Säkerhetsstrategier vid användning av bibliotek med öppen källkod

Jonatan Wackström

Kandidatavhandling
Handledare: Dragos Truscan
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
2022

Referat och nyckelord

De senaste åren har användningen av bibliotek med öppen källkod ökat betydligt. Användarvänliga verktyg för att installera och hantera kodbibliotek i mjukvaruprojekt uppmuntrar utvecklare att förlita sig på externa bibliotek. Tidigare studier rapporterar att cirka 80 procent av kod inom mjukvara kommer från tredjepartskällor, alltså är användningen av kodbibliotek en avgörande aspekt i utvecklingsprocessen av mjukvarusystem. Kodbibliotek tillåter utvecklare att använda pålitlig och noggrant testad kod och återanvändning av kod leder till snabbare utvecklingstakt samt tillåter utvecklare att fokusera på mjukvarans kärnfunktionalitet istället för den grundläggande funktionaliteten.

Användingen av öppen tredjepartskällkod medför även risker. Kodbibliotek är beroende av kollektivt skriven kod som publiceras och upprätthålls av enskilda aktörer. Nya sårbarheter i öppen källkod uppstår och rapporteras om dagligen. Majoriteten av sårbarheterna uppstår på grund av oavsiktliga misstag i kodbasen och tas ofta hand om innan slutanvändare eller sensitiv data påverkas. Däremot blir avsiktliga sårbarheter eller attacker vanligare och kodförråd utnyttjas för att effektivt distribuera skadlig kod.

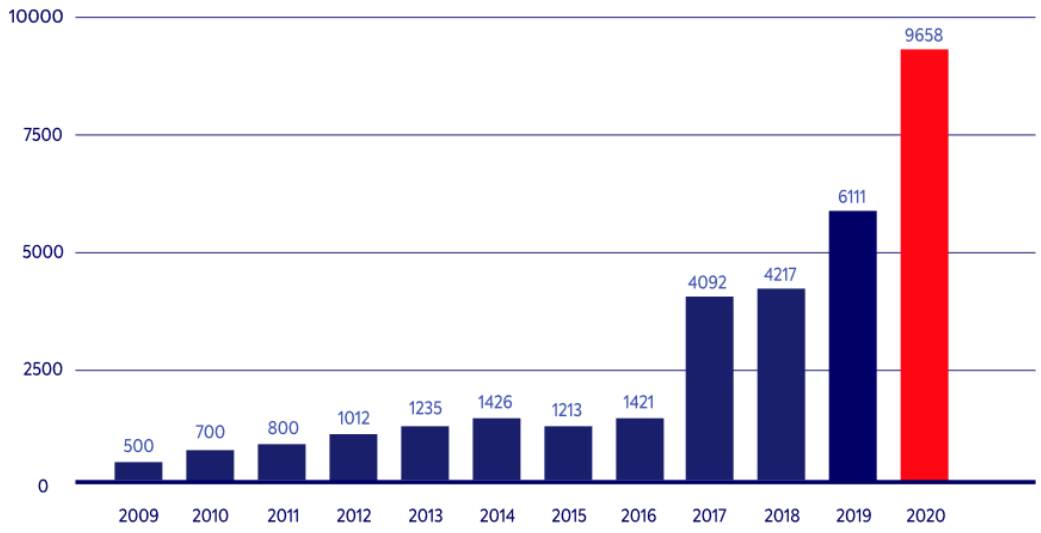
Nyckelord

Kodförråd, öppen källkod, kodbibliotek, sårbarheter, säkerhetsstrategier

Innehållsförteckning

1 Inledning

Användningen av bibliotek med öppen källkod introducerar risker för organisationer. Många organisationers användning av dessa kodbibliotek bygger enbart på förtroende för utvecklaren eller underhållaren bakom biblioteken [1]. Säkerhet och dataintegritet är en kritiskt aspekt för de flesta organisationer och säkerhetsläckor kan ha förödande konsekvenser för både organisationens verksamhet och anseende. Därför är det viktigt att införa omfattande säkerhetsstrategier och minimera risken att bli offer för sårbarheter eller avsiktliga attacker som introducerats genom bibliotek med öppen källkod. I figur 1 visas utvecklingen av antalet rapporterade sårbarheter i öppen källkod per år från år 2009 till 2020. En betydlig ökning de senaste fyra åren kan noteras.



Figur 1 Rapporterade sårbarheter i öppen källkod per år 2009-2020. [2]

Syftet med denna studie är att undersöka olika strategier för att mitigera risken att sårbar eller skadlig kod som introducerats av användningen av externa kodbibliotek används i mjukvara. I kapitel 2 presenteras historisk data om kodbibliotekens framfart de senaste åren. Därtill förklaras allmäna begrepp som används i studien. I kapitel 3 tas olika avsiktliga attackvektorer som i dagens läge används för att sprida skadlig kod inom kodbibliotek med öppen källkod upp.

I kapitel 4 diskuteras och sammanställs olika säkerhetsstrategier som kan tillämpas för att analysera och upptäcka sårbarheter eller attacker inom kodbibliotek och strategier för att minimera risken att skadlig kod används i utvecklad mjukvara.

Speciellt läggs fokus på de två populäraste programmeringsspråken, JavaScript och dess kodförråd Node Package Manager (npm), samt programmeringsspråket Python och dess kodförråd Python Package Index (PyPI). Miljöernas egenskaper och säkerhetsstrategier samt dess inbyggda verktyg för att mitigera sårbarheter evalueras och jämförs med varandra.

2 Bakgrund

I detta kapitel tas upp teori för de koncept och begrepp som behandlas i denna studie. Dessutom presenteras statistik för hur situationen har utvecklats under de senaste åren samt hur utvecklingen bidragit till att säkerhetsstrategier är en nödvänding del inom mjukvaruprojekt.

2.1 Öppen källkod

Mjukvara som är offentligt tillgänglig och som ofta är gratis att använda beskrivs som mjukvara med öppen källkod. Öppen källkod kan användas, inspekteras, förändras och förbättras av vem som helst. Användningen av mjukvara med öppen källkod tillåter utvecklare att fokusera på den egentliga funktionaliteten i mjukvaran istället för att spendera värdefull tid på trivial funktionalitet. Lättanvändna verktyg som tillåter utvecklare att implementera komplex funktionalitet med ett enda kommando har accelererat konsumptionen av öppen källkod avsevärt. I 2022 rapporterar GitHub, vilket är världens största tjänst för lagring av öppen källkod, att de lagrar totalt 290 miljoner kodförråd. [3]

Till mjukvara med öppen källkod hör även en licens. Licensen anger rättigheter och skyldigheter för användarna av mjukvaran. I regel tillåter licenserna användarna att utnyttja och distribuera källkoden fritt. Exempel på såna licenser är till exempel MIT som är skapat av Massachusetts Institute of Technology. MIT-licensen tillåter fri användning av den licensierade koden förutsatt att en kopia av licensen inkluderas i all mjukvara som använder den [4]. Undantag till regeln finns dock. Exempelvis GNU General Public License (GPL) kräver att mjukvara som skapas genom att modifiera GPL-licensierad kod publiceras offentligt i form av öppen källkod [5]. De flesta licenstyper innehåller också en sektion där det konstateras att mjukvaran förses utan någon som helst garanti eller skyldighet från upphovsmännen.

2.2 Kodbibliotek

Med användning av kodbibliotek avses användning av funktionalitet från tredjepartskod i mjukvara [1]. Användningen av tredjepartskod är en avgörande aspekt i utvecklingsprocessen av mjukvarusystem. Tredjepartskod levereras ofta i form av kodbibliotek som importeras i den existerande kodbasen. Medan det finns fördelar för utvecklare att skriva egen kod istället för att använda tredjepartskod från andra källor är det ofta inte hållbart i längden. Enligt Spinellis [6] är utvecklare betalda för att leverera slutresultat och inte för att återuppfinna hjulet.

2.3 Kodbiblioteksberoende

Kodbiblioteksberoende (eng. dependency) hänvisar till ett kodbibliotek eller en del av ett kodbibliotek som ett mjukvarusystem är beroende av för att fungera. GitHub för fram i sin årliga rapport för året 2019 att mjukvaruprojekt med öppen källkod i genomsnitt är beroende av 180 kodbibliotek [7]. Ett kodbibliotek kan vara beroende av ett eller flera kodbibliotek vilka i sitt fall kan vara beroende av ytterligare kodbibliotek. Strukturen av dessa beroenden bygger upp ett beroendeträd. Om bibliotek A är beroende av bibliotek B som är beroende av bibliotek C kallas biblioteket B för direkt beroende och bibliotek C för transitivt beroende.

Skillnader i antalet kodbiblioteksberoenden mellan programmeringsmiljöerna JavaScript och Python kan observeras. JavaScript-utvecklare tycks föredra användning av kodbibliotek även för de mest triviala funktionerna medan Python-utvecklare använder ett färre antal kodbibliotek i deras mjukvara [8]. I tabellerna 2 och 3 visas antalet direkta och transitiva kodbiblioteksberoenden för några populära bibliotek publicerade i kodförråden npm och PyPI samt antalet projekt som använder sig av och är i sin tur beroende av dem.

Bibliotek	Direkta	Transitiva	Används av
express@4.17.3 Webbramverk för Node.js	30	50	60 320
vue@3.2.31 Ramverk för användargränssnitt	5	21	52 377
aws-sdk@2.1096.0 Bibliotek för interagering med Amazon Web Services	9	14	18 536
eslint@8.11.0 Verktyg för kodanalys och kodstandarder	35	84	17 549
jest@27.5.1 Testramverk för JavaScript	3	335	10 124

Figur 2 Antalet dependencies i olika populära JavaScript-bibliotek.

Bibliotek	Direkta	Transitiva	Används av
django@4.0.3 Webbramverk för Python	3	0	6 439
pandas@3.2.31 Ramverk för dataanalys	3	1	2 169
boto3@1.21.22 Bibliotek för interagering med Amazon Web Services	3	9	4 350
pylint@2.12.2 Verktyg för kodanalys och kodstandarder	6	4	20 220
pytest@7.1.1 Testramverk för Python	6	1	16 169

Figur 3 Antalet dependencies i olika populära Python-bibliotek.

2.4 Kodförråd

Kodförråd tillåter utvecklare att ladda upp och publicera kod i form av kodbibliotek. Kodbiblioteken lagras i en centraliserad databas där andra utvecklare kan bläddra bland och använda biblioteken i sina egna projekt. Kodbiblioteken kan publiceras offentligt vilket tillåter vem som helst att hitta och ladda ned biblioteken. Biblioteken kan även publiceras privat vilket tillåter endast specifika användare eller användare inom en organistation att hantera och installera dem [9].

Utvecklare kan använda sig av pakethanteringsverktyg för att interagera med kodförråden. Pakethanteringsverktygen tillåter utvecklare att ladda ner, installera,

uppdatera och radera kodbibliotek i sina projekt [10]. Publicering av bibliotek sker ibland även med hjälp av dessa verktyg.

Varken npm eller PyPI implementerar någon sorts automatisk analys av bibliotek som publiceras i deras databas. Ansvaret för att rapportera bibliotek med skadlig kod som laddats upp läggs istället på enskilda användare av kodförråden. I praktiken kan vem som helst ladda upp ett bibliotek innehållande kod som raderar alla filer i ett filsystem omedelbart då biblioteket installeras.

I figur 7 visas utvecklingen av antalet kodbibliotek som lagras av kodförråden npm och PyPI från år 2011 till 2021. En betydlig ökning de senaste fem åren kan noteras för båda kodförråden.

2.4.1 npm

npm är ett kodförråd för JavaScript-paket och är världens största mjukvaruregister. Totalt upprätthåller npm över 1 800 000 kodbibliotek i sitt register [9]. Utvecklare kan använda sig av pakethanteringsverktyget npm CLI, vilket är ett kommandoradsgränssnittet, för att interagera med registret [10].

För att ladda upp och publicera ett bibliotek till npm krävs endast ett projekt innehållande en package.json-fil och ett npm-konto [9]. Filen package.json innehåller konfigurationen för ett projekt. Bland annat definierar filen kodbiblioteksberoenden [11] under nyckeln “dependencies”. Figur 5 visar ett exempel på innehållet av en package.json-fil.

```
{
  “name”: “app”
  “version”: “1.0.0”,
  “dependencies”: {
    “express”: “4.17.3”
  }
}
```

Figur 4 Exempel på innehåll av package.json-fil.

2.4.2 PyPI

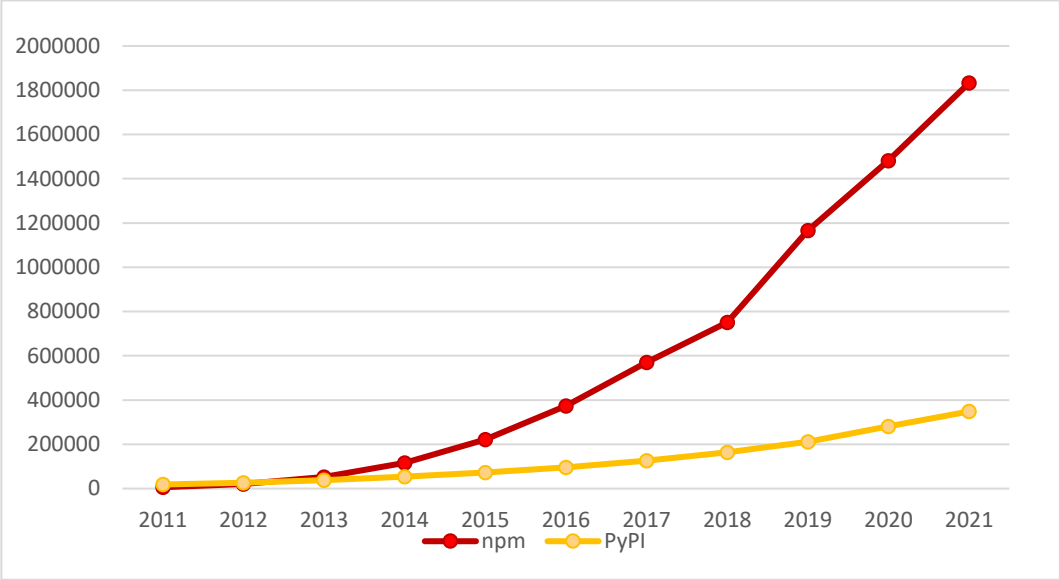
PyPI (Python Package Index) är ett kodförråd för programmeringsspråket Python. Totalt upprätthåller PyPI över 350 000 kodbibliotek i sitt register [12]. Utvecklare

kan använda sig av pakethanteringsverktyget pip, vilket är ett kommandoradsgränssnitt, för att interagera med registret [13].

För att ladda upp ett bibliotek till PyPI krävs förutom ett PyPI-konto ett projekt med en setup.py-, setup.cfg- och LICENSE.txt-fil [14]. Kodbiblioteksberoenden samt dess version för ett Python-projekt definieras i en skild requirements.txt-fil. Figur 6 visar ett exempel på innehållet av en requirements.txt-fil.

```
docopt==0.6.1
keyring==4.1.1
```

Figur 5 Exempel på innehåll av requirements.txt-fil.



Figur 6 Lagrade kodbibliotek i kodförråden npm samt PyPI från år 2011 till 2021. Bibliotek med flera versioner räknas som ett och samma bibliotek.

2.5 Semantisk versionsnumrering

Semantisk versionsnumrering består av regler om hur versioner hanteras inom mjukvara. Versionsnumret består av tre numeriska delar i formatet MAJOR.MINOR.PATCH som uppdateras enligt förändringarna i koden. Varje del betecknar följande [15]:

- **MAJOR:** En förändring introduceras som bryter bakåtkompatibiliteten.

- **MINOR:** En förändring som introducerar nya funktioner som är bakåtkompatibla.
- **PATCH:** En förändring som introducerar en bakåtkompatibel buggfix.

Med bakåtkompatibilitet menas att den uppdaterade versionen av ett bibliotek fungerar likadant som en föregående version och kräver inga ändringar av huvudprojektet för att vara kompatibel.

Både npm och PyPI rekommenderar utvecklare att använda sig av semantisk versionsnumrering för sina kodbibliotek men standarden är inte påtvingad. Däremot kräver de båda att versionsnumreringen inkrementeras sekventiellt [16] [17]. Ett bibliotek med versionsnummer 1.4.0 kan inte publiceras efter att versionen 1.5.0 har publicerats.

2.6 Open Web Application Security Project

Open Web Application Security Project eller OWASP [18] jobbar för att förbättra säkerheten av mjukvara. OWASP erbjuder gratis artiklar, metoder, strategier och dokumentation inom området säkerhet för webbapplikationer.

OWASP publiceras med jämna mellanrum en rapport över de tio mest kritiska riskerna för webbapplikationssäkerhet, “OWASP Top Ten”. Rapporten sammanställs av experter inom säkerhet som analyserar data från olika organisationer [19]. Enligt rapporten år 2017 rankades användningen av komponenter med kända sårbarheter på plats nummer nio. Rapporten som publicerades år 2021 visade att användningen av komponenter med kända sårbarheter hade stigit tre platser, till nummer sex. Rapporten motiverar förflyttningen genom att förklara att problemet med komponenter med kända sårbarheter är en fortsatt utmaning att testa emot och evaluera dess risker [19].

Plats	Säkerhetsrisk
A01	Brister i åtkomstkontroll.
A02	Kryptografiska brister.
A03	Injektion på grund av bland annat ovaliderad inmatningsdata och brister i parsning av data.
A04	Osäkra designmönster och implementeringar.
A05	Brister i säkerhetskonfigurering.
A06	Användning av sårbara eller föråldrade komponenter eller bibliotek.
A07	Brister inom identifiering och autentisering.
A08	Brister inom dataintegritet.
A09	Otillräcklig loggning och övervakning av sårbarheter.
A10	Server Side Request Forgery (SSRF).

Figur 7 OWASP Top Ten 2021 [19].

3 Sårbarheter

Med en sårbarhet avses i regel en säkerhetsbrist som oavsiktligt introducerats i mjukvara på grund av oförsiktighet eller brister i utvecklingsprocessen. Förutom oavsiktliga säkerhetsbrister existerar även sårbarheter i form av avsiktliga bakdörrar eller skadlig kod som inkluderats i mjukvaran. I det här kapitlet förklaras sårbarheter, sårbarhetsdatabaser och attackvektorer. Dessutom presenteras två tidigare sårbarheter som avsiktligt introducerats i kodbibliotek och strategier som skulle kunnat förhindra dem diskuteras.

3.1 Sårbarhetsdatabaser

Nya sårbarheter hittas dagligen i mjukvara. För att förenkla mitigationsprocessen för utvecklare och slutanvändare måste sårbarheterna avslöjas och lagras på en centraliserad plats. Det finns speciella sårbarhetsdatabaser, både öppna och kommersiella, som är specifikt inriktat på sårbarheter inom öppen källkod. Ett exempel på en sårbarhetsdatabaser inriktad på öppen källkod är OSV.dev [20]. OSV.dev sammanställer sårbarheter från olika källor, bland annat GitHub Security Advisories för sårbarheter gällande npm-bibliotek och Python Packaging Advisory Database för PyPI-bibliotek [20]. Figur 1 visar utvecklingen av antalet kända sårbarheter i öppen källkod från år 2009 till 2021.

Sårbarhetsdatabasers primära funktion är att informera användare och utvecklare om sårbarheter i mjukvara. Rapporterade sårbarheter blir tilldelade en unik identifierare och en kort beskrivning över sårbarheten. Vilka versionsnummer av biblioteket som påverkas inkluderas också i rapporten.

Eftersom vem som helst har tillgång till sårbarhetsdatabaser har även attackerare möjlighet att använda informationen till att strategiskt utnyttja sårbarheterna. Enligt Tenable [21] är tiden mellan att en sårbarhet rapporteras till att sårbarheten utnyttjas av attackerare endast 5.5 dagar. Däremot är tiden för att slutanvändare eller utvecklare identifierar och åtgärdar problemet i genomsnitt 13 dagar efter att sårbarheten upptäckts. Det betyder att attackerare i genomsnitt har ett sju dagars försprång att utnyttja en känd sårbarhet.

3.2 Attackvektorer

En attackvektor är en metod eller strategi för att utnyttja en sårbarhet i mjukvara. Exempel på traditionella attackvektorer utanför ett kodbibliotekskontext är nätfiske eller stöld av inloggningsuppgifter [22].

Till skillnad från oavsiktliga sårbarheter i öppen källkod existerar även sårbarheter som avsiktligen introduceras eller riktas mot ett kodbibliotek. Av GitHub Advisory Databases 6706 dokumenterade sårbarheter tillhör 335 sårbarheter kategorin CWE:506 [23], vilket betyder att sårbarhetens orsak beror på skadlig kod som inkluderats i biblioteket. Till avsiktliga attackvektorer hör bland annat typosquatting, beroendeförvirring och kodinjektion.

3.2.1 Typosquatting

Med typosquatting innebär att ett kodbibliotek som laddas upp till ett kodförråd ges ett namn som är väldigt liknande ett annat populärt bibliotek. För att förhindra att ett falskt paket ska bli upptäckt erbjuder de ofta exakt samma funktionalitet som det originella paketet och inkluderar utöver den legitima koden ofta skadlig kod. Attacken förlitar sig på att utvecklare oavsiktligt skriver in fel namn på paketet de vill installera. Typosquatting är en av de mest triviala attackvektorerna som används.

Kodförrådet npm har implementerat ett skydd mot typosquatting. Genom att reservera namn som liknar populära kodbiblioteks namn förhindras utvecklare att oavsiktligen använda bibliotek med skadlig kod i deras kodbas [24]. Exempelvis är namnet “mongose” reserverat (för det officiella biblitoteket “mongoose”) och är inte tillgängligt att använda för nya publicerade kodbibliotek.

3.2.2 Beroendeförvirring

Beroendeförvirring sker då ett pakethanteringsverktyg hittar ett kodbibiliotek med samma namn från flera kodförråd. Organisationer upprätthåller och använder ofta privata bibliotek som inte finns tillgängliga i offentliga kodförråd och endast är tillgängliga inom organisationen. Pakethanteringsverktygen har ett förinställt kodförråd (registry.npmjs.org för npm och pypi.org/simple för pip) men kan även

konfigureras att använda andra kodförråd. Kommandoradsargument `extra-index-url` används för pip och kommandot `npm config set registry <registry url>` för npm. som definierar på vilken URL eller webbadress verktyget ska hitta biblioteken som ska installeras.

På grund av den förinställda konfigurationen pakethanteringsverktyg installeras med, existerar en risk att bli offer för beroendeförvirring. Exempelvis söker pakethanteringsverktyget pip efter kodbiblioteksberoenden i följande ordning: [25]

- Kontrollerar om biblioteket existerar i de kodförråd som registrerats med kommandoargument `extra-index-url`.
- Kontrollerar om biblioteket existerar i det offentliga kodförrådet.
- Om biblioteket endast existerar i ett kodförråd, används den som källa.
- Om biblioteket existerar i flera kodförråd används det kodförråd som innehar den högsta versionen av biblioteket.

Detta innebär att ifall ett kodbibliotek med samma namn som det privata biblioteket laddas upp på ett offentligt kodförråd med en högre versionsnummer kommer pakethanteringsverktyget använda det offentliga kodförrådet istället för det avsedda privata kodförrådet. npm är inte utsatt för beroendeförvirring så länge som verktyget konfigurerats korrekt.

3.2.3 Kodinjektion

Med kodinjektion avses att skadlig kod läggs till i ett kodbibliotek. Tidigare exempel av denna attackvektor har använt sig av obfuskering, eller avsiktlig tillkrångling av den skadliga koden. Obfuskering används för att förhindra koden från att upptäckas från både automatiska analysverktyg och människor [26]. Attackerna som presenteras i sektion 3.3.1 och 3.3.2 baseras på kodinjektion.

3.3 Tidigare avsiktliga attacker

Det finns flertalet exempel på tidigare avsiktliga attacker som utförts med hjälp av kodbibliotek. Populära bibliotek som påverkar många organisationer och tjänster får ofta mer uppmärksamhet i media. Till dessa hör bland annat biblioteket *event-stream* vars mål var att komma över kryptovalutor och personlig information samt *node-ipc* som avsiktligen saboterades av sin underhållare.

3.3.1 event-stream

event-stream är ett JavaScript-bibliotek som publicerats i kodförrådet npm. Biblioteket utsattes år 2018 för en social engineering-attack. Attackeraren framställde sig som en underhållare av projektet och fick rättighet av den originala underhållaren att publicera nya versioner av biblioteket i kodförrådet.

Den skadliga koden lades till i form av kodinjektion och riktades mot företaget Copays mobilapp, en digital plånbok för kryptovalutor. Den skadliga koden kördes endast ifall denna specifika miljö upptäcktes. Koden var designad att exekvera följande funktioner:

- Upptäckte miljön som koden körs i. Om rätt utvecklingsmiljö ej upptäcks returnerar koden.
- Granskade saldot för Bitcoin och Bitcoin Cash i offrets digitala plånbok.
- Om saldot överstiger 100 Bitcoin eller 1000 Bitcoin Cash skickas summan till attackerarens plånbok och offrets personliga data samt privata nycklar skickas till en webbserver. Webbservern tillhör högst troligen attackeraren.

Eftersom koden endast kördes då en specifik utvecklingsmiljö upptäcktes påverkades andra utvecklare inte av attacken även om de hade installerat den [26]. Det ledde till att den skadliga koden inte upptäcktes förrän 76 dagar efter att versionen av biblioteket med skadlig kod hade laddats upp. Under dessa 76 dagar laddades ner och installerades biblioteket över 24 miljoner gånger.

Denna typ av attack kan delvis mitigeras genom att endast tillåta klienten att ansluta till kända adresser eller domännamn. Även låsning av versionsnummer för kodbiblioteket, som beskrivs i sektion 4.3, hade förhindrat denna attack.

3.3.2 node-ipc

node-ipc är en JavaScript-modul som I mars år 2022 blev saboterad av underhållaren. Underhållaren laddade upp en ny version, 9.2.2, på kodförrådet npm den 15 mars 2022. Den uppladdade versionen innehåller skadlig kod som riktas mot utvecklare och organisationer från Ryssland och Belarus. Koden är designad att exekvera följande funktioner: [27] [28]

- Koden har en 25% chans att köras.
- Skickar en förfrågan till en tredjeparts webbtjänst som returnerar klientens land utgående från IP-adress.
- Koden fortsätter endast ifall det returnerade landet är lika med “russia” eller “belarus”.
- Försöker skriva över alla filer i filsystemet med en textfil.

Denna typ av attack kan mitigeras genom att låsa versionsnummer eller genom att isolera mjukvaran från resten av värdsystemet med hjälp av en virtuell maskin eller container. Detta skulle leda till att inga filer på värdsystemet skulle påverkas.

4 Säkerhetsstrategier

Det finns olika sätt att mitigera risken av användning av kodbibliotek. Detta kapitel tar upp några säkerhetsstrategier vid användning av kodbibliotek med öppen källkod. Strategierna är sammanställda från företagen Microsoft [29], Snyk [30] och Contrast Security [1]. För utvecklare som konsumerar kodbibliotek är strategier som nogrann evaluering och val av kodbibliotek, utnyttjande av låsfiler och skanning av sårbara kodbibliotek relevanta. I det här kapitlet evalueras de ovannämnda strategierna samt diskuteras hur strategierna kan tillämpas vid användning av kodförråden npm och PyPI.

4.1 Val av kodbibliotek

Då man väljer ett kodbibliotek bör olika säkerhetsaspekter noga evalueras. Det lönar sig att överväga ifall man överhuvudtaget behöver introducera ett nytt kodbibliotek i sitt projekt eller om funktionaliteten är så trivial att det går att implementera i den egna kodbasen [6]. Om det finns flera kodbibliotek som erbjuder samma funktionalitet är det i regel bättre att välja det paket som är beroende av minst antal andra kodbibliotek. Det viktigaste är att vara medveten om antalet kodbibliotek och vilka kodbibliotek man använder i sitt projekt [29].

Enligt Gustavsson [31] kan arbetsbördan för att underhålla ett helt kodbiblioteket för att endast använda en enda funktion vara större än att att implementera denna funktion i den egna kodbasen. Ifall fler funktioner används kan arbetsbördan att upprätthålla den egna implementationen däremot vara större än att introducera ett kodbibliotek.

4.2 Analysera kodbibliotekets användningsgrad och kvalitet

Med användningsgrad avses hur populärt eller hur många användare kodbiblioteket har. Ett bibliotek som aktivt används är i regel grundligt beprövad och dokumenterad och är mindre sannolik att överges av underhållare. Användningsgraden kan bestämmas genom att granska antalet nedladdningar samt hur många projekt som är beroende av biblioteket.

Kvaliteten av ett kodbibliotek kan vara svår att bestämma. Använingsgraden är ofta en indikator att kvaliteten på projektet är godtyckligt. Utöver användingsgraden kan kvaliteten bestämmas genom att granska källkoden och dokumentationen.

En ofta avgörande aspekt är bibliotekets underhåll. Det är möjligt att få en snabb överblick över detta genom att granska när en ny version av bibliotek senast har lanserats och hur ofta nya versioner lanseras överlag [29]. Om bibliotekets källkod även finns tillgängligt på en versionshanteringsplattform, ta även dess uppdateringsintervall i beaktande. Ett projekt som inte har uppdaterats på en längre tid kan vara utsatt för sårbarheter.

Attackerare kan dock skapa kodbiblitoek som ser legitima ut och lockar användare att installera det för att sedan vid ett senare tillfälle använda dem för att distribuera skadlig kod [32]. Tidigare exempel existerar också där underhållare av kodbibliotek avsiktligen saboterat sina egna kodbibliotek. Se sektion 3.3.2.

4.3 Låsning av versionsnummer

Genom att låsa bibliotek till exakta versionsnummer försäkras att oavsiktliga uppdateringar inte kan introducera skadlig kod i projektet. Genom olika prefix kan regler för vilka versionsnummer av kodbibliotek som tillåts installeras i ett projekt definieras [11].

I figur 10 visas hur olika prefix påverkar vilka versionsnummer som tillåts. I sektion 2.3 beskrivs semantisk versionsnumrering.

Prefix	**Exempel**	**Beskrivning**
version	1.0.0	Måste matcha versionen exakt
>version	>1.0.0	Måste vara större än versionen
>=version	>=1.0.0	Måste vara större än eller lika med versionen
<version	<1.0.0	Måste vara mindre än versionen
<=version	<=1.0.0	Måste vara mindre än eller lika med versionen
~version	~1.0.0	Tillåter nyare versioner på PATCH-nivå enligt semantisk versionsnumrering
^version	^1.0.0	Tillåter nyare versioner på MINOR- och PATCH-nivå enligt semantisk versionsnumrering

Figur 8 Olika prefix som kan användas för att begränsa tillåtna versionsnummer.

4.4 Automatisk analys och skanning

Med det stora antalet kodbibliotek som importeras i dagens mjukvara är det inte hållbart för den enskilda utvecklaren att manuellt läsa och analysera all tredjepartskod som används. Därför finns olika analysverktyg som automatiskt skannar projekt för kända sårbarheter. Den här typen av automatisk skanning är beroende av att sårbarheter rapporteras och förs in i en sårbarhetsdatabas. Kända sårbarheter matchas med kodbiblioteken som är installerade i kodbasen.

4.4.1 npm

Kodförrådet npm använder sig av sårbarhetsdatabasen GitHub Advisory Database [33] för att analysera ett projekts sårbarheter. Funktionen är inbyggd i pakethanteringsverktyget npm. Sårbarhetsanalysen sker automatiskt då kodbibliotek installeras. I figur 11 och 12 visas hur utvecklaren informeras om sårbarheterna i kommandoradsgränssnittet.

```
$ npm install fresh
added 1 package, and audited 1 package in 1s

found 0 vulnerabilites
```

Figur 9 Information som visas då inga sårbarheter upptäckts.

```
$ npm install fresh@0.5.1
added 1 package, and audited 1 package in 1s

1 high severity vulnerability

Run `npm audit` for details.
```

Figur 10 Information som visas då en sårbarhet upptäckts.

Dessutom erbjuder npm ett inbyggt kommando *npm audit* för att visa ytterligare information om sårbarheten. I figur 13 visas utskriften för kommandot `npm audit`. Informationen innehåller typ av sårbarhet som upptäckts, hur allvarlig sårbarheten

är (låg, lindrig, hög eller kritisk), vilka versionsnummer av biblioteket som påverkas samt en länk till GitHub Advisory Database för denna sårbarhet.

Om en nyare version av biblioteket existerar där sårbarheten har åtgärdats visar kommandot även detta. Genom att använda kommandot `npm audit fix` försöker npm uppdatera alla bibliotek i projektet genom att uppdatera dem till versioner där sårbarheten åtgärdats.

```
$ npm audit

fresh  <0.5.2
Severity: high
Regular Expression Denial of Service in fresh –
https://github.com/advisories/GHSA-9qj9-36jm-prpv

fix available via `npm audit fix`
node_modules/fresh
```

Figur 11 Information som visas då kommandot 'npm audit' körs.

4.4.2 PyPI

Kodförrådet PyPI har inget inbyggt verktyg för analys av sårbarheter. Däremot finns modulen “pip-audit” som delvis implementerar samma funktionalitet som npm:s audit-kommando. Modulen analyserar installerade PyPI-bibliotek och matchar dem mot kända sårbarheter i Python Packaging Advisory Database [34]. I figur 14 visas utskriften för kommandot `pip-audit`.

Liksom npm:s kommando `npm audit fix` har pip-audit ett liknande kommando `pip-audit –fix` som automatiskt försöker åtgärda sårbarheter genom att uppdatera de påverkade kodbiblioteken till en nyare version.

```
$ pip-audit

Found 3 known vulnerabilities in 2 packages
Name         Version ID            Fix Versions
------------ ------- ------------- ------------
cryptography 2.8     PYSEC-2021-62 3.2.1
httplib2     0.14.0  PYSEC-2020-46 0.18.0
httplib2     0.14.0  PYSEC-2021-16 0.19.0
```

Figur 12 Information som visas då kommandot 'pip-audit' körs.

4.5 Andra rekommendationer

Enligt Contrast Security [1] bör utvecklare övervaka sårbarheter som påverkar kodbibliotek i användning och vidta åtgärder för att hålla sina kodbibliotek uppdaterade. Contrast Security hävdar också att organisationer skapa en lista som definierar vilka kodbibliotek får användas i organisationens mjukvara.

Microsoft [29] hävdar att organisationer bör skapa en strategi för att åtgärda rapporterade sårbarheter enligt samma princip som organisationens allmäna säkerhetsstrategi. Enligt Microsoft bör också aktiv testing av tredjepartskod utföras.

5 Konklusion

Återanvändning av kod är ett koncept som tillämpas i de flesta organisationer och andelen kod i mjukvara som kommer från tredjepartskällor överstiger i dagens läge 80 procent. Offentliga kodbibliotek introducerar en risk för organisationer. Studien presenterar olika sätt hur bibliotek med öppen källkod fungerar som en dörr in till en organisations mjukvarukedja.

I studien beskrivs olika avsiktliga attackvektorer som riktas mot användare av kodbibliotek eller som använder kodbibliotek och kodförråd som medel för distribuering. Rekommendationer på säkerhetsstrategier som kan implementeras och bästa praxis från olika företag och andra källor tas upp. Sårbarhetsdatabasers roll i mitigering av sårbarheter inom öppen källkod presenteras och dess effektivitet diskuteras.

Analysen inriktar sig på de två mest populära programmeringsspråken idag, JavaScript och Python. Programmeringsspråkens officiella pakethanteringsverktyg och kodförråd evaluerades och dess funktionalitet för att mitigera sårbarheter inom mjukvara jämfördes med varandra.

Referenser

[1] Contrast Security, ”The Unfortunate Reality of Insecure Libraries”.

[2] WhiteSource, ”The State of Open Source Security,” [Online]. Available: https://www.whitesourcesoftware.com/wp-content/media/2021/04/the-state-of-open-source-vulnerabilities-2021.pdf.

[3] ”Search Code,” GitHub, Inc., 2022. [Online]. Available: https://github.com/search.

[4] ”The MIT License,” [Online]. Available: https://opensource.org/licenses/MIT. [Använd 27 2 2022].

[5] ”GNU General Public License version 3,” [Online]. Available: https://opensource.org/licenses/GPL-3.0. [Använd 27 02 2022].

[6] D. Spinellis, ”APIs, Libraries, and Code,” 2012.

[7] GitHub, ”The State of the Octoverse | The State of the Octoverse celebrates a year of building across teams, time zones, and millions of merged pull requests.,” GitHub, 2019. [Online]. Available: https://octoverse.github.com/2019/.

[8] R. K. V. e. al., ”Security Issues in Language-based Software Ecosystems,” 2019.

[9] ”About npm,” npm, [Online]. Available: https://docs.npmjs.com/about-npm. [Använd 23 February 2022].

[10] ”npm CLI,” npm, [Online]. Available: https://docs.npmjs.com/cli/v8.

[11] ”package.json,” npm, [Online]. Available: https://docs.npmjs.com/cli/v8/configuring-npm/package-json.

[12] [Online]. Available: https://pypi.org/.

[13] ”pip documentation,” [Online]. Available: https://pip.pypa.io/en/stable/.

[14] [Online]. Available: https://packaging.python.org/en/latest/tutorials/packaging-projects.

[15] T. Preston-Werner, ”Semantic Versioning,” [Online]. Available: https://semver.org/.

[16] ”About semantic versioning | npm Docs,” [Online]. Available: https://docs.npmjs.com/about-semantic-versioning. [Använd 22 February 2022].

[17] ”PEP 440 -- Version Identification and Dependency Specification | Python.org,” [Online]. Available: https://www.python.org/dev/peps/pep-0440. [Använd 22 February 2022].

[18] ”Open Web Application Security Project,” [Online]. Available: https://owasp.org/.

[19] ”OWASP Top Ten Web Application Security Risks,” The OWASP Foundation Inc., 2021. [Online]. Available: https://owasp.org/www-project-top-ten/.

[20] ”OSV,” [Online]. Available: https://osv.dev.

[21] Tenable, ”Quantifying the attacker’s first-mover advantage,” Tenable, 2018. [Online]. Available: https://www.tenable.com/blog/quantifying-the-attacker-s-first-mover-advantage.

[22] ”What is an attack vector?,” Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/security/glossary/attack-vector/.

[23] ”GitHub Advisory Database,” GitHub, [Online]. Available: https://github.com/advisories?query=malicious+cwe%3A506.

[24] ”Security holding package,” [Online]. Available: https://github.com/npm/security-holder.

[25] Microsoft, ”3 Ways to Mitigate Risk When Using Private Package Feeds,” 23 09 2021. [Online]. Available: https://azure.microsoft.com/en-us/resources/3-

ways-to-mitigate-risk-using-private-package-feeds/.

[26] ”Details about the event-stream incident,” npm, 27 11 2018. [Online]. Available: https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident.

[27] [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-23812.

[28] [Online]. Available: https://github.com/RIAEvangelist/node-ipc/issues/233.

[29] ”Using Open Source,” Microsoft, [Online]. Available: https://www.microsoft.com/en-us/securityengineering/opensource.

[30] ”https://snyk.io/blog/ten-npm-security-best-practices/,” Snyk, 19 02 2019. [Online]. Available: https://snyk.io/blog/ten-npm-security-best-practices/.

[31] T. Gustavsson, ”Managing the Open Source Dependency,” *Computer,* vol. 53, nr 2, pp. 83-87, 2020.

[32] A. Sharma, ”Dev corrupts NPM libs 'colors' and 'faker' breaking thousands of apps,” Bleeping Computer, 09 01 2022. [Online]. Available: https://www.bleepingcomputer.com/news/security/dev-corrupts-npm-libs-colors-and-faker-breaking-thousands-of-apps/.

[33] ”GitHub Advisory Database,” GitHub, Inc., 2022. [Online]. Available: https://github.com/advisories.

[34] [Online]. Available: https://github.com/pypa/advisory-database.

[35] G. Podjarny, Securing Open Source Libraries, O’Reilly Media, 2019.

[36] B. Fitzgerald, 2004. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1310249.

[37] [Online]. Available: https://docs.npmjs.com/cli/v7/configuring-npm/package-json.

[38] [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.