IETF Hackathon

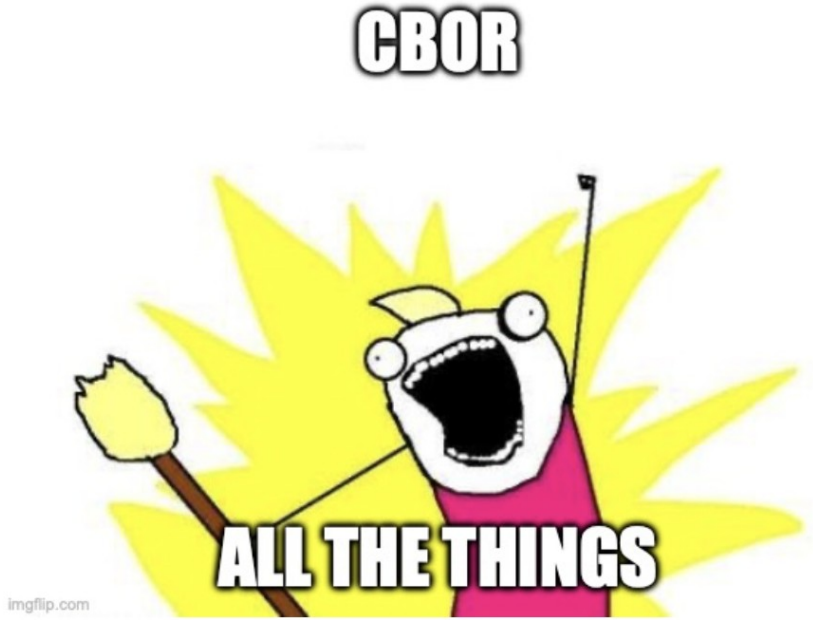
IETF 121

2–3 November 2024

Dublin, Ireland

Hackathon Plan

- SCRAPI – CBOR All The Things
- draft-ietf-scitt-scrapi
- Programmer accessibility balanced with the CBOR mission
- Implement the spec and updater the spec step-by-step



What got done

- We got done:
  - Fresh implementation of RFC 9290 in python
  - Fresh implementation of SCRAPI from first principles
  - End-to-end registration and retrieval of receipts from a live Transparency Service
  - Lots of improvement to the SCRAPI spec itself

I'd like a simple Python module that implements RFC9290. Specifically, I would like:
* a function that takes a dictionary and turns it into a 9290 CBOR object
* a function that takes a 9290 CBOR object and turns it into a dictionary

JSON Problem Details Example

json Copy

```
{
    "type": "https://example.com/error/validation-error",
    "title": "Validation Error",
    "detail": "Missing required field 'username'.",
    "instance": "/requests/12345",
    "response-code": 400
}
```

Corresponding RFC 9290 CDDL

In RFC 9290, each field has a numeric key:

- **instance** (1) - Corresponds to `"type"` in the JSON input.
- **title** (2)
- **detail** (3)
- **instance** (4)
- **response-code** (5)

Here's the CDDL to represent this structure:

cddl Copy

```
problem-details = {
    ? 1 => tstr,         ; URI identifying the type of problem (mapped from "type" in JSON)
    ? 2 => tstr,         ; Short human-readable summary of the problem
    ? 3 => tstr,         ; Detailed explanation of the problem
    ? 4 => tstr,         ; URI to identify the specific occurrence of the problem
    ? 5 => uint          ; CoAP response code representing the error
}
```

python

Copy

```python
import cbor2

# Define the mapping of field names to their numeric keys
FIELD_KEYS = {
    "type": 1,
    "title": 2,
    "detail": 3,
    "instance": 4,
    "response-code": 5
}

def encode_problem_details(data):
    """
    Encodes a dictionary into a CBOR object according to RFC 9290 using numeric keys.

    Parameters:
        data (dict): Dictionary containing RFC 9290 problem details.

    Returns:
        bytes: CBOR-encoded bytes representing the problem details.
    """
    # Map the dictionary keys to their numeric counterparts
    numeric_key_data = {FIELD_KEYS[k]: v for k, v in data.items() if k in FIELD_KEYS}
    return cbor2.dumps(numeric_key_data)

def decode_problem_details(cbor_data):
    """
    Decodes a CBOR-encoded problem details object with numeric keys into a dictionary.

    Parameters:
        cbor_data (bytes): CBOR-encoded bytes representing the problem details.

    Returns:
        dict: Dictionary containing RFC 9290 problem details with textual keys.
    """
    # Decode the CBOR data
    decoded_data = cbor2.loads(cbor_data)
    # Convert numeric keys back to their textual representation
    text_key_data = {key: decoded_data.get(FIELD_KEYS[key]) for key in FIELD_KEYS if FIELD_KEYS[key] in
    return text_key_data
```

Usage Example

python

Copy

```python
# Define problem details according to RFC 9290
problem_details = {
    "type": "https://example.com/error/validation-error",
    "title": "Validation Error",
    "detail": "Missing required field 'username'.",
    "instance": "/requests/12345",
    "response-code": 400
}

# Encode to CBOR
cbor_encoded = encode_problem_details(problem_details)
print("CBOR Encoded:", cbor_encoded)

# Decode back to dictionary
decoded_details = decode_problem_details(cbor_encoded)
print("Decoded Details:", decoded_details)
```

Imagine this JSON: what would be the CDDL? Assume all variables are tstr.

```
{
 "metamap": {
   "key": value",
   "key2": "value2"
 }
}
```

To represent this JSON structure in **CDDL**, where all variables are `tstr`, you can define it like this:

cddl Copy

```
my-object = {
    "metamap": meta-map
}

meta-map = {
    * key => value
}

key = tstr
value = tstr
```

Explanation:

- `my-object`: The main object containing the `"metamap"` field.
- `meta-map`: A map that allows multiple key-value pairs (`* key => value` indicates an open-ended set of key-value pairs).
- `key` and `value`: Both defined as `tstr`, meaning they are text strings.

Applying the science

- Nobuo-san from Sokendai applied SCRAPI to RFC9472 – Yang statements
- Specifically to see if real application concepts map well to SCITT concepts (eg Yang data model to SCITT Statements)
- Seems like the spec may be sufficient to cover the software supply chain needs. Nothing missing discovered, but not completely proven

What we learned

Working with COSE/CBOR is not as easy as it should be

- But the specs must be fine, if ChatGPT can do it
- It’s still worth doing the work to make SCRAPI consistent
- We had some good conversations about improving accessible tooling

SCRAPI is on the right track, needs more implementations to test it.

TEAM

Team members:

Jon Geater

Steve Lasker

Robin Bryce

Cedric Fournet

Roy Williams

Nobuo Aoki