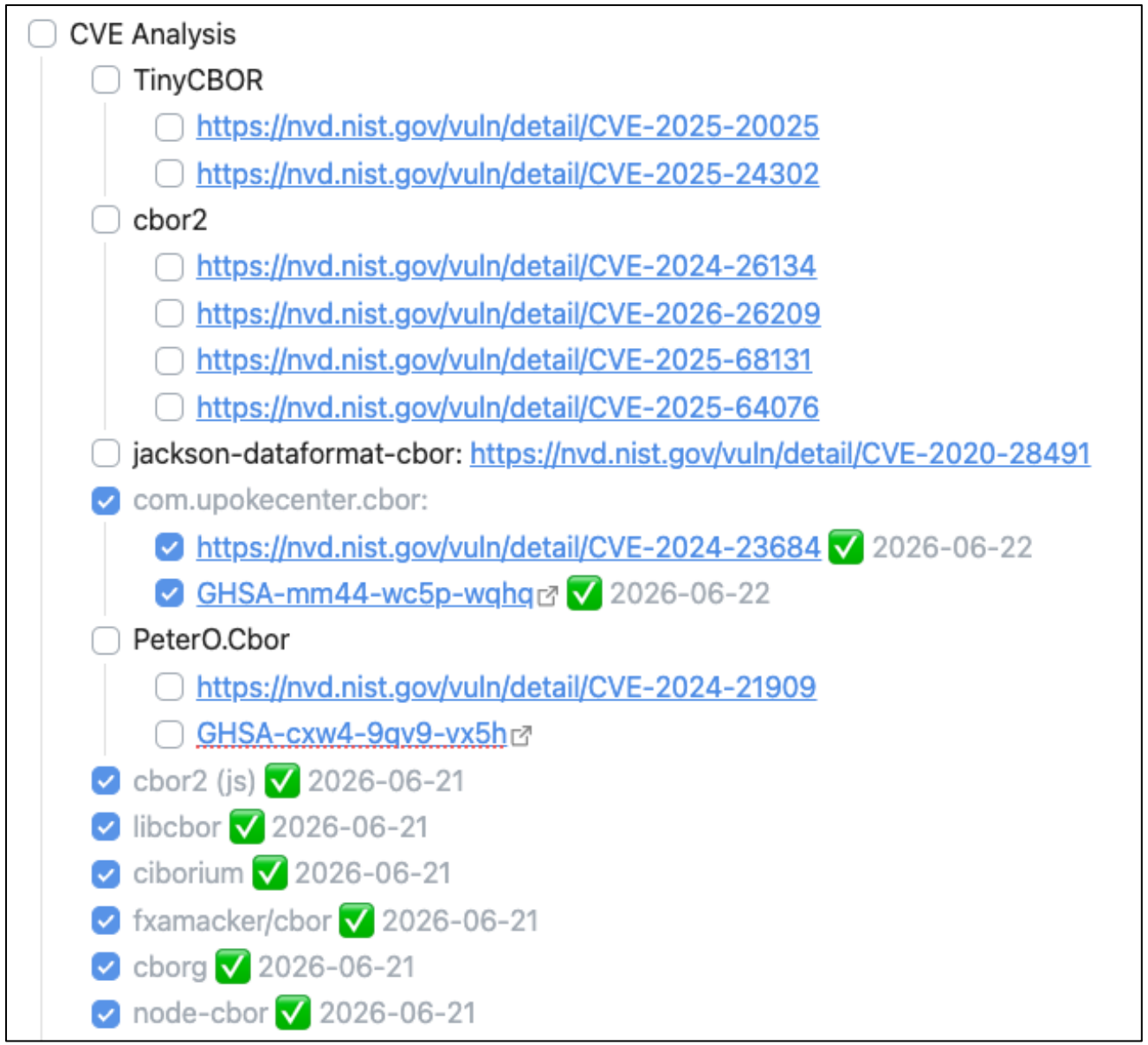



Analysis - What’s left?

- Talk was accepted, so the clock is ticking: Responsible disclosure for the crashes found with libcbor and tinycbor
- So lets dig deeper and earn some CVEs!
- …
- Redo the fuzzing run, using the (real) crashes/hangs of libcbor
- Lets’s redo the payloads again and try to find something worth presenting
- CVE analysis!

CVE Analysis – Overview

- 11 CVEs, 9 relevant (2 dups)
- Most (7/9) are DOS
  - Buffer overflow
  - Uncontrolled recursion
  - Inefficient Sorting
- Notable Exceptions:
  - Information Disclosure
  - Privilege Escalation



CVE Analysis – jackson-dataformat-cbor (CVE-2020-28491)

```
66  +     // 2.11.4: [dataformats-binary#186] Avoid OOME/DoS for bigger binary;
67  +     //  read only up to 250k
68  +     protected final static int LONGEST_NON_CHUNKED_BINARY = 250_000;
69  +
```

```
2437  -             byte[] b = new byte[len];
2438  -             if (_inputPtr >= _inputEnd) {
2439  -                 loadMoreGuaranteed();
2448  +             return _finishChunkedBytes();
2449  +         }
2450  +         // Non-chunked, contiguous
2451  +         if (len > LONGEST_NON_CHUNKED_BINARY) {
2452  +             // [dataformats-binary#186]: avoid immediate allocation for longest
2453  +             return _finishLongContiguousBytes(len);
2454  +         }
2455  +
2456  +         final byte[] b = new byte[len];
2457  +         final int expLen = len;
2458  +         if (_inputPtr >= _inputEnd) {
2459  +             if (!loadMore()) {
2460  +                 _reportIncompleteBinaryRead(expLen, 0);
```

CVE Analysis – jackson-dataformat-cbor (CVE-2020-28491)

- For chunked data, memory is eagerly allocated
- E.g.:
  0x5A7FFFFFFF -->
  010 11010 0111_1111 1111_1111 1111_1111 1111_1111
  2 26 2147483647
  → new byte[2147483647]
- This is interesting, but already included in my dataset

CVE Analysis – tinycbor (CVE-2025-20025, CVE-2025-24302)

- Both caused by recursion, both fixed in 0.6.1
- Fixed by introducing nesting limit
- Also interesting, but nesting was one of the first things to look for

```
┌──(venv)─(jakob㉿darth-kali)-[~/Documents/Masterarbeit/tinycbor]
└─$ git log --oneline v0.6.0..v0.6.1 | grep recursion
2455693 fix(pretty): advance to the end of the iterator if recursion limit was hit

┌──(venv)─(jakob㉿darth-kali)-[~/Documents/Masterarbeit/tinycbor]
└─$ git show e692445
commit e6924451a9d4cc45693a17b2b0cca64bedbc8434
Author: Thiago Macieira <thiago.macieira@intel.com>
Date:   Tue Feb 4 16:40:45 2025 -0800

    CBOR-to-JSON: Limit how deep we process nested containers

    1024 levels will probably be good enough for everyone, like
    cborparser.c. For those for whom it isn't, they can set the limit during
    the build.

    We already had this for the plain parser, so TinyCBOR wouldn't cause a
    stack overflow in case of a malformed stream (intentionally or not) when
    simply parsing and advancing over the stream. This same protection
    wasn't applied to the content converting from CBOR to JSON.

    Signed-off-by: Thiago Macieira <thiago.macieira@intel.com>
```

CVE Analysis – PeterO.Cbor (CVE-2024-21909)

Description

Impact

Due to this library's use of an inefficient algorithm, it is vulnerable to a denial of service attack when a maliciously crafted input is passed to `DecodeFromBytes` or other CBOR decoding mechanisms in this library.

Affected versions *include* versions 4.0.0 through 4.5.0.

This vulnerability was privately reported to me.

Patches

This issue has been fixed in version 4.5.1. Users should use the latest version of this library. (The latest version is not necessarily 4.5.1. Check the NuGet page to see the latest version's version number.)

Workarounds

Again, users should use the latest version of this library.

In the meantime, note that the inputs affected by this issue are all CBOR maps or contain CBOR maps. An input that decodes to a single CBOR object is not capable of containing a CBOR map if—

- it begins with a byte other than 0x80 through 0xDF, or
- it does not contain a byte in the range 0xa0 through 0xBF.

Such an input is not affected by this vulnerability and an application can choose to perform this check before passing it to a CBOR decoding mechanism.

CVE Analysis – PeterO.Cbor (CVE-2024-21909)

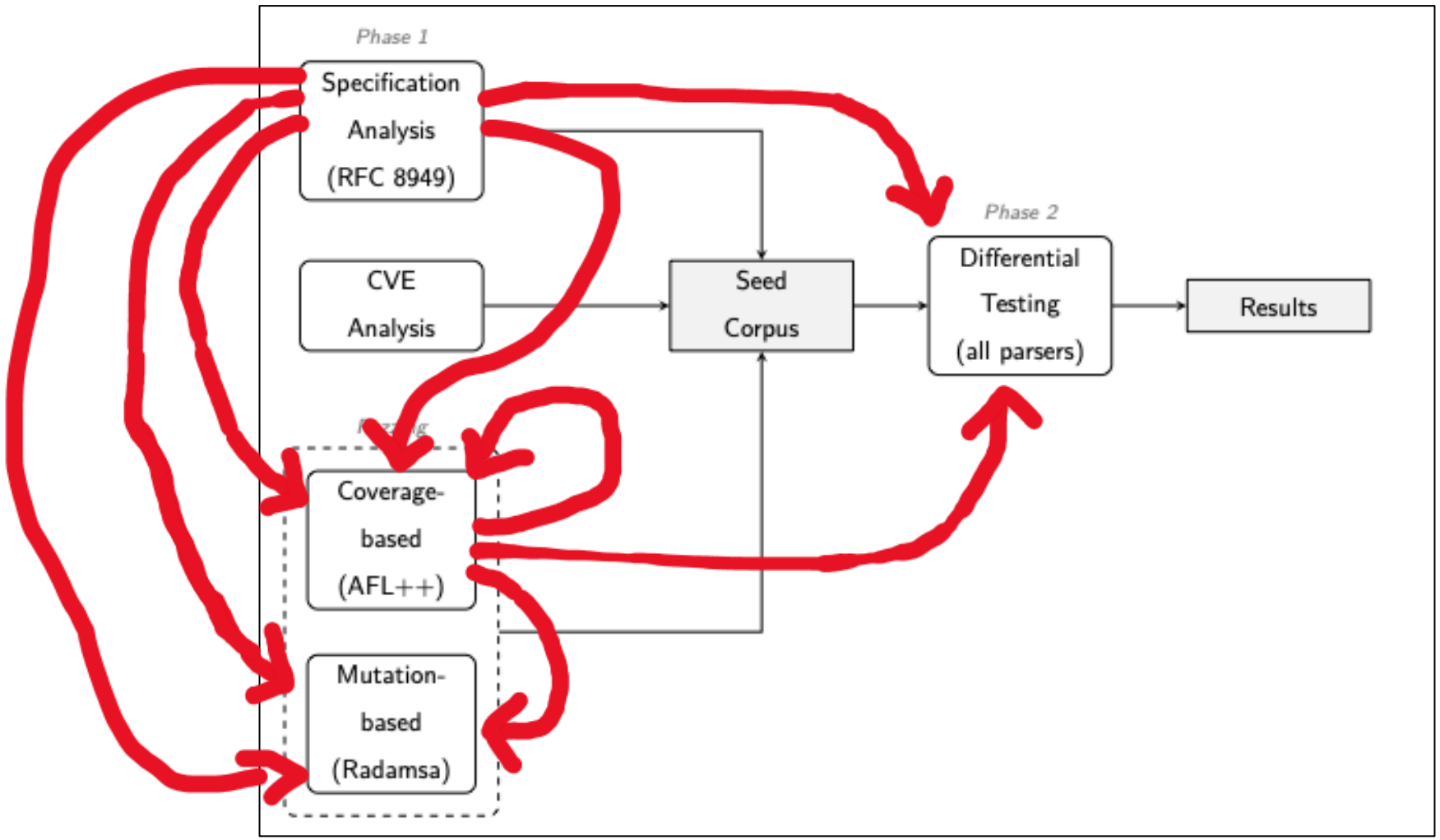
- Issue with the sorting of map: Sort() on a List runs in O(n log n)
- POC: very long maps, e.g. >200.000
- Interesting, probably won't occur too often. Still added to my collection

```
                16 + Version 4.5.1
                17 +
                18 + - Fix reported security issue
                19 +
16              20   Version 4.5:
17              21
18              22   - Add support for JSON Pointers and JSON Patches
```

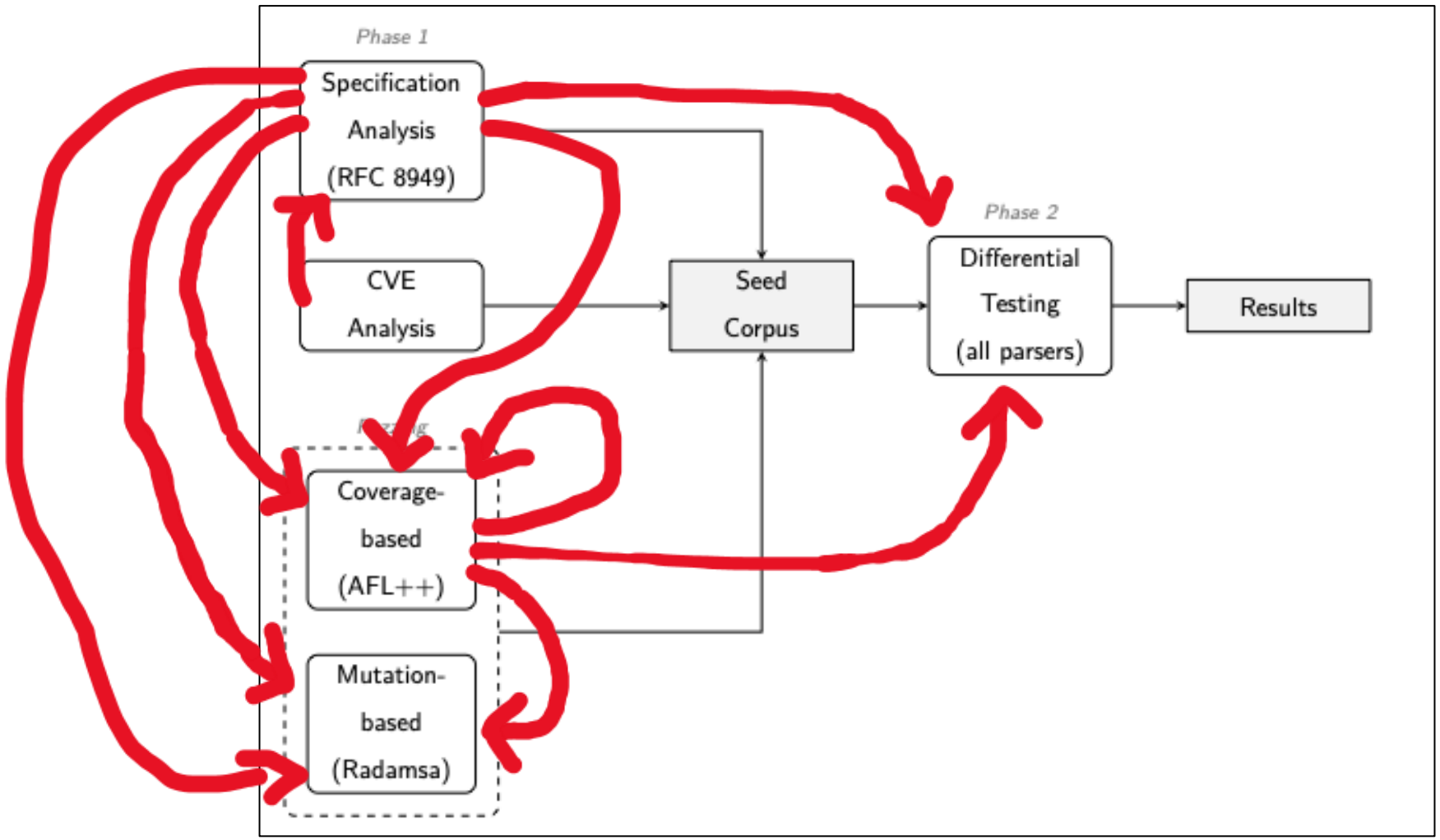
4 CBOR/PeterO/Cbor/CBORObject.cs

```
                   @@ -7945,9 +7945,9 @@ private static int MapCompare(
7945  7945              var sortedASet = new List<CBORObject>(mapA.Keys);
7946  7946              var sortedBSet = new List<CBORObject>(mapB.Keys);
7947  7947              // DebugUtility.Log("---sorting mapA's keys");
7948        -           sortedASet.Sort();
      7948  +           // sortedASet.Sort();
7949  7949              // DebugUtility.Log("---sorting mapB's keys");
7950        -           sortedBSet.Sort();
      7950  +           // sortedBSet.Sort();
7951  7951              // DebugUtility.Log("---done sorting");
7952  7952              listACount = sortedASet.Count;
7953  7953              listBCount = sortedBSet.Count;
```

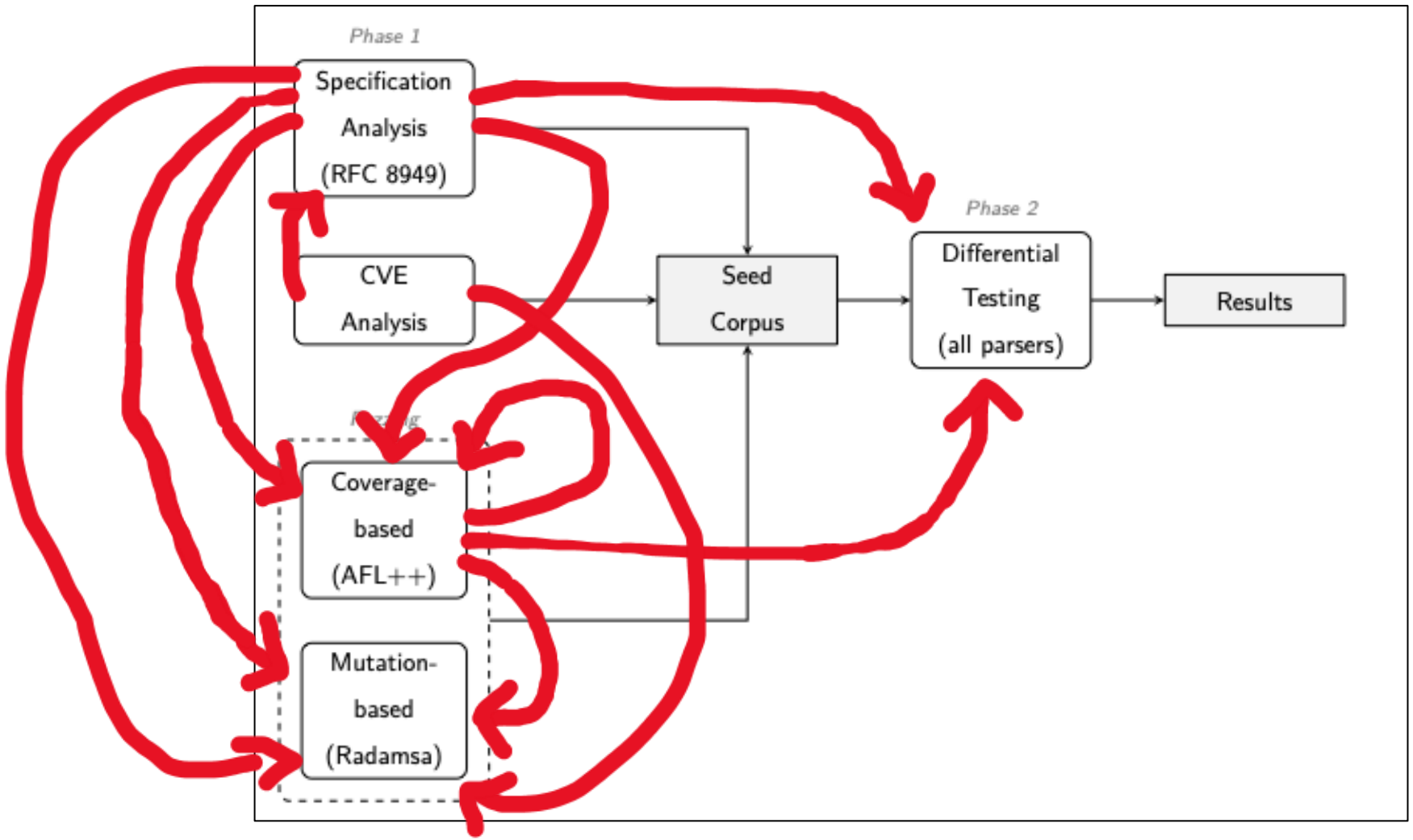
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Coverage-based (AFL++)
Mutation-based (Radamsa)

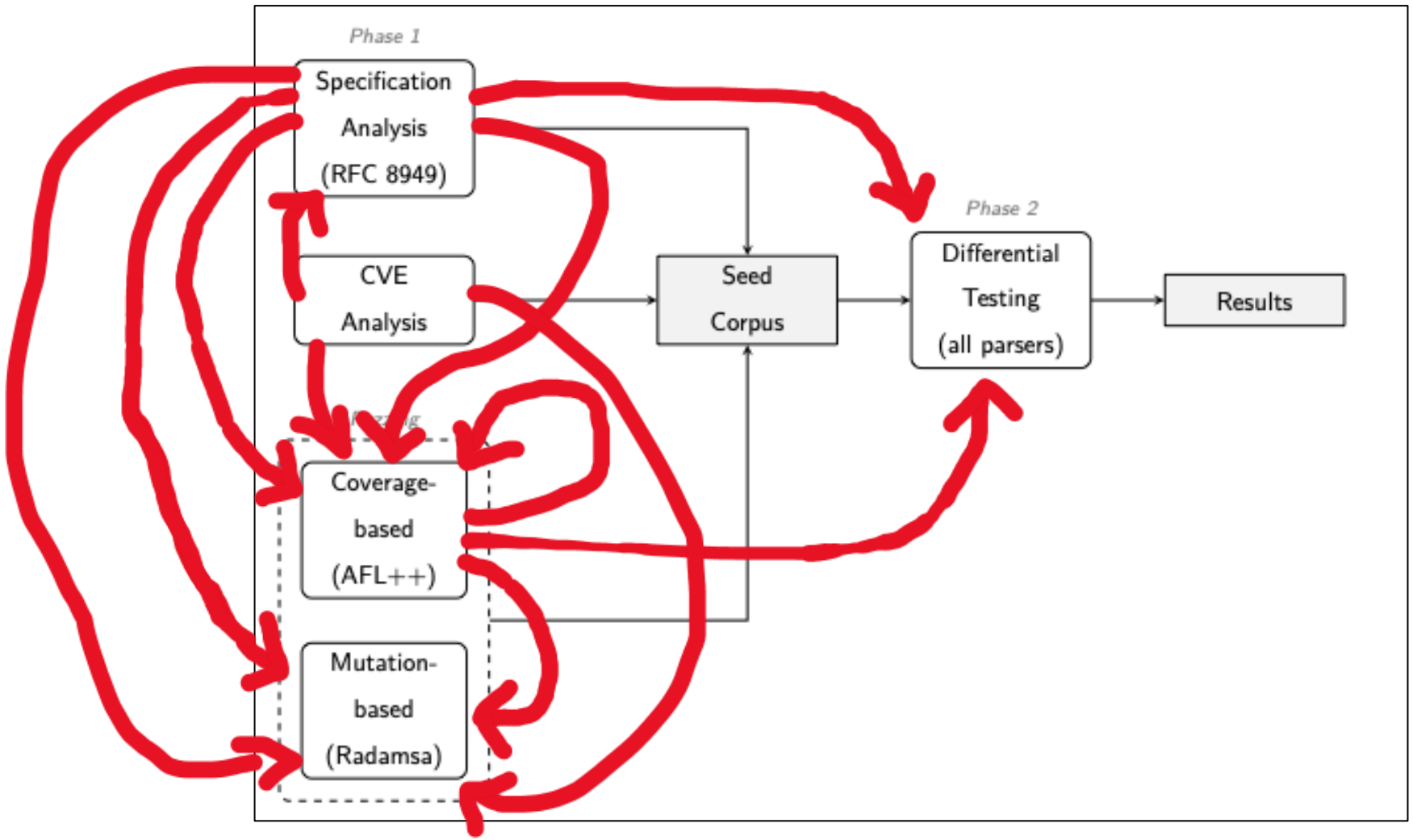
Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Coverage-based (AFL++)
Mutation-based (Radamsa)

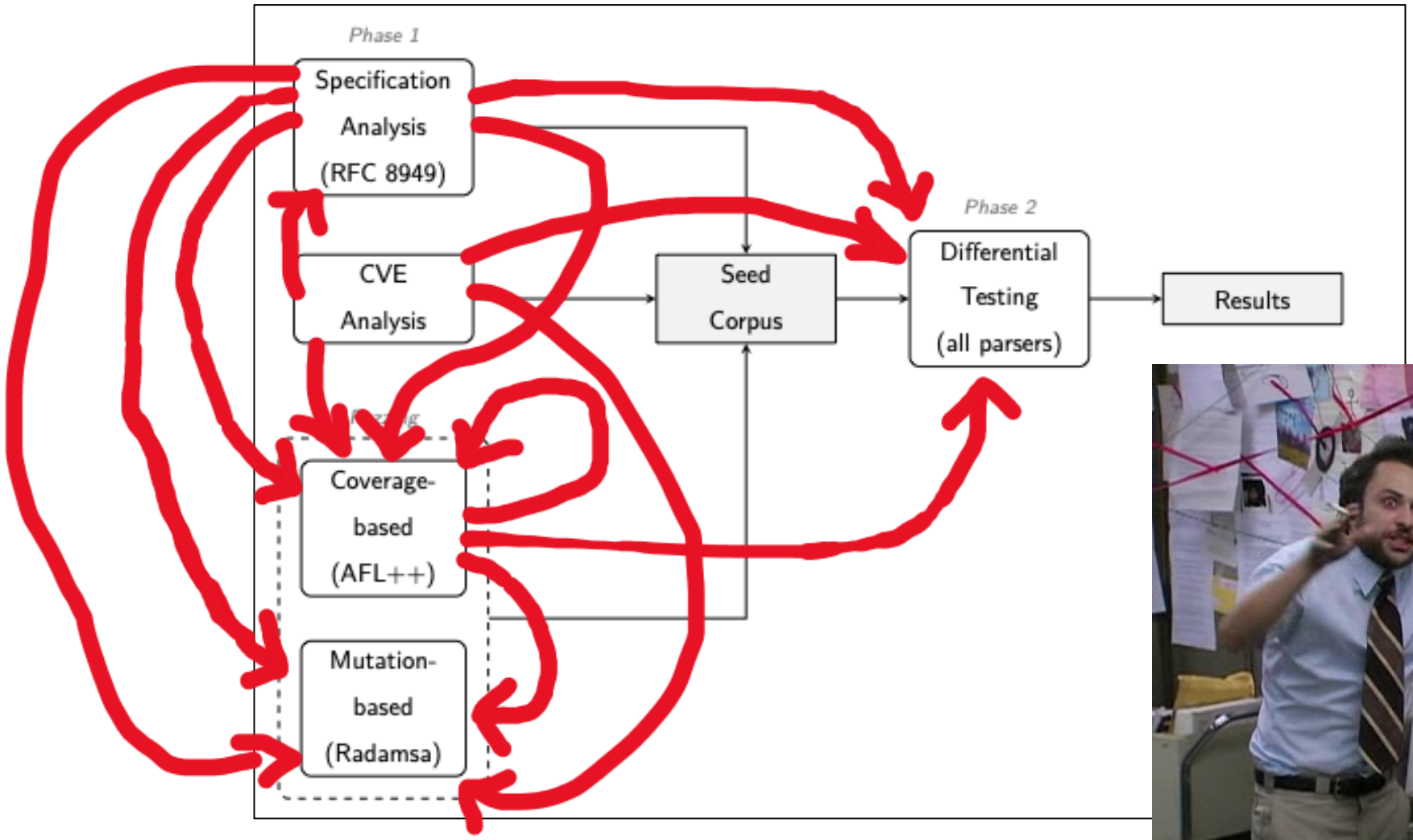
Workflow

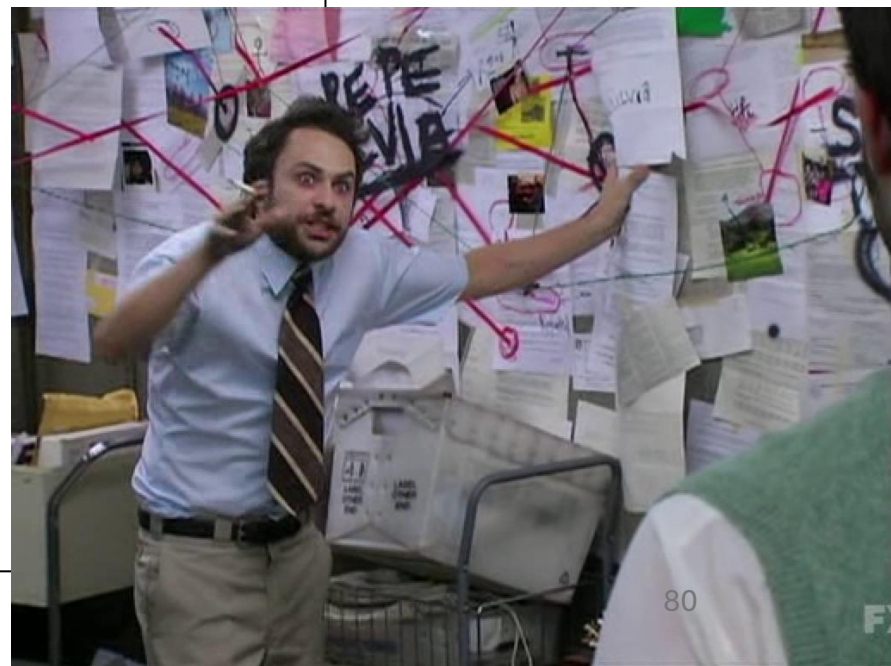

Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Coverage-based (AFL++)
Mutation-based (Radamsa)

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Coverage-based (AFL++)
Mutation-based (Radamsa)

Workflow


Phase 1
Specification Analysis (RFC 8949)
CVE Analysis
Seed Corpus
Phase 2
Differential Testing (all parsers)
Results
Coverage-based (AFL++)
Mutation-based (Radamsa)



Results!

Hundreds of CVEs, Fame & Fortune?

Results – Parser comparison

- Honestly: this is still very much ongoing
- There is so much to look at

Results – Parser comparison

- Honestly: this is still very much ongoing
- There is so much to look at
- Some examples

index	diagnostic / hex	class	tag	cbor2-py	libcbor	tinycbor	jackson	upokecenter	fxamacker	ciborium	petero	node-cbor	cborg	cbor2-js	interesting
8	fffe92	-	radamsa	accepted (0.01s)	rejected (0.00s)	rejected (0.00s)	rejected (0.09s)	rejected (0.02s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.05s)	rejected (0.06s)	yes
40	cab7cd8f96a081	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.09s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.06s)	rejected (0.05s)	rejected (0.06s)	yes
43	c097	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.10s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.05s)	rejected (0.06s)	yes
49	cab6c0	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.09s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.05s)	rejected (0.06s)	yes
56	c25a00002710ffffffffffffffffffffffffffffffff	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.09s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	yes
62	cb905fbbff	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.09s)	rejected (0.02s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	yes
65	d820766874e2808e78616d706c652e636f6d	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.09s)	rejected (0.02s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	yes
91	fffef3a0819b02629f0203daffff	-	radamsa	accepted (0.01s)	rejected (0.00s)	rejected (0.00s)	rejected (0.10s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	yes
95	cab4f3a081e281a5acf3a080baa200ff	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.10s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.05s)	rejected (0.05s)	rejected (0.06s)	yes
105	d8ffff68	-	radamsa	accepted (0.01s)	rejected (0.00s)	rejected (0.00s)	rejected (0.09s)	rejected (0.02s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.06s)	rejected (0.05s)	rejected (0.06s)	yes
180	c25a00002710ffe2ffffffffffffffffffffffffffff	-	radamsa	rejected (0.01s)	rejected (0.00s)	accepted (0.00s)	rejected (0.09s)	rejected (0.02s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	yes
30	a2000000	malformed		rejected (0.01s)	rejected (0.00s)	rejected (0.00s)	rejected (0.11s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	
31	c0	malformed		rejected (0.01s)	rejected (0.00s)	rejected (0.00s)	accepted (0.09s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.06s)	rejected (0.06s)	yes
32	5f4100	malformed		rejected (0.01s)	rejected (0.00s)	rejected (0.01s)	rejected (0.09s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.03s)	rejected (0.05s)	rejected (0.07s)	rejected (0.06s)	

Results – Parser comparison

index	diagnostic / hex	class	tag	cbor2-py	libcbor	tinycbor	jackson	upokecenter	fxamacker	ciborium	petero	node-cbor	cborg	cbor2-js	divergence
0	a9a9a9a9a9a99a49f40100	malformed	fuzz_libcbor	rejected (0.01s)	timeout (5.29s)	rejected (0.01s)	rejected (0.26s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.12s)	rejected (0.17s)	rejected (0.06s)	rejected (0.06s)	yes
1	9a5a002e2e2effffffffff2e2e2d2e925a00002e2e8888	malformed	fuzz_libcbor	rejected (0.02s)	timeout (6.55s)	rejected (0.08s)	rejected (0.42s)	rejected (0.06s)	rejected (0.02s)	rejected (0.00s)	rejected (0.22s)	rejected (0.39s)	rejected (0.09s)	rejected (0.07s)	yes
2	92a3829f929a292ef97e009280a39f929a2e2ef9ffffff	malformed	fuzz_libcbor	rejected (0.04s)	timeout (5.24s)	rejected (0.07s)	rejected (0.40s)	rejected (0.04s)	rejected (0.01s)	rejected (0.00s)	rejected (0.11s)	rejected (0.18s)	rejected (0.06s)	rejected (0.06s)	yes
3	bf9fbf9fbfffff800012bf9f9fbf000000000000009b00	malformed	fuzz_libcbor	rejected (0.03s)	rejected (3.10s)	rejected (0.00s)	rejected (0.13s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.07s)	rejected (0.06s)	rejected (0.06s)	
4	9a48ffffff928e9217ffffff928e9217187c640002fdf9	malformed	fuzz_libcbor	rejected (0.01s)	crash (2.94s)	rejected (0.01s)	rejected (0.21s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.08s)	rejected (0.15s)	rejected (0.07s)	rejected (0.07s)	yes
5	a9a9a9a99a5afff92ff	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.18s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
6	b999999a5afffffff9917	malformed	fuzz_libcbor	rejected (0.01s)	crash (3.50s)	rejected (0.01s)	rejected (0.22s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.09s)	rejected (0.16s)	rejected (0.06s)	rejected (0.06s)	yes
7	928e92809235353535353535353535353535353535353535	malformed	fuzz_libcbor	rejected (0.02s)	crash (4.01s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.08s)	rejected (0.06s)	rejected (0.06s)	yes
8	b7b7b7b7282828282828282f28282828282828284343	malformed	fuzz_libcbor	rejected (0.01s)	crash (3.33s)	rejected (0.01s)	rejected (0.18s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.07s)	rejected (0.08s)	rejected (0.06s)	rejected (0.06s)	yes
9	927b000000000000000000000a9a9a9a9a9a99a490100	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.95s)	rejected (0.01s)	rejected (0.18s)	rejected (0.05s)	rejected (0.00s)	rejected (0.00s)	rejected (0.08s)	rejected (0.12s)	rejected (0.06s)	rejected (0.06s)	yes
10	9a3a72363636363636923636392000192000100809a	malformed	fuzz_libcbor	rejected (0.01s)	crash (3.06s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
11	ac8ca1809a5858585858585858585858585858585858	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.38s)	rejected (0.01s)	rejected (0.22s)	rejected (0.05s)	rejected (0.00s)	rejected (0.00s)	rejected (0.07s)	rejected (0.12s)	rejected (0.07s)	rejected (0.08s)	yes
12	928e7149494949494949494949494949494949092891	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.12s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.10s)	rejected (0.06s)	rejected (0.06s)	yes
13	8292928001922e2e2e2e2ea9a9a9a9a97fffa9a9a9a97a	malformed	fuzz_libcbor	rejected (0.02s)	rejected (1.60s)	rejected (0.00s)	rejected (0.12s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.08s)	rejected (0.07s)	rejected (0.06s)	
14	92929280922e2e2e2e2ef97e00928092923bfffffffff	malformed	fuzz_libcbor	rejected (0.01s)	crash (3.11s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
15	a9a9a9a9aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.60s)	rejected (0.01s)	rejected (0.14s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.08s)	rejected (0.06s)	rejected (0.07s)	yes
16	c2818281818181819a4901000000000ffff802ea6a6a6	malformed	fuzz_libcbor	rejected (0.01s)	crash (2.78s)	accepted (0.01s)	rejected (0.14s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.09s)	rejected (0.07s)	rejected (0.06s)	yes
17	92d8d8d8d85b0000000000000000b7b7b7b77fb7b1b7b7	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.06s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.08s)	rejected (0.08s)	yes
18	a790a7a7a7c9a7a700a0a7a7879a4f4f4fa7a7a7	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.69s)	rejected (0.01s)	rejected (0.14s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.08s)	rejected (0.06s)	rejected (0.06s)	yes
19	92926d809a002e2e80929a000003e82ef97e009292002e	malformed	fuzz_libcbor	rejected (0.02s)	rejected (2.71s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.07s)	rejected (0.08s)	rejected (0.06s)	rejected (0.06s)	
20	9292a880929a49494949482e2e2e5ef97e5f5f05050505	malformed	fuzz_libcbor	rejected (0.01s)	crash (2.98s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
21	9292adadadadadadc400141414141414129141414ad01	malformed	fuzz_libcbor	rejected (0.02s)	rejected (3.07s)	rejected (0.01s)	rejected (0.19s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.09s)	rejected (0.18s)	rejected (0.07s)	rejected (0.07s)	
22	bf9fbf9fbfffff800012bf9fbf9fbfffff800012121212	malformed	fuzz_libcbor	rejected (0.03s)	crash (3.28s)	rejected (0.02s)	rejected (0.30s)	rejected (0.05s)	rejected (0.01s)	rejected (0.00s)	rejected (0.12s)	rejected (0.28s)	rejected (0.07s)	rejected (0.06s)	yes
23	91af92920707070707070707809292020202d20202020	malformed	fuzz_libcbor	rejected (0.03s)	crash (3.91s)	rejected (0.02s)	rejected (0.20s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.10s)	rejected (0.07s)	rejected (0.06s)	yes
24	ab9a4848484848484848484848484848b7484848484848	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.22s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.04s)	rejected (0.10s)	rejected (0.06s)	rejected (0.06s)	yes
25	925a000000002e2e2e2e2e2d2e9a5a000000022ef52e2e	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.26s)	rejected (0.03s)	rejected (0.20s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.13s)	rejected (0.17s)	rejected (0.09s)	rejected (0.07s)	yes
26	918e92807f928f92930d0d9a4f434400000d0d0d0d0d0d	malformed	fuzz_libcbor	rejected (0.01s)	crash (3.40s)	rejected (0.02s)	rejected (0.29s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.12s)	rejected (0.23s)	rejected (0.06s)	rejected (0.07s)	yes
27	c89a535353535353535353535553922e272e2e2e2e2e	malformed	fuzz_libcbor	rejected (0.04s)	crash (3.73s)	accepted (0.03s)	rejected (0.24s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.11s)	rejected (0.28s)	rejected (0.08s)	rejected (0.09s)	yes
28	92762ef97e928092772e2e2e2e2e2e641bffff00008000	malformed	fuzz_libcbor	rejected (0.03s)	crash (4.26s)	rejected (0.01s)	rejected (0.17s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.07s)	rejected (0.06s)	yes
29	889a525252aa889a52522ab7	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.69s)	rejected (0.01s)	rejected (0.21s)	rejected (0.04s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
30	a933333333339292a9a9a9a9a9a99a49010092c6c6c6c6	malformed	fuzz_libcbor	rejected (0.01s)	crash (2.77s)	rejected (0.02s)	rejected (0.17s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
31	a9a89a57ffffffff57575757	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.88s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
32	925a000000002e342e2e2e2d2e9a5a000000022ef52e2e	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.84s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.07s)	rejected (0.06s)	yes
33	a9a9a9a9aaaaaaaaaaaaaaaaaaaaaaaaaaaa4b4b4b4b4b	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.04s)	rejected (0.01s)	rejected (0.14s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.14s)	rejected (0.06s)	rejected (0.08s)	yes
34	9292928092772e2e2e2e2e47f97e929280772e2e2e9292	malformed	fuzz_libcbor	rejected (0.01s)	crash (2.91s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
35	8d8d8d8d9a505050505064505050505050505050505050	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.81s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
36	a9a9a9a9aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.17s)	rejected (0.01s)	rejected (0.19s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.07s)	rejected (0.15s)	rejected (0.07s)	rejected (0.11s)	yes
37	82bfc19a51f19f01	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.10s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
38	92909280929292909292909280928092922e2e2e2e2e2e2e	malformed	fuzz_libcbor	rejected (0.01s)	crash (2.80s)	rejected (0.02s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.07s)	yes
39	94947f9a55bfff5563555555555559a9a9a86941c17787a	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.08s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.05s)	rejected (0.09s)	rejected (0.07s)	rejected (0.07s)	yes
40	ac8ea19a576161616161616ac8ea19a576161616161616	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.86s)	rejected (0.01s)	rejected (0.16s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
41	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.14s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.06s)	rejected (0.10s)	rejected (0.07s)	rejected (0.07s)	yes
42	a9a9a9a9aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa	malformed	fuzz_libcbor	rejected (0.02s)	crash (2.96s)	rejected (0.01s)	rejected (0.34s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.07s)	yes
43	ab429280929a4f4f4f4f0b0b0b0b0b0b0b0b0b0b0b0b0b	malformed	fuzz_libcbor	rejected (0.01s)	crash (3.03s)	rejected (0.01s)	rejected (0.15s)	rejected (0.03s)	rejected (0.00s)	rejected (0.00s)	rejected (0.07s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes
44	9f9f9f009f9f9f9f04059f049f9f9fffff059f046e6ebf	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.18s)	rejected (0.01s)	rejected (0.14s)	rejected (0.03s)	rejected (0.01s)	rejected (0.00s)	rejected (0.06s)	rejected (0.10s)	rejected (0.07s)	rejected (0.06s)	yes
45	cecececececececece9a5151515151515151515151515151	malformed	fuzz_libcbor	rejected (0.02s)	crash (3.05s)	accepted (0.01s)	rejected (0.21s)	rejected (0.04s)	rejected (0.00s)	rejected (0.00s)	rejected (0.06s)	rejected (0.09s)	rejected (0.06s)	rejected (0.06s)	yes

Results – Parser comparison

- Honestly: this is still very much ongoing
- There is so much to look at
- Unfortunately, the most interesting results are still in the disclosure process

Summing Up

- Cbor is complicated → Reality is complicated
- libcbor is not safe against memory leakage by default, even their own example skips the necessary flag
- Most vulnerabilities are DOS

Summing Up

What I ordered	What I got
CVEs	Pain & suffering
bug bounty	Sleepless nights
Finished Thesis	Confusion

- Cbor is complicated → Reality is complicated
- libcbor is not safe against memory leakage by default, even their own example skips the necessary flag
- Most vulnerabilities are DOS
- 80/20 Rule still stands

Summing Up

What I ordered	What I got
CVEs	Pain & suffering
bug bounty	Sleepless nights
Finished Thesis	Confusion

- Cbor is complicated → Reality is complicated
- libcbor is not safe against memory leakage by default, even their own example skips the necessary flag
- Most vulnerabilities are DOS
- 80/20 Rule still stands
- Anyone looking for a Masters thesis? There is still so much to do!

915 libcbor rejected (0.00s)
915 tinycbor rejected (0.00s)
915 jackson rejected (0.21s)
915 upokecenter rejected (0.05s)
915 fxamacker rejected (0.00s)
915 ciborium rejected (0.00s)
915 petero rejected (0.05s)
915 node-cbor rejected (0.09s)
915 cborg rejected (0.10s)
915 cbor2-js rejected (0.11s)
916 cbor2-py rejected (0.02s)
916 libcbor rejected (0.00s)
916 tinycbor rejected (0.00s)
916 jackson rejected (0.22s)
916 upokecenter rejected (0.05s)
916 fxamacker rejected (0.00s)
916 ciborium rejected (0.00s)
916 petero rejected (0.05s)
916 node-cbor rejected (0.11s)
916 cborg rejected (0.12s)
916 cbor2-js rejected (0.14s)
917 cbor2-py accepted (0.02s)
917 libcbor rejected (0.00s)
917 tinycbor accepted (0.00s)

Questions?