**Black Widow: Final Report**

**The Problem**

The internet is infested with bad code. Not everyone is a security engineer or a professional software engineer, but everyone sure does have their opinion and post it freely on the internet. The problem with this is that search engines like google have no bias on whether an answer to a question is correct, as long as that page has the best search engine ranking it shows up first.

So let’s say I’m an engineer responsible for implementing encryption in my companies’ application. I’ve never used encryption but it’s ok I’m just going to google “Java Encryption” learn from there and use what I find to implement my own encryption.

Well turns out the majority of first few links in the “Java Encryption” google search turn up pages with vulnerable crypto implementations. This same problem propagates to a lot of similar issues with other programming languages and people giving you the advice to use functions that could have known vulnerabilities.

This entire project was very much inspired by the work of Manuel Egele and his presentation on his paper “An Empirical Study of the Cryptographic Misuse in Android Applications” where he pointed how bad this problem was with 88% of android apps misusing cryptography.

**The Solution**

This is where the inspiration to build Black Widow came about. Black Widow is a Service that consists of a Client and API. The client monitors what pages a user is on. If it detects that there is code or programming lingo being discussed it pings the API. The API will scan the page for known vulnerabilities and return any findings to the client. The client then displays this to the user.

The name Black Widow came about because we are trying to catch things in the web and there is no better analogy but that of a spider catching its prey in its web.

**Architecture**

**Front End**

The Client/Front end is rather simple. I built a Google Chrome Extension that communicates with black widow API and displays any results. It saves data into an SQLight database and attempts to be as invasive as possible only showing its self when a vulnerability is found. The beauty of Extensions is that they are all built with JavaScript and HTML making it as easy as making a web page to develop them. For the chrome extension, I have successfully implemented something called a content script and a popup script. But what is missing is the background script.

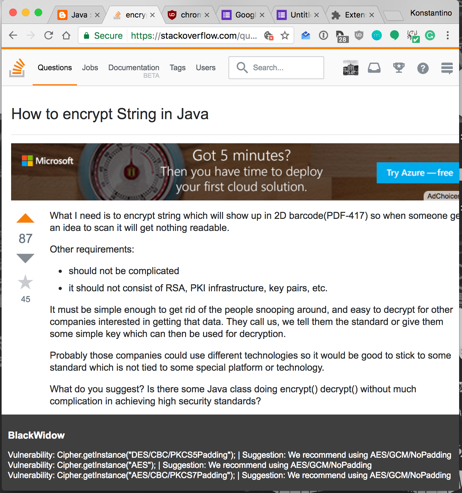
**Content Script**

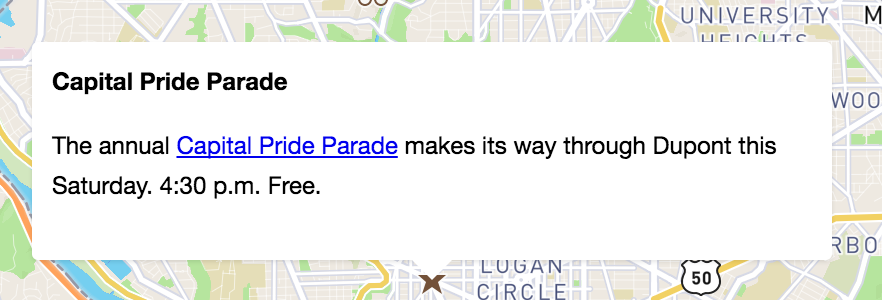
The content script runs JavaScript on each individual web page and allows you to modify each web page. So when you first visit a site the content script loads, and it goes ahead and checks the web page to see if there is code or programming lingo on the web page. If this is found only then does the client, ask the API to scan the page.

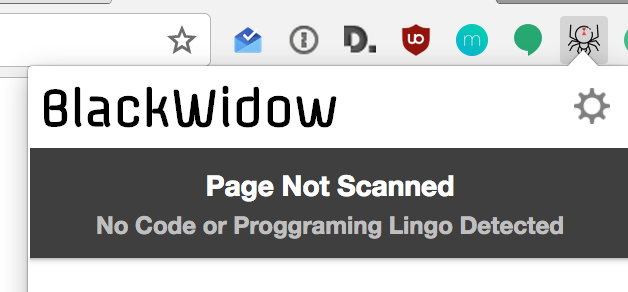
This is done in a very naïve manner where I simply convert the entire HTML page into a string and check If it contains keywords such as “Java”, “Python” or “<code>” which is the HTML element used by stack overflow to embed code on their web pages. This can and will be improved upon in the future as ensuring that a web page has code or possible vulnerabilities before pinging the API ensures that the service isn’t over flooded with costly API calls for no reason.

If code or programming lingo is detected, the API call is made using AJAX which is asynchronous and doesn’t cause any lag or noticeable delays to the user. Once the content script receives the information it is responsible for updating the state of the application, this is where a background script is needed but due to the time constraints of this project and a weird chrome API for communicating between these scripts this was not finished.

The content script is also responsible for highlighting the vulnerabilities in the page if some are found. This highlighting was actually the most difficult function to implement. I found a method to wrap a string with a div that would allow me to do this highlighting but certain web pages such a StackOverflow break strings up into pieces to ensure no XSS attacks can be executed. I will need to in the future implement a specific solution just for this. For the sake of the demo, I add a bar at the bottom of the screen with all vulnerabilities seen on the picture on the let but I would like to implement a red highlight and hover over solution often seen in which is on the picture to the right below:

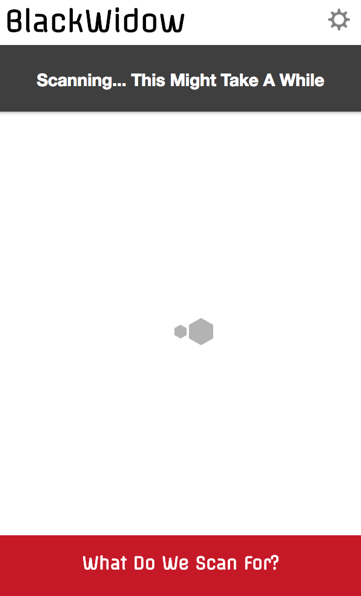




**Popup Script**

The popup script is what executes when you click the little chrome icon and a little popup window shows its self to the user.

This popup window has four states that matter to the user. The icon also changes according to the state to tell the user about it.

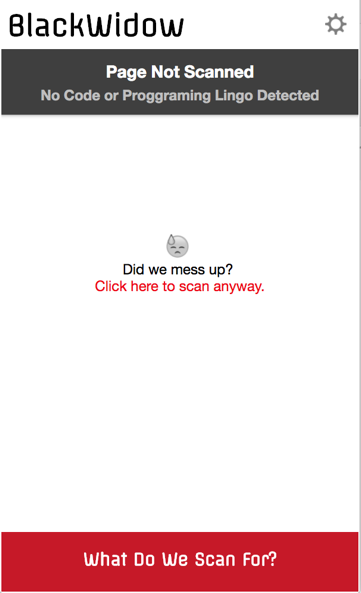
****

**Scanning state**

One of the states is the scanning state. When a user visits a page that has code in it, the client will ping the API. The API might have an answer if the page has previously been visited by another user. If not, once the service is at full scale and has a lot of modules, scanning may take a while so having a scanning state is key to show the users that the app is working. and the icon changes to show this:



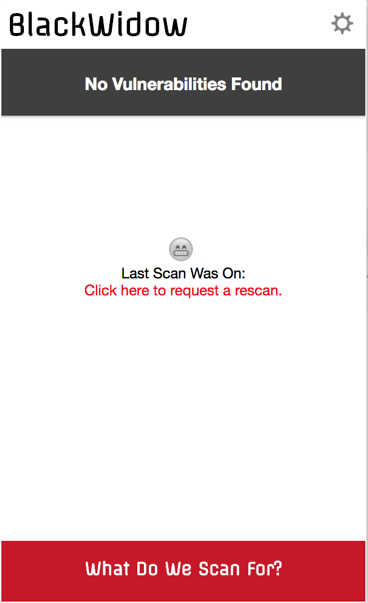
**Not Scanned State**

****Unless a user is actively developing they are very unlikely to visit a

Web page that contains code or coding advice. In such situations of

casual browsing there is no need to scan a web page. As discussed above this is done to ensure the server doesn’t get flooded with useless requests. But on the off chance that the detection software didn’t work properly we include a button to allow the users to force a scan to occur. In this state the icon is set to its default state:

/Users/ksparakis/Desktop/Screen Shot 2017-05-04 at 6.34.01 AM.png



**No Vulnerabilities Found State**

Given the situation that a page did have code, it was scanned and no vulnerabilities turned up, we have the no Vulnerabilities Found State. It shows the user when the last time a true scan occurred, just in case the webpage has been since updated they are allowed to request a Rescan. Also the icon is updated with a check mark to show that the page is good!



****

**Vulnerabilities Found State**

Given the situation where vulnerabilities are found on a webpage

We need to display this to the user. The popup activity will show

How many vulnerabilities were found and once it is fully implemented it will have a list describing each vulnerability.



**Future Work on the Front End**

**Missing a Background Script**

In order to fully sync the state of the application from the content script and popup. You need a background script. This script runs the entire time the chrome application is up. It has an api for communicating between popup and content script, and will need to be implemented in the future.

**“What Do We Scan For?” Button**

In all the images of the actual popup there is a “What Do We Scan For?” Button, this should link to a page with a description of every module and what it scans, allowing users to know what the application can and can not do. This page should be easily searchable but has not real use until more modules are created.

**Settings & User Management**

Eventually Since I am hoping to turn this into a product I would like to add user management to this. I plan to do so by leveraging Amazon Web services Congnito application which hosts and takes care of all the user management. The setting button should link to a page allowing users to change certain settings, login, logout and setup a payment method if that is the direction this application takes.

**Expanding to Other Web Browsers**

Because these web browser extensions are all built using, JavaScript and html, it is easy to remodel them a bit and create Mozilla, Safari and Internet Explorer versions. Of course each one of these extensions would have to be modified to account for each browsers API’s and slight differences but this type of work is not too difficult.

**Backend**

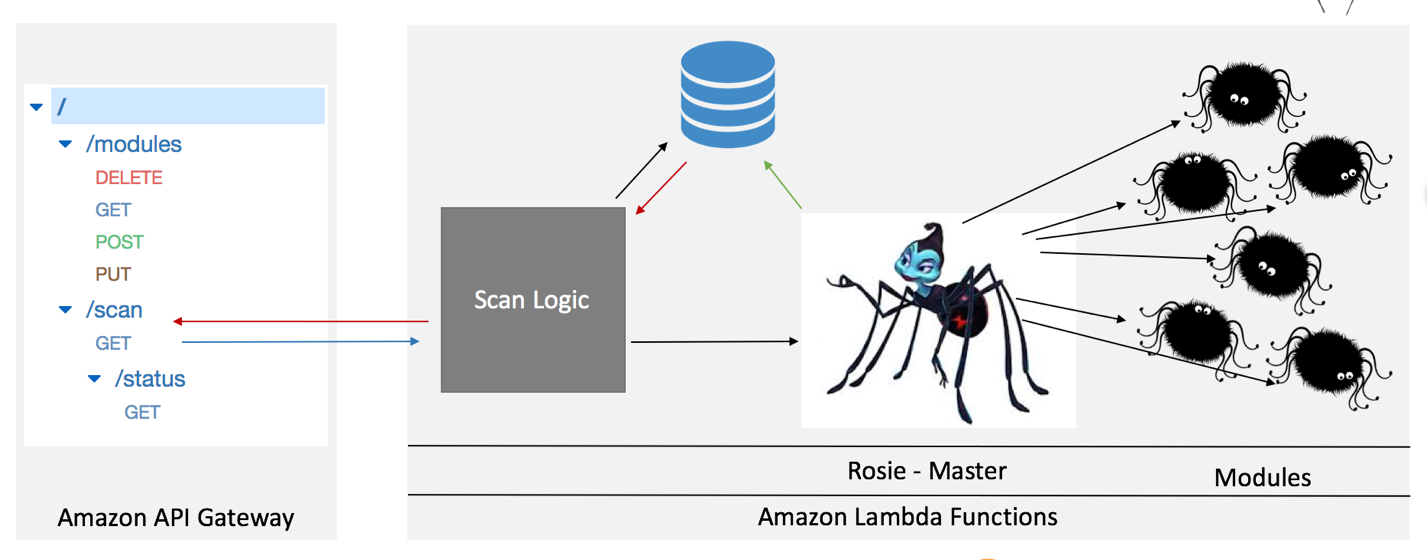
**AWS Cloud Deployment**

The backend is where all the magic happens with Black Widow. I decided to take a Serverless approach to this by using Amazon Web Services API Gateway and Lambda products. Lambda allows you to deploy a single function that takes in an input, be HTTP data or other inputs, and compute. Lambda allows you to specify the scalability of the function and they take care of everything you just worry about now much you want to spend. The API gateway allows you to link lambda functions in a RESTFUL API functionality which I use to deploy the Black Widow API.

By Serverless I do not mean there is are no servers running, these products simply allow you to not worry about scaling or managing servers, amazon takes care of that, and only worry about creating the application. I had been curious about this service but always assumed it was too expensive and just too good to be true. That is until I looked at a price analysis done by Andy Warzon [1.] an Amazon Engineer. comparing EC2 instance and Lambda, it turns out that until a specific breakeven point which is 295,000 function calls an hour then it is the equivalent of paying for an AWS M2.Large instance. But the issue is this M2.Large instance might not even be able to on its own handle 295,000 function calls an hour but the lambda services are fully scaled and can. The Lambda services are run you about 1/10th the cost especially at the small scale of 0-100 users this app will most likely have in its beginning stages and give you performance of a world class application.

With all this and my curiosity I was ready to try using these services and that is how I designed Black Widow to work. The obvious down fall here is that you experience vendor lock, with AWS and it would be extremely difficult to move away from AWS in future if you had a reason to.

Below is an image that describes the Back End Architecture:

****

**API & Database**

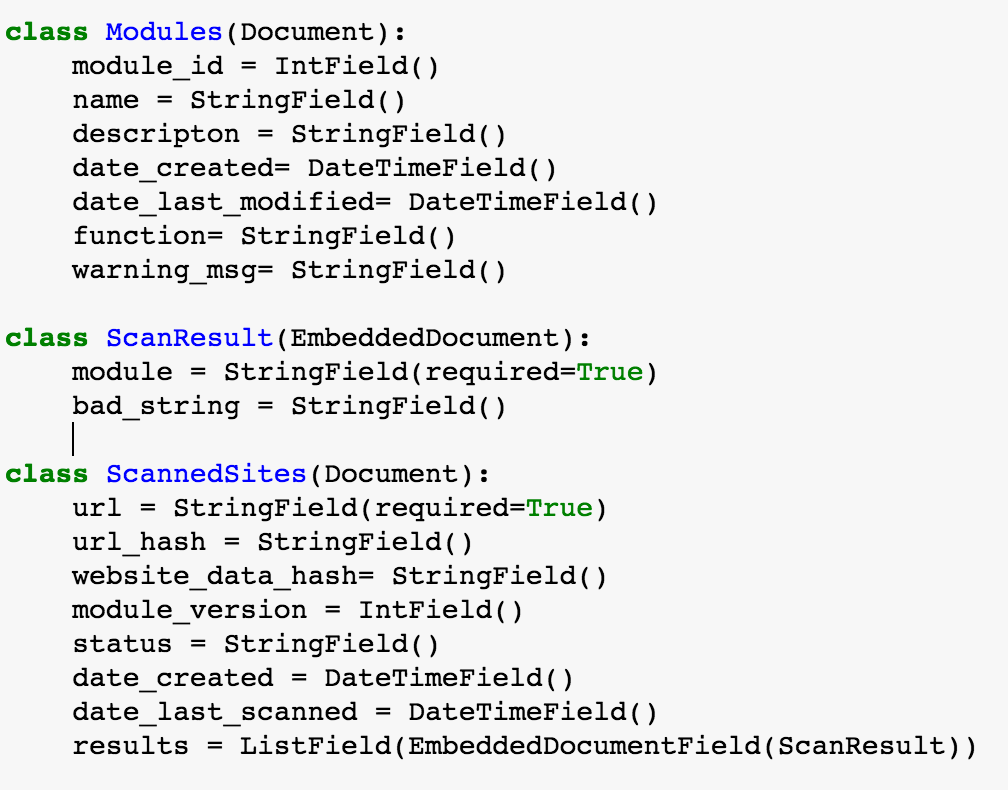
The bases of this API is the /scan function, this function should link to a scan logic function running in AWS which could potentially be built into Rosie the master function discussed below. This scan logic checks the database to see if the website has already scanned the information.

If a scan is executed once there are a lot of modules this could take some time to finish initiating so to match up the scanning state in the client they can call /scan/status to see if a scan is finished.

The modules section of the api is to add a dynamic functionality to adding and modify modules and is discussed in the section about modules below.

The database is a mango DB and the reason for this is because the san results of all the modules are better stored as a JSON string and there is no real need to a relational system. I did create a data model and implemented it using a MongoDB Engine which is an object relational python package for mongo. I have a Mongodb Database running on mlab.com which is free up to .5 GB. Unfortunately for the demo I skipped the database part since I only have one module and just rescan every time.

But here is the data model which needs further improvements in the future as it is very limited:

****

**Rosie The Master**

Rosie is named after the famous Bugs Life Black Widow Character. It is a Lambda Python Script that executes all the modules in an asynchronous manner allowing them to save their results to a database and run in a completely scalable manner as more and more modules are built.

Rosie is also in charge of making sure that if a new module is added or a module is edited that all websites in the database are rescanned.

**Modules**

The idea with the modules is to have a function that has one responsibility and it is to search for a vulnerability or issue in the html provided. All modules are to be built in python and the Html is provided via the beautiful soup library which is a powerful tool for parsing html. The modules once done executing execute a call back which saves whatever data into the database.

I am unsure as of now how to build these modules to be dynamic. I would like it be a system where any developer could just add their code via the API and it would be taken care of automatically in integrating a new module into the system. This could allow for Open Source contributions and so on. Does this mean making each module a lambda function on its own? This could be a possibility.

**Problems That Need to be Solved**

Along with not being sure how to manage all the modules, the biggest problem I think and have tried running my brain against. Is the issue of what do you do to monitor if a website has changed? Could you hash the website and periodically monitor this hash? But then any small change would trigger a rescan. If this application where to scale to have a lot of modules and a lot of users every rescan would become a costly endeavor.

I already for see one solution might be tagging modules to a specific language, and tagging a web page to that language as well and only scanning a webpage if the tags align.

**What Was I Able to Get Working?**

I definitely took a bite too big to chew and decided to use a lot of technologies outside of my comfort zone so I had a steep learning curve to push through to get things working. My goal was to create a working prototype that would scan a page, return that information to the user and display it. And given this goal I did manage to accomplish it.

I was able to successfully setup AWS’s Gateway API to point and launch an AWS Lambda Function. This for me was a very time consuming and big step. The API was definitely the simpler of the two to setup but learning how to deploy the service was a little tricky, but their process is very powerful. Their deployment model allows you to swap out API’s without interfering with customers using them. It also allows you to run multiple versions of your API’s as well.

Unfortunately for the sake of getting a prototype I had to ditch the whole asynchronous model for the modules and Rosie and chose to build this module directly into Rosie it’s self. Lambda was very difficult to figure out. I first setup a python jupyter notebook to test my module function locally first. Unfortunately for the time being I only implemented one module, and this module is used to identify improper use of Get.Instance() function in JavaScript.

**Module\_1**

The way module 1 works is for scanning of bad instances of Get.Instance(“”) which is used to initialize a cipher in Java. Online you often see bad implementations making the call as such:

*Get.Instance(“DES/CBC/PKCS5Padding”);*

DES is broken and there are a lot of other bad blockciphers and encryption modes being used commonly found in all types of examples online.

You also see a lot of instances where people use it as such:

*Get.Instance(“AES”);*

Although AES is secure, not including padding and encryption mode sets this up to automatically encrypt with ECB which through the penguin we know is bad at hiding repeating data.

So what module 1 does, is if it detects the misuse of this it will tell the user the string it detected, and the proper string it should be which I would argue using the strongest method is the best which is:

*Get.Instance(“AES/GCM/NoPadding”);*

**Conclusion**

This project was a massive undertaking but I managed to learn a lot of new technologies and the underlying problem I believe is a very important one that no one seems to be trying to solve. Is this the solution to fix everything? I can not fully answer that, could creating a wiki with proper implementations be a better solution? Potentially? But overall this project seems to have been a success with more future work to come. It’s advantage is that it taps into what developers are already used to doing and corrects them if anything seems wrong. Feel free to install the chrome extension, by opening up the developer tools in the chrome settings and linking the source code and try it!

**Extra Note**

I have even taken the time to go about and attempt pointing out these issues in a stack overflow post that I found is the first google result shown when googling “Java encryption.”

<https://stackoverflow.com/questions/1205135/how-to-encrypt-string-in-java/43779197#43779197>

**Have a great summer and thank you for everything!**

Sources

1. <https://www.trek10.com/blog/lambda-cost/>