Application architecture

Input Machine

Infrastructure

The machine that receives the starting inputs is located in GCloud. It is running an Apache NiFi instance. The following security measures are set in place:

- Confidentiality
    - NiFi is secured with TLS, thus, the data that is sent to and sent by input.itapp.eus is completely confidential.
- Integrity
    - NiFi is secured with TLS, maintaining data integrity in all the connections from and to the input.itapp.eus
- Authentication
    - Not anyone can connect to NiFi to start and stop processors. The 8080 port on which NiFi runs only listens to localhost connections. You need to tunnel though SSH to the input.itapp.eus machine to be able to connect to NiFi. You need an SSH key to authenticate.

Entries

We have three data sources that send information in a variety of ways, through calls to some APIs and loading data from local, to the machine that collects the Entries, input.itapp.eus.

These entries are JSON formatted.

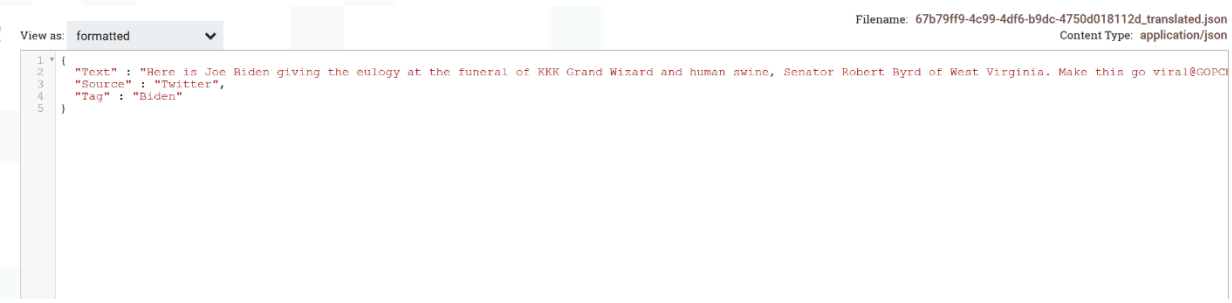
- Validation & Sanitization
    - We ensure that the file that NiFi receives is a JSON.
    - JSONs that do not have the fields we need are discarded.

- The JSON is parsed through a whitelist that filters characters we do not allow.

Outputs

The machine input.itapp.eus sends the JSONs through HTTP via a Remote Process Group in NiFi once they have been treated and enriched to the gateway machine, gateway.itapp.eus.

- Validation & Sanitization
  - Before sending the JSON we check its structure and characters once again.
    - We ensure the enrichment we have done is there.
    - We ensure the flowfiles those JSON create are not empty.



Gateway Machine

Infrastructure

The machine that receives data from input.itapp.eus, gateway.itapp.eus, is located in GCloud. It is running an Apache NiFi instance. The following security measures are set in place:

- Confidentiality
  - NiFi is secured with TLS, thus, the data that is sent to and sent by gateway.itapp.eus is completely confidential.
- Integrity
  - Nifi is secured with TLS, maintaining data integrity in all the connections from and to the gateway.itapp.eus
- Authentication
  - Not anyone can connect to NiFi to start and stop processors. The 8080 port on which NiFi runs only listens to localhost connections. You need to tunnel though SSH to the gateway.itapp.eus machine to be able to connect to NiFi. You need an SSH key to authenticate.
  - Not anyone can send data to Gateway's NiFi. The iptables configuration in gateway.itapp.eus blocks all incoming traffic except for the input.itapp.eus machine.

Entries

We have only one input channel communication where data comes from the input.itapp.eus machine.

The entries are JSON formatted.

- Validation
  - We ensure that the file that NiFi receives is a JSON.

Outputs

The machine gatway.itapp.eus sends the JSONs through HTTP via a Remote Process Group in NiFi once they have been treated to the cloud NiFi cluster (node1.itapp.eus, node2.itapp.eus, node3.itapp.eus).

Once we have checked the input for gateway is a JSON file we trust its origin and just send it to cloud.itapp.eus.

Cloud Machine (Nifi-Cluster)

Infrastructure

The cluster is composed by three machines that receive data from the gateway machine. The cluster is located in GCloud. It is running an Apache NiFi Cluster. The following security measures are set in place:

- Confidentiality
    - NiFi Cluster is secured with TLS, thus, the data that is sent to and sent by all the nodes is completely confidential.
- Integrity
    - Nifi is secured with TLS, maintaining data integrity in all the connections from and to all the nodes.
- Authentication
    - Not anyone can connect to NiFi to start and stop processors. The 8080 port on which NiFi runs only listens to localhost connections. You need to tunnel though SSH to all the nodes to be able to connect to NiFi. You need an SSH key to authenticate.
    - Not anyone can send data to Cloud's NiFi. The iptables configuration in cloud.itapp.eus blocks all incoming traffic except for the gateway.itapp.eus machine.

Entries

We have only one input channel communication where data comes from the gateway.itapp.eus machine.

The entries are JSON formatted.

- Validation
    - We ensure that the file that NiFi receives is a JSON file.
    - Once we convert it to JSON we rerun all the checks from input.itapp.eus.

Outputs

The cluster sends the JSONs once they have been treated and enriched to the Hadoop Cluster, which is located in the same machines.

- Validation & Sanitization
    - Before sending the JSON we check its structure and characters once again.
        - We ensure the enrichment we have done is there.

The JSONs are grouped into groups of 10 to lower the number of files on HDFS.

Hadoop cluster

Infrastructure

The cluster is composed by three machines that receive data from the NiFi cluster. The cluster is located in GCloud. It is running a Hadoop Cluster. The following security measures are set in place:

- Confidentiality
    - Hadoop Cluster is secured with TLS, thus, the data that is sent to and sent by all the nodes is completely confidential.
- Integrity
    - Hadoop is secured with TLS, maintaining data integrity in all the connections from and to all the nodes.
- Authentication
    - Not anyone can connect to Hadoop to start and stop processors. You need an SSH key to authenticate yourself in the machine and stop Hadoop.
    - Not anyone can write data to Hadoop. Only processes which are owned by users that are in the supergroup group of the Hadoop Cluster can write data in it. Thus, only our NiFi should be able to write there. LEAST PRIVILEGE

Entries

We have only one input channel communication where data comes from the NiFi Cluster.

The entries are JSON formatted.

There are no security checks for the files in Hadoop.

Outputs

The cluster stores all the data in specific folders.

Spark cluster

Infrastructure

The cluster is composed by three machines located in AWS. It is running a Spark Cluster.

The following security measures are set in place:

- Confidentiality
  - The Spark Cluster is secured with TLS, thus, the data that is sent to and sent by all the nodes is completely confidential.
- Integrity
  - Spark is secured with TLS, maintaining data integrity in all the connections from and to all the nodes.
- Authentication
  - Not anyone can connect to Spark to execute notebooks. We use Spark's Log4j implementation to enable username/password based logins to the web UI (still under construction).

Entries

The notebooks we execute for the training fetch data from the Hadoop cluster. This data is trusted and thus it is not verified.

- Authentication
  - To be able to read from the Hadoop cluster the user that executes the Spark process needs to be added to the Hadoop's cluster supergroup.

```
df = spark.read.option("inferSchema", "true").json('/home/ubuntu/export/*.json')
df = df.select("Text").where("Tag == 'Trump'")
df.show()
```

Outputs

The cluster will create the model that will be used in the project which we do not check on save.

WEB APP

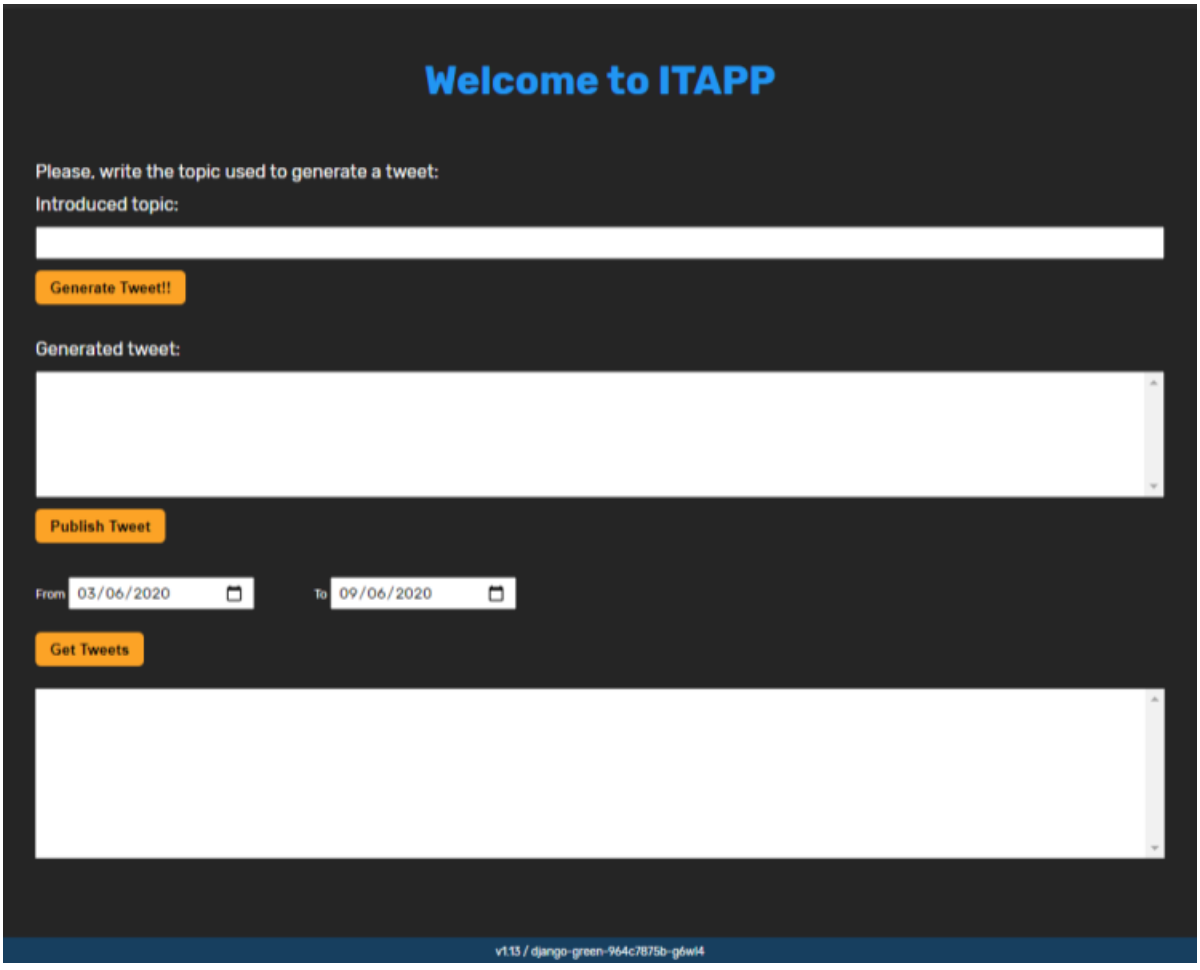
Infrastructure

The Django based web application runs over a machine located on Google Cloud. Anybody on the internet can access to it.

- Confidentiality
    - The web application is secured with TLS, thus, the data exchanges that happen with users are completely confidential.
- Integrity
    - The same way, TLS maintains the integrity of the messages that are exchanged with the web application.
- Authentication
    - Maybe we will add a login screen via Key Cloak, for now the web app allows anyone to interact with it.

Entries

The web application has three main data entries/inputs. Those are the imports and loading of the model the backend does, the user input fields on the application and the tweet loading it does from elastic search.



- Input testing
    - All the inputs present in the web application will be fuzz tested to avoid undesirable behaviors.
- Validation & Sanitization
    - All the input fields have limited characters you can input to them.
    - Once an input is sent to the backend it is filtered comparing it against regular expressions.
    - The data received from elastic search is checked:
        - It should be a JSON.
        - The dates of the first and last tweet should coincide with what the user inputted.

Outputs

The web application has two main data outputs. It posts the curated tweets to a Twitter account we have set up and saves those exact tweets on elastic search.

- The user's edition of the generated tweet is limited to known characters.
- Once an input is sent to the backend it is filtered comparing it against regular expressions.
- A JSON is formed with that tweet that respects the JSON structure we are using in elastic search.
    - We ensure the generated tweet has that structure.
- The new tweet is sent to twitter, elastic search and HDFS.

Secure coding practices

Code security

The development of the code of the web application Itapp will follow the following Secure Software Design Principles in order to create the most secure software possible. Those design principles will be checked against SonarQube, the static code analysis tool we have used.

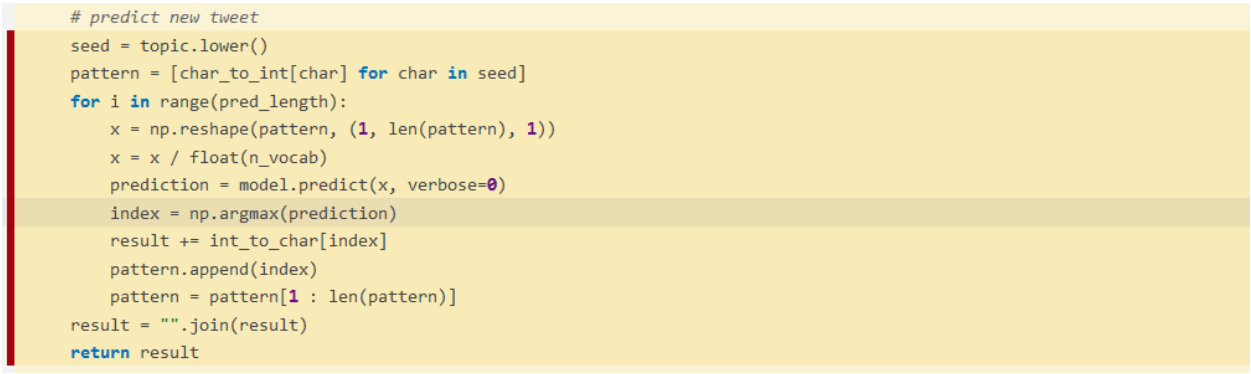
Code styling, zero smells, zero bugs

Itapp has been developed according to the needs we established and the warnings SonarQube was giving us. That way the code has ended up with no code smells nor bugs.

These SonarQube scans have been automatized to pass with the pipeline's execution, although they cannot stop the pipeline itself. The scans are manually checked after each execution.

To exemplify some of the changes made to the code here are some unused variable warnings that have been fixed:

Before



After

Economy of mechanism:

The code will be segmented into different parts that work in unison, each of the parts having a concrete goal and being the smallest and simplest possible. This translates to using the least lines of code possible and following the guidelines on code complexity set by SonarQube.

Code complexity is measured in the amount of nested checks a piece of code has.

For example, our code had the following code complexity issue solved:

```
if request.method == "POST":
    inp = Input()
    if "introduced_topic" in request.POST:
        topic = request.POST["introduced_topic"]
        # sanitization + validation
        topic = inp.sanizate(topic)
        print("TOPIC: " + topic)
        is_valid = inp.validate_topic(topic)
        # generate tweet
        if is_valid:
            tweet = predict_tweet(topic)
        else:
            error = True
            alert_message = "Introduced topic is not valid! Allowed: [a-zA-Z0-9] and [.,;] and single quote"
    elif "generated_tweet" in request.POST:
        generated_tweet = request.POST["generated_tweet"]
        tweet = generated_tweet
        # sanitization + validation
        generated_tweet = inp.sanizate(generated_tweet)
        is_valid = inp.validate_tweet(generated_tweet)
        if is_valid:
            # post tweet on twitter
            print(generated_tweet)
            # save tweet on elastic
            insert_tweet(generated_tweet)
            # save tweet on hdfs
            insert_hdfs_tweet(generated_tweet, "twitter")
        else:
            error = True
            alert_message = "Introduced tweet is not valid! Allowed: [a-zA-Z0-9] and [.,;] and single quote"
    elif "from" in request.POST:
        filtered_tweets = query_timestamp_range(
            request.POST["from"], request.POST["to"]
        )
        tweet_list = []
        for hit in filtered_tweets:
            print(hit.tweet)
            tw = TweetModel(hit.tweet, str(hit.date).split("T")[0])
            tweet_list.append(tw)
    topic_form = TopicForm(initial={"introduced_topic": topic})
    tweet_form = TweetForm(initial={"generated_tweet": tweet})
elif request.method == "GET":
    topic = ""
    tweet = ""
    tweet_list = []
    topic_form = TopicForm()
    tweet_form = TweetForm()
else:
    return redirect("/")
```

According to SonarQube the complexity of the nested elifs on the code in views.py was to large, thus, we refactored the code so the complexity could be was reduced. Some of the nested elifs where taken out and became their own functions.

```python
@csrf_exempt
def home_view(request):
    global topic, tweet, tweet_list
    alert_message = ""
    error = False
    if request.method == "POST":
        if "introduced_topic" in request.POST:
            error, alert_message = introduced_topic(request)
        elif "generated_tweet" in request.POST:
            error, alert_message = publish_generated_tweet(request)
        elif "from" in request.POST:
            get_tweet_list(request)
        topic_form = TopicForm(initial={"introduced_topic": topic})
        tweet_form = TweetForm(initial={"generated_tweet": tweet})
    elif request.method == "GET":
        topic = ""
        tweet = ""
        tweet_list = []
        topic_form = TopicForm()
        tweet_form = TweetForm()
    else:
        return redirect("/")

    return render(
        request,
        "index.html",
        {
            "tweet_list": tweet_list,
            "topic_form": topic_form,
            "tweet_form": tweet_form,
            "hostname": socket.gethostname(),
            "error": error,
            "alert_message": alert_message,
        },
    )
```

Fail-safe defaults

The code follows whitelisting methods for creating the if structures that decide what the application should do according to the inputs it receives and if the validation and sanitization break or not.

This avoids unwanted execution patterns as the only permitted execution paths are those the developers desire.

Also, all the errors some code pieces could raise are encapsulated in try catch statements.

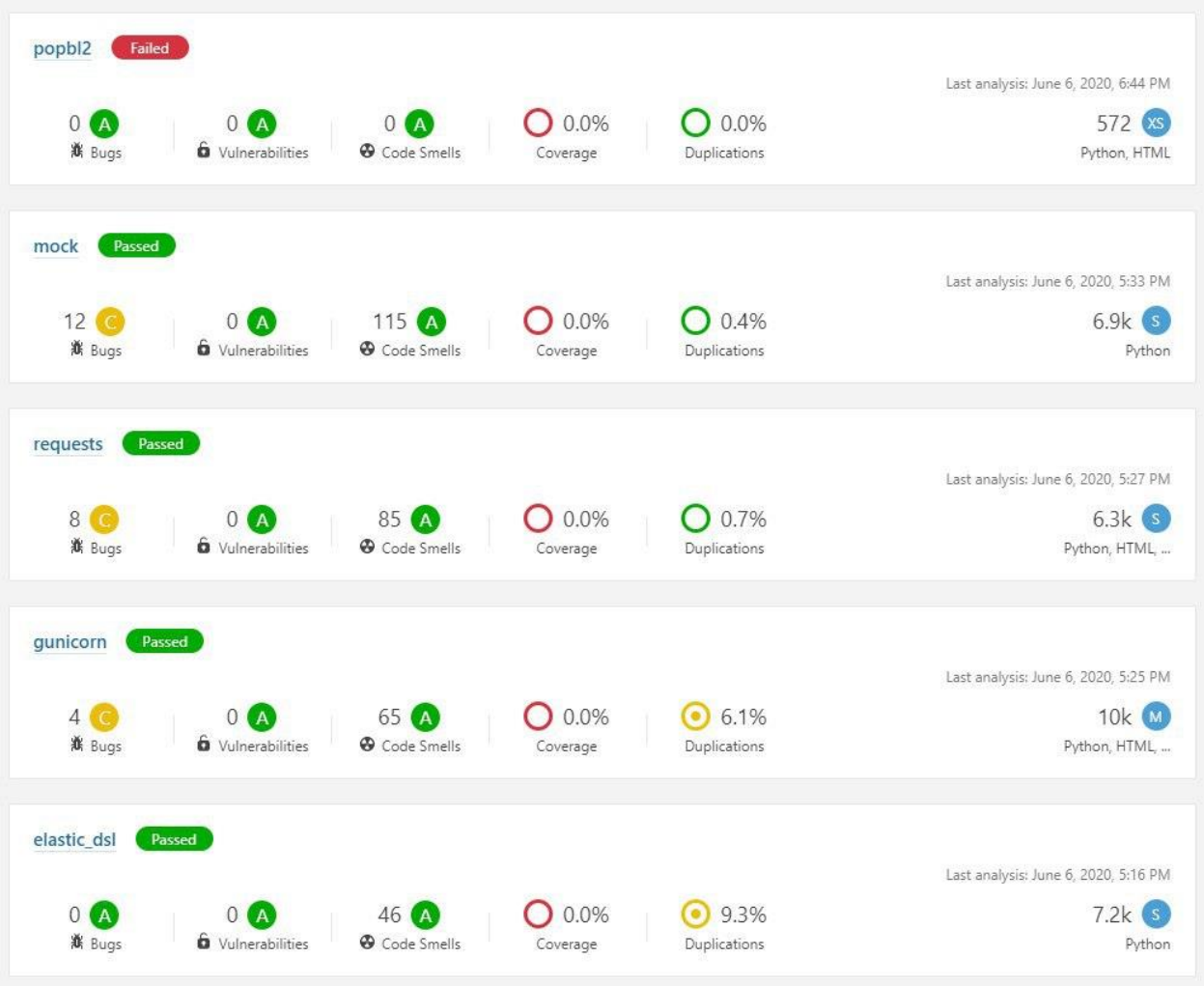
```
def insert_tweet(tweet_content):
    try:
        # create conection
        connections.create_connection(hosts=["node1.itapp.eus:9200"])
        # create the mappings in elasticsearch
        Tweet.init()
        # create and save the tweet
        tweet = Tweet(tweet=tweet_content, date=datetime.now())
        tweet.save()
    except Exception:
        pass
```

In the following piece of code belonging to the web apps backend we can see the only permitted course of action, if the action taken by the webpage is not known, is to reload the webpage.

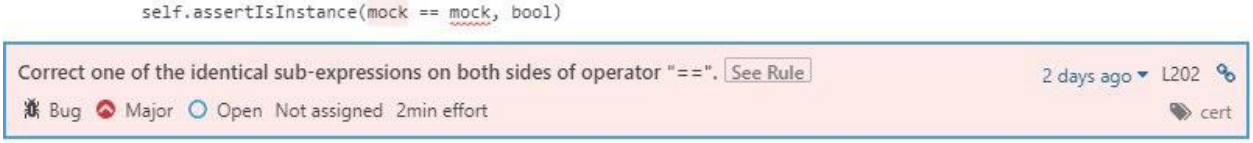
```
if request.method == "POST":
    if "introduced_topic" in request.POST:
        error, alert_message = introduced_topic(request)
    elif "generated_tweet" in request.POST:
        error, alert_message = publish_generated_tweet(request)
    elif "from" in request.POST:
        get_tweet_list(request)
    topic_form = TopicForm(initial={"introduced_topic": topic})
    tweet_form = TweetForm(initial={"generated_tweet": tweet})
elif request.method == "GET":
    topic = ""
    tweet = ""
    tweet_list = []
    topic_form = TopicForm()
    tweet_form = TweetForm()
else:
    return redirect("/")
```

Vulnerabilities in third party code

All the code that has been reused from other projects, including all the python libraries, have been uploaded to SonarQube to test their robustness (code smells, bugs, etc.) and vulnerabilities that could be present in them have been searched for.



The bugs and the code smells the python libraries had are mainly format dependent, meaning they make use of bad coding styling in the projects. They do not represent any major threats for the application.



The search for vulnerabilities in these libraries and Django (which could not be uploaded to SonarQube due to Nginx's size constraints) did not see any results, meaning no vulnerabilities were found. For this search the open issue list in each of the projects' GitHubs were checked along with webpages like CVE from MITRE.

Python-requests : Security Vulnerabilities

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9
Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending
Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2018-18074	522			2018-10-09	2019-10-02	5.0	None	Remote	Low	Not required	Partial	None	None
	The Requests package before 2.20.0 for Python sends an HTTP Authorization header to an http URI upon receiving a same-hostname https-to-http redirect, which makes it easier for remote attackers to discover credentials by sniffing the network.													
2	CVE-2015-2296				2015-03-18	2016-07-15	6.8	None	Remote	Medium	Not required	Partial	Partial	Partial
	The resolve_redirects function in sessions.py in requests 2.1.0 through 2.5.3 allows remote attackers to conduct session fixation attacks via a cookie without a host value in a redirect.													

Total number of vulnerabilities : 2 Page : 1 (This Page)