——

–, —

Report Verifiable Credentials

Systems and Computer Engineering at Carleton
Ottawa, Canada

June, 2024

Artin Biniek, Arvin Samiei

Abstract

The w3c verifiable credentials provides a way of providing a verifiable means of ensuring ones identity. For instance veryifing ones identity for the legal age for driving via drivers licence.Verifiable credentials have certain specifications to follow for different version such as V2.0 and V1.1. However Verifiable credentials lack the necessary testing to cover BDD scenarios and constrained combinatorial scenarios. On top of this there are no current means ensuring that the json schemas provided to a validator ensures its validating w3c v1.1 and v2.0. There is also no support for a generator that efficiently produces v1.1 and v2.0 of verifiable credentials along with abiding with its specifications. In this research the task is to provide a means of validating such credentials so that it ensures its abiding by the specification as outlined in w3c. Another task is to generate these credentials so that it ensures n-way combinations is covered. Finally various input test cases will be created to cover the bdd scenarios along with combinatorial scenarios of the w3c verifiable credentials.

Keywords: W3c(world wide web consortium),VC(Verifiable credentials),BDD(behavioral driven development),Combinatorial testing

1 Motivation

The aim of this document is to report on the four main approaches that were performed with respect to VC 1.1 and VC 2.0 for w3c which involves VC validation, VC generation, BDD testing, and combinatorial testing. Each of these approaches will further discuss the purpose of this approach,what the approach is,tools/packages that were used to replicate this approach,the set up and running the script, the pseudo code and explanation of each aspect of the code.

2 Literature review

2.1 Digital identities and verifiable credentials[2]

[1] The paper discusses that verifiable credentials and digital wallets offer a convenient,secure,along with a privacy oriented alternative then to the current physical along with digital identity management systems. The paper further discusses how the current issues with authentication involve disclosing too much personal information raising privacy concerns along with having an increase in verification time. This paper relates to our research as we are similarly performing studies on verifiable credentials specifically with the w3c verifiable credentials data model.However the way in which our paper differs is that it does not focus on the benefits or disadvantages of VC compared to modern authentication rather it aims to validate the json-ld model provided by w3c VC.

3 VC Validation

3.1 Purpose of VC Validation

The purpose of conducting VC validation is to validate the JSON schema and check if it abides with the specifications of either VC 1.1 and VC 2.0. Another reason for creating the validation script is report any failures of the existing schema and give clear results of what has failed/passed.

3.2 Tools/packages that were utilized

1. Tools involved in development environment:
    - IDE:Visual studio code
    - programming language:Python
    - Command line: Ubuntu commandline interface
2. Validation script standard library packages:
    - json:Allows for parsing json strings and coverting it to python objects

- os:Allows for referencing files from a directory and can perform functionality like reading along with writing to its path
- validators:A validation library that validates a value not requring defining a schema [2].
- dateutil and prarser:date util is a python library while parse is used for parsing out dates into datetime objects.

3. utils.py:

   - Verifiable credential:A constant defined in utils.py
   - Verifiable presentation:A constant defined in utils.py
   - VC-SCHEMA-1:The context of VC1.1
   - VC-SCHEMA-2:The context of VC2.0
   - DATETIMETYPE1:The datetime object that is used for VC1.1
   - DATETIMETYPE2:The datetime object that is used by VC2.0
   - required attributes:The json schema that defines the required child attributes for parent attributes
   - Restriction:Is a class function
   - special-reqs:A defined json schema that ensures that "type" in credentialStatus is a Url.

3.3 Set up and how to run script

To setup the script ensure both utils.py and maximusvalidator.py are both in the same directory,ensure the test cases that are being added are within the proper paths so that the validator script can reference it properly. For V2.0 of w3c verifiable credentials the path name is ./tests/input/ for V1.1 it is ./vc-data-model-1.0/input/. Then to execute the script "python3 maximusvalidator.py"

3.4 The Approach

The approach of the VC validator works by the following it first checks if the context is present,is an array,and has the required VC JSON-LD schema URL as the first element. Then for each key it checks if all the required attribute's if any are present for the parent attribute its checking for. Then it checks for each key(required attribute) has its value type being correct which is all performed through recursive calls.

3.5 Pseudo Code and Explanation

Algorithm 1 VC validator

```
1: Read VC version and configurations
2: Validate Context
3: if Context is not an array then
4:     log error
5: end if
6: if Context does not include the main json-ld schema then
7:     log error
8: end if
9: if Context URIs are not valid then
10:     log error
11: end if
12: if Context includes reserved keywords (like "Verifiablecredential") then
13:     log error
14: end if
15: function VALIDATE-ATTRIBUTES(attribute)
16:     if All required properties are not present then
17:         log error
18:     end if
19:     if All restrictions are not satisfied then
20:         log error
21:     end if
22:     if Attribute's type is a basic type then
23:         Check the attribute matches the type
24:     end if
25:     if Attribute includes sub-attributes then
26:         for each sub-attribute in attribute do
27:             VALIDATE-ATTRIBUTES(sub-attribute)
28:         end for
29:     end if
30: end function
```

4 VC generator

4.1 Purpose of conducting this

The purpose of conducting vc generation is to satisfy the following to ensure that automated test generation is able to consider n-way combinations of VC's to be covered, to produce various unit test cases involving verifiable credentials for both Vc 1.1 and VC2.0,and to esnsure the reliability of validation checking.

4.2 Tools/packages that were utilized

1. Standard library imports:
   - datetime: is used for handling date and time objects
   - itertools: Provides functions for creating iterators for efficient looping,particularly for generating combinations of attribute values
   - json:Is used for parsing and generating json
   - os:provides functions for interacting with the file system such as reading and writing
   - random:Used for generating random values
2. Util.py:
   - Verifiable-Credential: is a constant representing verifiable credential
   - required-attributes: is a dictionary
   - Verifiable-presentation: is a constant representing verifiable presentation
   - BIT-STRING-STATUS-URL:A constant representing a specific URL for bit string status
   - get-json-schema:A function to retrieve a json schema
   - VC-SCHEMA2:A constant representing VC V2.0 schema.

4.3 Set up and how to run script

To run and setup the script ensure that validvcGenerator.py,and utils.py is within the same directory.Then once this step is covered run the following command "python3 validVCGenerator.py".The next step is to either enter a for producing and storing VC's for VC 2.0 that abide by the w3c specifications or to enter b for producing and sotring VC's for VC1.1 that abide by the w3c specications.

4.4 The Approach

The approach of the VC generation first generates the values for each key in the JSON-LD schema. It does so by first iterating through each key in the JSON-LD schema then checks the type of each key and generates the appropriate values based on the type. If the type is @id it generates a URL and an object with a key that is a URL. If he type is a datetime it generates a datetime value. If its anything else it returns a predefined string. The second step is the combination of attribute values. It combines all possible values for the required attributes. It then ensures that all possible variations of the attributes are considered. The final step is selecting and storing the schemas. It does this by randomly selecting 50 schema from the generated combinations. Then it stores each selected schema in a separate file. The files are then automatically placed in a created directory "generatedVC's"

5 VC Generation Pseudocode

Algorithm 2 Generate All Possible Attribute Combinations

```
function GENERATE-BASIC-TYPE(attribute)
    if Type is id then
        return [random_id, {'id': random_id}]
    end if
    if Type is datetime then
        return [current_datetime]
    end if
    return [random_string]
end function
function GENERATE-ATTRIBUTES(attribute)
    Initialize req_attributes_map and arbitrary_attributes_map to store required and arbitrary attributes
    if Attribute has sub-attributes then
        for each req_attr in required sub-attributes do
            req_attributes_map.append(Generate-Attributes(req_attr))
        end for
        for each arb_attr in arbitrary sub-attributes do
            arbitrary_attributes_map.append(Generate-Attributes(arb_attr))
        end for
    end if
    if Attribute does not have sub-attributes then
        for each req_prop in required properties do
            req_attributes_map.append(Generate-Basic-Type(req_prop))
        end for
        for each arb_prop in arbitrary properties do
            arbitrary_attributes_map.append(Generate-Basic-Type(arb_prop))
        end for
    end if
end function
Generate all possible combinations of attributes:
for each required attribute do
    Include one of its possible values in each combination.
end for
for each arbitrary attribute do
    Choose either not to include it in the combination or include one of its possible values.
end for
return all the combinations
```

6 Conclusion

To conclude the report aimed to cover the following features that were implemented with regards to VC 1.1 and VC2.0 for w3c. The approaches involving VC validation,VC generation, BDD scenario testing, and Combinatorial testing. Where each of these features were specifically expanded upon to cover the aspects involving the purpose of the implementation of the said feature,the tools/packages that were utilized for the implementation of the feature,the set up and how the script is ran, the approach, and the pseudo code and explanation of the code itself.

References

[1] J. Sedlmeir, R. Smethurst, A. Rieger, and G. Fridgen, "Digital identities and verifiable credentials," *Business & Information Systems Engineering*, vol. 63, no. 5, October 2021. [Online]. Available: https://doi.org/10.1007/s12599-021-00722-y

[2] Validators, *Validators Documentation*, https://validators.readthedocs.io/en/latest/, 2024, https://validators.readthedocs.io/en/latest/.