Finding the Weak Spot of NPMJS Packages

CS224W Final Report Yoonji Shin

Abstract

Popular open source communities such as npmjs has network of packages with dependencies on other packages. As a result, manipulating a package has a cascading effect and impact a large part of the network of packages un-usable. In this study, we analyze the characteristics of npmjs network and identify vulnerabilities and experiment the impact of manipulating target packages. Network model for npmjs network was created using the dependency and devdependency lists for each packages and created Erdos-Renyi random network for comparison. Modified version of SIR model was used to identify the spread of impact and identified packages that can break 62.991% of the network after just 16 iterations. The least active package has highest chance of being unnoticed of modifications. A model for computing activity level based on the revision history of project wiki page is proposed and the package with the highest influence and least activity is proposed as the vulnerability target for the NPM package library.

1. Introduction

With the ever-increasing number of projects in the open source community, finding and selecting a quality library packages to used in the project could be a key to building a successful software. With the introduction of npm, the package manager for JavaScript that allows you to find, share, and reuse packages of code, hundreds of thousands of developers has contributed to create and share the packages that depend on each other.

Recently, there has been an incident where a developer broke Node, Babel, and thousands of projects with just 11 lines of Javascript. He removed a popular package called "left-pad," which pads out the left-hand-side of strings with zeroes or spaces, and with left-pad removed from NPM, widely used bits of open source infrastructure were unable to obtain the dependency, and thus fell over during development and deployment [1].

The complex layers of dependencies mean that dependencies are liability and changes to the underlying dependent packages can have propagating influence to the stability of other projects. Thus, understanding of the influence of the npm packages can help developers in choosing dependency packages for their project, in choosing the project to contribute on, and also in understanding the potential measure of risk for the open source community before making the propagating changes.

In this paper, we will be observing the characteristics and structure of the NPM package dependency network and identify the “weak spot” that can have a huge cascading effects on the network if the package was removed or manipulated from the network. The way that manipulated or removed packages can impact the entire npmjs network will be analyzed by using the following two models:

Part 1. Identifying weak area based on the influence spread model

1. **Modified SIR model of disease spreading**

```
Algorithm: Pseudo-code for simulating the modified SIR model on npmjs graph
Input: initial set of infected nodes I
S ← V\I // susceptible nodes
while I≠∅ do
    foreach node u ∈ V do
        if u∈S then
            foreach (u, v) ∈ E with v ∈ I do
                S ← S\{u}, I ← I ∪ {u}, and break for loop
```

In the traditional SIR model, every node can be either susceptible, infected, or recovered and Every infected neighbor of a susceptible node infects the susceptible node with probability β and infected nodes can recover with probability δ and the links between nodes are bidirectional—if two nodes are linked, both nodes can influence each other.

In our npmjs network, each npm package is a node and the links between nodes are directional—that is, if package A has listed package B as its dependency, package B can influence package A but not the other way around unless package A, too, lists package B as its dependency. Also, if any of the listed dependency is manipulated or removed from the network, the original package has $\beta=1$ and $\delta=0$ since impacted package cannot recover itself and if any of the dependency is missing, the package is not valid anymore. We will be comparing the npmjs graph with Erdos-Renyi random graph with the same number of nodes and links on the proportion infected for each graph, proportion of trials that result in epidemics, number of iterations to reach epidemics using pairwise Chi-Squares test and Mann-Whitney U test.

2. **Applied Threshold model**

In the traditional threshold model, each individual i has a threshold $t"$ that determines her behavior in the following way. If there are at least $t"$ individuals that are rioting, then she will join the riot, otherwise she stays inactive. To create a model, we will ignore the specifics of the dependency packages and only consider the number of listed dependent packages as the threshold for the package to join the riot—in our case, to become a non-functional package. We will then compute the average threshold for the npmjs network and create a random network with the same number of nodes, same number of directed edges, and the same average threshold that is following the standard distribution for the threshold and simulate the impact by having initial manipulated/removed nodes and compare the result with npmjs network.

We then identify the "weak areas" based on these categories and analyze and compare the impact of manipulating such nodes:

- Node with the highest degree centrality
- Top 10 nodes with the highest degree centrality
- Top 10 nodes with the highest number of downloads

Part 2. Identifying the target package for attack based on its activity level on the project wiki page revision history

The least active package has highest chance of being unnoticed of modifications. When a package is not updated for a prolonged time and when there is little activity on the project Wikipedia page for modification of the project, it is likely that less people are focusing on the NPM package and higher chance of an attack being unnoticed for longer period of time. A model for computing activity level based on the revision history of project wiki page is proposed and the package with the highest influence and least activity is proposed as the vulnerability target for the NPM package library. Past 2 years worth of revision history is collected by crawling the wiki page of the NPM package library.

To compute the activity level, the following criteria have been considered

- Frequency of modification.
- Amount of modification in relation to the total document size.
- Impactful contribution—that is long lasting and not getting overwritten. Any changes that was overwritten within 1 day is ignored. If part of the change was removed or overwritten within 1 day, only the remaining part and newly modified part is considered for calculation.

2. Prior Work

2.1. Link-based classification

The authors proposed a framework for modeling a link based model that supports discriminative models describing both the link distributions and the attributes of linked objects which means, it can capture and predict the entity categories using both the entity information itself and the link information. They used mode-link, count-link, and the binary link where the mode-link is simply taking the mode of its neighbor categories, the count-link counts the number of neighbors with that category, and the binary-link checks whether the entity has a neighbor with that category to achieve a significantly higher F1 scores than the traditional content-only model.

However, the framework model that authors created is based on the assumption that each entity is independent and besides from the explicitly noted linkages, the content of the entity. For our study, we will be building the link based classification framework as suggested by the authors, but with the following modification—while the authors treated the linkage between the entities as un- weighted, we will be creating npmjs network model with weighted, directional linkages. We believe that assigning different weight for each link based on the download count will provide more accurate representation in the impact analysis. Also, since the exhaustive iterative algorithm is too costly and inefficient to run on our large data set consists of 383413 packages and 899668, we have optimized the algorithm for creating the link based model of the network.

3. Data

3.1. Dataset

We have downloaded the snapshot of entire npmjs library as of November 2016. The npmjs library data consists of 383413 packages. Out of the total 383413 packages, there were 28493 unreachable

packages—2547 of which failed in reading the package and 25946 of them failed to find the latest version. We suspect that this is most likely caused by a bug in indexing in npmjs library database. For our modeling of network, we only used the 354919 packages that was readable and had the link for the latest version

3.2. Data Processing

https://registry.npmjs.org/-/all - returns a JSON document containing the set of all packages. Each package information consists of the name, author, url, description, tags, license, bugs, users, repository, versions, and timestamp. We trimmed the unnecessary additional information and for each package, we retrieved additional metadata consisting of the version information, dependency package information and devdependency package information. Dependency package identifies the npm packages that are necessary for the target package to run the package, and devdependency package lists the npm packages that are not necessary for developing and building the package.

We then create the following model networks for npmjs:

1. A model networks for npmjs by using each package as a node and adding a directed edge for each of the packages listed as dependency
2. A model network for npmjs by using each package as a node and by adding a directed edge for both the packages listed under dependency and packages listed as devdependency
3. A model npmjs by using each package as a node and by adding a directed edge for both the packages listed under dependency and packages listed as devdependency and having each edge weighed based on the number of downloads for the package the edge is directing to.

3.3. Data Statistics

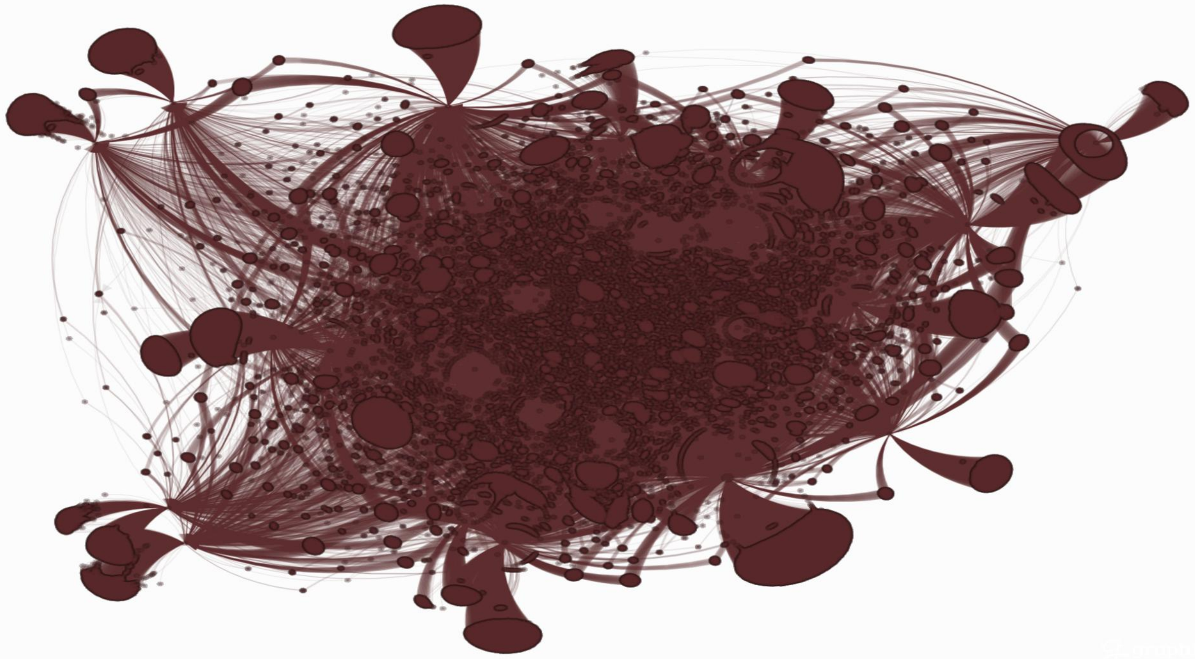
npmjs library data statistics

Total Packages	383,413
Total Valid Packages	354,919
Total Dependency Packages (Edge Count for Dependency Graph)	899,668
Total DevDependency Packages (Edge Count for DevDependency Graph)	2,289,960

4. Experiments & Analysis

4.1. Creating network models

Network model for npmjs network was created using the dependency lists for each packages and created. Before processing, all non-valid NPM packages (missing download link for the package, dangling endpoint, etc.) has been removed. Each NPM package is modeled as nodes and dependency is deemed to be a directed edge from the package that uses (depends on) another package and equal weight has been given.



[Figure 1. Visualization of NPMJS network graph]

To compare the result of computation, Erdos-Renyi random network for comparison with the same number of nodes and directed edges where the directed edges were randomly assigned.

npmjs network graph statistics		***random network graph statistics***	
Total Node Count	354.919	*Total Node Count*	354.919
Total Directed Edge Count	899,668	*Total Directed Edge Count*	899,668
Clustering Coefficint	.0330321	*Clustering Coefficint*	4.296e-06

4.2. Identifying Weak Area--Infecting the nodes with the highest degree centrality on Modified SIR model of disease spreading

From the npmjs network model, top 10 nodes with the highest degree centrality were selected and modified SIR model of disease spreading simulation was performed. Also, same simulation was performed on the Erdos-Renyi random graph for comparison.

Top 10 packages with the highest degree centrality			
npmjs network		Erdos-Renyi random network	
lodash	0.0732554	*31339*	4.4742e-05
request	0.0446074	*255973*	4.4742e-05
async	0.0408829	*279054*	4.4742e-05
underscore	0.0337939	*15332*	4.21101e-05
express	*0.0298556*	*16321*	4.21101e-05
commander	*0.0293262*	*20513*	4.21101e-05
chalk	*0.029227*	*33677*	4.21101e-05
debug	*0.0261233*	*67643*	4.21101e-05
bluebird	*0.023956*	*90513*	4.21101e-05
mkdirp	*0.0191935*	*180866*	4.21101e-05

On the npmjs network, the node with the highest degree of centrality was ‘lodash’ package with the degree of centrality: 0.0732554015002. As shown on the figure 2, After 16 iterations, it has infected 62.991% of the total npmjs network, and there was no change in the infected nodes indicating that the node was a part of a disjoint cluster and it has successfully infected all nodes in the cluster that the package belonged to.

```
     [Manipulating ‘lodash’ package on npmjs network - S: Susceptible, I: Infected, R: Recovered]
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 1, 'S': 383412, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIII    {'I': 28088, 'S': 355325, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIII    {'I': 75381, 'S': 308032, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 216622, 'S': 166791, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 236323, 'S': 147090, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 240367, 'S': 143046, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241175, 'S': 142238, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241374, 'S': 142039, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241464, 'S': 141949, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241483, 'S': 141930, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241496, 'S': 141917, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241505, 'S': 141908, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241506, 'S': 141907, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241509, 'S': 141904, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241515, 'S': 141898, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241516, 'S': 141897, 'R': 0}
```

The results were remarkably similar among the 10 packages with the highest degree of centrality, as the iteration made no further infections after 15 iterations and while there was minimal difference in spread of the infection, the total number of final infected nodes was 241,516 nodes for all 10 packages, indicating that all 10 packages with highest degree centrality were included in a disjoint cluster composed of 241,546 nodes (62.991% of the npmjs network). We also experimented with manipulating all of the top 10 packages with the highest degree centrality instead of removing just one package, and the result was identical to removing just the ‘lodash’ package which has the highest degree centrality.

```
      [Manipulating Erdos-Renyi random network - S: Susceptible, I: Infected, R: Recovered]
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 1, 'S': 379956, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 18, 'S': 379939, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 101, 'S': 379856, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 489, 'S': 379468, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 2311, 'S': 377646, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSII    {'I': 10649, 'S': 369308, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIII    {'I': 46723, 'S': 333234, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 166941, 'S': 213016, 'R': 0}
SSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 333816, 'S': 46141, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 377011, 'S': 2946, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379691, 'S': 266, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379815, 'S': 142, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379818, 'S': 139, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379820, 'S': 137, 'R': 0}
```

We ran the same experiment on Erdos-Renyi random network. When manipulating the package with the highest degree centrality, Compared to the npmjs network, the initial spread of the infection was slower as shown on above, however, unlike npmjs network which stopped the spread after 62.991% infection, the final infection rate was 99.999% after 14 iterations. 137 nodes didn’t get infected in the end. This result indicates that compared to the npmjs graph which had a large disjoint central cluster, Erdos-Renyi random network had only small disjoint clusters totaling 137 nodes and the links were sparse.

4.3. Identifying Weak Area--Infecting the nodes with the highest closeness centrality on Modified SIR model of disease spreading

From the npmjs network model, top 10 nodes with the highest closeness centrality were selected and modified SIR model of disease spreading simulation was performed. Also, same simulation was performed on the Erdos-Renyi random graph for comparison.

Top 10 packages with the highest closeness centrality			
npmjs network		Erdos-Renyi random network	
merino	0.26664	*31339*	4.4742e-05
big-bertha	0.266068	*255973*	4.4742e-05
generator-arc	0.240286	*279054*	4.4742e-05
web-component-tester	0.233	*15332*	4.21101e-05
async	0.231248	*16321*	4.21101e-05
esbox	0.226092	*20513*	4.21101e-05
neyka	0.222979	*33677*	4.21101e-05
zero.lib	0.22086	*67643*	4.21101e-05
yeoman-generator	0.219042	*90513*	4.21101e-05
kibana-with-account	0.218213	*180866*	4.21101e-05

On the npmjs network, the node with the highest closeness centrality was ‘merino package with the closeness centrality: 0.26664. We ran the modified SIR model for disease spreading.

```
      [Manipulating 'merino' package on npmjs network - S: Susceptible, I: Infected, R: Recovered]
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSI    {'I': 7369, 'S': 376044, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIII    {'I': 42681, 'S': 340732, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 209653, 'S': 173760, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 235111, 'S': 148302, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 240183, 'S': 143230, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241150, 'S': 142263, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241358, 'S': 142055, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241453, 'S': 141960, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241482, 'S': 141931, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241496, 'S': 141917, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241505, 'S': 141908, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241506, 'S': 141907, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241509, 'S': 141904, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241515, 'S': 141898, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241516, 'S': 141897, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 241516, 'S': 141897, 'R': 0}
```

As shown above, after 16 iterations, it has infected 62.991% of the total npmjs network, and there was no change in the infected nodes indicating that the node was a part of a disjoint cluster and it has successfully infected all nodes in the cluster that the package belonged to.

```
      [Manipulating Erdos-Renyi random network - S: Susceptible, I: Infected, R: Recovered]
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 1, 'S': 379956, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 18, 'S': 379939, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 101, 'S': 379856, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 489, 'S': 379468, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS    {'I': 2311, 'S': 377646, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSII    {'I': 10649, 'S': 369308, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIII    {'I': 46723, 'S': 333234, 'R': 0}
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 166941, 'S': 213016, 'R': 0}
SSSSSSSIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 333816, 'S': 46141, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 377011, 'S': 2946, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379691, 'S': 266, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379815, 'S': 142, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379818, 'S': 139, 'R': 0}
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII    {'I': 379820, 'S': 137, 'R': 0}
```

Again, the same experiment was run on Erdos-Renyi random network. When manipulating the package with the highest closeness centrality, Compared to the npmjs network, the initial spread of the infection

was slower as shown above, however, unlike npmjs network which stopped the spread after 62.991% infection, the final infection rate was 99.999% after 14 iterations. There were 137 nodes that didn't get infected in the end. This result indicates that compared to the npmjs graph which had a large disjoint central cluster, Erdos-Renyi random network had only small disjoint clusters totaling 137 nodes and the links were sparse.

As shown above, all top 10 packages from the highest degree centrality and highest closeness centrality had a very close result – all packages succeeded in infecting 62.991% of nodes within 16 to 18 iterations and then no more infection happened. On the other hand, for the Erdos-Renyi random graph, the initial spread was slow, but within 15~18 iterations, it successfully infected over 99% of nodes in the graph. This indicates that NPMJS network has highly dense disjoint cluster with 379820 nodes. The result matches with the visual representation of the NPMJS network graph in part 1.

4.4. Identifying the target page for attack based on its activity level on the project wiki page revision history

From the above experiment, top 8 packages (which completed infecting 62.991% of nodes within 16 iterations) have been marked as the "weak area" of npmjs library. Past 2 years worth of revision history is collected for each of the packages in the "weak area" by crawling the wiki page of the NPM package library. Wikipedia provides an open API for accessing the full content of the page and the full history of revision data. It follows standard REST API format for data retrieval and outputs a standard XML file.

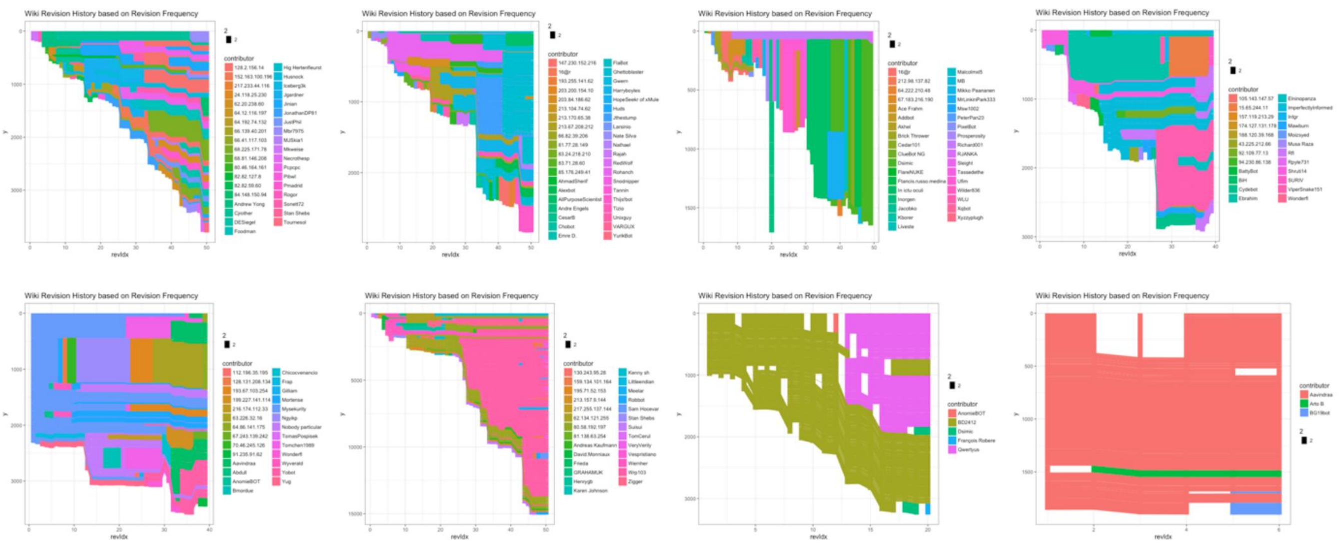
For this package, the following data fields are included (per each revision history): timestamp, username, content. The format of REST API call is:
http://en.wikipedia.org/w/api.php?action=query&prop=revisions&rvprop=timestamp|user|
content&rvlimit=max&format=xml&rvdir=newer&titles=TITLE_OF_THE_PAGE

In order to process the collected data, WikiSaxHandler is used to process and iterate through the large XML file to retrieve necessary data and build the data frames without creating the full parse tree of XML which would be much more time demanding. Some of the wikipedia revision history files can run as large as multi-megabytes of data especially if it is one of the popular and lengthy page that gets frequently revised, thus this was a necessary choice.

R project was written for handling the crawled Wikipedia revision history data and computing the activity level. Once the data is stored in the data frame, revision version comparison is made. The sentences are stripped to remove the whitespace or punctuations, and the previous version's sentence is compared to the newer version's sentence to mark the changes. If any part of the sentence has been modified by more than 1 word, the entire sentence will be marked as modified, and the new contributor will be the owner of the sentence segment.

$$activity\ level = \left| \sum \frac{\#of\ sentences\ modified}{\#\ of\ total\ sentences} * \log\left(\frac{modification\ age}{365 * 2}\right) \right|$$

Above model was used to compute the activity level and the weak area npmjs packages were sorted in ascending order to find the package with lowest visibility.



- x_0: previous revision index
- x_1: current revision index
- y_0: previous location of the revised sentence location by the same contributor. If does not exist, then the location of the newly contributed sentence (in case of new contributor)
- y_1: y_0 + amount of contribution (text length)

Above figure is the visualization of the revision history for the 8 npmjs packages in the weak area where x axis indicates the revision index and each legend indicates different contributor(user) and y is the modified content (sentences) in descending activity level order (left-top figure has the highest activity level and right-bottom figure has the lowest activity level).

The activity level ranged from 0.316017 with only 3 revision history to 4.91089 with 42 revision history in the last 2 years. mkdirp is has the lowest activity level of 3 revisions in the past 2 years with the activity level of 0.316017 and thus "weak spot" with highest influence in the npmjs library.

5. Conclusion

We have observed that npmjs library network has a large cluster which occupies 62.991% of the total network with high clustering coefficient. Even though this is singly directed network, we observed that removing one of the top 10 package with high degree of centrality from the cluster can effectively make 241,546 nodes (62.991% of the npmjs network) impossible to run. If one is targeting an attack, mkdirp is the most vulnerable package as this has 0.019 degree centrality. It can shut down 62.991% of the network after 16 iterations. At the same time, there has been only 3 modifications on the package wiki page resulting in the lowest activity level of 0.316017 which indicates low visibility for the package. Therefore, mkdirp package is the package with a very low visibility and high influence on the npmjs network and is ideal candidate for vulnerability.

6. Future Study

For this study, we only collected data for the latest version of npmjs packages and ignored the versions for packages listed in the dependency and devdependency. However, an author of a package can choose to

- Not specify any version for the package, which will default to the latest available package
- Specify only the major version of the package
- Specify the major version and minor version of the package

This indicates that if there is a package that was popular in the past and became less popular in the later versions, we may observe different behavior on impact of manipulating the latest version versus previous versions depending on the trend of how other package owners specify the dependency package version.

Furthermore, only the “revision history” for the wiki page was considered in computing the activity level of the package but the “view count trend” can also indicate how the package is actively being watched and will be a good addition to improve accuracy of the activity level.

7. References

[1] C. Williams, How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript. TheRegister. 2016 <http://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/>

[2] Lu, Q. & Getoor, L. (2003). Link-based classification. Proceedings of the Twentieth International Conference on Machine Learning.

[3] Abraham D Flaxman, Alan M Frieze, and Juan Vera. A geometric preferential attachment model of networks. Internet Mathematics, 3(2):187–205, 2006.

[4] Yang-Yu Liu, Jean-Jacques Slotine, and Albert-Laszlo Barabasi. Controllability of complex networks. Nature, 473(7346):167–173, 2011

[5] W. Chen, Y. Yuan, L. Zhang. Scalable Influence Maximization in Social Networks under the Linear Threshold Model. In Proc. ICDM, 2010.

[6] A. Goyal, W. Lu, L. S.V. Lakshmanan. SIMPATH: An Efficient Algorithm for Influence Maximization under the Linear Threshold Model. In Proc. ICDM, 2011.