IDENTIFICATION OF DEPENDENCY-BASED ATTACKS ON NODE.JS

Brian Pfretzschner
Technical University of Darmstadt, Germany

Lotfi ben Othmane
Iowa State University, USA

August 31, 2017
1. Located in Ames, Iowa – 6h from Chicago
2. 36 600 students in 2016 (4% annual increase)
3. 50 Faculty members in the Electrical and Computer Engineering Department
4. More than 200 funded Ph.D. students
   ➔ Open for Excellent candidates
USE OF DEPENDENCIES IN NODE.JS

Node Package Manager (NPM)
500,000 packages
QUESTIONS

1. How attackers can exploit third-party dependencies in Node.js environment?

2. How to mitigate such attacks?
USE OF MALICIOUS DEPENDENCIES
WEAKNESSES OF ENVIRONMENT

• Use of global variables
• Allows manipulation of loaded module cache
• Allows for monkey-patching

```javascript
1 function MyClass () {}
2 MyClass . prototype . someFunction = function () {
3 // Initial implementation
4 };
6 function performMonkeyPatch () {
7 var originalFunction = MyClass . prototype . someFunction ;
8 MyClass . prototype . someFunction = function () {
9 // New monkey-patched implementation
10 // originalFunction can be invoked here
```
ATTACK - GLOBAL LEAKAGE

Example: Redefinition of async library

1. var async = require('async');
2. var map = async.map;
3. async.map = function () {
4. var events = arguments[0];
5. leak(events);
6. return map.apply(this, arguments);
7. };
8. function leak() { ...}
GLOBAL MANIPULATION ATTACK

Manipulation of the validation function

1. > require='./index.js'; // Contains the Manip . attack
2. > validator . isEmail ('name@example.de" hello =" world ');
3. true

4. // Content of ./index.js
5. var validator = require('validator ');
6. validator . isEmail = function ( val ){
7.   if ( val === 'de" hello =" world ') return true ;
8.   else return isEmail . apply (this , arguments );
9. };

1. require('./malicious-lib');
2. require.cache[require.resolve('victim-lib')]
3. = require.cache[require.resolve('./malicious-lib')];
ATTACK SCENARIOS

- Dependency-based Attack
  - Start A New Dependency
  - Promote the Dependency
  - Social Engineering / Corruption
- Inject A Dependency
  - Exploit NPM Vulnerabilities
- Modify An Existing Dependency
  - Attack Developer Accounts (e.g., GitHub)
  - Attack Development Resources
- Direct or Indirect Installation
IDENTIFICATION OF ATTACKS

- Extended T.J. Watson Libraries for Analysis (WALA) code analysis to detect the attacks.
- We use control flow and data flow of WALA to detect the attacks.
- The analysis is integrated to OpenWhisk ➔ Cloud providers analyze your code before deployed to the cloud.
CODE ANALYSIS RESULTS

• We developed 20 test cases for the 4 attacks

• Most of the analyses (4 for each of the 20 cases) end in less than a second except 2, which take little over 2 min)

• Global manipulation detection performs badly for global leak examples. (false positive is high)
CONCLUSION

- Node.js and JavaScript support
  - global variables
  - monkey-patching
  - loaded modules cache

- Dependency-based attacks:
  - leakage of global variables
  - manipulation of global variables
  - manipulation of local variables
  - manipulation of the dependency-tree

- We developed code analysis to detect these attacks when code is loaded to the cloud platform
Thank you

Lotfi ben Othmane
othmanel@iastate.edu