naev

Utilizing functional equivalence to establish a security friendly nonlinear versioning system

Thomas Quig and Zach Oldham

Overview

Background

Problem Statement

naev overview

Automatic version selection

Interface-based Equivalence

Test-based Equivalence

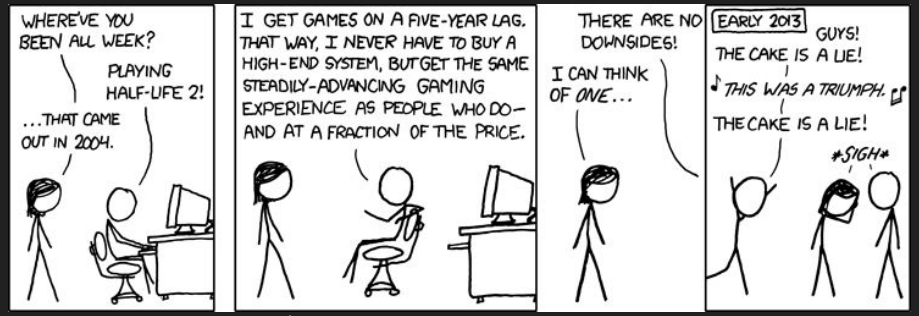
Background - Package Managers

- Help you quickly install packages to assist in development
- Three types
  - System-level (apt), Language-level (npm), Hybrid (pip)

- npm
  - Node Package Manager
  - node.js javascript framework (and packages)

- Issues with npm
  - package-lock.json

Background - Technical Lag



- Delaying updating your packages
  - Many reasons people do so (cost too great, use not enough)
- Bugs vs Vulnerabilities
  - Bugs can be avoided
    - Just don't reach the buggy condition
  - Vulnerabilities often cannot be avoided
- Patches often go uninstalled, which can have a cascading effect*
  - Dependencies
- Patch Racing**

* Small World with High Risks (Zimmerman et al.)

** Windows of Vulnerability (Arbaugh et al.)

Background - Semantic Versioning

A.B.C

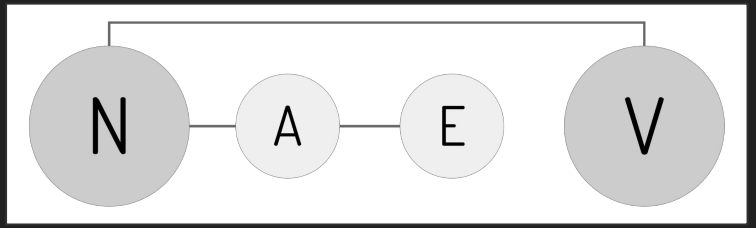
- Major Versions (A)
  - Typically releases that will break legacy code
- Minor Versions (B)
  - May break legacy code, may not.
  - If major versions are a BIG deal (Python), then minor versions will often also break code
- Patch Versions
  - Sometimes fit in with minor versions, **should not** break code
- Breaking code
  - What defines broken?
- Nonadherence

Problem Statement

Due to semantic versioning nonadherence, users often hesitate to install patch versions of code as they fear breaking changes. The technical lag created by this nonadherence creates large, rippling windows of vulnerability within vulnerable packages and their dependencies.

There exists no standardized method of retroactively inserting patches into old versions of code such that developers can ensure that users will adopt the changes, and users can ensure that the changes will not break their code.

Our Solution - NAEV



- Nonlinear Automatic Equivalence Verification

“Make sure it will still work before you update to it”

- Two components
  - Automatic Version Selection
  - Functional Equivalence Verification

- The bulk of the work was to establish an abstraction
  - Implementation was to show proof of concept and effectiveness.

Check out https://naev.page (“I love branding” -Thomas)

The issue with Traditional Package Versioning

Once a version is published, there is no way to change it

- NPM allows arbitrary versions, but it’s not encouraged

Bugs in older versions usually remain

- Only fixed in newer versions after discovery

For security issues, this is a compounding effect

- Many users use older versions

Nonlinear Package Versioning

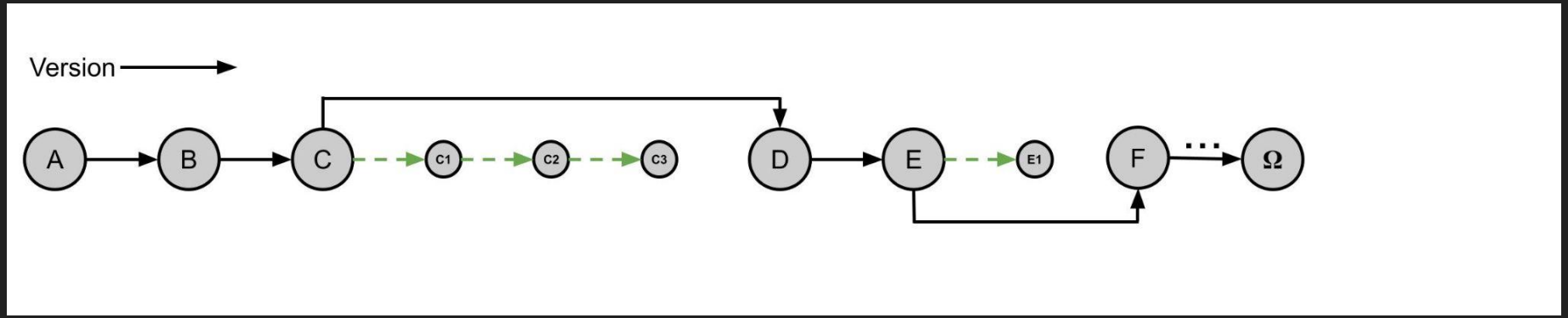
Allow patch versions to be published easily **without** it overwriting current versions.

The 20th release is no longer the 20th version.

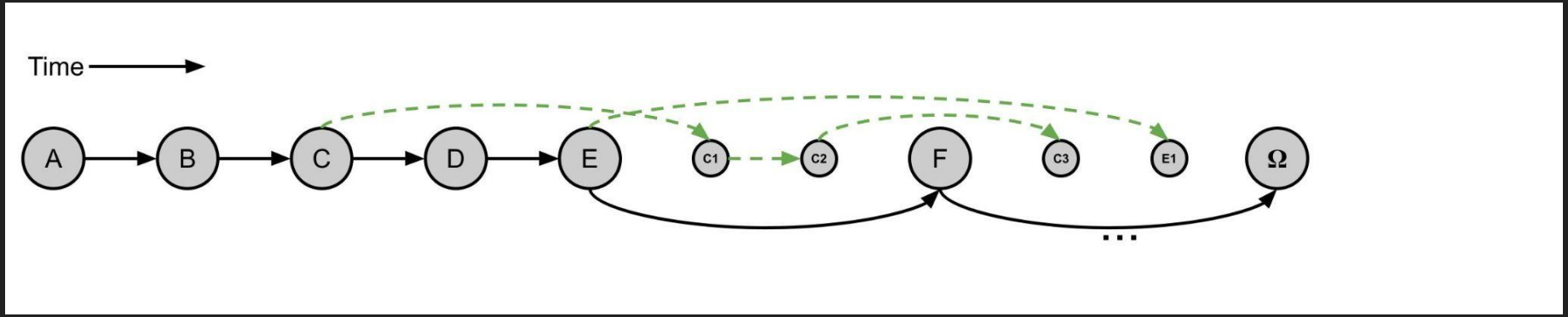
You can bodge this on npm with tags.

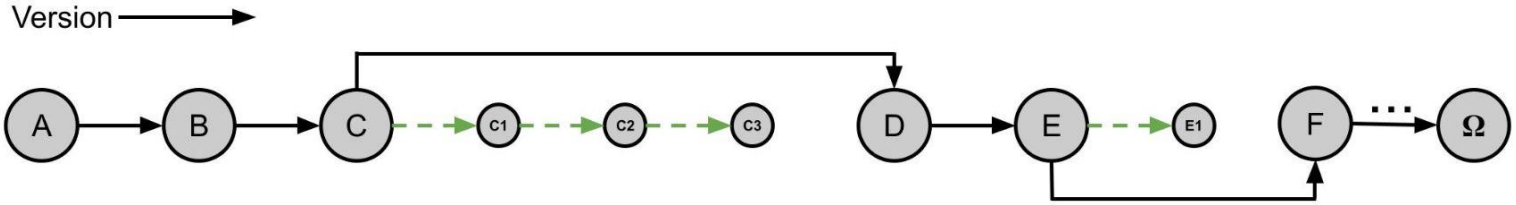
You can do this on github with branches

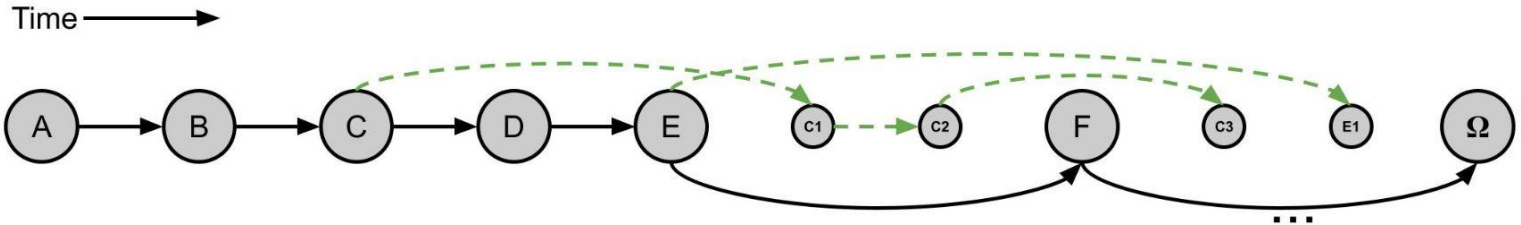
NAEV releases with respect to VERSION

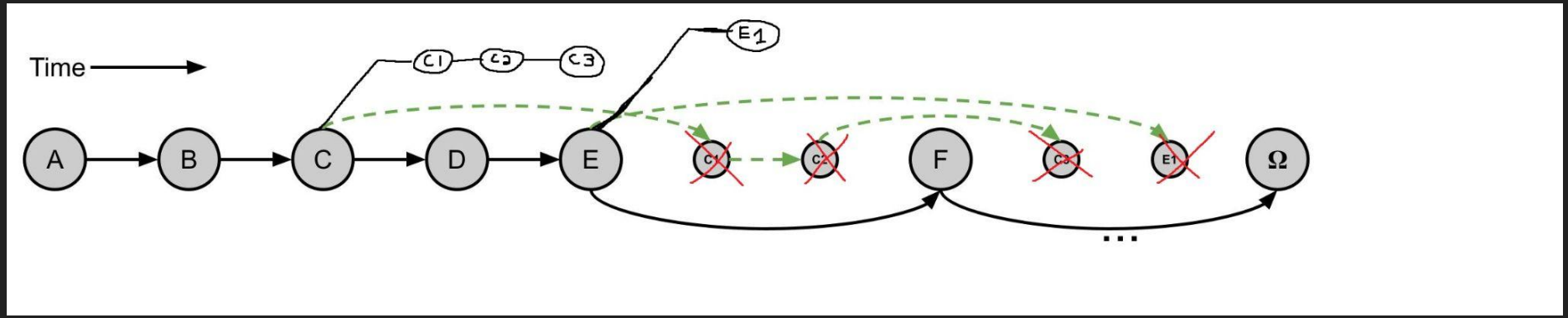


NAEV releases with respect to time




Version
A
B
C
C1
C2
C3
D
E
E1
F
Ω


Time
A
B
C
D
E
C1
C2
F
C3
E1
Ω


Time
A
B
C
D
E
C1
C2
C3
E1
F
Ω
...

Automatic Version Selection

Nonlinear versioning alone doesn’t reduce technical lag

- People are lazy

Current package managers allow version ranges

- Requires trust in developer

Two methods to minimize technical lag

- Allow Security patches
- Allow Functionally Equivalent Versions

Allow Security Patches

Security patches assumed non-breaking

Select most recent security patch of selected version

Coupled with traditional version selection:

- Take highest version based on traditional rules
- Take most recent security patch of that version

Minimal Risk

Big impact if security patches are targeted

- Focus on most used versions
- Focus on highly vulnerable versions

What is Functional Equivalence

Given program A and program B, is it possible to determine if those programs have the exact same output state for every possible input state.

- Undecidable problem

- We want to get as close as possible
  - Reasonable certainty
  - Requires domain-specific knowledge

Allow Functionally Equivalent Versions

Select highest version equivalent to a given version

- Equivalence is approximated - Can specify a confidence level
- Select most recent security patch

Coupled with traditional version specification:

- Find highest version based on traditional rules
- Select highest version equivalent above the confidence level
- Take most recent security path

Allows specification of allowable risk

Big impact

- Maximizes security with no effort
- Pairs with ‘choose a version and forget’ mentality

Our Test Package - naev-npm

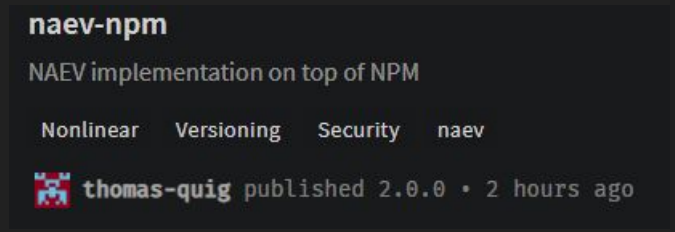
Very basic npm package

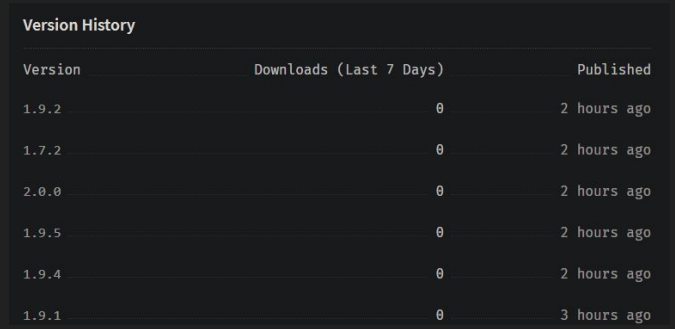
~dozen methods

Jest unit tests

Easiest testing framework to work with

26 releases with known ground truth equivalence





Functional Equivalence Verification Implementation

Implement several analysis tools to compare versions

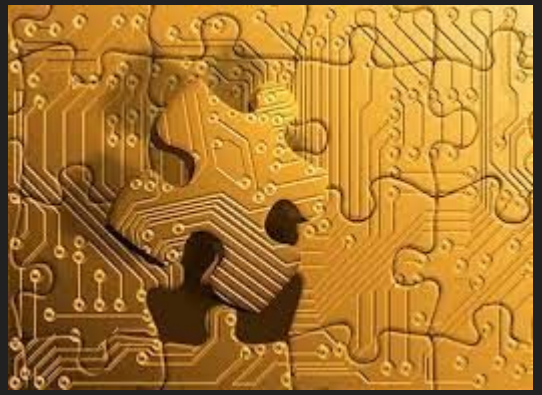
- Interface-Based analysis
- Test-Based analysis

No one method is guaranteed (undecidable)

We are striving for reasonable confidence

Functional equivalence for non-malicious inputs.

Interface-Based Functional Equivalence



Key Idea: Interfaces roughly describe a package

Adding things is non-breaking

Modifying things might be breaking

Removing things is breaking

Detecting interface changes allows equivalence approximation

- Not perfect, even with parsing oracle

Interface-Based Functional Equivalence - Implementation

Only examine ‘exported’ items - other items hidden

Identify changes in:

- Classes - class properties, class functions, class function parameters
- Variables
- Functions, function parameters

Approximate breaking ‘likelihood’ given delta counts

Many Assumptions - arbitrary javascript parsing is hard

Interface-Based Functional Equivalence - Results (1)

Interface Equivalence developed prior to test package

- Jest requires different format than expected, had to manually convert

Successfully compared every version of package, generated confidence number

- Ranged from 0.715 to 1.0

Interface-Based Functional Equivalence - Results (2)

	1.0.0	1.1.0	1.2.0	1.3.0	1.3.1	1.3.2	1.3.3	1.4.0	1.5.0	1.5.1	1.6.0	1.6.1	1.6.2	1.6.3	1.6.4	1.7.0	1.7.1	1.7.2	1.8.0	1.8.1	1.9.0	1.9.1	1.9.2	1.9.4	1.9.5	2.0.0
1.0.0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.1.0		1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.985	0.985	0.985	0.985	0.865	0.865	0.865	0.865	0.865	0.865	0.715	0.715	0.715	0.715
1.2.0			1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.3.0				1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.3.1					1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.3.2						1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.3.3							1.000	1.000	1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.4.0								1.000	1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.5.0									1.000	1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.892	0.892	0.892	0.892	0.892	0.892	0.772	0.772	0.772	0.772
1.5.1										1.000	1.000	1.000	0.988	0.988	0.988	0.988	0.896	0.896	0.896	0.896	0.896	0.896	0.781	0.781	0.781	0.781
1.6.0											1.000	1.000	0.988	0.988	0.988	0.988	0.896	0.896	0.896	0.896	0.896	0.896	0.781	0.781	0.781	0.781
1.6.1												1.000	0.988	0.988	0.988	0.988	0.896	0.896	0.896	0.896	0.896	0.896	0.781	0.781	0.781	0.781
1.6.2													1.000	1.000	1.000	1.000	0.904	0.904	0.904	0.904	0.904	0.904	0.784	0.784	0.784	0.784
1.6.3														1.000	1.000	1.000	0.904	0.904	0.904	0.904	0.904	0.904	0.784	0.784	0.784	0.784
1.6.4															1.000	1.000	0.904	0.904	0.904	0.904	0.904	0.904	0.784	0.784	0.784	0.784
1.7.0																1.000	0.938	0.938	0.938	0.938	0.714	0.938	0.862	0.862	0.862	0.862
1.7.1																	1.000	1.000	1.000	1.000	0.785	1.000	0.923	0.923	0.923	0.923
1.7.2																		1.000	1.000	1.000	0.785	1.000	0.923	0.923	0.923	0.923
1.8.0																			1.000	1.000	0.785	1.000	0.923	0.923	0.923	0.923
1.8.1																				1.000	0.785	1.000	0.923	0.923	0.923	0.923
1.9.0																					1.000	1.000	0.912	0.912	0.912	0.912
1.9.1																						1.000	0.923	0.923	0.923	0.923
1.9.2																							1.000	1.000	1.000	1.000
1.9.4																								1.000	1.000	1.000
1.9.5																									1.000	1.000
2.0.0																										1.000

Interface-Based Functional Equivalence - Results (3)

Captured general trend - more distant versions less equivalent

Cons:

- Far too optimistic in general
- Fails to detect non-interface changes

Pros:

- Detected changes are definitely breaking
- very lightweight

Test-Based Functional Equivalence

Give developers the power to check their code's compatibility

Run old tests on both versions

Given that identical tests passed, the versions are 'functionally identical'*

*100% Depends on how good your tests are at code coverage

Test-Based Functional Equivalence - Implementation

1. Grab packages from npm (skippable if src already downloaded)
2. Clone lower version test suite into higher version
3. Run both tests (run the lower version test suite on the lower and higher version)
4. Diff the outputs of the two, if they are past some margin, the versions are considered different.

Test-Based Functional Equivalence - Implementation (2)

- Why run the lower tests?
  - Allows for new features without defining the changes as breaking
- Specific implementation details
  - Test checker built in python
  - Runs jest tests on npm packages
    - Loads the json result file to compare
- Finding packages that actually meet these criteria is difficult
  - Implementing dynamic test analysis is out of scope for the semester.
  - Most developers don't upload tests when they publish their package.
  - Lots of crawling

Test-Based Functional Equivalence - Results

"1.1.0": ["1.2.0", "1.3.0", "1.3.1", "1.3.2", "1.3.3", "1.6.0", "1.6.1", "1.6.2"],
"1.2.0": ["1.3.0", "1.3.1", "1.3.2", "1.3.3", "1.6.0", "1.6.1", "1.6.2"],
"1.3.0": ["1.3.1", "1.3.2", "1.3.3", "1.6.0", "1.6.1", "1.6.2"],
"1.3.1": ["1.3.2", "1.3.3", "1.6.0", "1.6.1", "1.6.2"],
"1.3.2": ["1.3.3", "1.6.0", "1.6.1", "1.6.2"],
"1.3.3": ["1.6.0", "1.6.1", "1.6.2"],
"1.4.0": ["1.8.0", "1.8.1", "1.9.0", "1.9.1", "1.9.2", "1.9.4", "1.9.5", "2.0.0"],
"1.5.0": ["1.5.1"],
"1.5.1": [],
"1.6.0": ["1.6.1", "1.6.2"],
"1.6.1": ["1.6.2"],
"1.6.2": [],
"1.6.3": ["1.6.4", "1.7.0", "1.7.1", "1.7.2"],
"1.6.4": ["1.7.0", "1.7.1", "1.7.2"],
"1.7.0": ["1.7.1", "1.7.2"],
"1.7.1": ["1.7.2"],
"1.7.2": [],
"1.8.0": ["1.8.1", "1.9.1", "1.9.2", "1.9.4", "1.9.5"],
"1.8.1": ["1.9.1", "1.9.2", "1.9.4", "1.9.5"],
"1.9.0": ["1.9.1", "1.9.2", "1.9.4", "1.9.5"],
"1.9.1": ["1.9.2", "1.9.4", "1.9.5"],
"1.9.2": ["1.9.4", "1.9.5"],
"1.9.4": ["1.9.5"],
"1.9.5": []

Test-Based Functional Equivalence - Results (2)

naev-npm : the actual **live** npm package we were using for this

https://www.npmjs.com/package/naev-npm

- Adheres to proper semantic versioning
  - 1.3.x are all functionally Identical, Minor releases are not
- 1.3.x can also go to 1.6
  - Neat and also unplanned.

- Runtime
  - Subset (1.3.0 to 1.6.0)
    - Extremely slow (00:04:31.561)
  - Sequential test
    - Much faster (00:01:15.223)

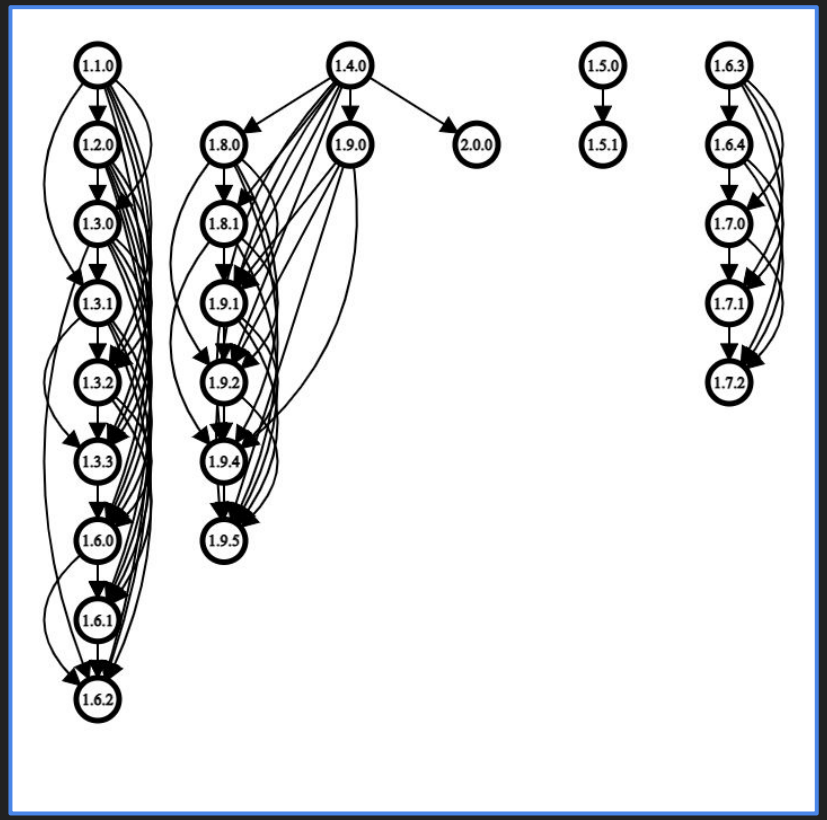
Full test (325 comparisons) took
real 105m33.008s
user 46m57.031s
sys 57m5.641s

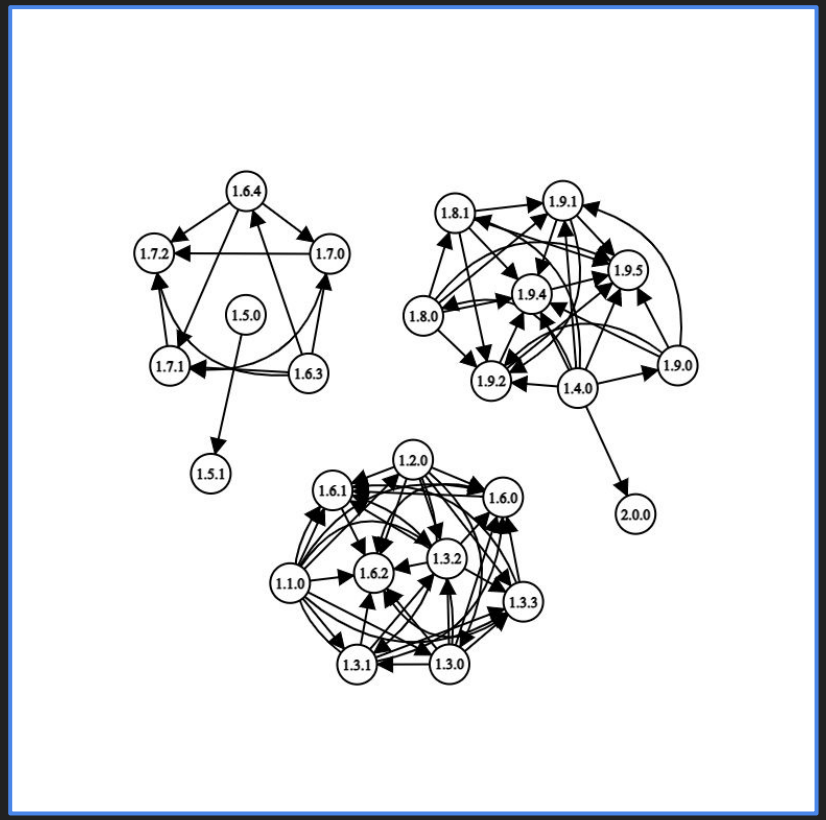
The Coolest Screenshot I’ve Taken all Semester

```
Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 17 passed, 18 total
Snapshots:   0 total
Time:        3.323 s
Test results written to: ../testResultU.json
Error 2 occured while comparing 1.9.5 and 2.0.0, check errlist
{'1.1.0': ['1.2.0', '1.3.0', '1.3.1', '1.3.2', '1.3.3', '1.6.0', '1.6.1', '1.6.2'], '1.2.0': ['1.3.0', '1.3.1', '1.3.2', '1.3.3', '1.6.0', '1.6.1', '1.6.2'], '1.3.0': ['1.3.1'
, '1.3.2', '1.3.3', '1.6.0', '1.6.1', '1.6.2'], '1.3.1': ['1.3.2', '1.3.3', '1.6.0', '1.6.1', '1.6.2'], '1.3.2': ['1.3.3', '1.6.0', '1.6.1', '1.6.2'], '1.3.3': ['1.6.0', '1.6.
1', '1.6.2'], '1.4.0': ['1.8.0', '1.8.1', '1.9.0', '1.9.1', '1.9.2', '1.9.4', '1.9.5', '2.0.0'], '1.5.0': ['1.5.1'], '1.5.1': [], '1.6.0': ['1.6.1', '1.6.2'], '1.6.1': ['1.6.2
'], '1.6.2': [], '1.6.3': ['1.6.4', '1.7.0', '1.7.1', '1.7.2'], '1.6.4': ['1.7.0', '1.7.1', '1.7.2'], '1.7.0': ['1.7.1', '1.7.2'], '1.7.1': ['1.7.2'], '1.7.2': [], '1.8.0': ['
1.8.1', '1.9.1', '1.9.2', '1.9.4', '1.9.5'], '1.8.1': ['1.9.1', '1.9.2', '1.9.4', '1.9.5'], '1.9.0': ['1.9.1', '1.9.2', '1.9.4', '1.9.5'], '1.9.1': ['1.9.2', '1.9.4', '1.9.5']
, '1.9.2': ['1.9.4', '1.9.5'], '1.9.4': ['1.9.5'], '1.9.5': []}

real    105m33.008s
user    46m57.031s
sys     57m5.641s
tquig@THOMAS-PC:~/Developer/CS-563/naev/src$
```

The coolest images I’ve made all semester





Limitations

- Not 100% accurate, should be used in conjunction with other tests.

- Very limited scope at the moment
  - Implementation was on npm and jest, whereas this is actually an abstraction for all package managers. This may not generalize as well as we would want it to in reality. (Might not work on X, Y, or Z)

- Testing
  - Finding live packages to test this on is nearly impossible because of the specific requirements for packages.
  - Researcher created packages designed to demonstrate test capabilities, instead of having standard functionality.

Conclusion

naev provides a useful abstraction for security-focused package managers

naev offers a hands-off solution to package version selection

- Aims to maximize security with little effort from developers

Functional equivalence is useful for identifying breaking changes

- Our implementations show it is difficult, but feasible to automate
- The accuracy-complexity tradeoff is severe

Questions?