Dipartimento di Ingegneria e Scienza dell'Informazione
Departement of Information Engineering ans Computer Science

Bachelor's Degree in
Computer Science

Final dissertation

HTTP STL - Security testing language and implementation

Sottotitolo (alcune volte lungo - opzionale)

Supervisor
Silvio Ranise

Student
Matteo Bitussi

Co-Supervisors
Andrea Bisegna
Roberto Carbone

Academic year 2021/2022

Ringraziamenti

...thanks to... TODO (in italiano)

Contents

Abstract

This thesis covers the work started in my internship at FBK in the context of Single-Sign-On (SSO) protocols testing. SSO protocols are becoming more and more poular these days, and are being used in very sensitive applications such as SPID or CIE. Seeing the vast number of different implementations that are being used from whatever type of service, there is the need to test them in order to ensure that (at least) the known most common vulnerabilities are avoided.
To avoid having to manually test each implementation, an automaic tool is necessary. Moreover, a standard language to define these test suites has to be defined. This is what will be shown in this paper.

1 Background

The idea of a pentesting tool used to test SSO implementations such as OAuth, OIDC and SAML was previously discussed by my colleagues Stefano Facchini [4], Claudio Grisenti [5] and Wendy Barreto [3], which developed a plugin in Burp with the intent of automating the the testing of OIDC and OAuth protocols.

1.1 Burp Suite

Burp is one of the most used application security testing software for web security testing. It works by the use of a proxy server over which a browser redirect the traffic to. Burp has access to the proxy, it can sniff HTTP packets and can edit them. Burp also gives the possibility of creating custom plugins giving to the developers access to the java API.

2 Design of the language

In the Design chapter I am going to talk about how the language and the plugin have been designed and how they work.

2.1 Introduction

The idea was to think of a language that could implement all the possible actions which a security tester would be wanting to do on a multipart webapp test. One of the objective was to think of a language that could define tests that could be defined once, but tested over multiple websites. For example, a series of tests to verify the well-known vulnerabilities of a particular protocol could be defined and then used on any type of website. I had to decide how to write and define the actual tests, i thought i could define a proper language with a dedicated parser, but it was not worth the effort, as there are already some well-tested alternatives available. I found a great alternative: i used JSON as a base over which write the tests. It is a convinient way of defining gerarchical sturctures like tests could be. The idea behind this language is that a specific message can be intercepted and checked or edited in some way, to do this we define various types The gerarchical structure and the details of the language will be discussed in the next charapter.

2.2 Test example: PKCE Downgrade

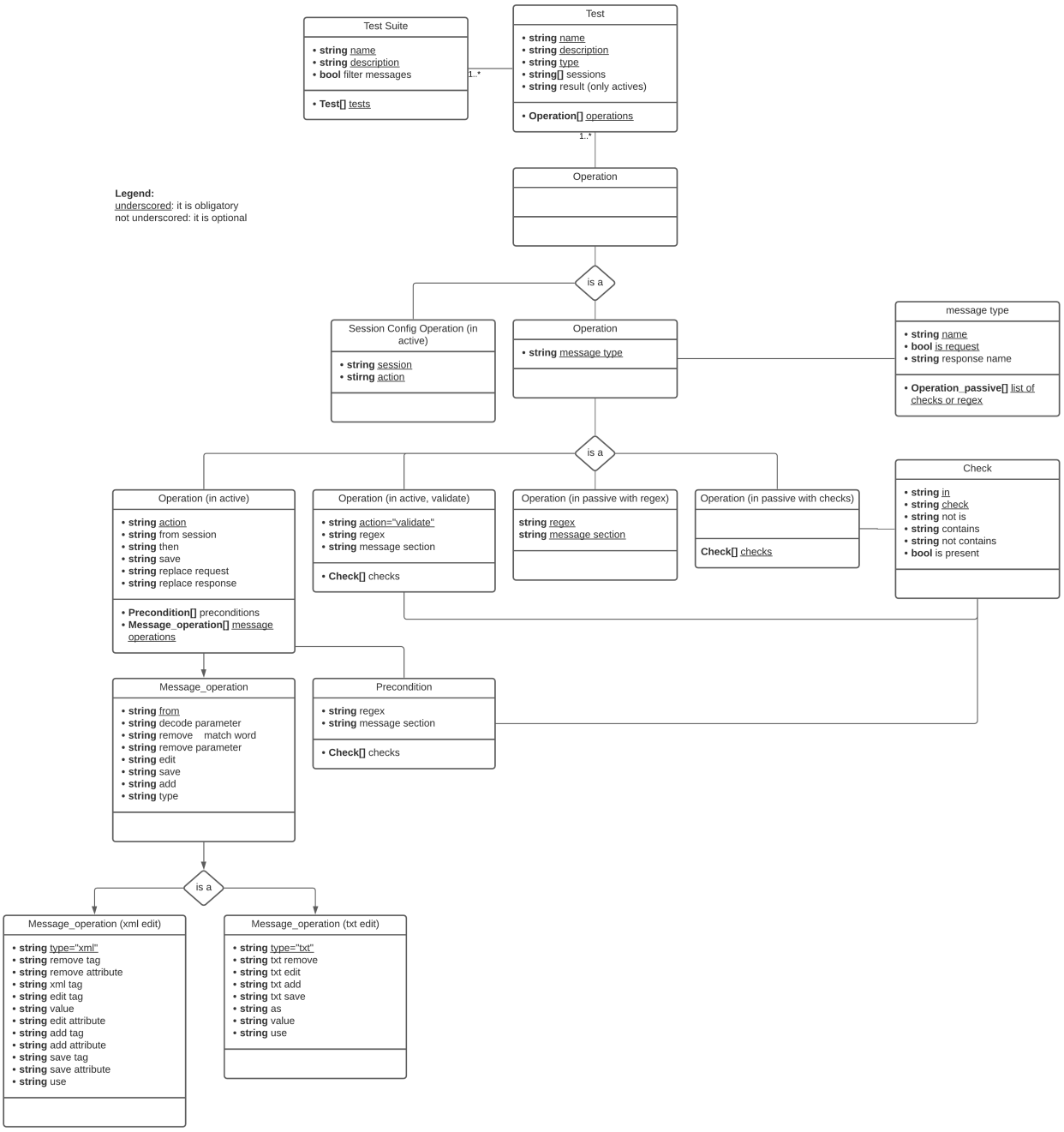
I want to introduce the language with an example. Due to its complexity, having a real example before the explanation of all its components could be helpful to understand their use. The implemented test has as objective to test an OAuth vulnerability where removing the parameter "code_challenge" from the url of an authorization request message will be downgrading the authentication proces in a way that PKCE will not be used if the service is vulnerable.

```
1 {
2     "test suite": {
3         "name": "OAuth Active tests",
4         "description": "A series of tests to test OAuth's well-known
    ↪ vulnerabilities"
5     },
6     "tests": [
7         {
```

```
8             "test": {
9                 "name": "PKCE Downgrade",
10                "description": "Tries to remove code_challenge parameter",
11                "type": "active",
12                "sessions": [
13                    "s1"
14                ],
15                "operations": [
16                    {
17                        "session": "s1",
18                        "action": "start"
19                    },
20                    {
21                        "action": "intercept",
22                        "from session": "s1",
23                        "then": "forward",
24                        "message type": "authorization request",
25                        "preconditions": [
26                            {
27                                "in": "url",
28                                "check param": "code_challenge",
29                                "is present": true
30                            }
31                        ],
32                        "message operations": [
33                            {
34                                "from": "url",
35                                "remove parameter": "code_challenge"
36                            }
37                        ]
38                    }
39                ],
40                "result": "incorrect flow s1"
41            }
42
43        }
44    ]
45 }
```

The first Operation defined in this test at line 17 is an operation that is used to start the session (and the browser). The automated browser will execute a series of actions defined by the user in a session track. The actions in this case will do a complete login in a website that uses OAuth as SSO login option. During the execution of the actions, language's Operations will be executed. At line 21 there is an Operation used to intercept an "authorization request" message, that is defined in an apoosite file where all Message Types are defined. Once an authorization request message is intercepted, the preconditions at line 32 are executed, checking that the parameter we want to test the vulnerability is used. This is done because the parameter "code_challenge" is not an optional parameter for the OAuth protocol, so, if it is not present, i want the test to result "not applicable" instead of failed or passed. The next part of the example is the Message Operation at line 35, where I tell to remove from the intercepted message's url the "code_challenge" parameter. The last part of the test is the definition of the result, the result is part of the evaluation of a test, in this case it is set to "incorrect flow s1", this means that I want the test to be considered passed if the execution of the session s1 is incorrect that is, if the execution of the session s1 encounters an error or an unexpected page.

2.3 Language structure



2.3.1 Test suite

The test suite is the main component which contains all the other one, it is composed by:

- Test suite name, the name of the test suite
- Test suite description, the description of the test suite
- Tests, which is a list containing the tests to be executed

2.3.2 Test

The Test object is the one that actually defines a test. As said earlier, a test is contained in a Test Suite, and has various items:

- name
- description

- type, it can be "active" or "passive"
- sessions, which is a list of the sessions which are needed in this test
- result, (only for actives) it defines the conditions over which the test is considered passed or not.
- operations, a list of operation objects which will be executed in the Test object

it can be defined either as an active or a passive test, depending on the type of actions it has to do on the intercepted messages. If a test doesn't need to manipulate the flow or the content of the messages, then it is considered passive, otherwise it is considered active. The list of Operations contained in a Test is executed iteratively one after the other.

2.3.3 Operation

The operation object is the thing that define what a test actually does. As shown in the image above, an operation could be either a standard operation or a session config operation, the latter is used to manage the sessions for the active tests (i.e. start, stop, pause). Depending on the type of test which an Operation is defined into, the standard Operation can be active or passive. In both cases, an operation has to contain the **message type** which defines the type of message to be intercepted in that particular operation (more info in the dedicated paragraph).
A **passive** operation has as objective to verify the presence (or absence) of some text or parameters in the intercepted message, it should contain one of the following options:

- A list of Check objects, which are then executed to check the presence of some text or parameter
- A regex inspection, which executes a inspection considering the intercepted message as plain text and executing a regex over it, if the regex has a match, the operation is considered passed, otherwise failed. Note that when a regex is used, it has to be specified also the message section over which to be executed (boy,head, url)

If the Test where the operations are defined is an **active** test, so if the intercepted messages need to be manipulated in some way, an active Operation has to be defined. It is composed by:

- action, the action it has to do (intercept, validate)
- from session, from which session to expect the message to be intercepted
- then, the action to to after the receiving and manipulation of the message (forward or drop)
- replace request (or response), specify a previously saved message in order to replace it to the intercepted one
- preconditions, a list of Precondition objects
- message operations, a list of Message Operation objects, which will do the actual manipulation of the intercepted message

If the action is set to "**validate**" the operation becomes like a passive operation, because its objective is just to verify that some messages are as expected. It will contain or a regex or a list of checks to be done.

2.3.4 Message Operation

The message operation is the Object that actually does the manipulations on the intercepted messages. It is composed by:

- from, the message section to work on
- decode parameter (optional) it indicates which parameter or string to be decoded before processed

- encodings (optional) the list of encodings to be applied to the parameter or text to be decoded. The supported encodings are base64, deflate, url
- remove match word (optional), remove text from te specified section in the matched message, it uses a regex
- edit, edit the matched text
- save, (optional) used to save an entire message in a variable in a way it can be used in future operations
- add, (optional) add some text after the matched text
- type (optional) specify the type of edit you want to do over a decoded parameter

In a message operation there is the possibility to specify a parameter or some text to be decoded before manipulation, to do that specify with "decode parameter" the parameter to be decoded and with "encodings" the encodings necessary to decode the parameter. The parameter (or text) decoded, at the end of the Message operation will be encoded again automatically. The decoded parameter can be manipulated by means of the "**type**" tag, there is the possibility to intepreter the decoded parameter as plain text, and to edit it using some actions:

- txt remove
- txt edit
- txt add
- txt save

All the previous tags accept a regex, and whatever that regex matches will be edited or added or saved.

Another possibility is to interpeter the decoded text as xml, assigning the type tag "xml". This way we have various possible operations to be done on the xml:

- remove tag
- remove attribute
- edit tag
- edit attribute
- add tag
- add attribute
- save tag
- save attribute

2.3.5 Message type definition

The message type definition is needed in order to define some types of message that will be later used in the language to intercept them. The message type definition is not actually part of the language, but it is stored in a file in the burp folder. Anyway, the definition of the type of messages uses the same Objects as the language. A message type object is defined using these tags:

- name, the name that will be used in the language to reffer to this message type
- is request, se to true if the searched message is a request, false otherwise

- response name, the name that will be used in the language to reffer to the response of the searched message
- checks, a list of Check objects used to identify the message. If evaluated to true, the message is considered found

This is an example that defines the saml request and the saml response messages

```
1  {
2      "message_types": [
3          {
4              "name": "saml request",
5              "is request": true,
6              "checks": [
7                  {
8                      "in": "url",
9                      "check param": "SAMLRequest",
10                     "is present": true
11                 }
12             ]
13         },
14         {
15             "name": "saml response",
16             "is request": true,
17             "checks": [
18                 {
19                     "in": "body",
20                     "check param": "SAMLResponse",
21                     "is present": true
22                 }
23             ]
24         }
25     ]
26 }
```

So, if "saml request" is used in an Operation, the message having the parameter SAMLRequest in his url will be intercepted an processed by the Operation.

2.4 The oracle

The ensemble of all parts of the language that decide the result of the tests is called Oracle, the oracle decides whether a test should be considered passed or failed. I decided to build the oracle in a way that can be almost fully customized by the user. There are various components that belong to the oracle. If we take the example from above, we can see that the test has a tag "result" which value is "incorrect flow s1", this means that the oracle will evaluate the test as passed if and only if the execution of the session track of the session s1 will be incorrect, this means that if the browser will encounter some type of page which was not meant to encounter, the test will be considered passed. With "was not meant to encounter" i mean that the actions in the session track cannot be done, because the objects to be pressed in the page are not present. With this check i also added the possibility to do preconditions on the tests, they are useful to check if a given test is applicable or not. For example, at line 25 in the example above, I want to consider the test "not applicable" if i can't find the parameter code_challenge in the authorization request message. This means that is not possible to apply the given test if the preconditions are not satisfied.

2.5 Sessions

A session is defined by a session track, which is a series of commands that the browser will execute automatically during execution of the Tests. There is the possibility of defining and using more than one session, in a way that (i.e.) reply tests can be executed. As said in the previous sections, a "from session" tag can be specified in the Operation, this will tell in which of the available session search

the desired message. To define the session track I have used and extended the idea used in the Micro Id Gym tool [5][4], adding some options like "wait" and "clear cookies" functionalities. The syntax of the session track is based on the plain text export of Katalon Recorder[2]. An example of a session track:

```
open | https://www.google.com/ |
click | id=L2AGLb |
click | link=Accedi |
click | id=identifierId |
type | id=identifierId | matteo.bitussi@studenti.unitn.it
click | id=identifierNext |
click | id=clid |
type | id=clid | matteo.bitussi@unitn.it
click | id=inputPassword |
type | id=inputPassword | password
click | id=btnAccedi |
click | link=Gmail |
```

This session track will do the login on the Unitn website using some credentials and password. The actions supported are:

- open | url |, to open an url
- click | id=, link=, xpath= |, to click on a http object with the given id, link or xpath
- type | id= | text, to write on a given http element the given text
- wait | milliseconds, to make the execution of the session wait for a given time
- clear cookies |, to make the browser of the session clear all of the cookies in it

3 Implementation

In this chapter I will describe the implementation of the language and the plugin, and also the problems faced and the solutions adopted.

3.1 The plugin

For the implementation of the Burp's plugin I have decided to start from a work done by Wendy Barreto [3], which did a similar plugin for OIDC and OAuth SSO protocols, this was a good base to start with my implementation. The plugin code is written in Java, I used the Burp's interface classes to interact with it. The standard usage of Burp Suite is based on the execution of a browser which connects to the Burp's proxy, in a way that all the packets can be intercepted, viewed or edited and forwared or dropped from the Burp interface. The tester would do some actions on the browser and watch the flowing packets in Burp and then check them or edit them. With the plugin the idea is the same, but the operation done on the browser and the checks or edits on the messages are made automatically, in a way that the tester doesn't have to do them by itself.

3.1.1 Interface

* insert image *

3.1.2 Test execution

The test execution differs from static to dinamic, as static tests don't need the the edit of the messages, the execution of the session track is done once, the messages are saved and the tests are executed on the saved messages. I have also added the possibility of exporting the saved messages to a file, in a way that they can be imported in the plugin and tested again. On the other hand, active tests needs to edit the messages, so the execution of the track has to be repeated for each test.

3.1.3 Decoding & encoding of parameters

As said in the previous chapter, the encoding and decoding of parameters is possible. To do that, a list of encodings has to be provided, i.e. url, base64, deflate. Once the specified message is intercepted, the parameter is taken and decoded following the order of the provided encodings. To do that, i used part of the code of SAML Raider [1] which did the decoding of SAML Requests and responses parameters. I've taken that part of the code and edited it to fit the plugin. SAML Raider is a Burp's plugin used to manage SAML certificates.

3.1.4 SAML certificate managing

In SAML Requests and responses there is sometime the need to remove or edit the certificate associated to that request or response, so, to speed up the process I did add a specific tag in the language to remove or edit the certificate signature. There is still the possibility of doing it by editing the SAML request or response with a regex, but this way is more convinient. To do this, i used a part of the code of SAML Raider [1], editing it to fit my needs.

3.1.5 Oracle

The oracle is based on three main components:

- Evaluation of the complete (or incomplete) execution of the session track
- Evaluation of the Precondition objects
- Evaluation of the Validate objects

If all of the above conditions are met, the test is considered passed, otherwise it is considered failed. The oracle can be built for example using Validate objects verifying that some intercepted messages satisfy some conditions like having a particular parameter or string in them. To identify abnormal pages like error pages the session track evaluation should be sufficient, because if some of the actions could not be executed means that the original "flow" of pages was not followed.

3.1.6 Session managing

The sessions are managed independently, each session is basically a browser that is launched when a session is started. Each session can follow a different session track defined in the apposite tabs. Every session is ran in a separated thread to make parallelism possible. By the use of specific commands in the language, is possible to do some actions on each session, like stop it, pause it, or clear its cookies. Each browser uses a different proxy port, so that it is possible to know from which session the messages come from.

4 Uses cases

In this chapter I will talk about some examples of use cases in which my language could be used.

4.1 SAML Use-Case

My work has been used by Sofia Zanrosso, for her bachelor thesis, her objective was to search for SAML vulnerabilities and to define a series of well-known test to verify them. She defined all the tests using my language and tested multiple SAML implementation with it.

4.2 OAuth & OIDC Use-Case

5 Related work

In this chapter i will talk about other tools and works that are related to mine. I will then compare them to my language.

5.1 SSO Testing language and Plugin

This is a similar plugin and language done by Wendy Barreto [3] to test OAuth and OIDC SSO protocols in a more dinamic way. Previously in this thesis i said that I have taken part of her work which was a good base to start with my work, editing it to suit my needs. My idea of a language that could be used for any type of test over HTTP was born when I used her plugin, which was limited to OAuth and OIDC tests. I wanted to enlarge the possible tests without a restiction on a specific protocol. One of the things that I used is the interface of her plugin, adding some buttons and tabs to deal with multiple session tracks and added functionalities.

Action	Old language	New language
Custom message filtering	Only on active tests	supported
Edit string	only by regex	supported with regex and check construct
Remove string	only by regex	supported with regex and check construct
Add string	not supported	supported
Check parameter	only with regex	with regex and check construct
Multiple operations in single message	not supported	supported
Saving and reusing of values and messages	not supported	supported
Multiple sessions in single test	not supported	supported
Custom oracle definition	not supported	by using regex and checks

5.2 Micro ID Gym

Another plugin for burp that was developed by two my colleagues was Micro ID Gym, a tool to test OIDC and OAuth implementations. This plugin was used by Wendy Barreto to do her work. The old plugin of Stefano and Claudio was based on a track, which defined some actions to be done by the browser, which was a selenium istance. The plugin checks the messages and based on the test defined in the plugin tells if there is a vulnerability or not. Starting from the first version of Stefano, to the last of Wendy, the plugin was improved. In the first two versions of Stefano and Claudio the plugin had its tests hard-coded, in a way that ony the supported tests could be executed, with little settings to change. If a new test had to be implemented, the plugin had to be recompiled. This version of the plugin worked well, but as said, the tests could not be customized or adapted by the user.

5.3 Last plugin version

Wendy improved the plugin by removing the staticity of the test, adding the possibility to customly define all of the tests with the use of a JSON language.
This is the last version of the plugin which i started working to. The plugin supported the definition of passive and active tests: passive tests are tests where the messages are not edited, active tests are tests where there could be an edit of one or more messages. The availabe test actions worked well, but there were some limitations on the possible actions, especially in the active tests. For example:

- Limited oracle for the verification of active tests, having just the verification of the correct execution of the operation and a check for the string ”error” on the last page of the browser

- The filtering of which message to check or edit for static tests was limited: (only "Authorization grant message", "Response messages", "Request messages" and "All messages")
- Only regex were supported to search something in a message
- Unable to work over encoded parameters
- Impossibility of doing multiple operations on a single message
- Impossibility of saving a parameter and using it somewhere else

Some of which stated as future works in Wendy's thesis. Moreover, the language was thought to be used with tests for OIDC and OAuth, other SSO protocols such as SAML could not be tested, because of the fact that SAML parameters are encoded, so editing them or verifying them is not possible. This is the biggest limitation that made me decide to redesign the language.

6 Conclusions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Bibliography

[1]

[2] Katalon recorder sample plugin. https://github.com/katalon-studio/katalon-recorder-sample-plugin. Last accessed 19/11/2021.

[3] Wendy Barreto. Design and implementation of an attack pattern language for the automated pentesting of oauth/oidc deployments. Master's thesis, Università degli Studi di Trento, 2021.

[4] Stefano Faccini. Design and implementation of an automated tool for checking saml sso vulnerabilities and spid compliance. Master's thesis, Università degli Studi di Trento, 2019/2020.

[5] Claudio Grisenti. A pentesting tool for oauth and oidc deployments. Master's thesis, Università degli Studi di Trento, 2019/2020.

Allegato A Titolo primo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

A.1 Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

A.1.1 Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Allegato B Titolo secondo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

B.1 Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

B.1.1 Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.