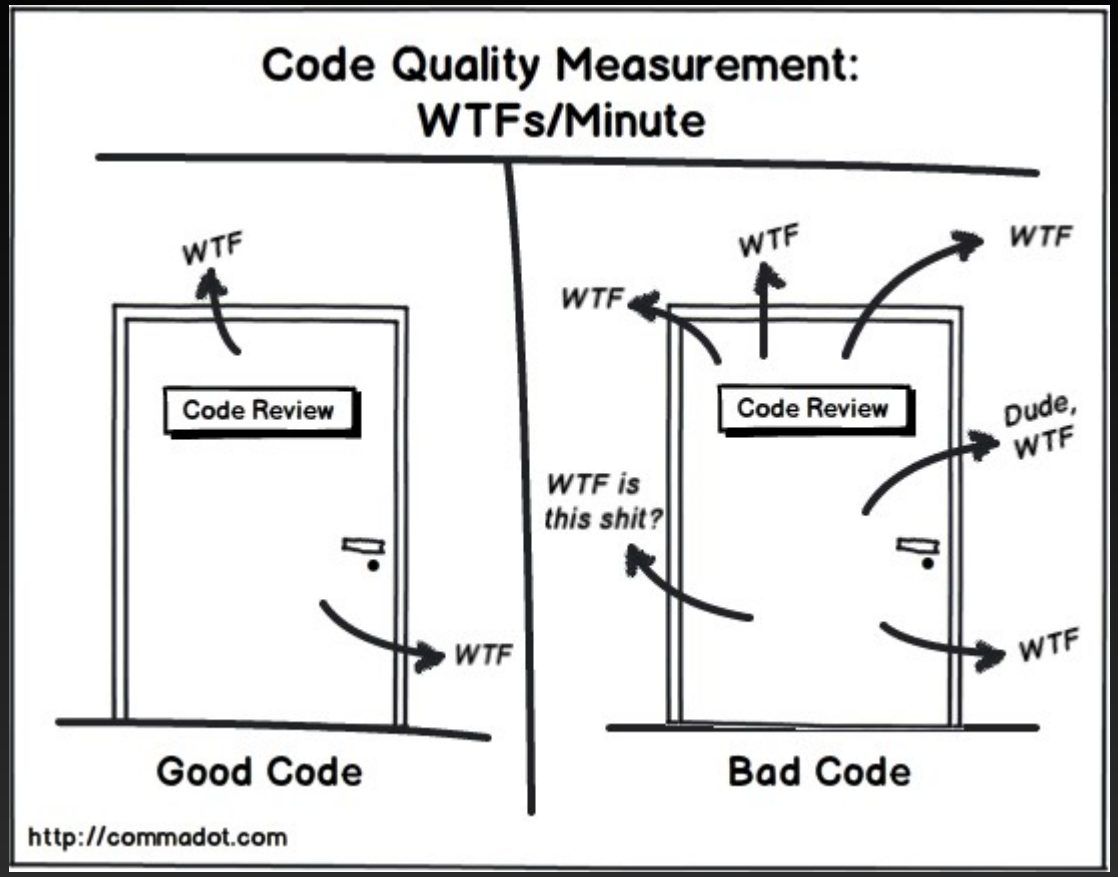


¿¡ PERO QUIÉN (#~(#& PROGRAMÓ ESTA $3& !?

SEGURIDAD EN LA CADENA DE SUMINISTRO DE PROYECTOS DE SOFTWARE

CRISTIÁN ROJAS, CSSLP, ISO27KLA/LI

PARENTAL ADVISORY EXPLICIT CONTENT

SOCHISI
DIGITAL SECURITY THINK TANK

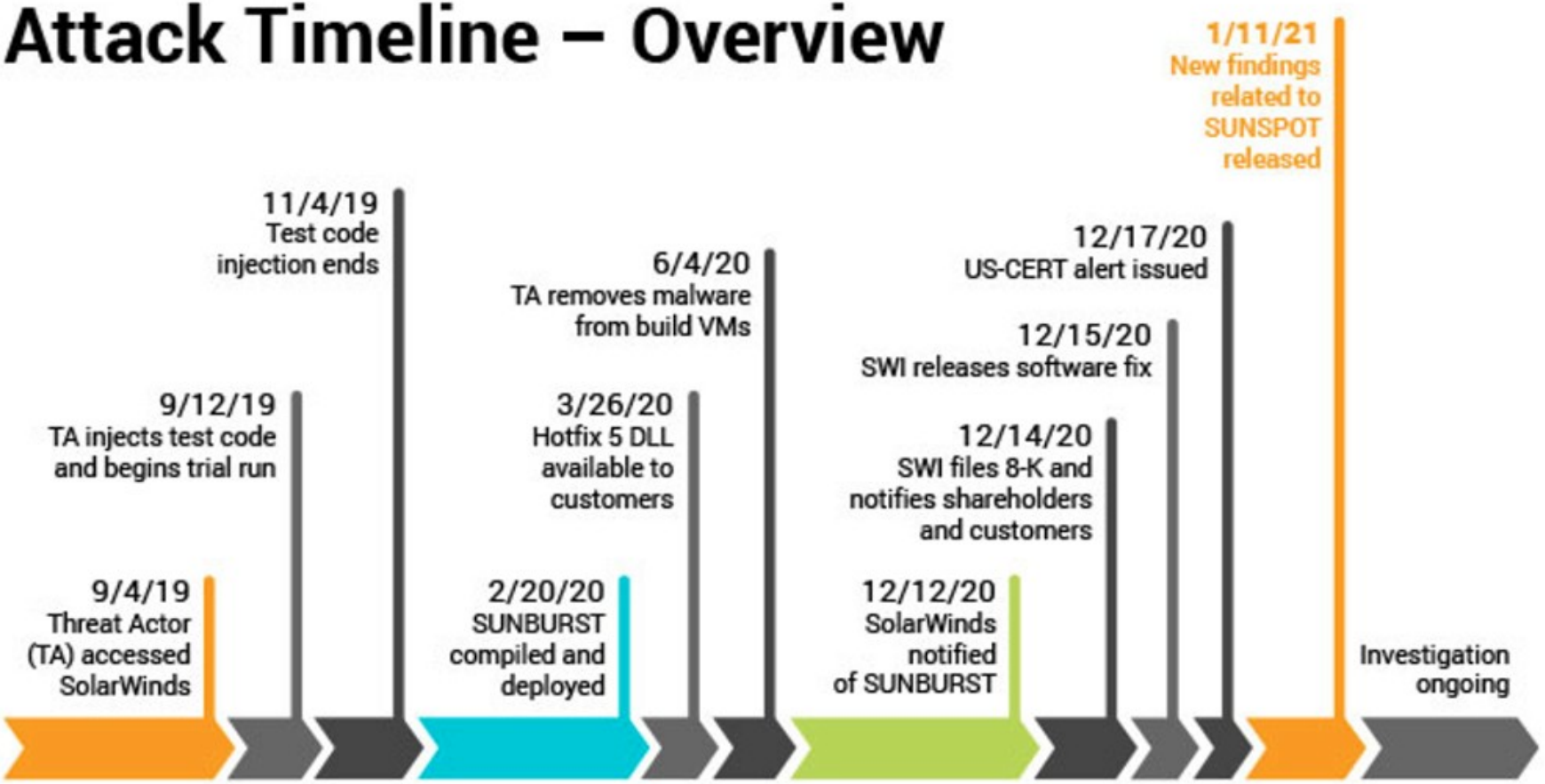
https://diccionariochileno.cl/term/Que+chucha

It's FOSS, "BLM Effect: Linux Kernel to Adopt an Inclusive Code Language, Blocks Terms like Blacklist-Whitelist and Master-Slave"

```
8db4808d2 (Jacob Herrington        2020-01-16 14:02:09 -0600   1) # coding: utf-8
8db4808d2 (Jacob Herrington        2020-01-16 14:02:09 -0600   2)
b8e68600d (Mac Siri                2019-08-05 16:39:09 -0400   3) git_source(:github) { |name| "https://github.com/#{name}.git" }
^301c6080 (Mac Siri                2018-02-28 16:11:08 -0500   4) source "https://rubygems.org"
30554b84f (rhymes                  2020-02-11 23:25:04 +0100   5) ruby File.read(File.join(File.dirname(__FILE__), ".ruby-version")).s
trip
^301c6080 (Mac Siri                2018-02-28 16:11:08 -0500   6)
aed0869f2 (Michael Kohl            2018-07-21 04:25:25 +0700   7) group :production do
bd10c9086 (Molly Struve            2020-03-16 10:10:01 -0400   8)   gem "hypershield", "~> 0.2.0" # Allow admins to query data via int
ernal
46b1b1d28 (Anna Buianova           2019-04-25 14:56:15 +0300   9)   gem "nakayoshi_fork", "~> 0.0.4" # solves CoW friendly problem on
MRI 2.2 and later
ff929fcf1 (rhymes                  2019-05-21 00:53:31 +0200  10)   gem "rack-host-redirect", "~> 1.3" # Lean and simple host redirect
ion via Rack middleware
aed0869f2 (Michael Kohl            2018-07-21 04:25:25 +0700  11) end
aed0869f2 (Michael Kohl            2018-07-21 04:25:25 +0700  12)
```

[rswanson@pawneepr ~] git blame Gemfile_

Attack Timeline – Overview



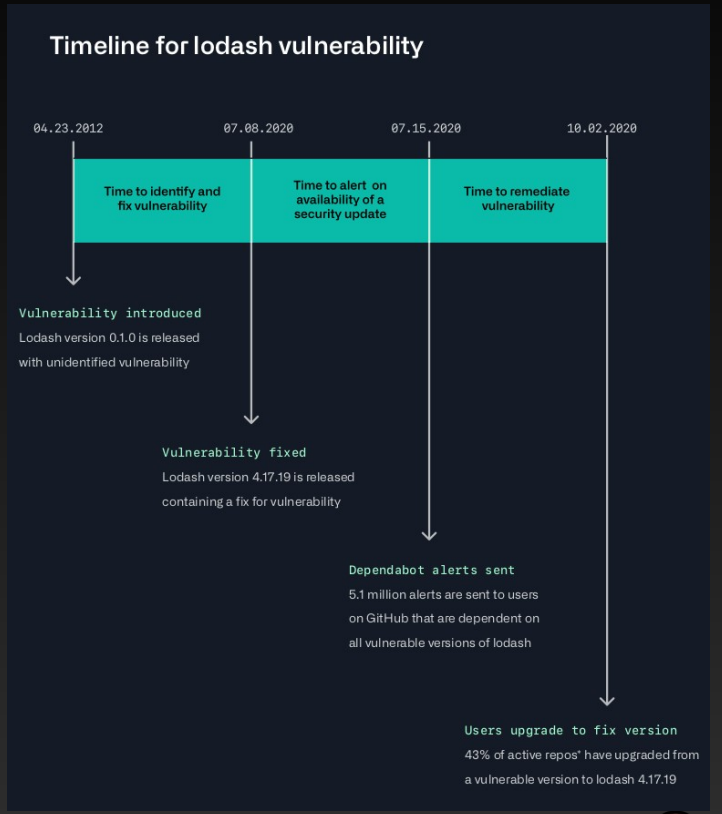
All events, dates, and times approximate and subject to change; pending completed investigation.

HelpNetSecurity, “SolarWinds hack investigation reveals new Sunspot malware”

Bloomberg Quicktake, “Arnold Schwarzenegger Gets COVID-19 Vaccine: 'Come With Me If You Want to Live'”

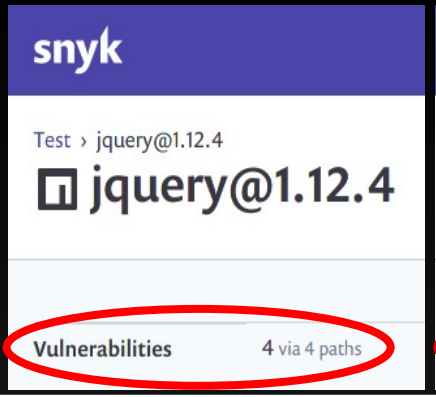
“Opensource es más seguro que propietario”

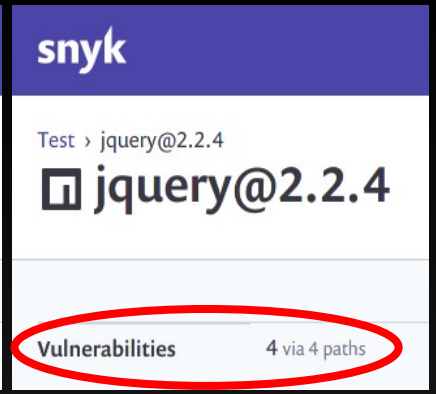
- Ley de Linus: “Dada la suficiente cantidad de ojos, todos los bugs se hacen obvios”
- ¿Todos?
  - Una vulnerabilidad toma en promedio… **¡232 semanas en ser corregida!**
  - Consecuencia: Dependencias sin parchar por mucho tiempo
- ¿Por qué?



GitHub, “The 2020 state of the Octoverse” (reporte de seguridad)

El peligro del End-Of-Life (EOL)



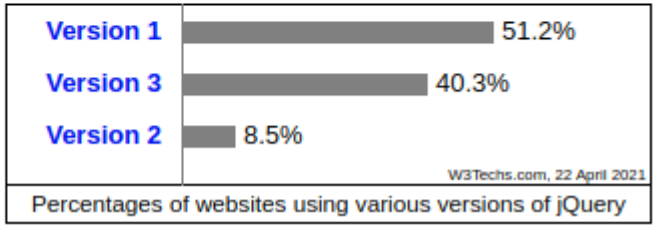


We don't give any guarantees for jQuery 1.x/2.x - they're officially unsupported - but if a serious vulnerability was detected (& sent to security@jquery.com) we might reconsider a patch.

Versions of jQuery

This diagram shows the percentages of websites using various versions of jQuery.

How to read the diagram:
Version 1 is used by 51.2% of all the websites who use jQuery.



Percentages of websites using various versions of jQuery

https://w3techs.com/technologies/details/js-jquery
https://github.com/jquery/jquery.com/issues/162

Solución: SCA y parchado basado en SEMVER

- SCA: Catastro automático de dependencias
  - Y a veces, autoparchado
- Herramientas conocidas:
  - OWASP Dependency Check (Java, .NET)
  - npm audit (Node)
  - Safety (Python)
  - Bundler Audit (Ruby)
  - Sonatype OSS Index (multiplataforma)
  - Snyk (multiplataforma)
  - DependaBot (GitHub)
  - Dependency Scanning (GitLab Ultimate)

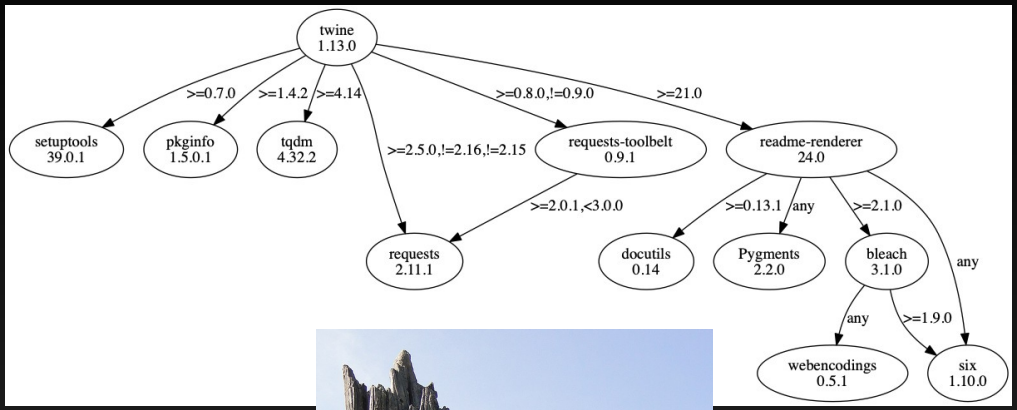
Ejemplos de upgrade automático y riesgos de DoS


```
"foo": "~3.2.17"
"foo": "^3.2.17" ⚠️
```


```
gem 'foo', "~>3.2.17"
gem 'foo', "~>3.2" ⚠️
```

El Solarwinds de los devs

- 3 tipos de ataques:
  - Repositorio infiltrado
  - Repositorio preparado
  - Confusión de dependencias
- Casos:
  - Ruby: `pretty_color`, `ruby-bitcoin`
  - NPM: `amzn`, `serverless-slack-app`



Michael Burch, “Be afraid of the Ruby on Rails Supply Chain”
Sonatype, “PyPI and npm flooded with over 5,000 dependency confusion copycats”

¿Cómo evitarlo?

- Usar repositorios locales
  - Ej., Sonatype Nexus
  - Mejor aún si incluyen funciones anti-malware o de lista negras
- Fijar versiones de paquetes
  - Adiós del todo al autoparchado y los ^, ~, ~>...
- Solución Solarwinds-style: Builds paralelos

¿Y ese código que copypasteaste de StackOverflow?

- Vulnerabilidades típicas:
  - Faltas de validación (input y output)
  - Falta de sanitización
  - Uso de código obsoleto
- Qué hacer:
  - Hacer SAST al menos en el IDE
  - Deuda de seguridad = Deuda técnica

El problema con usar plataformas E-Commerce/CMS

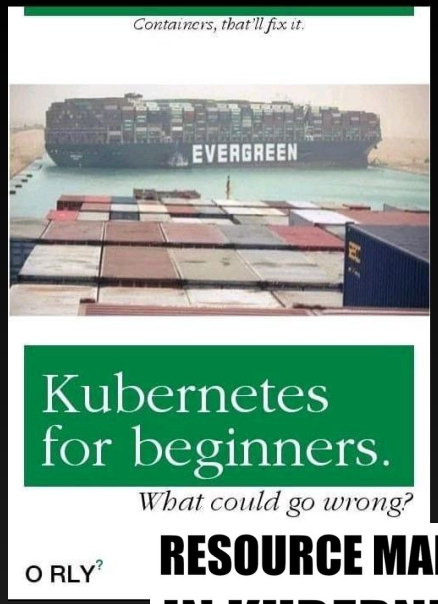
- Son tan fáciles de instalar y usar que nadie pregunta "quién (#~(#&"
- En su configuración por defecto son muy inseguros
- Tampoco hay una preocupación mayor por mantenerlos actualizados en sus parches de seguridad
  - Menos con sus plugins, que es por donde más ataques llegan
- Ejemplos:
  - Wordpress
  - WooCommerce

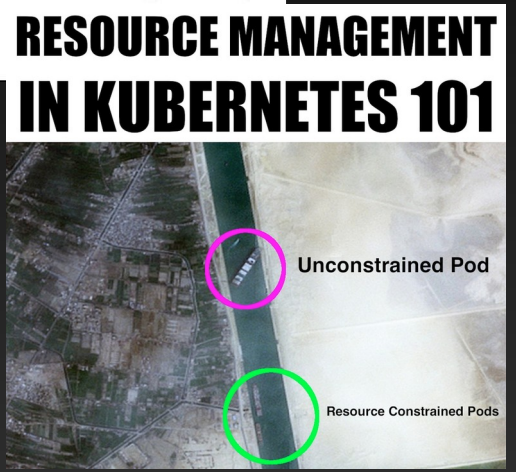
woocommerce/includes/class-wc-install.php

```
1  // Shop manager role.
2  add_role(
3          'shop_manager',        // Internal name of the new role
4          'Shop manager',        // The label for displaying
5          array(                 // Capabilities
6                  ⋮
7                  'read_private_posts'     => true,
8                  'edit_users'             => true,
9                  'edit_posts'             => true,
10                 ⋮
11         )
12 );
```

Los contenedores son código también

- Llevar un repo local de imágenes
- Limitar el acceso a recursos (CPU, memoria, red...)
- ~~`USER root`~~ → `USER jenkins`
- Fijar en el Dockerfile:
  - El tag de la imagen (no usar `latest`)
  - Las versiones de los paquetes a instalar (apt, dnf, apk...)
- Herramientas: Hadolint, Trivy

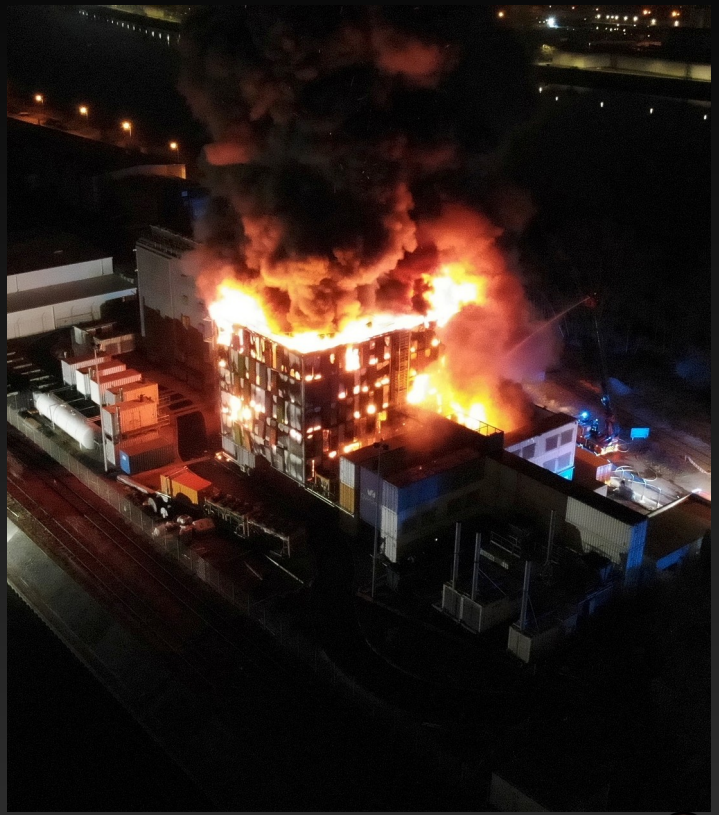




Prevasio, "Operation Red Kangaroo: Industry's First Dynamic Analysis of 4M Public Docker Container Images"

Cloud: Activo crítico

- Responsabilidad compartida
- Ningún acceso sin MFA
- Infrastructure/Security as Code
- Usar herramientas de seguridad ofrecidas por proveedores
  - GuardDuty, Inspector, Macie (AWS)
  - Security Center (Azure)
  - Command Center (Google)
- Ojo con la disponibilidad: Los Cloud también se caen



Infosecurity Magazine, "OVH Data Center Fire Impacts Cyber-criminals"

¿Quién (#~(#& es el externo que nos hace software?

- Muchas empresas usan código desarrollado por externos
  - Primera pregunta: ¿De quién (#~(#& es ése código?
- ¿Les exigimos niveles adecuados de seguridad?
- Herramientas:
  - OWASP Software Component Verification Standard (SCVS)
  - OWASP Secure Software Contract Annex

Conclusiones

- Empezar, de alguna manera, a decir "¡¿quién (#~(#&?!"
  - Preocuparnos del código del que dependemos, tanto de su calidad como de su procedencia
  - Recuerden siempre: Al menos el 80% de nuestro software son dependencias
- Mantener catastros de nuestras dependencias y poner atención a eventos EOL
- Aprovechar herramientas automatizadas (SCA) para análisis de dependencias
  - E incluir sus pruebas tanto en el IDE como en los pipelines CI/CD

Muchas gracias por su atención.

A continuación, ronda de preguntas y respuestas.

linkedin.com/in/injcristianrojas

injcristianrojas@gmail.com

{SECURITY DEV DAY}