De event-stream, npm, y Arquitecturas que Invitan al Desastre

Publicado en: SG #58 (/REVISTA/58)

COLUMNA INVITADA (/SECCION-REVISTA/COLUMNA-INVITADA)

Autor: Gunnar Wolf (/buzz/autores/gunnar-wolf)

En los últimos días de noviembre pasado se dio a conocer una noticia que armó revuelo — Y cuya intersección me atañe e interesa mucho en lo personal: Seguridad informática, software libre, integración de proyectos libres, arquitecturas de gestión de paquetes, mecanismos de compilación y distribución… Cuando me invitaron a participar con este texto adicionalmente a mi columna habitual, lo primero que pensé es, "¿cómo no escribí mi columna con ese tema? ¿Será que ya lo anticipábamos tanto que no me resultó de suficiente importancia?"

Y es que tiene mucho, muchísimo tiempo que venimos anticipando que esto ocurra. Y ha ocurrido, aunque de formas menos graves.

¿Qué pasó con event-stream?

Con este texto no busco reportar de forma ágil y oportuna, sino que hacer un pequeño análisis sobre las causas, y algunas recomendaciones para evitar que casos como este se repitan. El tema fue ya cubierto oportunamente por SG Buzz [1] al muy poco tiempo de reportarse. Hay, además, un análisis mucho más completo en LWN.net [2]. Para quien quiera ver detalles mucho más finos, Zach Schneider presentó en su blog personal otro artículo [3] incluyendo un desglose

técnico de cómo se realizó el ataque. Me salto, pues, tanto de los detalles como me es posible.

Así que, resumiendo tremendamente: event-stream es una biblioteca de Javascript, muy frecuentemente utilizada para la manipulación de flujos (streams) de datos en aplicaciones basadas en Node. Si bien es una biblioteca que goza de muy alta popularidad, como frecuentemente pasa, su desarrollador original fue perdiendo motivación para desarrollarla, indicando que "ya ni siquiera la utilizaba". Esto es muy comprensible: A lo largo de los ocho años de vida que tiene este proyecto, ¿cómo se le puede exigir a una persona comprometerse a mantener los mismos intereses y proyectos de vida?

En septiembre de 2018, un usuario con alias rightgctrl solicitó al autor de event-stream el convertirse en su nuevo mantenedor. Esto ocurre todo el tiempo en distintos proyectos. Este nuevo mantenedor inició con un muy buen ritmo de actividad, subió nuevas versiones. Como es frecuente, una de esas nuevas versiones incluía una dependencia sobre una nueva biblioteca — flatmap-stream. Esta biblioteca contiene código legítimo y válido, lo único que podría llamar la atención al respecto es que no tenía ninguna instalación previa a su inclusión — Pero no resulta incomprensible que un desarrollador intente modularizar más su código, separando alguna funcionalidad para poderla utilizar sin depender de todo el código…

Hasta aquí todo bien. Ahora, flatmap-stream se distribuía tanto en el formato fuente, como minificada. ¿Qué es esto? Dado que el código Javascript muy frecuentemente se transmite a clientes Web (sean éstos navegadores, aplicaciones celulares, o incluso aplicaciones nativas en diversos entornos de escritorio), se busca que la descarga sea tan corta y ágil como sea posible. La minificación es, a fines prácticos, una compilación hacia el mismo lenguaje: Elimina todos los comentarios, todo el espacio en blanco, y reemplaza los nombres descriptivos de funciones y variables por el más corto posible. El código minificado, si bien semánticamente idéntico al código fuente, no puede considerarse modificable o, siquiera, legible por otra persona.

La estrategia de nuestro inteligente atacante, pues, fue no incluir nada malicioso en el código fuente, pero sí lo hizo en la versión minificada. Y otro punto importante: ¿Qué es lo que buscaba atacar? No era al grueso de usuarios de event-stream… Su uso es demasiado genérico. Sin embargo, identificó a una aplicación en particular que tiene a event-stream entre sus dependencias: copay-dash, un gestor de billeteras de criptomonedas.

Habiendo leído esto último, estoy seguro de que, como decimos en México, ya les cayó el veinte respecto el efecto de este ataque: Lo que logra es transferir el control de la billetera que tenga el desafortunado usuario que corra el código malicioso a rightgctrl.

¡Ah! Y un punto importante, específico a la cultura de desarrollo de npm: La búsqueda de un punto vulnerable en una aplicación relacionada con manejo de billeteras… Puede no haber sido tan difícil. El La cultura de npm apunta a que se empaqueten como biblioteca independientes pedacitos incluso triviales de cinco o diez líneas de código. Es imposible que un sólo desarrollador controle el conjunto de dependencias de ese proyecto: copay-dash incluye… ¡1277 dependencias de 378 mantenedores diferentes!4

¿Cómo podría haberse evitado?

Los usuarios afectados no pidieron actualizar manualmente las dependencias de sus sistemas. Esto se debe a la cultura de moverse rápido y romper cosas que ha sido conscientemente adoptada por la comunidad de desarrolladores de Node, particularmente por conducto de su gestor de paquetes, npm. Este gestor busca ofrecerle a los desarrolladores siempre las últimas versiones disponibles de todas sus bibliotecas, actualizándolas automáticamente desde fuentes sin relación explícita de confianza entre sí, hospedadas en cualquier lugar de la red, y bastando con el mero hecho de listarlos en un registro central. La página de npm5 dice ser el registro de software más grande del mundo, con más de 800,000 bloques de construcción (módulos independientes).

El blog de Schneider sugiere como primer medida de mitigación a este tipo de ataques el uso de lock-files. Estos no son archivos de bloqueo en el sentido del control de escritura concurrente a archivos — Son archivos que, en vez de permitir que npm actualice automáticamente, especifiquen la versión específica que requieren de cada biblioteca. Esta estrategia es frecuentemente utilizada con gestores de paquetes específicos a lenguaje, e incluso tiene soporte en npm mismo, pero relativamente poca gente lo usa al ser antitético a la citada cultura de moverse rápido.

Esa recomendación, me parece, es una buena mitigación — Pero no previene ni cura el problema. Mi recomendación se aleja un par de pasos de la gratificación instantántea que brindan los gestores de paquetes específicos a lenguaje (npm para Node/Javascript, Gems para Ruby, Pip para Python, etcétera.

Una visión consistente, coherente, completa del sistema

En lo particular, abrevo mis prácticas de la cultura de las distribuciones de Linux, en particular de Debian. Desde hace varios años he defendido las ventajas de la filosofía de empaquetamiento a nivel sistema: Si bien tiene sentido que un desarrollador tenga las últimas versiones de las bibliotecas que emplea y conozca los detalles de su funcionamiento, no puede esperarse lo mismo de los usuarios finales de una herramienta.

Si estoy instalando un paquete determinado para gestionar mi billetera Bitcoin (o para cualquier otra cosa), no debería siquiera importarme en qué lenguaje está escrito, y mucho menos si la voy a instalar utilizando una u otra infraestructuras de empaquetamiento. La mayor parte de los usuarios están intersados en resolver su problema desde una única interfaz de gestión de software. De acuerdo, en el caso deNode, muchas de sus aplicaciones están pensadas para instalarse de forma centralizada, en un servidor, por un administrador de sistemas y no un usuario final. Aún así — Si se espera que dicho administrador emplee el desarrollo (en contraposición a utilizarlo como base para un desarrollo adicional), ¿para qué obligarlo a aprender otra idiosincracia más, otra manera más de gestionar un pequeño pedacito del sistema?

Por el trabajo que realizo con Debian, desde hace más de una década he venido lamentando el abandono de la consistencia y coherencia a nivel sistema que caracteriza a dicha distribución, y

al uso de Linux en general. Uno de los principales puntos que me atrajo hacia Debian hace ya muchos ayeres es conocer y enamorarme de su documento de políticas del sistema: Todos los paquetes que forman parte del sistema siguen una lógica común; el usuario sabe de antemano dónde está cada archivo de configuración, datos, programas, y se cumple un conjunto de expectativas que permiten el uso de un entorno operativo plenamente integrado.

Insisto — Un sistema empaquetado como lo presenta una distribución completa no busca reemplazar, ni lograría hacerlo con la oportunidad y agilidad necesarias, a los sistemas de paquetes por lenguaje. Éstos son el vehículo correcto para el desarrollador, para quien tiene que estar permanentemente en lo último — y sin filtros que exijan un nivel mínimo de madurez en cada uno de los componentes. Pero no podemos esperar que la respuesta sea la misma para los usuarios finales de nuestros desarrollos: Para ellos, estabilidad y predictibilidad valen muchísimo más.

A lo largo de las décadas en que se ha desarrollado el movimiento de software libre, han existido unos cuantos casos de software "troyanizado" de diferentes maneras, como event-stream, el caso que aquí nos ocupa. Afortunadamente, han sido pocos, y su impacto negativo mayormente ha logrado contenerse. Sin embargo, con la masificación del uso de los sistemas de empaquetamiento de modelos más ágiles y orientados naturalmente a desarrolladores más que a usuarios, como npm, muchos usuarios están quedando expuestos a este tipo de ataques.

Está en todos nosotros, como profesionales del ramo, racionalizar el uso de estas herramientas y contribuir con una cultura de seguridad informática en todo el mundo.

Referencias

1. SG Buzz: Crónica de la inyección de código malicioso a biblioteca de JavaScript https://sg.com.mx/buzz/cronica-de-la-inyeccion-de-codigo-malicioso-biblioteca-de-javascript (https://sg.com.mx/buzz/cronica-de-la-inyeccion-de-codigo-malicioso-biblioteca-de-javascript)

2. LWN.net: event-stream, npm, and trust https://lwn.net/Articles/773121/ (https://lwn.net/Articles/773121/)

3. Zach Schneider: event-stream vulnerability explained https://schneid.io/blog/event-stream-vulnerability-explained/ (https://schneid.io/blog/event-stream-vulnerability-explained/)

4. NPMGraph, http://npm.broofa.com/?q=copay-dash (http://npm.broofa.com/?q=copay-dash)

Log in (/user/login?destination=/revista/58/de-event-stream-npm-y-arquitecturas-que-invitan-al-desastre-0%23comment-form) or

register (/user/register?destination=/revista/58/de-event-stream-npm-y-arquitecturas-que-invitan-al-desastre-0%23comment-form)

to post comments

SEARCH

USER ACCOUNT MENU

Log in (/user/login)

SUSCRÍBETE A NUESTRO NEWSLETTER

Suscríbete para recibir noticias de Software Guru en tu correo.

Email Address

Nombre

Apellido(s)

SUSCRIBIRME

OPORTUNIDADES DE EMPLEO

- Desarrollador Java (https://sgtalento.com/content/desarrollador-java-4)
- Analista de Negocios (https://sgtalento.com/content/analista-de-negocios)
- iOS Engineer (https://sgtalento.com/content/ios-engineer)
- Senior Java Engineer (https://sgtalento.com/content/senior-java-engineer)
- Software Test Engineer (https://sgtalento.com/content/software-test-engineer)
- Full-Stack .NET Engineer (https://sgtalento.com/content/full-stack-net-engineer)
- JavaScript Engineer (https://sgtalento.com/content/javascript-engineer)
- Database Reporting Engineer (https://sgtalento.com/content/database-reporting-engineer)
- Rails Engineer (https://sgtalento.com/content/rails-engineer)
- Front End Engineer (https://sgtalento.com/content/front-end-engineer-1)

RECOMENDAMOS

- Monoliths, Migrations and Microservices (https://www.youtube.com/watch?v=gOZFmFNl1uk)
- Bye bye Mongo, Hello Postgres (https://www.theguardian.com/info/2018/nov/30/bye-bye-mongo-hello-postgres)

Software Guru es el medio preferido por las personas de habla hispana interesadas en construir software de alto desempeño.

> *Conocimiento para construir software grandioso.*

Más servicios de SG

Best Place to Code (https://bestplacetocode.com)
Data Day (https://sg.com.mx/dataday)
Dev Day 4 Women (https://devday4w.com)
Hackatour (https://sg.com.mx/hackatour)
Dev Relations (https://devrel.sg.com.mx)