System Administration

Week 04, Segment 4

Package Management Pitfalls

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/615/
Software Types

CS615 - System Administration

Add-on or Third-Party Applications
(web browser, database, programming languages, ...)

System Software
(device drivers, loadable modules, libraries, ...)

Applications/Utilities
(shell, common unix tools, daemons, compiler, ...)

Kernel

Firmware

Hardware

Package Management

Operating System
echo "import antigravity" | python
“Native” language package managers

“What’s pip?”
“A python package manager”
“How do I install it?”
“easy_install pip”
“What’s easy_install?”
“A python package manager”
"Native" language package managers

What is Bower?
"A package manager"
"How do I install it?"
"Use npm"
"What's npm?"
"A package manager"
"...."
“Native” language package managers
Dependencies, Integrity, and Trust

OS provider repositories:
• yum update/yum install
• apt-get
• pkgin

Language-specific community repositories:
• gem install foo
• go get github.com/randomAccount/randomRepository
• npm install -g foo
• perl -MCPAN -e 'install Something::YouWant'
• pip install foo

What could possibly go wrong?
$ curl http://somewhere/script.sh | sudo bash

isn’t any better nor worse than

$ wget http://somewhere/some.tar.gz
$ tar zxf some.tar.gz
$ cd some
$ ./configure
$ make
$ sudo make install

or

$ gem install foo
$ go get github.com/randomAccount/randomRepository
$ npm install -g foo
$ perl -MCPAN -e 'install Something::YouWant'
$ pip install foo

or

$ brew install whatever
A diagram illustrating the process of verifying the integrity of an RPM (Red Hat Package Manager) package. The process involves:

1. The RPM package contains a lead, signature, and header (metadata) followed by the payload.
2. A public key is used to verify the signature.
3. A hash (H(x)) is calculated for the header and payload.
4. The hash is compared with the signature.
5. If the hashes match, the RPM integrity is OK. Otherwise, it has been tampered with.

The diagram shows the flow from the public key, through the verify integrity stage, to the compare hashes stage, and finally to the conclusion of RPM integrity being OK or tampered with.
Remember Left-Pad?

```javascript
module.exports = leftpad;

function leftpad (str, len, ch) {
    str = String(str);
    var i = -1;
    if (!ch && ch !== 0)
        ch = ' ';  
    len = len - str.length;
    while (++i < len) {
        str = ch + str;
    }
    return str;
}
```

https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/
Dependencies, Integrity, and Trust

https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610
Step 1: GitHub Dorking

14,144,094 code results
Step 2: npm-publish

Publish a package

Version 7.x (Current release)

Synopsis

npm publish [<tarball>|<folder>] [--tag <tag>] [--access <public|rest>]

Publishes '.' if no argument supplied
Sets tag 'latest' if no --tag specified

Description

Publishes a package to the registry so that it can be installed by name.

By default npm will publish to the public registry. This can be overridden by specifying a different default registry or using a scope in the name (see package.json).
Step 3: ...wait...
Step 4:

To resolve packages by name and version, npm talks to a registry website that implements the Commons Package Registry specification for reading package info.

- `npm install [@scope/]@<name>`

  Do a `<name>@<tag>` install, where `<tag>` is the "tag" configured in `config`. The config's default value is `latest`.

  In most cases, this will install the version of the package tagged as `latest` on the npm registry.

- `npm is configured by the npm public registry at https://registry.npmjs.org by default.

  Use of the npm public registry is subject to terms of use available at https://www.npmjs.com/policies/terms.

  You can configure npm to use any compatible registry you like, and even run your own registry. Use of someone else's registry may be governed by their terms of use.
Dependencies, Integrity, and Trust

Dependencies are called dependencies because you *depend* on them.

Mirroring untrusted, unverified dependencies does not solve any of your problems.

Integrity verification is meaningless without assurance of trust.

Dependency trust and integrity is recursive.
Questions

• Research the cited repository incidents — how would you protect your environment from similar compromises or impact?

• What repositories do the different package managers we’ve seen use by default? How do we know that we can trust them?

• If you use native language package managers, how could you build cross-dependencies with the native OS package manager?

Coming up: multiuser fundamentals and authentication basics
Links

Software Installation and Package Management:

Left-pad: https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/

Dependency Confusion: https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

RPM File Format Details: