**CSCI4360/CSCI4308 (Computer Science Research/Adv Topics in PDC (Cloud Computing))- Final Report**

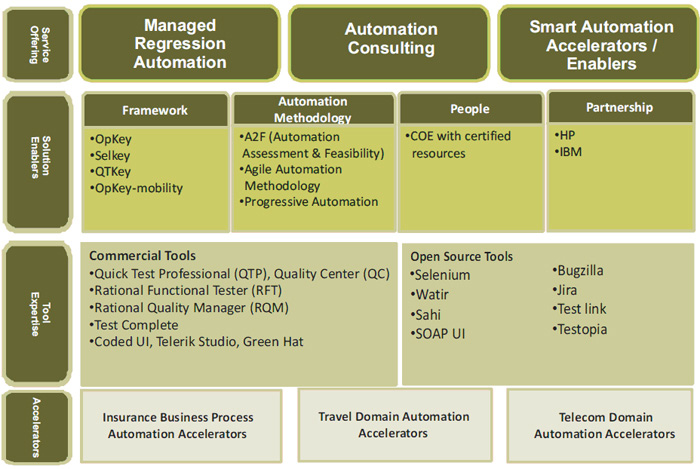
**Project Title:** Service to Test Web Applications in a Sanitized Cloud Environment

**Team Members**: Lisa Blackerby, Ashley Mains, Warnitris May, Sam Riley, Adam Sturchio,

**Introduction and Current Practices**

Currently, Automation Testing is a common section of Quality Assurance that all companies are required to implement in software development to maintain customer satisfaction in online web applications. With the cost of this software development fluctuating rapidly depending on the need of the software, outsourcing software testing is not a new concept. Many companies offer testing services that are not only comprehensive, but also affordable for any sized company looking to create an online footprint.

Through research, our team found than many companies who offer the services of software quality assurance testing, share a common characteristic in that these services create testing frameworks utilizing a local virtual machine environment. One such example is CresTech, a worldwide testing service, utilizes a comprehensive and sweeping set of solutions for automation testing with a large source of commercial and automation tools:



Resource pulled from: <http://crestechglobal.com/solutions/test-automation/>

While their tool kit is wide and varied, It is interesting to note that utilization of cloud computing is not present as part of their solutions package. Additionally, with an increased interest in cloud hosting, it is intriguing to see how automation testing would fair when utilizing some of the same tools and testing methodologies.

With this knowledge, our team decided to utilize the Google Cloud Computing environment by automating the process of creating a virtual machine to create secure testing environments for web applications. By using a series of automated scripts, we created two separate web applications that use a php/mySQL framework on a LAMP server and REACT/Django REST Framework respectively and ran a series of security and quality assurance tests on them in sanitized environments.

**Proposed Methods and Evaluations**

As stated in the introduction, our goal was to automate the process of creating a virtual machine in the cloud in order to create a secure and repetitively sanitized testing environment for web applications. Additionally, we wanted to automate the process of setting up the web applications on a cloud instance to create sanitized environments for testing purposes. The next series of sections will define in detail the work accomplished in building our test web applications along with the cloud automation accomplished over the course of the semester.

**Web Application Creation of Different Frameworks onto Cloud Virtual Networks**

Our team decided to create 2 web applications utilizing vastly different frameworks. The reason behind this decision was to see if we could automate any kind of web application onto a cloud virtual machine.

The first web application was a was a registration system that allowed users to create a username, password, and email for a new account. Users were then permitted to login to their created account via username and password. The frontend of this application was created using php, and the database backend was created using MySQL. Our team then utilized a WAMP server for successful implementation of the application on a local machine. For the testing site to perform various functions, multiple php files were required. The first was a “register.php” page, that allows users to create a username and password for their account. The “login.php” file allows users to login with their created credentials, and the server.php file facilitates the form and database validation. Finally, the index.php, provides some necessary structure and execution of a “landing” page.

The second web application was a to-do list built using a ReactJS/Django REST Framework. Users can navigate to the React website to add, remove, and mark various to-do tasks as complete. Additionally, an admin of the website can log into Django’s back end interface to directly control the SQLite3 database, users, and website API.

Created by Facebook and first hosted by the social media website in 2011, REACT is an open source JavaScript library which allows developers to create large web application which can change data without reloading the page. Additionally, ReactJS makes it easy to create new features without rewriting existing code on an individual technology stack. As a result, only a single file, App.js, was needed to perform all the necessary frontend functions for our to-do list.

The rest of the work on this web application comes from the Django REST Framework. This python-based toolkit simplifies common practices in web development. Additionally—by default—the framework comes with a SQLite3 database, but has multi-database support, making it useful for many different technology stacks. Instructions to install and set up both frameworks can be found in Appendix \_\_.

For the back end, 6 files needed to be either created or altered: models.py, admin.py, serializers.py, views.py, urls.py, and settings.py. the models.py file is the definitive source of information about the data. It contains the essential fields and behaviors of the data being stored. Each model is mapped to a single database table. The file admin.py was altered to ensure the backend to the to-do application was easily accessible, but still secure. serializers.py allows for complex data such as query sets and model instances to be converted into native python datatypes which can then be rendered into JSON types, which is needed to function with ReactJS. The views.py file is the website controller that performs functions conceptually similar to GET and POST that one would see in other frameworks. In order for Django, React, and the database to interact with the appropriate configuration, url.py and settings.py needed to be altered to whitelist React’s designated port. For purposes of automating the cloud virtual machine that will host this application, it is necessary to ensure these last two files are configured appropriately to each individual cloud machine. This will be better defined in a later section.

**Automation of the Cloud Virtual Machine**

For the automation of a virtual cloud machine, we found it possible to create a batch script that generates the cloud virtual machine and a separate script that sets up and configures the local host appropriately. Some alterations need to be made to the script before running it, however. For instance, the script that creates a google cloud virtual instance requires an argument to be given for the virtual machine. Additionally, it is required to rename the PROJ variable. Additionally, the location of the files that need to be uploaded into the instance must be edited appropriately. Once the virtual machine is created, the script tried to establish a connection to the instance until one is made. The final action of the create virtual machine script is to run uptime, curl, and whoami commands on the instance to ensure proper functionality.

The local host script requires a little more configuration depending on the web application being hosted, but what is common is that python and selenium for python needs to be installed on each virtual machine. Additionally, the home directory of each script must be altered to match the local machine that runs the automation scripts. From here, the scripts that run the appropriate installations for our web applications are called.

**Automation of Sanitized Web Applications**

For our account creation web application, the tricky part came in with converting the first web application from a WAMP server to a LAMP server. For ease of use, part of our team worked on automating the creation of a cloud virtual machine on an Ubuntu 15.1 Linux machine. Because of this, it was necessary to convert the local web application to be functional on in a Linux environment. We accomplished this by creating a script that installed all the necessary configurations needed for a LAMP server then uploaded the appropriate php files to view. From there, we were able to call a selenium script that tested the creation of a user. As of this writing, there is an issue with communicating with the database, which the selenium script picks up on, but for all intents and purposes, we found it possible to automate the creation of a php/mysql server.

The one web application we were not able to automate its creation was the to-do list application. Despite being able to run its selenium tests from an automatically created virtual machine, we encountered several configuration issues along the way. While it is easy to create and host a web-application on a local virtual machine using React and Django, it is not as simple to host the web application on a cloud virtual machine in a testing capacity. One reason is that while in development mode, both framework’s servers are required to run simultaneously. This requires at least 3 command line interfaces to be open at one time.

Additionally, since no one is utilizing cloud virtual environments for testing, the only source of knowledge to rely on is how hosting is done in production. The common practice for hosting React pages on Google Cloud is in the App Engine instead of on a cloud instance virtual machine using an apache server. It is possible to run a Django application on a Linux server, utilizing apache to run the webhosting, but to configure both React and Django to work simultaneously in this environment will require more research into the resource documentation.

Finally, the firewall on the virtual machine must be appropriately configured to accept communication to the development server’s designated ports. Both issues are easily resolved in manual implementation, but when trying to automate for the sake of a sanitized web application for each test, this proved difficult to configure properly in a batch script.

**Security Testing of Sanitized Web Applications**

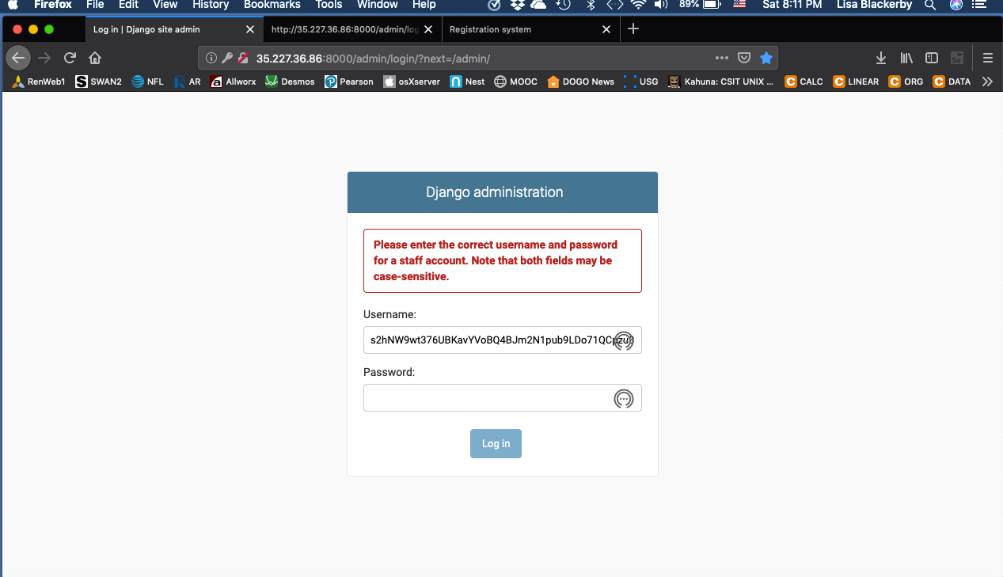
While we were not able to automate the security testing of each web application, our team made several attempts to break into both cloud hosted web applications using a series of SQL-injection attacks and a few XSS techniques to see if google cloud would allow such attacks to occur repeatedly on their network. First common logins and passwords were tested, then blind truth SQL statements. When both these failed, a Blind SQL Injection Technique was attempted. A Blind SQL Injection Technique is:

*“a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages but has not mitigated the code that is vulnerable to SQL injection.*

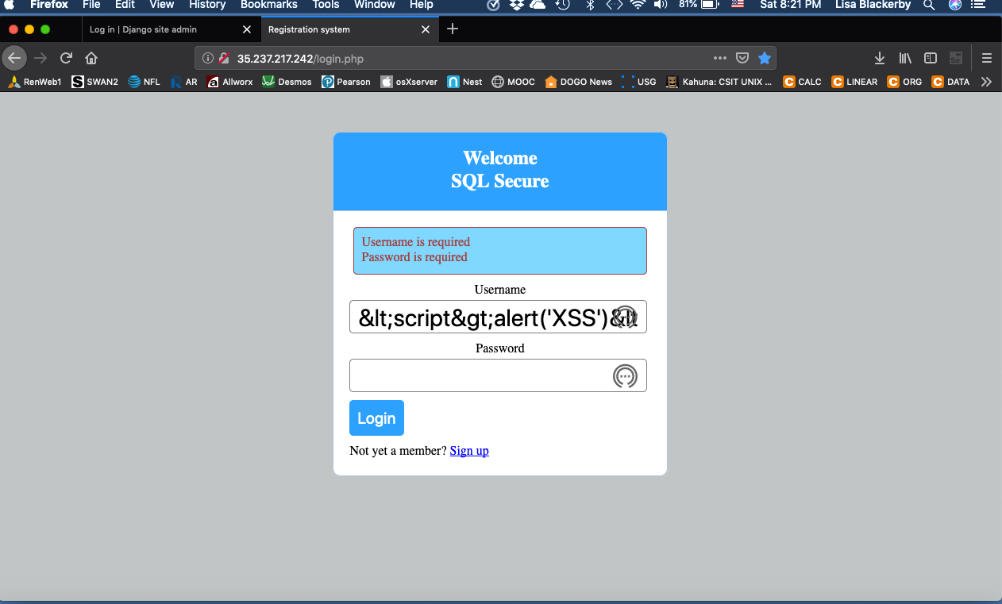
*When an attacker exploits SQL injection, sometimes the web application displays error messages from the database complaining that the SQL Query's syntax is incorrect. Blind SQL injection is nearly identical to normal* [*SQL Injection*](https://www.owasp.org/index.php/SQL_Injection)*, the only difference being the way the data is retrieved from the database. When the database does not output data to the web page, an attacker is forced to steal data by asking the database a series of true or false questions. This makes exploiting the SQL Injection vulnerability more difficult, but not impossible. “*

While we did not find success in breaking into the applications, we did find from monitoring our Django development server logs that the requests were getting through the cloud network to the local server and being denied at that point. We believe that with more time and knowledge on how to do varied attacks, successfully breaking into a web application could be possible.

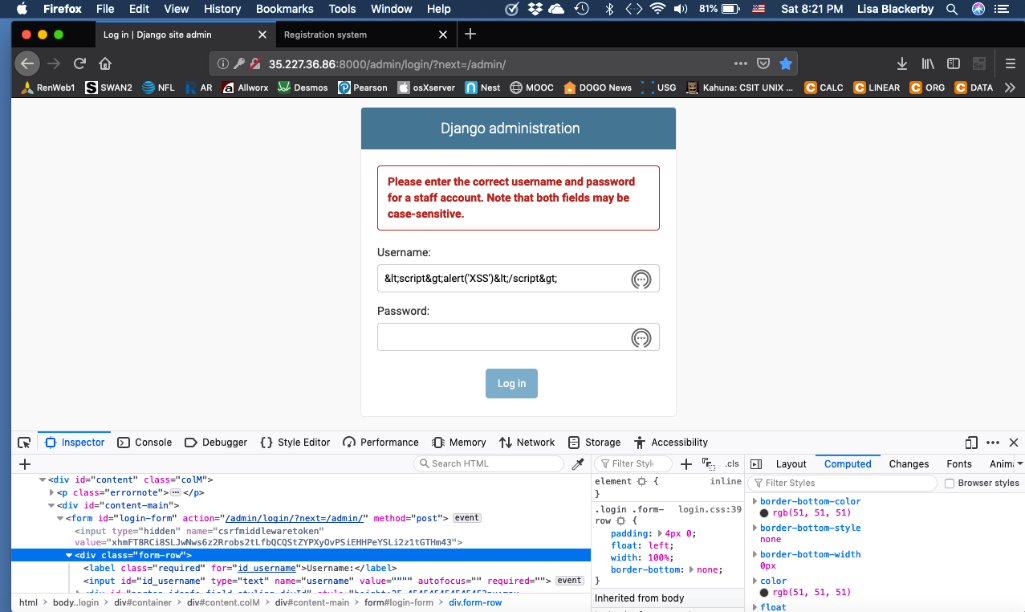
*Attempt to determine the database used in Django*



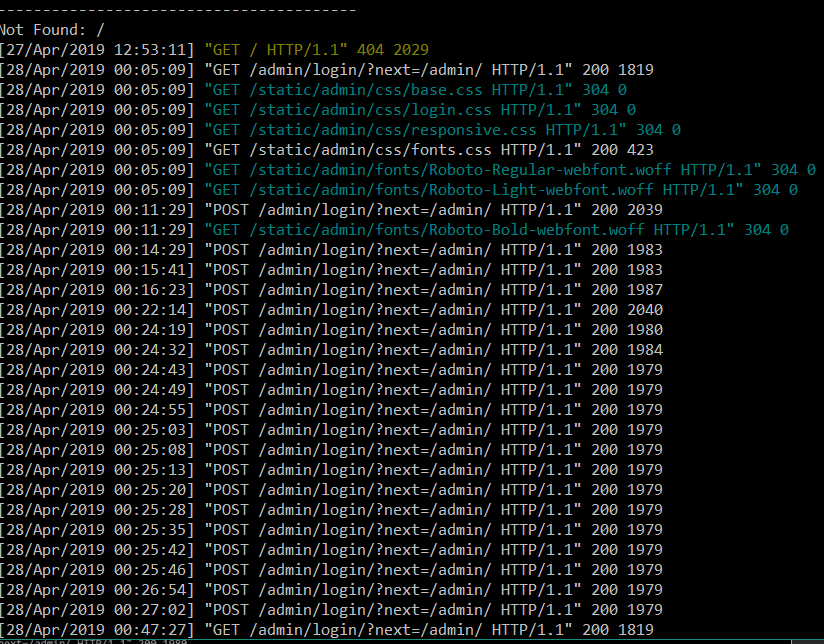
*Attempt at an XSS to occur on php/mySQL web application.*

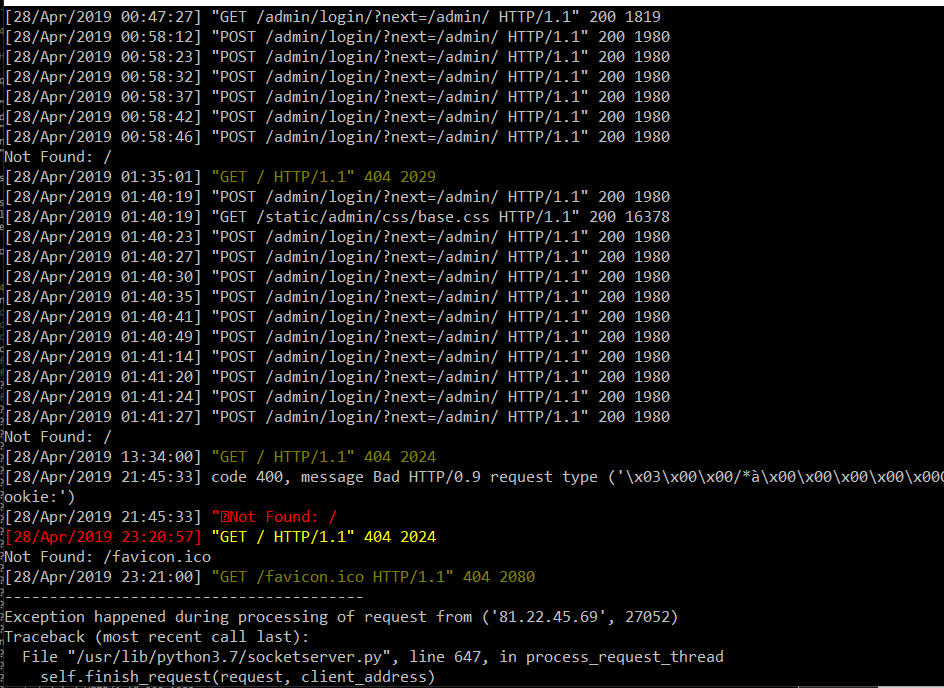


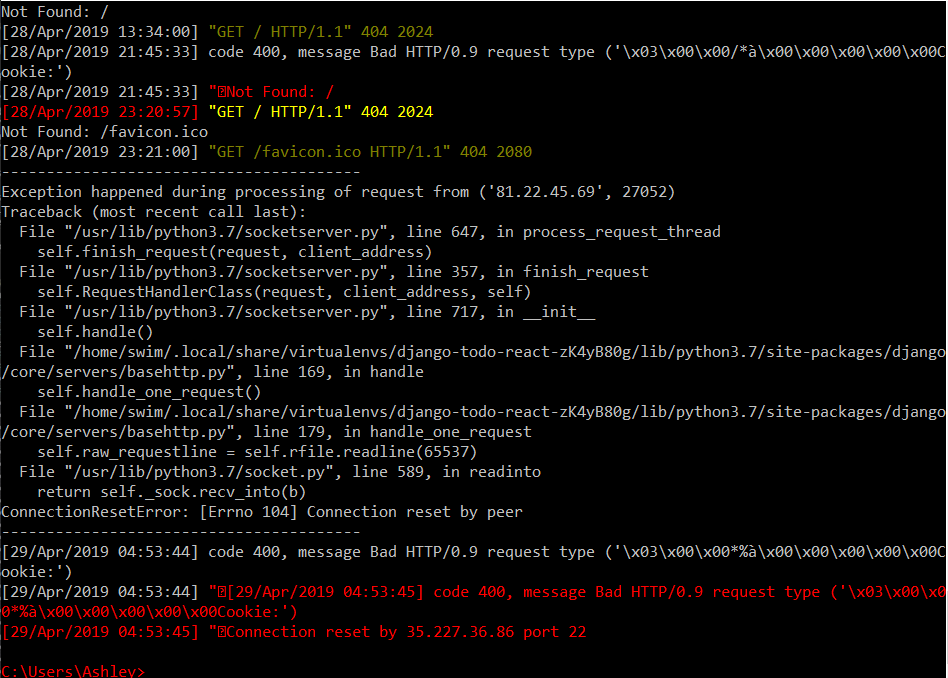
*Attempt at an XSS to occur on Django admin interface*



*The following is the Django server results from the truth injection and XSS tests:*







**Conclusion**

While we were unable to complete every objective to its fullest intent, we believe that we have provided a good set of groundwork to continue this research and that it would be possible to use cloud virtual machines to host sanitized web application for automation testing. We also conclude that it could be possible to create automated scripts that test the security of a web application, because the google cloud network does not pick up on these sorts of attacks. As far the automated creation of both web applications, we believe with continued research full automation is possible on a google cloud virtual instance. We believe our issue lied not in the scripting itself, but in the source code that grants permission to the databases and outside networks.

**Appendix**

**README for gcloud\_create\_vm**

|  |
| --- |
| gcloud\_create\_vm.sh operation & usage: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| 1.script requires name you want to give vm as an argument; |
|  |

|  |
| --- |
| 2.script can be reused by anyone ,but it requires you to rename PROJ variable (changing ZONE variable is optional); |
|  |

|  |
| --- |
| 3.after vm is created, the script tries to establish a connection to vm until one is made |
|  |

|  |
| --- |
| 4.in order to upload needed files to vm change the second to last line to the files you want to include |
|  |

5.the script runs uptime, curl, & whoami commands on vm to check functionality

**README for Local Host Setup**

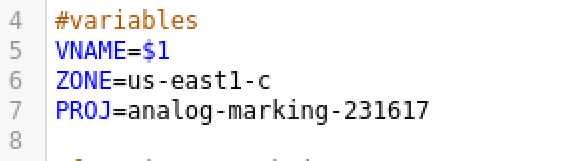
You’ll need to install python and Selenium for python.

This link has some instructions:

<https://www.youtube.com/watch?v=CriSHYMtg9M>

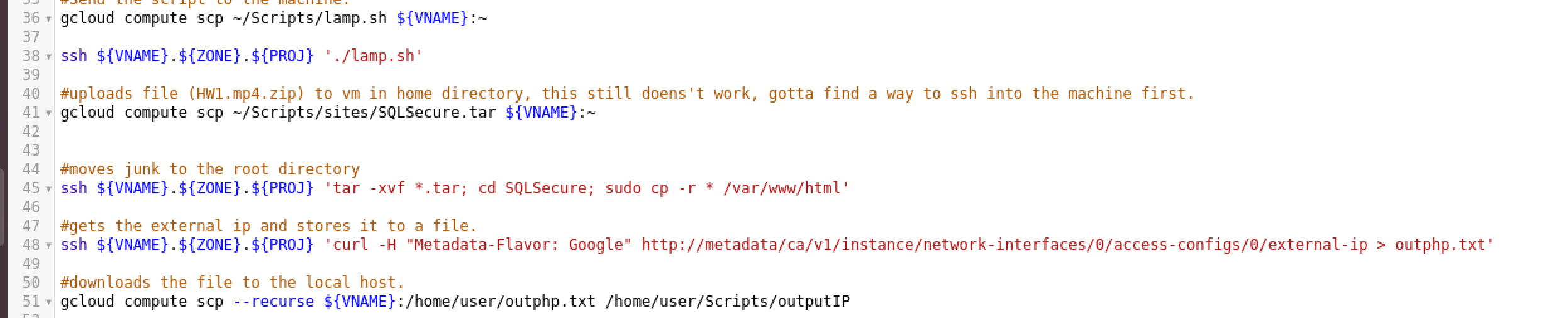
Configuring Scripts:

You’ll need to change the ZONE and PROJ variables to your preferences in each script. They are located near the top under #variables.



The directories managed on the host machine to require minimum alterations. You’ll need to change the home directory to your own in each script.

In the phpAuto.sh script you’ll want to manage the directories on line 36, 41, and 51.



On line 51, the first directory is on the cloud machine. You’ll want to change the username to your own username. The second directory is on your local machine, you’ll want to adjust that to match as well.

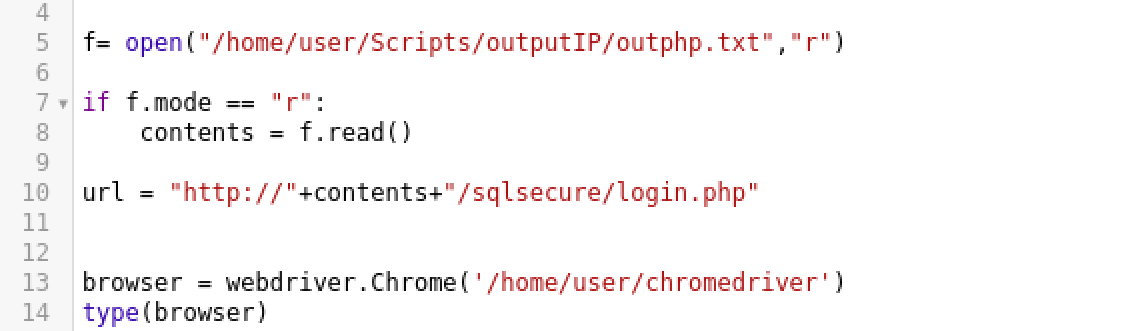
As of this writing there is no automation script for the React-django environment, our hope is to add it later.

You’ll want to change all the directories in CloudComputing.sh to match your home directories.



The directories for the python scripts will need to be altered as well.

In TestPhp.py you’ll want to handle the directory on line 5 and line 13 to your own home directories.



Normal HTML and bootstrap sites are easiest, that’s located in the thisworks.sh script and like the others, you’ll want to adjust the ZONE and PROJ variables along with the directories to match yours. Then writing a python script to test it would be simple.

**README for php/MySQL setup on cloud virtual instance**

Resource to manually set up can be found [here](https://gist.github.com/EmpireWorld/737fbb9f403d4dd66dee1364d866ba7e). Otherwise, use lamp.sh script.

**README for REACT/Django setup on cloud virtual instance**

**README for Security Testing**

The following SQL injections were attempted on both web application:

Attempted SQL injections - true statements, from the following list:

or 1=1  
or 1=1--  
or 1=1#  
or 1=1/\*  
admin' --  
admin' #  
admin'/\*  
admin' or '1'='1  
admin' or '1'='1'--  
admin' or '1'='1'#  
admin' or '1'='1'/\*  
admin'or 1=1 or ''='  
admin' or 1=1  
admin' or 1=1--  
admin' or 1=1#  
admin' or 1=1/\*  
admin') or ('1'='1  
admin') or ('1'='1'--  
admin') or ('1'='1'#  
admin') or ('1'='1'/\*  
admin') or '1'='1  
admin') or '1'='1'--  
admin') or '1'='1'#  
admin') or '1'='1'/\*  
1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055  
admin" --  
admin" #  
admin"/\*  
admin" or "1"="1  
admin" or "1"="1"--  
admin" or "1"="1"#  
admin" or "1"="1"/\*  
admin"or 1=1 or ""="  
admin" or 1=1  
admin" or 1=1--  
admin" or 1=1#  
admin" or 1=1/\*  
admin") or ("1"="1  
admin") or ("1"="1"--  
admin") or ("1"="1"#  
admin") or ("1"="1"/\*  
admin") or "1"="1  
admin") or "1"="1"--  
admin") or "1"="1"#  
admin") or "1"="1"/\*  
1234 " AND 1=0 UNION ALL SELECT "admin", "81dc9bdb52d04dc20036dbd8313ed055