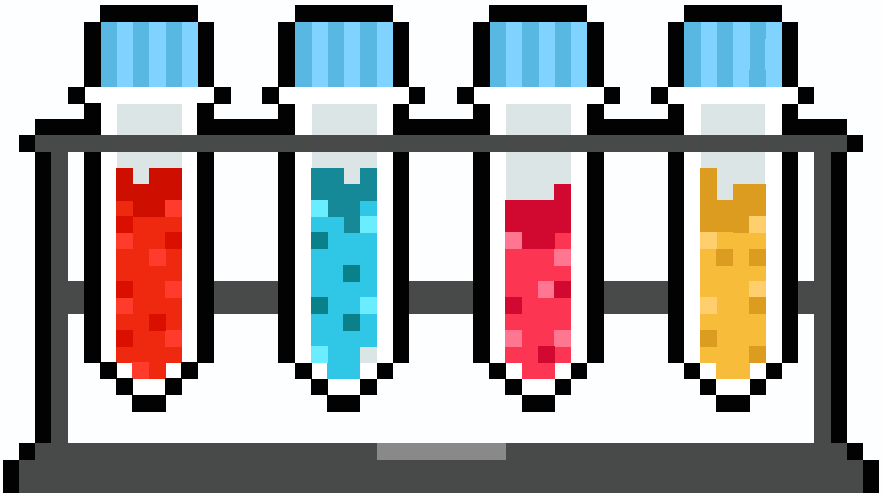


Principer för konstruktion av datamängder med mjukvarusårbarheter

För utvärdering av mjukvaruverktyg som identifierar sårbarheter

Christian Vestlund, Christian Gustavsson, Daniel Eidenskog

FOI-R--5891--SE

December 2025

FOI

Christian Vestlund, Christian Gustavsson, Daniel Eidenskog

Principer för konstruktion av datamängder med mjukvarusårbarheter

För utvärdering av mjukvaruverktyg som identifierar sårbarheter

Titel	Principer för konstruktion av datamängder med mjukvarusårbarheter – För utvärdering av mjukvaruverktyg som identifierar sårbarheter
Title	Principles for Constructing Datasets with Vulnerable Software - For Evaluation of Tools that Identify Software Vulnerabilities
Rapportnr/Report no	FOI-R--5891--SE
Månad/Month	December
Utgivningsår/Year	2025
Antal sidor/Pages	48
ISSN	1650-1942
Uppdragsgivare/Client	Försvarsmakten
Forskningsområde	Cyberförsvar och cybersäkerhet
FoT-område	Operationer i cyberdomänen
Projektnr/Project no	E38562
Godkänd av/Approved by	Linda Sjödin
Ansvarig avdelning	Cyberförsvar och Ledningsteknik
Exportkontroll	Innehållet är granskat och omfattar ingen information som är underställd exportkontrollagstiftningen.

Bild/Cover	Shutterstock

Sammanfattning

Verktyg som kan identifiera sårbarheter i mjukvara spelar en viktig roll för att skapa säker mjukvara. Det är dock inte uppenbart hur bra sådana verktyg är på att hitta olika typer av sårbarheter eller hur verktygen bör utvärderas. I forskningslitteraturen används olika datamängder med sårbarheter för att utvärdera och jämföra verktyg.

Denna rapport presenterar en litteraturstudie som undersöker vilka datamängder som finns och hur dessa presenteras i litteraturen. Bland annat undersöks vilken kritik som presenteras mot dem och vilka egenskaper hos datamängderna som lyfts fram i publikationerna. I studien har 54 olika datamängder eller verktyg som kan generera datamängder identifierats. Därtill har tio typer av kritik och 22 önskvärda egenskaper identifierats.

Resultaten visar en trend mot att datamängder som baseras på sårbarheter i verklig mjukvara har blivit vanligare medan det skapas en mindre andel datamängder med konstruerade eller injicerade sårbarheter. Brist på realism är dessutom en vanlig kritik mot datamängder, vilket också avspeglas i att realism är en av de vanligaste önskvärda egenskaperna. Studiens resultat visar dock att det saknas en etablerad gemensam bild över vilka egenskaper som behövs för att skapa bra datamängder.

Nyckelord: mjukvarusäkerhet, verktyg, datamängder, sårbarheter

Summary

Tools that can detect software vulnerabilities play a critical role in creating secure software. However, it is not clear how effective such tools are at identifying different types of vulnerabilities or how these tools should be evaluated. In research literature, various types of datasets with vulnerable code are used to evaluate and compare such tools.

This report presents a literature study that identifies datasets and examines how they are presented in research literature. This study also examines the criticism against datasets and the characteristics highlighted in the publications. The study has identified 54 different datasets or tools that can generate datasets. Additionally, ten types of criticisms and 22 desirable characteristics have been identified.

The results show a trend toward datasets based on vulnerabilities in real software becoming more common, while fewer datasets with constructed or injected vulnerabilities are being created. Moreover, lack of realism is a common criticism against datasets, which is also reflected in realism being one of the most common desirable characteristics. This study's results indicate a lack of an established common understanding of characteristics needed to create high-quality datasets.

Keywords: software security, tools, datasets, vulnerabilities

Innehåll

1 Inledning

Sårbar mjukvara utgör en stor risk i dagens digitaliserade samhälle, där enskilda sårbarheter kan innebära att samhällskritiska funktioner störs ut. Detta belyses också av Jen Easterly, tidigare chef på amerikanska CISA[1] [1] (författarnas översättning):

> Vi har normaliserat det faktum att tekniska produkter släpps ut på marknaden med dussintals, hundratals eller tusentals defekter, trots att en så dålig konstruktion skulle vara oacceptabel inom alla andra kritiska områden. [...]
>
> Vi har normaliserat det faktum att säkerhet delegeras till ”IT-människorna” i mindre organisationer eller till en informationssäkerhetschef i större företag, men få har resurserna, inflytandet eller ansvarsskyldigheten för att ge incitament till att använda produkter där säkerhet prioriteras korrekt gentemot kostnad, marknadsintroduktion och funktioner.

Verktyg som kan identifiera mjukvarusårbarheter har en viktig roll i att stärka säkerheten i mjukvara genom att förenkla arbetet med att hitta sårbarheterna innan dessa sprids och mjukvaran tas i drift. Att skapa evidens för säkerhetsnivån i IT-systemen är en viktig del i säkerhetsarbetet [2], och där kan tillförlitliga och effektiva verktyg vara viktiga pusselbitar.

I FOI-rapporten *Tekniker och verktyg som identifierar mjukvarusårbarheter* av Gustavsson m.fl. undersöktes forskningspublikationer mellan 2014 och 2024 för att identifiera verktyg för att identifiera sårbarheter [3]. Två av slutsatserna i rapporten var att utvärderingen av verktygen varierade stort och att verktygen med högst mognadsgrad fokuserar på få och närbesläktade sårbarheter.

Att jämföra verktyg för att analysera mjukvara är inte en enkel uppgift. Det saknas standardiserade tillvägagångssätt för att jämföra verktyg [4] och jämförelser i akademiska publikationer har stora variationer [5], [6]. I vissa fall används datamängder, men vilka som används varierar mellan publikationer. I andra fall

[1] Cybersecurity and Infrastructure Security Agency

används verklig mjukvara, vilket medför problem när de mjukvaror som används för testning ändras. Jensen m.fl. genomförde en inledande studie av datamängder som kan nyttjas för att utvärdera verktyg där en av studiens slutsatser var att fortsatt forskning behövs för att avgöra vad som utgör en bra datamängd [7].

Fokus för den här rapporten är att identifiera de principer som bör användas vid konstruktion av datamängder med sårbarheter som ska användas för utvärdering av verktyg som identifierar mjukvarusårbarheter. Målsättningen med principerna är att de ska leda till datamängder som är bättre lämpade för utvärderingen.

1.1 Syfte och mål

Rapporten ingår i ett projekt vars syfte är att bygga kunskap om olika tekniker och verktyg som kan användas för att identifiera mjukvarusårbarheter. Målet med studien är att undersöka principer för att konstruera datamängder med sårbarheter. I längden ska principerna kunna användas för att utvärdera eller konstruera datamängder för utvärdering av verktyg.

Rapporten besvarar nedanstående forskningsfrågor som utgår från tidigare identifierad kritik mot datamängder med sårbarheter:

1. Vilka begränsningar finns i befintliga datamängder med sårbar mjukvara?
2. Vilka principer bör användas vid konstruktionen av datamängder med sårbar mjukvara för att förbättra relevans, kompletthet och användbarhet?
3. Vilka principer har utvärderats för befintliga datamängder med sårbar mjukvara?

Frågorna ovan besvaras genom att undersöka både befintliga datamängder och verktyg som skapar datamängder för teständamål.

Rapporten vänder sig främst till personer som har kunskap om mjukvarusäkerhet och säkerhetstestning av mjukvara.

1.2 Avgränsningar

Rapporten gör inte anspråk på att vara heltäckande utan ger en översiktlig bild av kritik mot tillgängliga datamängder med sårbarheter. Genomgången av kritik mot datamängder fokuserar på akademiska publikationer, vilket innebär att kritik som presenterats i annan litteratur inte beaktas.

2 Metod

För att identifiera underlag som kan nyttjas för att besvara forskningsfrågorna genomfördes en litteraturstudie för att hitta akademiska publikationer[2] som tar upp datamängder som är avsedda att testa verktyg för sårbarhetsupptäckt eller verktyg som skapar sådana datamängder. De sökta datamängderna innehåller samlingar av sårbar mjukvara i någon form, såsom källkod eller binärer. Publikationer om de identifierade datamängderna och verktygen användes sedan som indata för att besvara forskningsfrågorna i studien.

För att bredda möjligheten att identifiera datamängder genomfördes sökningar i två separata spår utifrån två olika söksträngar. De två spåren är avsedda att hitta datamängder dels genom publikationer som direkt beskriver sådana datamängder, dels genom publikationer som indirekt tar upp datamängder, exempelvis genom beskrivningar av vilka datamängder som används vid testning av verktyg som identifierar mjukvarusårbarheter.

Valet att även genomföra sökningar efter publikationer som jämför verktyg som identifierar mjukvarusårbarheter följer av att forskargruppen[3] ansåg det troligt att dessa publikationer även diskuterar hur verktygen har utvärderats och vilka datamängder som använts för detta. Studien är inkluderande i termer av datamängder som används av olika typer av verktyg, exempelvis statiska kodanalysverktyg, fuzzers och AI-baserade[4] analysverktyg. Det finns närliggande ämnen, exempelvis datamängder som enbart berör buggar och inte sårbarheter, som skulle kunna vara relevanta men som exkluderats i denna litteraturstudie.

Litteraturstudien baseras på den metod som Kitchenham m.fl. tagit fram för systematiska litteraturstudier inom mjukvaruteknik [8]. Metoden som de presenterar passar dock inte fullt ut för denna studie då litteratursökningarna i detta fall huvudsakligen varit ämnade till att identifiera omnämnanden av befintliga datamängder och verktyg för att skapa datamängder utan något behov

[2] I praktiken utgörs de hittade publikationerna huvudsakligen av journalartiklar och konferensbidrag.

[3] Forskargruppen utgörs av författarna till denna rapport.

[4] AI används här som ett samlingsbegrepp för ett flertal tekniker, såsom maskininlärning och andra metoder inom artificiell intelligens.

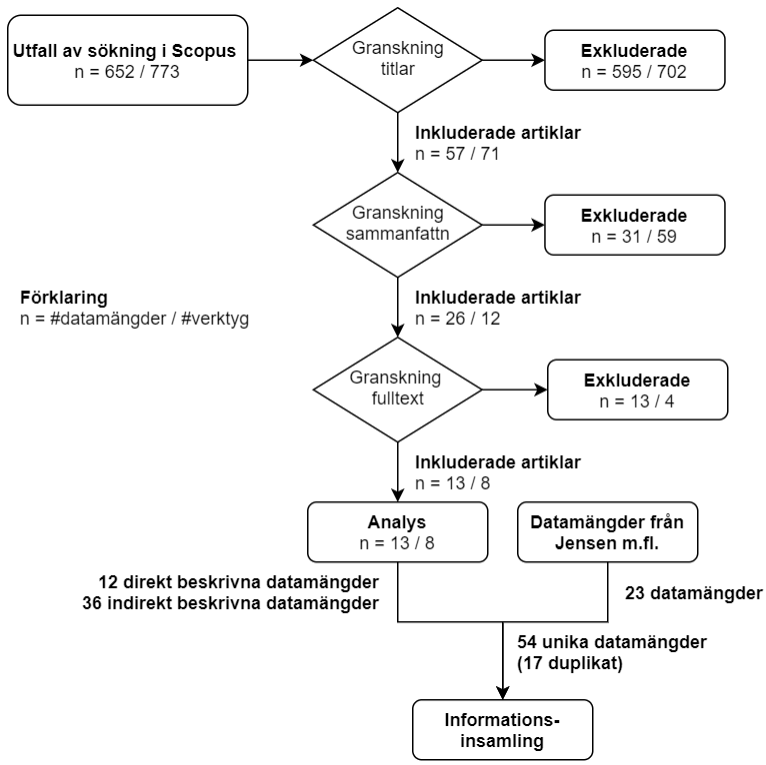
av en djupare kvalitetsbedömning av dessa källor i sig. Den djupare analysen kommer först i det efterföljande steget, där ursprungskällorna för datamängder och verktyg används för att samla in information.

Rapportens metod består av följande huvudmoment:

1. *Litteratursökningar*
   Två separata sökningar genomfördes i Scopus med hjälp av utvalda sökord inom de två spåren.
2. *Gallring på titlar*
   En första gallring genomfördes genom att titlarna granskades och uppenbart ointressanta träffar togs bort.
3. *Gallring på sammanfattningar*
   En andra gallring genomfördes genom granskning av sammanfattningarna.
4. *Gallring på fulltext*
   En sista gallring genomfördes genom att publikationerna lästes i sin helhet för att gallra bort eventuella kvarvarande ointressanta publikationer.
5. *Analys*
   De kvarstående publikationerna analyserades för att identifiera datamängder som beskrivs direkt eller indirekt[5]. Identifierade publikationer som direkt beskriver datamängder kompletterades med refererade publikationer som beskriver de indirekt omnämnda datamängderna. Dessa publikationer användes sedan tillsammans med tidigare identifierade datamängder från Jensen m.fl. [7] som indata till nästa steg.
6. *Informationsinsamling*
   Den sammanlagda mängden av publikationer som beskriver datamängder och verktyg som skapar datamängder granskades och analyserades. Analysen bygger på en tematisering för att samla in och sammanställa information som är relevant för att besvara forskningsfrågorna.

[5] Med indirekt beskrivna datamängder menas sådana som omnämnts i identifierade publikationer men som beskrivs i en refererad publikation.

Metoden från Kitchenham m.fl. [8] följs i princip fram till steg 4 för att sedan endast utgöra inspiration för de aktiviteter som gjorts i steg 5. En illustration av metoden återfinns i figur 2.1. Följande avsnitt beskriver huvudmomenten i mer detalj.



Figur 2.1: Övergripande beskrivning av processen för litteratursökning, gallring och analys.

2.1 Sökningar

Söksträngarna byggdes upp av termer som satts samman utifrån rapportens syfte. Termerna konkretiserades genom tidigare kunskap hos forskargruppen samt den FOI-rapport [3] och det memo [7] som tidigare publicerats inom projektet. Därtill utfördes pilotsökningar i Scopus samt i Google Scholar för att undersöka söksträngarnas lämplighet.

Söktermer för att hitta publikationer om datamängder återges i tabell 2.1.[6] Söktermer för verktyg för att identifiera sårbarheter återges i tabell 2.2.[7]

Sökningarna begränsades till publikationer från år 2018 och senare skrivna på engelska inom området *computer science*. Publikationer inom dokumenttyperna *book chapter* och *conference review* uteslöts.

Tabell 2.1: Söktermer, där OR användes för att sätta ihop raderna inom kolumnen, medan AND användes mellan de fyra första kolumnerna och AND NOT för den sista kolumnen. Sökningen gjordes på titel, sammanfattning och nyckelord.

Innehåll	Användning	Datamängd	Syfte	Avgränsning
software	tool*	dataset*	vulnerab*	predict*
code	analys*	data set		detection
source code	analyz*	repository		
binary	evaluating			

[6] Söksträng på Scopus format för datamängder: TITLE-ABS-KEY ((software OR code OR ”source code” OR binary) AND (tool* OR analys* OR analyz* OR evaluating) AND (dataset* OR ”data set” OR repository) AND (vulnerab*) AND NOT (predict* OR detection)) AND PUBYEAR > 2018 AND PUBYEAR < 2027 AND (LIMIT-TO (SUBJAREA , ”COMP”)) AND (EXCLUDE (DOCTYPE , ”cr”) OR EXCLUDE (DOCTYPE , ”ch”)) AND (LIMIT-TO (LANGUAGE , ”English”))

[7] Söksträng på Scopus format för verktyg: TITLE-ABS-KEY ((survey OR review OR ”literature study”) AND (”static analysis” OR ”dynamic analysis” OR fuzz* OR tool*) AND (code OR ”source code” OR software OR binary) AND (secur* OR vulnerab*) AND NOT (predict* OR ”code review” OR ”smart contract” OR ”blockchain” OR web)) AND PUBYEAR > 2018 AND PUBYEAR < 2027 AND (LIMIT-TO (SUBJAREA , ”COMP”)) AND (EXCLUDE (DOCTYPE , ”cr”) OR EXCLUDE (DOCTYPE , ”ch”)) AND (LIMIT-TO (LANGUAGE , ”English”))

Tabell 2.2: Söktermer, där OR användes för att sätta ihop raderna inom kolumnen, medan AND användes mellan de fyra första kolumnerna och AND NOT för den sista kolumnen. Sökningen gjordes på titel, sammanfattning och nyckelord.

Artikeltyp	Teknik	Mjukvara	Syfte	Avgränsning
survey	static analysis	code	secur*	predict*
review	dynamic analysis	source code	vulnerab*	code review
literature study	fuzz*	software		smart contract
	tool*	binary		blockchain
				web

De slutliga sökningarna genomfördes i Scopus och resulterade i 652 publikationer om datamängder och 773 publikationer om verktyg.

2.2 Gallringar

Gallring bland sökträffarna genomfördes manuellt enligt steg 2–4 i metodbeskrivningen ovan. Gallringen utgick från kriterier som togs fram av forskargruppen utifrån studiens syfte och forskningsfrågor. I varje steg skapades samsyn kring kriterierna genom att hela forskargruppen bedömde en gemensam delmängd av publikationerna och diskuterade avvikelser i bedömningarna.

För att en publikation om datamängder skulle inkluderas användes följande kriterier i samtliga gallringssteg:

- Publikationen handlar om datamängder med sårbar kod.
- Datamängderna i publikationen kan användas för att utvärdera verktyg som identifierar mjukvarusårbarheter.
- Publikationens datamängd handlar inte enbart om webbsårbarheter eller generella mjukvarubuggar.

Efter titelgallring återstod 57 publikationer om datamängder. Efter gallring på sammanfattningar återstod 26 publikationer som blev 13 publikationer efter gallring på fulltext. De återstående publikationerna gick vidare till analys.

För att en publikation om verktyg skulle inkluderas användes följande kriterier i samtliga gallringssteg:

- Publikationen innehåller en översikt eller jämförelse av verktyg för att identifiera mjukvarusårbarheter.
- Publikationen diskuterar datamängder som använts för att utvärdera verktyg, eller diskuterar generering av testfall för att utvärdera verktyg.
- Publikationen handlar inte enbart om webbsårbarheter.
- Publikationen handlar inte enbart om att upptäcka eller analysera skadlig kod.
- Publikationen handlar inte enbart om enskilda verktyg för att identifiera mjukvarusårbarheter.

Efter titelgallring återstod 71 publikationer om verktyg. Efter gallring på sammanfattningar återstod tolv publikationer som blev åtta publikationer efter gallring på fulltext. De återstående publikationerna gick vidare till analys.

2.3 Analys

Fulltexten för de identifierade publikationerna analyserades för att hitta beskrivningar och omnämnanden av datamängder av intresse för studien. Utifrån de 21 publikationer som återstod efter fulltextgranskning identifierades tolv direkt beskrivna datamängder och 36 indirekt beskrivna datamängder, det vill säga totalt 48 datamängder.

2.4 Informationsinsamling

I detta steg kompletterades de 48 identifierade datamängderna och verktygen från litteraturstudien med 23[8] datamängder och verktyg som tas upp av Jensen m.fl. [7]. Sjutton av dessa återfanns i både litteraturstudiens resultat och i Jensen m.fl. Totalt är det således 54 datamängder och verktyg som identifierats genom de två kombinerade studierna och som undersöks i informationsinsamlingen. Viss mängd icke-akademisk litteratur används i informationsinsamlingen då några av de 36 indirekt beskrivna resultaten presenteras i sådana publikationer.

Den information som samlades in från publikationerna inkluderar:

- Hur datamängden har konstruerats, t.ex. principer och metoder som använts.
- Vilka egenskaper hos datamängder som publikationen lyfter fram.
- Fördelar som poängteras med publikationens datamängd.
- Motivering till varför publikationens datamängd tagits fram.
- Kritik som riktas mot andra datamängder, t.ex. vad dessa innehåller eller avseende de metoder och principer som använts vid framtagandet.

Informationen som samlats in strukturerades efter vilken typ av information (det vill säga enligt ovanstående punktlista) och analyserades för att hitta likheter och diskrepanser mellan beskrivningar och resonemang i de olika publikationerna. Informationen analyserades genom tematisering[9] på principer, egenskaper och motiveringar för datamängder samt på fördelar och kritiker som presenteras i materialet. Resultatet av denna analys presenteras i nästa kapitel.

[8] Unified Bug Dataset och Learning Realistic Bugs som listas i Jensen m.fl. inkluderas inte i denna studie då datamängderna bedömdes ha fokus på generella buggar och inte sårbarheter.

[9] Tematiseringen har i praktiken inneburit en gruppering i relativt breda men sammanhängande grupper av informartion.

3 Resultat

Detta kapitel redovisar resultaten för genomgången av den akademiska litteraturen. Kapitlet är upplagt så att avsnitt 3.1 presenterar datamängder som identifierats i litteraturstudien, avsnitt 3.2 beskriver kritik som identifierats mot datamängder och avsnitt 3.3 beskriver önskvärda egenskaper hos datamängder som identifierats. Publikationerna som inkluderades efter litteraturgenomgången listas i tabell A.1 (i bilaga A).

3.1 Datamängder

Inom akademisk forskning används ett stort antal datamängder med sårbar mjukvara för att utvärdera och träna verktyg. I studien identifierades 54 datamängder eller verktyg för att generera datamängder med testfall. Tabell 3.1 presenterar en översikt av datamängderna och hur de konstruerats. I många fall har datamängderna varit huvudsyftet med publikationen men i flera fall har datamängderna skapats för att utvärdera ett verktyg. Alla identifierade datamängder har inte ett akademiskt ursprung. Exempelvis SARD och OWASP benchmark har tagits fram av NIST[10] (i samarbete med andra organisationer) respektive OWASP.[11]

Tabell 3.1: Datamängder och verktyg för att generera datamängder.

Typ av sårbarheter	Namn eller författare	Referens
Verkliga sårbarheter	Big-Vul	[9]
	Branconaro & Losiouk	[10]
	BugAnaBench	[11]
	BugHunter	[12]
	Cao m.fl.	[13]
	CrossVul	[14]

Fortsättning på nästa sida

[10] National Institute of Standards and Technology. `https://www.nist.gov`.
[11] Open Worldwide Application Security Project. `https://www.owasp.org`.

Typ av sårbarheter	Namn eller författare	Referens
	CVEFixes	[15]
	D2A	[16]
	Devign datamängd	[17]
	DiverseVul	[18]
	Eceiza m.fl.	[5]
	FUNDED datamängd	[19]
	Fuzzbench	[20]
	IoTVulCode	[21]
	LVDAndro	[22]
	MegaVul	[23]
	Pereira m.fl.	[24]
	Ponta m.fl.	[25]
	PrimeVul	[26]
	REEF	[27]
	ReposVul	[28]
	ReVeal	[6]
	Secbench	[29]
	SecBench.js	[30]
	UniFuzz	[31]
	Vul4J	[32]
	VulData7	[33]
	VulinOSS	[34]
	VulnMiner datamängd	[35]
Injicerade sårbarheter	Apocalypse	[36]
	BOSS	[37]
	Bug Injector	[38]
	EvilCoder	[39]
	IntJect	[40]
	LAVA	[41][12]
	mua-fuzzer-benchmark	[42]
	Magma	[43]
	Zheng m.fl.	[44]
Syntetiska sårbarheter	Cyber Grand Challenge	[45]
	DataRaceBench	[46]
	FormAI	[47]

Fortsättning på nästa sida

[12] Publikationen om LAVA innehåller två datamängder, LAVA-1 och LAVA-M.

Typ av sårbarheter	Namn eller författare	Referens
	Ghera	[48]
	Juliet	[49][13]
	OWASP Benchmark	[51]
	Vulnerability Test Suite Generator (VTSG)	[52]
	Ziems & Wu	[53]
Hybrider	Russel m.fl.	[54]
	GoBench	[55]
	MVDSC-* [14]	[56]
	SARD	[50]
	SATE VI	[57]
	SySeVR datamängd	[58]
	VulDeePecker datamängd	[59]
	VulDetectBench	[60]

Datamängder kan placeras in i fyra olika kategorier beroende på hur sårbarheterna i datamängden skapats:

Syntetiska sårbarheter Datamängder som består av sårbar kod som skapats i syfte att implementera en eller flera sårbarheter i ett källkodsexempel.

Verkliga sårbarheter Datamängder som består av sårbar kod som samlats in från verkliga exempel. Exempelvis genom CVE-databaser som refererar till specifika instanser av sårbar kod och patchar som åtgärdar sårbarheter.

Injicerade sårbarheter Datamängder som består av sårbar kod som skapats genom att injicera sårbarheter i verklig kod.[15]

Hybrider Datamängder som består av testfall som skapats på minst två av ovanstående sätt. Ett exempel är SARD som både består av syntetiska sårbarheter och verkliga sårbarheter [50].

I denna rapport används begreppet *verkliga sårbarheter* för att benämna sårbarheter som upptäckts i verklig kod. Syntetiska och injicerade sårbarheter är inte verkliga sårbarheter men kan däremot jämföras med sårbarheter i verklig kod

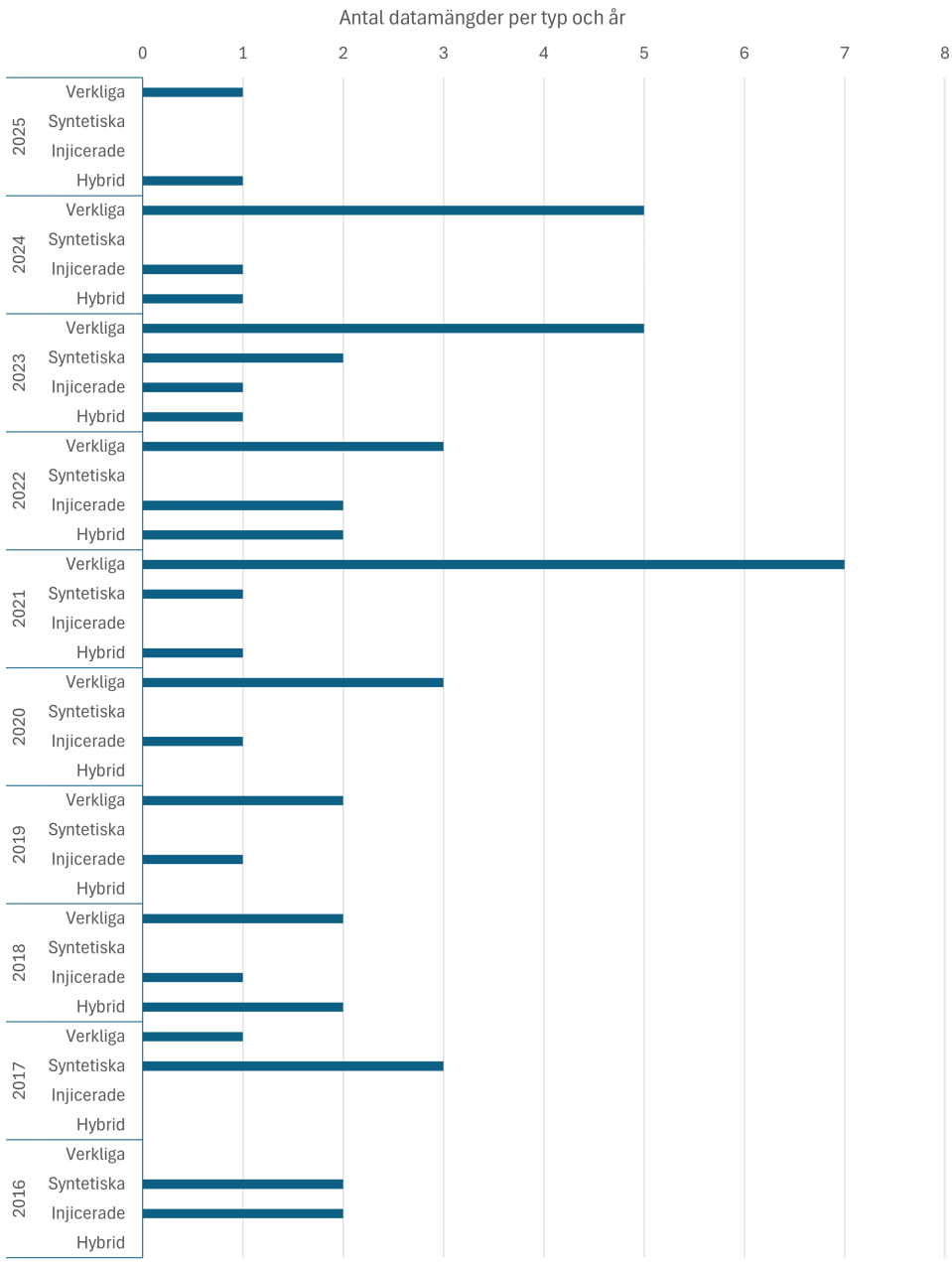
[13] Juliet refererar till testsviter för C, C++ och Java som ingår i SARD [50].
[14] Publikationen innehåller tre datamängder: MVDSC-C, MVDSC-C-Adv, MVDSC-C-Mixed.
[15] Med *verklig kod* avses källkod som är framtagen för att driftsätta i produktionsmiljö.

för att bedöma hur *realistiska* sårbarheterna är. Över hälften, knappt 54 %, av datamängderna i studien baseras på verkliga sårbarheter. En sjättedel, knappt 17 %, baseras på injicerade sårbarheter, knappt 15 % av datamängderna baseras på syntetiska sårbarheter och knappt 15 % av datamängderna är hybrider som består av flera typer av sårbarheter. Figur 3.1 illustrerar hur många datamängder av olika typer som skapats eller uppdaterats ett visst år. Det är tydligt att datamängder som baseras på verkliga sårbarheter har skapats i allt större utsträckning efter 2020.

3.2 Kritik mot datamängder

Flera publikationer lyfter kritik mot olika datamängder, ofta som en del av att motivera skapandet av en egen datamängd. Kritiken som lyfts är relativt utspridd mellan publikationerna, med några enstaka publikationer som lyfter flera typer av kritik. All kritik är inte nödvändigtvis relevant för alla typer av datamängder utan beror på i vilket syfte en datamängd används. Exempelvis använder AI-baserade verktyg befintliga datamängder för träning och har därför ett behov av att datamängderna är korrekt annoterade och inte innehåller extra kod för instrumentering, det vill säga kod som lagts till enbart för att övervaka eller spåra körningen. Annotering spelar inte lika stor roll för andra typer av verktyg, exempelvis statiska analysverktyg. Kritiken som listas nedan är presenterade utifrån det som författarna upplever som vanligast:

Orealistiska sårbarheter En vanlig kritik mot syntetiska datamängder är att sårbarheterna i dem inte är tillräckligt realistiska [19], [23], [61]. Liknande kritik förekommer mot datamängder som skapas genom injicering av sårbarheter i verklig kod, framförallt kopplat till att det inte alltid är känt om sårbarheterna kan exploateras [27], [44], [57] eller ens känt hur många sårbarheter som finns i mjukvaran [43]. Vissa publikationer är mer specifika, exempelvis genom att påpeka att granulariteten i många datamängder inte avspeglar hur sårbarheter faktiskt uppträder i verkliga program, eftersom representationer på enbart funktions- eller filnivå missar sårbarheter som sträcker sig över flera rader, funktioner eller filer [28]. Chakraborty m.fl. exemplifierar kritiken med ett isolerat och simpelt exempel på buffertöverskridning som används i VulDeePecker och SySeVR [6].



Figur 3.1: Antal datamängder per typ och år.

Obalans mellan sårbar kod och neutral kod Fördelningen mellan sårbar kod och neutral kod återkommer som kritik i flera publikationer. Ni m.fl. lyfter att fördelningen mellan sårbar kod och neutral kod i datamängden Devign är orealistisk [23]. Liknande kritik lyfts mot CVEFixes av Yadav och Wilson [37] och som generell kritik mot datamängder av Chakraborty m.fl. [6]. Afanador & Irvine analyserade datamängderna Juliet C/C++, Juliet Java, CGC och OWASP benchmark och kom fram till att datamängderna inte avspeglade sårbarheterna som publicerades i NVD:s CVE-databas under åren 2014–2019 [4].

Obalans i sårbarhetstyper Vilka typer av sårbarheter som ingår i en datamängd och vilken fördelning sårbarhetstyperna har lyfts i flera fall upp som en kritik. Ni m.fl. använder datamängden ReVeal som ett exempel på datamängd med en begränsad mängd typer av sårbarheter [23]. Fan m.fl. påpekar att det saknas tillgängliga datamängder för att undersöka fler typer av sårbarheter [9].

För få sårbarheter eller sårbara mjukvaror Att en datamängd har för få sårbarheter eller för få sårbara mjukvaror är ett problem som främst uppstår vid träning av AI-modeller. Yadav och Wilson lyfter kritiken att befintliga datamängder, framförallt LAVA-M och Big-Vul, innehåller för få exempel av en viss sårbarhet för att kunna träna en AI-modell [37]. Li m.fl. påpekar att deras tidigare datamängd VulDeePecker innehåller för få typer av sårbarheter för att utvärdera verktyget SySeVR [58].

Onåbara sårbarheter En kritik som främst lyfts mot syntetiska och injicerade sårbarheter är att dessa inte alltid är exploaterbara. Delaitre m.fl. beskriver ett scenario där ett program innehåller mer än en sårbarhet varav den första sårbarheten hindrar att efterföljande sårbarheter exekverar, exempelvis vid användning av statiska analysverktyg [57].

Låg kvalitet Låg kvalitet omfattar kritik där exempelvis koden i datamängden inte kompilerar, metadata är felaktig, etc. Ni m.fl. lyfter flera exempel på låg kvalitet i Big-Vul, exempelvis inkompletta funktioner, felaktigt sammanslagna funktioner och missade commit-meddelanden [23]. Zhou m.fl. lyfter att datamängderna från VulDeePecker och SySeVR innehåller felmärkt data [56]. Bhuiyan m.fl. lyfter att datamängder som består av automatiskt insamlade sårbarheter innehåller brus [30]. Bruset kan i sin tur

negativt påverka träningen av AI-baserade verktyg [62]. Chakraborty m.fl. anmärker att annoteringar som baseras på statiska analysverktyg också ärver nackdelarna från de statiska analysverktygen, såsom att de innehåller falska positiver där kod felaktigt markerats som sårbar [6].

Saknad annotering eller metadata En kritik som främst omnämns i publikationer som rör träning av AI-modeller är att det saknas annotering eller metadata i datamängderna. Mer konkret kan det innebära att det saknas information om var sårbarheten finns i koden, vilket exempelvis lyfts som kritik mot datamängden VulinOSS av Yadav och Wilson [37] och mot Juliet av Partenza m.fl. [63]. Pereira m.fl. skapade en datamängd baserat på projekt med öppen källkod skriven i C/C++ med motiveringen att det behövs datamängder med metadata som exempelvis kan användas för träning av AI-modeller [24]. Li m.fl. lyfter bristen på fingranulär information och metadata som en generell brist hos datamängder [64].

Saknar åtgärdad kod Ytterligare en kritik som främst lyfts i anslutning till träning av AI-modeller är att det saknas åtgärdad källkod [37].

Kända mönster vid injicering av sårbarheter En kritik som främst är riktad mot datamängder som skapas genom injicering av sårbarheter är att injiceringen kan lämna spår i den ursprungliga koden. Exempelvis LAVA lägger till funktionsanrop, vilket påverkar verktyg som använder datamängden för träning [37], [44].

Utdaterad datamängd Flera publikationer lyfter att många datamängder är utdaterade och innehåller få nya sårbarheter [23], [65]. I vissa fall motiveras kritiken med att datamängderna inte tar hänsyn till hur mjukvara och sårbarheter utvecklas över tid [27].

3.3 Önskvärda egenskaper hos datamängder

I detta avsnitt sammanställs de egenskaper som datamängder bör ha utifrån publikationerna som identifierats i studien. Egenskaper hos datamängder adresseras inte på ett enhetligt vis mellan publikationerna då vissa publikationer explicit diskuterar egenskaper som behövs medan andra publikationer främst kritiserar bristen på vissa egenskaper i datamängder. Flera publikationer beskriver

samma egenskap men inte nödvändigtvis med samma termer. För att hantera detta har rapportens författare grupperat ihop egenskaper som beskrivs på liknande sätt och därmed anses vara näraliggande. Nedan presenteras de identifierade egenskaperna, ordnade så att de egenskaper som författarna upplevde som vanligast kommer först.

Realism Realistiska datamängder syftar till att spegla hur sårbarheter faktiskt ser ut och uppträder i verkliga system. Flera arbeten framhåller att utvärderande jämförelser måste ligga nära verkligheten, då syntetiska eller förenklade testfall saknar den komplexitet som kännetecknar sårbarheter i verklig mjukvara [4], [11], [14], [28], [31]. Realism omfattar även att sårbarheter kräver specifika tillstånd och icke-triviala indata för att triggas, eftersom verkliga sårbarheter ofta uppträder först under djupa körvägar eller sammansatta interaktioner i mjukvaran. Sådana egenskaper kan endast fångas av realistiska testfall [30], [32], [36], [38]. Slutligen betonar vissa publikationer vikten av lämplig granularitet, där både för grova och för syntetiska kodstrukturer riskerar att dölja eller förvränga sårbarhetens verkliga form [4], [14], [28].

Information om både sårbar och åtgärdad mjukvara Egenskapen förstås som att datamängden består av före- och efter-par som utgör facit (eng. ground truth) för att bedöma om ett verktyg lyckats återskapa, lokalisera eller reparera en sårbarhet på rätt sätt [33], [66]. Flera publikationer samlar verklig data i form av en sårbar och en åtgärdad (eng. patched) variant av varje kodexempel [10], [30], [32]. Syntetiskt konstruerade datamängder, som Juliet Test Suite [49], [67], följer samma struktur, även om koden inte är baserad på verkliga sårbarheter.

Bevis för sårbarhet Datamängden tillhandahåller körbara bevis på att datamängdens sårbarheter faktiskt kan utlösas, vanligtvis i form av ett specifikt testfall som triggar sårbarheten vid körning. Detta demonstrerar att sårbarheterna är exploaterbara i praktiken och fungerar som facit för att avgöra om ett verktyg eller en modell korrekt identifierar eller åtgärdar dem [30], [32], [36], [38], [44], [48].

Mångfald i sammansättning Att datamängden innehåller en variation av sårbarhetstyper, kodstrukturer och programmiljöer framhålls som viktigt i flera arbeten. Några publikationer tar upp att homogena testfall riskerar att

begränsa värdet hos datamängden då den inte speglar den variation av felmönster och kodmiljöer som verktygen förväntas hantera i praktiken [38], [44]. Andra lyfter behovet av språklig och strukturell mångfald, eftersom sårbarheter uttrycks och yttrar sig olika beroende på programmeringsspråk, idiom och systemarkitektur [14], [68]. Slutligen framhålls att variation i programstorlek och funktionell komplexitet är nödvändig för att undvika att verktyg testas på triviala eller snäva kodbaser och därmed får svårigheter att prestera i verkliga miljöer [5], [31], [43]. Även domänspecifik variation, såsom olika typer av Android-sårbarheter i Ghera, lyfts som en väg att återspegla verklighetens bredd [48].

Tillräcklig omfattning En viss volym av testfall krävs för att utvärderingar av sårbarhetsdetektion ska ge tillförlitliga resultat. Små datamängder riskerar att ge slumpkänsliga eller missvisande resultat, vilket visats för bland annat fuzzningsverktyg [5]. Inom maskininlärning framhålls behovet av storskaliga träningsdatamängder för att säkerställa modeller som kan generalisera bortom specifika datapunkter [68]. Datamängdernas omfattning beskrivs därför som en förutsättning för en robust analys och rättvisande utvärderingar [11], [14].

Metadata Sårbarheterna i datamängderna bör vara väldokumenterade. Det beskrivs ofta som att de ska åtföljas av strukturerad och detaljerad information som gör sårbarheterna möjliga att spåra, förstå och reproducera i sitt sammanhang. I litteraturen beskrivs särskilt värdet av metadata som CVE- och CWE-klasser, patch- och commit-information, fil- och funktionskontext, versionsuppgifter samt exekverings- och verifieringsdata. Tillgång till sådan information underlättar typ- och trendanalys, reproducerbar utvärdering och förståelse av sårbarheters uppkomst och åtgärd [10], [14], [23], [30], [33].

Repeterbarhet Att sårbarheter kan återskapas konsekvent och med samma resultat vid varje körning framhålls som en central egenskap. Detta kräver stabila och verifierbara testfall som tydligt utlöser sårbarheten [10], [32], [36], liksom kontrollerade och deterministiska experimentmiljöer där verktygens beteende kan jämföras rättvist över tid [20]. Flera arbeten betonar också behovet av upprepade körningar för att undvika slumpvariationer, särskilt i fuzzingbaserade studier där enstaka resultat kan vara missvisande [31]. I vissa arbeten betonas dessutom att jämförande

tester bör utformas så att reproducerbara resultat kan uppnås även när olika analysverktyg används, vilket gör egenskapen central för rättvisande och jämförbara utvärderingar [4].

Lättanvänt För att underlätta användning av datamängderna framhålls att de bör vara både färdiga att använda (eng. ready-to-use), det vill säga direkt användbara utan anpassning, och lättanvända (eng. easy-to-use), med en struktur som gör testfallen enkla att förstå och integrera [48]. En bredare användbarhet kan också handla om att datamängderna är plattformsoberoende, så att de kan användas för utvärderingar i olika miljöer utan att jämförbarheten påverkas [4], [20], [31], [43].

Distribution av sårbar och neutral kod För att möjliggöra maskininlärningsbaserad klassificering av sårbarheter behöver en datamängd innehålla både sårbara och icke-sårbara kodexempel [35]. Neutrala exempel, där det inte finns identifierade sårbarheter, fyller rollen som negativ klass och utgör därmed en förutsättning för träning och rättvis utvärdering av detektionsmodeller.

Uppdaterbarhet Statiska eller föråldrade datamängder riskerar att ge missvisande utvärderingar. Att en datamängd kan uppdateras löpande lyfts som en relevant egenskap eftersom information om sårbarheter förändras över tid och nya programversioner, patchar och exploateringsmetoder tillkommer. För att förbli relevanta bör datamängder därför vara konstruerade så att de enkelt kan byggas ut och uppdateras [11].

Verktygs- och teknikagnostiskt En datamängd bör vara utformad så att den inte gynnar eller förutsätter en särskild analysmetod, utan kan användas av olika typer av verktyg på likvärdiga villkor. Detta säkerställer att observerade resultat speglar verktygens faktiska kapacitet snarare än hur datamängden råkar vara konstruerad. Ett arbete lyfter fram det som en central princip och betonar att utvärderingar inte ska byggas runt antaganden om hur sårbarheter upptäcks [48]. Liknande krav framkommer i arbeten som framhåller behovet av neutrala och standardiserade testmiljöer, där resultaten inte påverkas av teknikberoende variationer mellan verktyg [20], [31].

Annoterat En korrekt annotering lyfts som en särskilt viktig egenskap för AI-inriktade datamängder, där testfall behöver vara korrekt märkta som

sårbara eller neutrala för att möjliggöra tillförlitlig träning och utvärdering [35], [65]. I dessa sammanhang används ofta etiketter baserade på CVE- eller CWE-information [14], [53]. Egenskapen överlappar med egenskaperna om tillgång till metadata samt information om både sårbar och åtgärdad mjukvara.

Rättvist För att en datamängd ska ge meningsfulla jämförelser mellan analysverktyg krävs att testfallen inte systematiskt gynnar eller missgynnar vissa metoder. Enbart en publikation framhåller att en hög kvalitet på ett jämförelsetest innebär att en den använda datamängden måste vara rättvis, med en balanserad problemprofil och sakna snedvridningar som påverkar jämförelser [4]. Även på testfallsnivå betonas att sårbarheterna måste vara konstruerade så att de är rimligt upptäckbara av flera typer av verktyg, för att undvika att enskilda tekniker favoriseras [37].

Granskat Enbart en publikation nämner explicit att en datamängd bör granskas systematiskt för att säkerställa att varje sårbarhet existerar, att den är korrekt klassificerad och åtföljs av relevant metadata [30]. I någon mån överlappar egenskapen med egenskaperna bevis för sårbarhet och metainformation om sårbarheter, men ingen annan publikation har lyft behovet av en övergripande granskning av innehållet.

Automatisk generering Utvärderingar av analysverktyg gynnas av stora och varierade datamängder, något som automatiserade sårbarhetsinjektionsmetoder kan tillhandahålla i stor skala och med kontrollerade egenskaper [44]. När injektionen är deterministisk kan samma sårbarhetstyper återskapas konsekvent över olika kodbaser och experimentuppsättningar [44].

Funktionsspecifik Egenskapen innebär att testfall bör fokusera på den specifika funktionalitet som ger upphov till sårbarheten, och undvika överflödig kod eller alternativa API-anrop som kan dölja dess karaktär eller påverka analysresultat negativt. Detta är särskilt viktigt i plattformar med omfattande API-ytor och flera parallella uttryckssätt, som Android [48], [69].

Kontextberoende Sårbarheter måste förstås i den specifika kontext där de uppstår, eftersom samma typ av sårbarhet kan ta sig olika uttryck beroende på omgivande API:er, komponenter, plattformsnivåer eller

applikationsmönster. Därför bör en datamängd skilja mellan likartade sårbarheter som uppträder i olika sammanhang, och representera dessa som separata fall när kontexten påverkar hur sårbarheten identifieras eller exploateras [48]. Detta är särskilt relevant i miljöer som Android, där säkerheten påverkas av många lager och komponenter, och där olika analysnivåer kan avslöja olika sårbarhetsmönster [48], [69].

Skalbarhet En datamängd bör fungera för jämförelsetester även när testade verktyg utsätts för en ökande grad av parallellism och belastning, exempelvis genom att datastorlek eller antalet programtrådar varieras för att avslöja fel som bara uppstår under mer krävande förhållanden [46]. Skalbarhet innebär också att datamängden kan användas i större experimentuppsättningar, där många verktyg och målprogram körs över längre tidsperioder utan att stabilitet, reproducerbarhet eller jämförbarhet går förlorad [20], [31].

Versionsspecifik För att undvika felaktiga utvärderingar behöver testfall i en datamängd vara knuten till de ramverks- eller plattformsversioner där sårbarheterna kan reproduceras. En sårbarhet kan vara exploaterbar i en version, men patchad i en senare version. Detta kan vara särskilt viktigt i miljöer med snabb API-utveckling och fragmentering, där versionstillhörighet påverkar både sårbarheters förekomst och verktygens möjlighet att identifiera dem på ett korrekt sätt [48].

Öppen datamängd Öppenhet framhålls som en relevant egenskap som möjliggör transparens, återanvändbarhet och gemensam vidareutveckling av datamängden [48]. Datamängder bör vara fritt tillgängliga för alla som vill använda eller bidra till den, med öppen åtkomst till testfall, dokumentation och tillhörande resurser så att innehållet kan granskas, förbättras och integreras i olika forsknings- och utvecklingsmiljöer.

Praktisk En datamängd bör vara praktisk att använda i utvärderingar, vilket innebär att åtminstone någon sårbarhet i varje målprogram ska kunna upptäckas inom rimlig tid. Detta framhålls som nödvändigt för att undvika orimliga körningstider och för att möjliggöra realistiska och jämförbara experiment, eftersom alltför svåråtkomliga sårbarheter riskerar att skapa oproportionerliga utvärderingskostnader [31].

Dokumenterad En datamängd bör åtföljas av tydlig och relevant dokumentation som beskriver sårbarheterna, hur sårbarheten skapats,

versioner för ramverk eller plattformar och instruktioner för att identifiera och exploatera sårbarheterna [48]. Dokumentationen behöver omfatta både en teknisk förklaring av felet och praktiska instruktioner kring versioner, miljöer och steg för att identifiera och utlösa sårbarheten, eftersom detta krävs för att användare ska kunna förstå, återskapa och korrekt utvärdera testfallen.

4 Diskussion

I detta kapitel diskuteras datamängders utformning och begränsningar samt definitioner av datamängders egenskaper i avsnitt 4.1 respektive avsnitt 4.2. Framtida forskningsmöjligheter om datamängder diskuteras i avsnitt 4.3.

4.1 Datamängders utformning

Det är tydligt att det saknas samsyn om vilka datamängder som ska användas vid utvärdering av verktyg som identifierar mjukvarusårbarheter. I vissa publikationer har forskare skapat nya datamängder för att adressera avsaknad av datamängder eller brister i befintliga datamängder, exempelvis Pereira m.fl. [24] och Li m.fl. [31]. I andra fall har forskare utvecklat ett verktyg och en egen datamängd för att utvärdera verktyget, exempelvis Li m.fl. som utvecklade verktyget VulDeePecker [59]. Publikationer om AI-baserade metoder för sårbarhetsdetektion behöver ofta konstruera egna datamängder, eftersom olika modelltyper kräver olika former av annotering för träning och utvärdering. Klassificeringsmodeller behöver exempelvis etiketter som anger om kod är sårbar eller inte, medan modeller för lokalisering eller reparation dessutom måste ha information om var sårbarheten uppstår och hur den åtgärdas [6]. Befintliga datamängder saknar ofta en eller flera av dessa etiketter, vilket gör att forskare vanligtvis bygger egna datamängder eller omannoterar befintliga exempel — till exempel genom att reducera metadata till binära sårbarhetsetiketter för att möjliggöra modellträning [13].

Shihab & Alanezis sammanställning av verktyg och utvärderingar visar tydligt på hur spretigt området är avseende programmeringsspråk, sårbarheter och datamängdsanvändning [70]. Få publikationer belyser dock bristen på samsyn. Ett undantag är Mitra och Ranganath som i sitt arbete med skapandet av Ghera observerade att det saknades riktlinjer och önskvärda egenskaper för datamängder [48]. Bristen på samsyn kan delvis härledas till att datamängder används till olika syften, men det är också troligt att datamängder väljs, skapas eller anpassas för att lyfta fram ens egen forskning.

Två begränsningar med datamängder är både att mjukvaruutveckling i termer av språk och programmeringsmönster förändras över tid och nya sårbarheter hittas i ny och gammal mjukvara. Okända sårbarheter i datamängder som baseras på verklig kod är ett uppenbart problem för verktyg som tränas på datamängder [11]. Uppdaterbarhet nämns knappt i publikationerna, trots att det kan kännas som en grundläggande egenskap för att datamängder ska förbli relevanta över tid [11]. De riskerar annars att spegla ett historiskt och delvis överspelat sårbarhetslandskap, där verktyg som identifierar mjukvarusårbarheter främst utvärderas mot kända och redan åtgärdade sårbarheter istället för aktuella och framväxande hot.

En intressant observation är att datamängder ibland skapas genom att använda statiska analysverktyg. Exempelvis använder skaparna av VulnMiner verktygen FlawFinder, CppCheck och Rats för att annotera en datamängd med målet att datamängden ska kunna användas av AI-modeller för att detektera sårbarheter [35]. Samtidigt är det osannolikt att statiska analysverktyg hittar alla sårbarheter och genererar dessutom ibland falska detektioner. Metoden riskerar att både saknade och felaktiga annoteringar ger bristfälliga träningsdata och missvisande utvärderingar.

4.2 Egenskapernas definitioner

Långt ifrån alla egenskaper som identifierats i denna studie har en tydlig definition. Många av de egenskaper som identifierats är relativt självförklarande, men det finns också egenskaper vars innebörd beror på datamängdens användning eller som inte beskrivs i tillräcklig detalj. Vid praktisk konstruktion av datamängder riskerar identifierade egenskaper att vara inkompatibla med varandra. Exempelvis kan krav på realism och versionsspecificitet krocka med önskemål om enkel användbarhet, precis som automatisk generering av stora mängder injicerade testfall kan stå i motsats till kravet på kontextberoende eller funktionsspecifik representation av sårbarheter. När egenskaperna saknar entydiga definitioner blir det svårt att avgöra vad som ska prioriteras och hur avvägningar bör hanteras. Detta understryker behovet av ett mer systematiskt ramverk för hur datamängders kvalitetsegenskaper ska tolkas, viktas och balanseras i olika användningsfall.

Realism framhålls ofta som en central egenskap hos sårbarhetsdatamängder och lyfts närmast som en guldstandard inom forskningsområdet. Samtidigt definieras

begreppet sällan på ett tydligt sätt. När realism konkretiseras handlar det oftast om att datamängden bevarar den kod- och kontextinformation som påverkar hur sårbarheter uppstår i praktiken, såsom repository-historik, beroenden och patchinformation [28]. Diskussionen kring LAVA visar dock att många datamängder fortfarande bygger på artificiella sårbarheter som saknar sådan kontext, vilket leder till testfall som är lätta att detektera men dåligt representerar verkliga hot [41]. Detta visar att begreppet realism kräver en mer precis teknisk definition för att vara användbart som kvalitetskriterium i framtida arbete.

Samtidigt är realism som egenskap inte oproblematisk. Testfall som hämtas från öppna källor, som GitHub, speglar främst de projekt som råkar vara aktiva eller väldokumenterade, snarare än den fulla bredd av mjukvara som verktyg för sårbarhetsdetektion förväntas hantera. Sådana datamängder riskerar därför att missa variation, domänspecifika sårbarheter och komplexa kodmiljöer som sällan förekommer i öppna ekosystem. Krav på realism kan också komma i konflikt med andra egenskaper. Egenskaper som korrekt annotering, rik metainformation och versionsspecificitet kan vara svåra egenskaper att uppnå om källorna inte tillhandahåller en fullständig och korrekt dokumentation. Detta innebär att realism inte i sig garanterar vare sig heltäckning eller systematisk bredd, och att syntetiska eller injicerade testfall i vissa fall kan erbjuda viktig kompletterande variation.

Det är intressant att också betrakta studiens resultat utifrån vad som inte identifierats. En egenskap som inte framträder i publikationerna handlar om sårbarheters kritikalitet (eng. severity). Det diskuteras i ett arbete av Gkortzis m.fl., i form av CVSS-poäng, påverkan på konfidentialitet, integritet och tillgänglighet [34]. Däremot beskriver de inte kritikalitet som en självständig egenskap. Det är anmärkningsvärt eftersom kritikalitet i många fall är relevant för hur sårbarheter bör vägas i en datamängd: mer allvarliga brister uppvisar ofta andra mönster och konsekvenser än mindre kritiska [34]. Samtidigt finns det förklaringar till varför kritikalitet inte framträder som en tydlig datamängdsegenskap. Allvarlighetsgrad bygger på faktiska konsekvenser och kan därför i praktiken bara bedömas för verkliga, observerade sårbarheter. Dessutom kan samma sårbarhetstyp få olika kritikalitet beroende på var den uppträder, vilket gör egenskapen kontextberoende och svår att generalisera. Kritikalitet blir därmed en svåranvänd egenskap, särskilt för syntetiska eller injicerade testfall.

4.3 Vägen framåt

Studiens resultat visar tydligt på att det behövs fortsatt forskning kring utformning av datamängder. Framtida forskning bör inkludera undersökningar om vilka egenskaper som är önskvärda eller inte önskvärda för olika typer av datamängder. Det vore även intressant att undersöka om en datamängd kan användas oavsett typ av verktyg eller om det behövs specialiserade datamängder för olika typer av verktyg.

Att verifiera förmågan hos verktyg som identifierar mjukvarusårbarheter är nödvändigt och kräver oberoende utvärderingar. Även om publicerade studier genomgår en vetenskaplig granskning finns det en risk att forskare testar sina verktyg mot flera datamängder men endast redovisar de resultat som visar deras egna verktyg i gynnsam dager. Detta kan leda till en överskattad bild av prestandan och begränsad generaliserbarhet [4], [31]. Avsaknaden av standardiserade utvärderingsmiljöer försvårar också meningsfulla jämförelser [20]. Utvärderingar som genomförs på flera etablerade datamängder samt med egenutvecklade testfall är nödvändiga för att bekräfta resultat och minska sannolikheten för metodberoende snedvridningar.

Att datamängder är öppet tillgängliga och möjliga för fler att löpande bidra till är en tilltalande egenskap. Gemensam granskning och vidareutveckling kan främja transparens, kvalitetsförbättringar och reproducerbarhet [48]. Samtidigt kan öppna datamängder medföra utmaningar. Det finns en risk för överanpassning till enkla och välkända testfall och bidragens kvalitet skulle kunna variera mycket. Öppen publicering av sårbara kodexempel kan dessutom kräva noggrann hantering för att undvika oavsiktliga säkerhetsrisker, särskilt när datamängderna innehåller reproducerbara exploateringssteg eller ännu inte fullt åtgärdade sårbarheter. Även en öppen datamängd skulle behöva modererande insatser.

Det är också värt att reflektera över att den här studien inte identifierar alla datamängder som används vid utveckling och testning av verktyg som identifierar mjukvarusårbarheter. Att sammanställa, annotera och underhålla omfattande eller högkvalitativa datamängder är resurskrävande. Öppen tillgänglighet innebär dessutom att utvecklare riskerar att optimera sina verktyg specifikt mot publicerade testfall snarare än mot verkliga och okända kodbaser. Detta kan leda till en överanpassning till testfall (eng. benchmark overfitting), där verktyg

framstår som mer effektiva än de faktiskt är. Tillsammans kan dessa faktorer bidra till att vissa organisationer avstår från att publicera sina mest omfattande eller realistiska datamängder, trots deras potentiella värde för forskningsfältet. Det kan också förklara varför flera datamängder får betraktas som övergivna av sina tillhandahållare och inte längre uppdateras med nya sårbarheter [7].

5 Slutsats

Målet med denna studie var att undersöka principer för att konstruera datamängder med sårbarheter. Studiens resultat visar på att forskningen om hur datamängder bör utformas lämnar en del att önska. Det saknas en etablerad bild över vilka egenskaper som behövs för att skapa en bra datamängd för att utvärdera verktyg som identifierar sårbarheter. Framtida forskning bör fokusera på att skapa en taxonomi och ensa bilden om nödvändiga och önskvärda egenskaper beroende på datamängdens ändamål.

Nedan redovisas övergripande svar på studiens forskningsfrågor.

Vilka begränsningar finns i befintliga datamängder med sårbarheter?

Kritiken varierar beroende på typen av datamängd och ändamålet med att använda datamängden. Den vanligaste kritiken handlar om för få sårbarhetstyper eller för få sårbarheter till antalet. För datamängder som används vid träning av AI-baserade verktyg är bristande annotering en återkommande kritik.

Vilka principer bör användas vid konstruktionen av datamängder med sårbarheter för att förbättra relevans, kompletthet och användbarhet?

Egenskaper såsom information om både sårbar och åtgärdad mjukvara, bevis för sårbarhet, realism och mångfald i sammansättning återkommer som önskvärda egenskaper i flera publikationer. Andra egenskaper beror i många fall på ändamålet med datamängden. Exempelvis kräver AI-baserade verktyg en stor mängd exempel med sårbar och icke-sårbar kod, vilket inte nödvändigtvis är lika kritiskt för datamängder som enbart fokuserar på att utvärdera statiska analysverktyg.

Vilka principer har utvärderats för befintliga datamängder med sårbarheter?

Inom vissa användningsområden utvärderas egenskaper i större utsträckning, exempelvis vid träning av AI-modeller där egenskaper som omfattning och annotering är kritiska. För andra användningsområden har få egenskaper utvärderats i praktiken. Forskning inom området tar hänsyn till egenskaper när datamängder skapas men det är sällan som egenskaperna i sig utvärderas eller diskuteras.

Referenser

[1] J. Easterly, *Unsafe at Any CPU Speed - The Designed-In Dangers of Technology and What We Can do About It*, Distinguished Lecture, Carnegie Mellon University, 2023.

[2] D. Eidenskog och C. Vestlund, ”Säkerhetsevidens för IT-system – En inledande studie om bevisföring för systematisk säkerhet”, Totalförsvarets forskningsinstitut, FOI-R--5686--SE, 2024.

[3] C. Gustavsson, C. Vestlund, V. Andersson, L. Nyholm, C. Jensen och D. Eidenskog, ”Tekniker och verktyg som identifierar mjukvarusårbarheter”, Totalförsvarets forskningsinstitut, FOI-R--5692--SE, 2024.

[4] K. Afanador och C. Irvine, ”Representativeness in the Benchmark for Vulnerability Analysis Tools (B-VAT)”, i *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020.

[5] M. Eceiza, J. L. Flores och M. Iturbe, ”Improving fuzzing assessment methods through the analysis of metrics and experimental conditions”, *Computers & Security*, årg. 124, s. 102 946, 2023. DOI: https://doi.org/10.1016/j.cose.2022.102946.

[6] S. Chakraborty, R. Krishna, Y. Ding och B. Ray, ”Deep learning based vulnerability detection: Are we there yet?”, *IEEE Transactions on Software Engineering*, årg. 48, nr 9, s. 3280–3296, 2021. DOI: https://doi.org/10.1109/TSE.2021.3087402.

[7] C. Jensen, C. Vestlund, C. Gustavsson, V. Andersson, L. Nyholm och D. Eidenskog, ”Inventering av testfall för verktyg som identifierar mjukvarusårbarheter”, Totalförsvarets forskningsinstitut, FOI Memo 8673, 2024.

[8] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey och S. Linkman, ”Systematic literature reviews in software engineering – A systematic literature review”, *Information and Software Technology*, årg. 51, nr 1, s. 7–15, 2009. DOI: https://doi.org/10.1016/j.infsof.2008.09.009.

[9] J. Fan, Y. Li, S. Wang och T. N. Nguyen, ”A C/C++ code vulnerability dataset with code changes and CVE summaries”, i *Proceedings of the 17th international conference on mining software repositories*, 2020, s. 508–512. DOI: https://doi.org/10.1145/3379597.3387501.

[10] E. Braconaro och E. Losiouk, ”A Dataset for Evaluating LLMs Vulnerability Repair Performance in Android Applications: Data/Toolset paper”, i *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY ’25, Pittsburgh, PA, USA: Association for Computing Machinery, 2025, s. 353–358. DOI: https://doi.org/10.1145/3714393.3726486.

[11] Z. Jiang, R. Li och C. Tang, ”BugAnaBench: benchmark for software vulnerability analysis and its construction method”, i *Second International Symposium on Computer Technology and Information Science (ISCTIS 2022)*, SPIE, vol. 12474, 2022, s. 40–45. DOI: https://doi.org/10.1117/12.2653414.

[12] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth och T. Gyimóthy, ”An automatically created novel bug dataset and its validation in bug prediction”, *Journal of Systems and Software*, årg. 169, s. 110 691, 2020. DOI: https://doi.org/10.1016/j.jss.2020.110691.

[13] S. Cao, X. Sun, L. Bo, Y. Wei och B. Li, ”Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection”, *Information and Software Technology*, årg. 136, s. 106 576, 2021. DOI: https://doi.org/10.1016/j.infsof.2021.106576.

[14] G. Nikitopoulos, K. Dritsa, P. Louridas och D. Mitropoulos, ”CrossVul: a cross-language vulnerability dataset with commit data”, i *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021, Athens, Greece: Association for Computing Machinery, 2021, s. 1565–1569. DOI: https://doi.org/10.1145/3468264.3473122.

[15] G. Bhandari, A. Naseer och L. Moonen, ”CVEfixes: automated collection of vulnerabilities and their fixes from open-source software”, i *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE 2021, Athens, Greece: Association for Computing Machinery, 2021, s. 30–39. DOI: https://doi.org/10.1145/3475960.3475985.

[16] Y. Zheng, S. Pujar, B. Lewis, L. Buratti, E. Epstein, B. Yang, J. Laredo, A. Morari och Z. Su, ”D2a: A dataset built for ai-based vulnerability detection methods using differential analysis”, i *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2021, s. 111–120. DOI: https://doi.org/10.1109/ICSE-SEIP52600.2021.00020.

[17] Y. Zhou, S. Liu, J. Siow, X. Du och Y. Liu, ”Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks”, *Advances in neural information processing systems*, årg. 32, 2019.

[18] Y. Chen, Z. Ding, L. Alowain, X. Chen och D. Wagner, ”Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection”, i *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, s. 654–668. DOI: https://doi.org/10.1145/3607199.3607242.

[19] H. Wang, G. Ye, Z. Tang, S. H. Tan, S. Huang, D. Fang, Y. Feng, L. Bian och Z. Wang, ”Combining graph-based learning with automated data collection for code vulnerability detection”, *IEEE Transactions on Information Forensics and Security*, årg. 16, s. 1943–1958, 2020. DOI: https://doi.org/10.1109/TIFS.2020.3044773.

[20] J. Metzman, L. Szekeres, L. Simon, R. Sprabery och A. Arya, ”Fuzzbench: an open fuzzer benchmarking platform and service”, i *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2021, s. 1393–1403. DOI: https://doi.org/10.1145/3468264.3473932.

[21] G. P. Bhandari, G. Assres, N. Gavric, A. Shalaginov och T.-M. Grønli, ”IoTvulCode: AI-enabled vulnerability detection in software products designed for IoT applications”, *International Journal of Information Security*, årg. 23, nr 4, s. 2677–2690, 2024. DOI: https://doi.org/10.1007/s10207-024-00848-6.

[22] J. Senanayake, H. K. Kalutarage, M. O. Al-Kadri, L. Piras och A. Petrovski, ”Labelled vulnerability dataset on android source code (lvdandro) to develop AI-based code vulnerability detection models.”, i *SECRYPT*, 2023, s. 659–666. DOI: https://doi.org/10.5220/0012060400003555.

[23] C. Ni, L. Shen, X. Yang, Y. Zhu och S. Wang, ”MegaVul: A C/C++ vulnerability dataset with comprehensive code representations”, i *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, s. 738–742. DOI: https://doi.org/10.1145/3643991.3644886.

[24] J. D. Pereira, J. H. Antunes och M. Vieira, ”A software vulnerability dataset of large open source c/c++ projects”, i *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE, 2022, s. 152–163. DOI: https://doi.org/10.1109/PRDC55274.2022.00029.

[25] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi och C. Dangremont, ”A manually-curated dataset of fixes to vulnerabilities of open-source software”, i *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, IEEE, 2019, s. 383–387. DOI: https://doi.org/10.1109/MSR.2019.00064.

[26] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray och Y. Chen, ”Vulnerability detection with code language models: How far are we?”, *arXiv preprint arXiv:2403.18624*, 2024. DOI: https://doi.org/10.48550/arXiv.2403.18624.

[27] C. Wang, Z. Li, Y. Pena, S. Gao, S. Chen, S. Wang, C. Gao och M. R. Lyu, ”REEF: A framework for collecting real-world vulnerabilities and fixes”, i *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2023, s. 1952–1962. DOI: https://doi.org/10.1109/ASE56229.2023.00199.

[28] X. Wang, R. Hu, C. Gao, X.-C. Wen, Y. Chen och Q. Liao, ”Reposvul: A repository-level high-quality vulnerability dataset”, i *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, s. 472–483. DOI: https://doi.org/10.1145/3639478.3647634.

[29] S. Reis och R. Abreu, ”SECBENCH: A Database of Real Security Vulnerabilities.”, i *SecSE@ ESORICS*, 2017, s. 69–85.

[30] M. H. M. Bhuiyan, A. S. Parthasarathy, N. Vasilakis, M. Pradel och C.-A. Staicu, ”SecBench.js: An executable security benchmark suite for server-side JavaScript”, i *2023 IEEE/ACM 45th International Conference on*

Software Engineering (ICSE), IEEE, 2023, s. 1059–1070. DOI: https://doi.org/10.1109/ICSE48619.2023.00096.

[31] Y. Li, S. Ji, Y. Chen, S. Liang, W.-H. Lee, Y. Chen, C. Lyu, C. Wu, R. Beyah, P. Cheng m. fl., "{UNIFUZZ}: A holistic and pragmatic {Metrics-Driven} platform for evaluating fuzzers", i *30th USENIX Security Symposium (USENIX Security 21)*, 2021, s. 2777–2794. DOI: https://doi.org/10.48550/arXiv.2010.01785.

[32] Q.-C. Bui, R. Scandariato och N. E. D. Ferreyra, "Vul4j: A dataset of reproducible java vulnerabilities geared towards the study of program repair techniques", i *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, s. 464–468. DOI: https://doi.org/10.1145/3524842.3528482.

[33] M. Jimenez, Y. Le Traon och M. Papadakis, "[Engineering Paper] Enabling the Continuous Analysis of Security Vulnerabilities with VulData7", i *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2018, s. 56–61. DOI: https://doi.org/10.1109/SCAM.2018.00014.

[34] A. Gkortzis, D. Mitropoulos och D. Spinellis, "VulinOSS: a dataset of security vulnerabilities in open-source systems", i *Proceedings of the 15th International conference on mining software repositories*, 2018, s. 18–21. DOI: https://doi.org/10.1145/3196398.3196454.

[35] G. Bhandari, N. Gavric och A. Shalaginov, "VulnMiner: A comprehensive framework for vulnerability collection from C/C++ source code projects", *Software Impacts*, årg. 22, s. 100 713, 2024. DOI: https://doi.org/10.1016/j.simpa.2024.100713.

[36] S. Roy, A. Pandey, B. Dolan-Gavitt och Y. Hu, "Bug synthesis: Challenging bug-finding tools with deep faults", i *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, s. 224–234. DOI: https://doi.org/10.1145/3236024.3236084.

[37] A. S. Yadav och J. N. Wilson, "BOSS: A dataset to train ML-based systems to repair programs with out-of-bounds write flaws", i *Proceedings of the 5th ACM/IEEE International Workshop on Automated Program Repair*, 2024, s. 26–33. DOI: https://doi.org/10.1145/3643788.3648013.

[38] V. Kashyap, J. Ruchti, L. Kot, E. Turetsky, R. Swords, S. A. Pan, J. Henry, D. Melski och E. Schulte, ”Automated customized bug-benchmark generation”, i *2019 19th international working conference on source code analysis and manipulation (SCAM)*, IEEE, 2019, s. 103–114. DOI: https://doi.org/10.1109/SCAM.2019.00020.

[39] J. Pewny och T. Holz, ”EvilCoder: automated bug insertion”, i *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, s. 214–225. DOI: https://doi.org/10.1145/2991079.2991103.

[40] B. Petit, A. Khanfir, E. Soremekun, G. Perrouin och M. Papadakis, ”Intject: Vulnerability intent bug seeding”, i *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, 2022, s. 19–30. DOI: https://doi.org/10.1109/QRS57517.2022.00013.

[41] B. Dolan-Gavitt, P. Hulin, E. Kirda, T. Leek, A. Mambretti, W. Robertson, F. Ulrich och R. Whelan, ”LAVA: Large-Scale Automated Vulnerability Addition”, i *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, s. 110–121. DOI: 10.1109/SP.2016.15.

[42] P. Görz, B. Mathis, K. Hassler, E. Güler, T. Holz, A. Zeller och R. Gopinath, ”Systematic Assessment of Fuzzers using Mutation Analysis”, i *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, 2023-08.

[43] A. Hazimeh, A. Herrera och M. Payer, ”Magma: A ground-truth fuzzing benchmark”, *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, årg. 4, nr 3, s. 1–29, 2020. DOI: https://doi.org/10.1145/3410220.3456276.

[44] T. Zheng, Z. Tong, P. Yi och Y. Wu, ”Automated generation of bug samples based on source code analysis”, i *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, 2022, s. 51–60. DOI: https://doi.org/10.1109/APSEC57359.2022.00017.

[45] B. Caswell. ”Cyber Grand Challenge Corpus”. Lunge Technology, utg. (u.å.), URL: http://www.lungetech.com/cgc-corpus/. Besökt: 2025-11-27.

[46] C. Liao, P.-H. Lin, J. Asplund, M. Schordan och I. Karlin, ”DataRaceBench: a benchmark suite for systematic evaluation of data race detection tools”, i *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, s. 1–14. DOI: https://doi.org/10.1145/3126908.3126958.

[47] N. Tihanyi, T. Bisztray, R. Jain, M. A. Ferrag, L. C. Cordeiro och V. Mavroeidis, ”The FormAI dataset: Generative AI in software security through the lens of formal verification”, i *Proceedings of the 19th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2023, s. 33–43. DOI: https://doi.org/10.1145/3617555.3617874.

[48] J. Mitra och V.-P. Ranganath, ”Ghera: A repository of android app vulnerability benchmarks”, i *Proceedings of the 13th international conference on predictive models and data analytics in software engineering*, 2017, s. 43–52. DOI: https://doi.org/10.1145/3127005.3127010.

[49] P. E. Black, *Juliet 1.3 test suite: Changes from 1.2*. US Department of Commerce, National Institute of Standards och Technology, 2018.

[50] P. E. Black, *The Software Assurance Reference Dataset (SARD)*. National Institute of Standards och Technology, 2025.

[51] OWASP. ”OWASP Benchmark”. (u.å.), URL: https://owasp.org/www-project-benchmark/. Besökt: 2025-11-24.

[52] P. E. Black, W. Mentzer, E. Fong och B. Stivalet, *Vulnerability Test Suite Generator (VTSG) Version 3*. US Department of Commerce, National Institute of Standards och Technology, 2023. DOI: https://doi.org/10.6028/NIST.IR.8493.

[53] N. Ziems och S. Wu, ”Security vulnerability detection using deep learning natural language processing”, i *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2021, s. 1–6. DOI: https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484500.

[54] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood och M. McConley, ”Automated Vulnerability Detection in Source Code Using Deep Representation Learning”, i *2018 17th IEEE International Conference on Machine Learning and Applications*

(ICMLA), 2018, s. 757–762. DOI: https://doi.org/10.1109/ICMLA.2018.00120.

[55] T. Yuan, G. Li, J. Lu, C. Liu, L. Li och J. Xue, ”GoBench: A benchmark suite of real-world go concurrency bugs”, i *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2021, s. 187–199. DOI: https://doi.org/10.1109/CGO51591.2021.9370317.

[56] X. Zhou och R. M. Verma, ”Vulnerability detection via multimodal learning: Datasets and analysis”, i *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, 2022, s. 1225–1227. DOI: https://doi.org/10.1145/3488932.3527288.

[57] A. Delaitre, P. E. Black, D. Cupif, G. Haben, L. Alex-Kevin, V. Okun och Y. Prono, ”SATE VI Report: Bug injection and collection”, 2023.

[58] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu och Z. Chen, ”SySeVR: A framework for using deep learning to detect software vulnerabilities”, *IEEE Transactions on Dependable and Secure Computing*, årg. 19, nr 4, s. 2244–2258, 2021. DOI: https://doi.org/10.1109/TDSC.2021.3051525.

[59] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng och Y. Zhong, ”VulDeePecker: A deep learning-based system for vulnerability detection”, *arXiv preprint arXiv:1801.01681*, 2018. DOI: https://doi.org/10.48550/arXiv.1801.01681.

[60] Y. Liu, L. Gao, M. Yang, Y. Xie, P. Chen, X. Zhang och W. Chen, ”Vuldetectbench: Evaluating the deep capability of vulnerability detection with large language models”, *arXiv preprint arXiv:2406.07595*, 2024.

[61] M. Saletta och C. Ferretti, ”A neural embedding for source code: Security analysis and CWE lists”, i *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, IEEE, 2020, s. 523–530. DOI: https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00095.

[62] X. Nie, N. Li, K. Wang, S. Wang, X. Luo och H. Wang, ”Understanding and Tackling Label Errors in Deep Learning-Based Vulnerability Detection (Experience Paper)”, i *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2023)*, 2023, s. 52–63. DOI: https://doi.org/10.1145/3597926.3598037.

[63] G. Partenza, T. Amburgey, L. Deng, J. Dehlinger och S. Chakraborty, ”Automatic identification of vulnerable code: Investigations with an ast-based neural network”, i *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, 2021, s. 1475–1482. DOI: https://doi.org/10.1109/COMPSAC51774.2021.00219.

[64] X. Li, S. Moreschini, Z. Zhang, F. Palomba och D. Taibi, ”The anatomy of a vulnerability database: A systematic mapping study”, *Journal of Systems and Software*, årg. 201, s. 111 679, 2023. DOI: https://doi.org/10.1016/j.jss.2023.111679.

[65] S. Jin, ”Large Language Models for Vulnerability Detection in Static Code Analysis: A Survey”, i *2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, IEEE, 2025, s. 473–479. DOI: https://doi.org/10.1109/ICAIBD64986.2025.11082007.

[66] J. Qiu, Y. Jiang, Y. Miao, W. Luo, L. Pan och X. Zheng, ”A survey of coverage-guided greybox fuzzing with deep neural models”, *Information and Software Technology*, s. 107 797, 2025. DOI: https://doi.org/10.1016/j.infsof.2025.107797.

[67] A. Adhikari och P. Kulkarni, ”Survey of techniques to detect common weaknesses in program binaries”, *Cyber Security and Applications*, årg. 3, s. 100 061, 2025. DOI: https://doi.org/10.1016/j.csa.2024.100061.

[68] S. M. Taghavi Far och F. Feyzi, ”Large language models for software vulnerability detection: a guide for researchers on models, methods, techniques, datasets, and metrics”, *International Journal of Information Security*, årg. 24, nr 2, s. 78, 2025. DOI: https://doi.org/10.1007/s10207-025-00992-7.

[69] J. Senanayake, H. Kalutarage, M. O. Al-Kadri, A. Petrovski och L. Piras, ”Android source code vulnerability detection: a systematic literature

review”, *ACM Computing Surveys*, årg. 55, nr 9, s. 1–37, 2023. DOI: https://doi.org/10.1145/3556974.

[70] A. Shihab och M. Alanezi, ”Exploring Datasets In Software Vulnerability Analysis: A Review”, i *2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, IEEE, 2024, s. 1–6. DOI: https://doi.org/10.1109/eSmarTA62850.2024.10638858.

[71] P. Zhou och Y. Gao, ”Detecting prototype pollution for node.js: vulnerability review and new fuzzing inputs”, *Computers & Security*, årg. 137, s. 103 625, 2024. DOI: https://doi.org/10.1016/j.cose.2023.103625.

A Forskningsartiklar i litteraturstudien

Tabell A.1: Akademiska publikationer inkluderade i litteraturstudien.

Publikation	Referens
K. Afanador och C. Irvine, ”Representativeness in the Benchmark for Vulnerability Analysis Tools (B-VAT)”, i *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020	[4]
M. Eceiza, J. L. Flores och M. Iturbe, ”Improving fuzzing assessment methods through the analysis of metrics and experimental conditions”, *Computers & Security*, årg. 124, s. 102 946, 2023. DOI: https://doi.org/10.1016/j.cose.2022.102946	[5]
J. Fan, Y. Li, S. Wang m. fl., ”A C/C++ code vulnerability dataset with code changes and CVE summaries”, i *Proceedings of the 17th international conference on mining software repositories*, 2020, s. 508–512. DOI: https://doi.org/10.1145/3379597.3387501	[9]
E. Braconaro och E. Losiouk, ”A Dataset for Evaluating LLMs Vulnerability Repair Performance in Android Applications: Data/Toolset paper”, i *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY ’25, Pittsburgh, PA, USA: Association for Computing Machinery, 2025, s. 353–358. DOI: https://doi.org/10.1145/3714393.3726486	[10]
Z. Jiang, R. Li och C. Tang, ”BugAnaBench: benchmark for software vulnerability analysis and its construction method”, i *Second International Symposium on Computer Technology and Information Science (ISCTIS 2022)*, SPIE, vol. 12474, 2022, s. 40–45. DOI: https://doi.org/10.1117/12.2653414	[11]
C. Ni, L. Shen, X. Yang m. fl., ”MegaVul: A C/C++ vulnerability dataset with comprehensive code representations”, i *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, s. 738–742. DOI: https://doi.org/10.1145/3643991.3644886	[23]

Fortsättning på nästa sida

Publikation	Referens
J. D. Pereira, J. H. Antunes och M. Vieira, ”A software vulnerability dataset of large open source c/c++ projects”, i *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE, 2022, s. 152–163. DOI: https://doi.org/10.1109/PRDC55274.2022.00029	[24]
C. Wang, Z. Li, Y. Pena m. fl., ”REEF: A framework for collecting real-world vulnerabilities and fixes”, i *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2023, s. 1952–1962. DOI: https://doi.org/10.1109/ASE56229.2023.00199	[27]
M. H. M. Bhuiyan, A. S. Parthasarathy, N. Vasilakis m. fl., ”SecBench.js: An executable security benchmark suite for server-side JavaScript”, i *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE, 2023, s. 1059–1070. DOI: https://doi.org/10.1109/ICSE48619.2023.00096	[30]
Q.-C. Bui, R. Scandariato och N. E. D. Ferreyra, ”Vul4j: A dataset of reproducible java vulnerabilities geared towards the study of program repair techniques”, i *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, s. 464–468. DOI: https://doi.org/10.1145/3524842.3528482	[32]
G. Bhandari, N. Gavric och A. Shalaginov, ”VulnMiner: A comprehensive framework for vulnerability collection from C/C++ source code projects”, *Software Impacts*, årg. 22, s. 100 713, 2024. DOI: https://doi.org/10.1016/j.simpa.2024.100713	[35]
A. S. Yadav och J. N. Wilson, ”BOSS: A dataset to train ML-based systems to repair programs with out-of-bounds write flaws”, i *Proceedings of the 5th ACM/IEEE International Workshop on Automated Program Repair*, 2024, s. 26–33. DOI: https://doi.org/10.1145/3643788.3648013	[37]

Fortsättning på nästa sida

Publikation	Referens
G. Partenza, T. Amburgey, L. Deng m. fl., ”Automatic identification of vulnerable code: Investigations with an ast-based neural network”, i *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, 2021, s. 1475–1482. DOI: https://doi.org/10.1109/COMPSAC51774.2021.00219	[63]
X. Li, S. Moreschini, Z. Zhang m. fl., ”The anatomy of a vulnerability database: A systematic mapping study”, *Journal of Systems and Software*, årg. 201, s. 111 679, 2023. DOI: https://doi.org/10.1016/j.jss.2023.111679	[64]
S. Jin, ”Large Language Models for Vulnerability Detection in Static Code Analysis: A Survey”, i *2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, IEEE, 2025, s. 473–479. DOI: https://doi.org/10.1109/ICAIBD64986.2025.11082007	[65]
J. Qiu, Y. Jiang, Y. Miao m. fl., ”A survey of coverage-guided grey-box fuzzing with deep neural models”, *Information and Software Technology*, s. 107 797, 2025. DOI: https://doi.org/10.1016/j.infsof.2025.107797	[66]
A. Adhikari och P. Kulkarni, ”Survey of techniques to detect common weaknesses in program binaries”, *Cyber Security and Applications*, årg. 3, s. 100 061, 2025. DOI: https://doi.org/10.1016/j.csa.2024.100061	[67]
S. M. Taghavi Far och F. Feyzi, ”Large language models for software vulnerability detection: a guide for researchers on models, methods, techniques, datasets, and metrics”, *International Journal of Information Security*, årg. 24, nr 2, s. 78, 2025. DOI: https://doi.org/10.1007/s10207-025-00992-7	[68]
J. Senanayake, H. Kalutarage, M. O. Al-Kadri m. fl., ”Android source code vulnerability detection: a systematic literature review”, *ACM Computing Surveys*, årg. 55, nr 9, s. 1–37, 2023. DOI: https://doi.org/10.1145/3556974	[69]

Fortsättning på nästa sida

Publikation	Referens
A. Shihab och M. Alanezi, ”Exploring Datasets In Software Vulnerability Analysis: A Review”, i *2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)*, IEEE, 2024, s. 1–6. DOI: https://doi.org/10.1109/eSmarTA62850.2024.10638858	[70]
P. Zhou och Y. Gao, ”Detecting prototype pollution for node.js: vulnerability review and new fuzzing inputs”, *Computers & Security*, årg. 137, s. 103 625, 2024. DOI: https://doi.org/10.1016/j.cose.2023.103625	[71]

ISSN 1650-1942