Making Software More Secure and Security Engineers' Lives Easier

Rui Abreu (U.Porto)

@rmaranhao

The 9th International School on Software Engineering (ISE 2023)
July 10-12, 2023

Outline

Outline

About me

Research Overview

Collection of SAST Tools

Software Vulnerability Detection + AI

Best Practices For Patch Documentation

Alert Prioritization

Infrastructure-as-code (IaC) scripts

Fixing Vulnerabilities Potentially Hinders Software Maintainability

Why focusing on security (during your PhD)?

📈 The number of new vulnerabilities is growing over time and it takes a long time to patch vulnerabilities regardless of their severity.

🧑‍💻 Lack of security experts (gap of 3 million jobs globally).

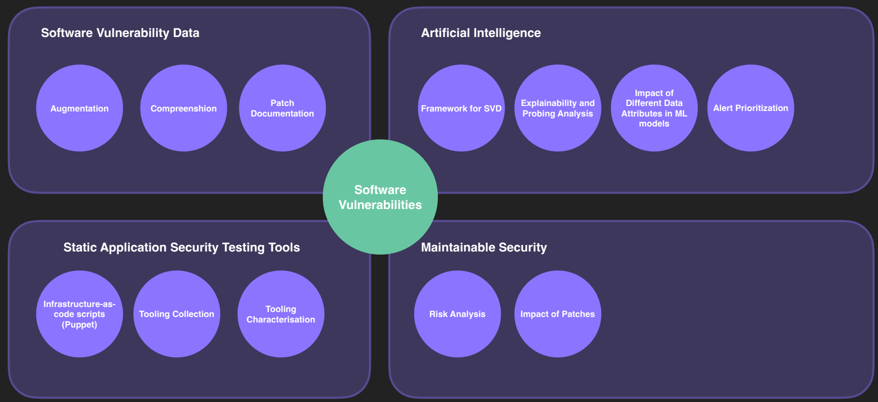
🙅 Adoption is still low (high false positive rates, lack of education and training, lack of actionability, poor usability).

🤨 Knowledge is not structured, updated and centralised.

⚖️ No fair comparison between tools (difficult to trust and know how they fair).

💸 Costs associated with software vulnerabilities.

Research Overview



@rmaranhao

Outline

- About me
- Research Overview
- Collection of SAST Tools
- Software Vulnerability Detection + AI
- Best Practices For Patch Documentation
- Alert Prioritisation
- Infrastructure-as-code (IaC) scripts
- Fixing Vulnerabilities Potentially Hinders Software Maintainability

Systematic Survey on SAST Tools

(On the road to improve static analyzers adoption for security)

SAST tools: Static Application Security Testing tools (aka, static analysers for security).

Two of the main issues for the low adoption of SAST tools are:

1) the lack of complete documentation (approaches, performance rates, scalability, coverage);

2) the lack of structured, updated and centralised knowledge.

@rmaranhao

Systematic Survey on SAST Tools

(On the road to improve static analyzers adoption for security)

In order to get a better overview of the SAST scope, we ran a systematic survey on the topic to answer the following research questions:

RQ1: What are the underlying techniques used by SASTs?

RQ2: Which classes of vulnerabilities and programming languages are covered by the existing SASTs?

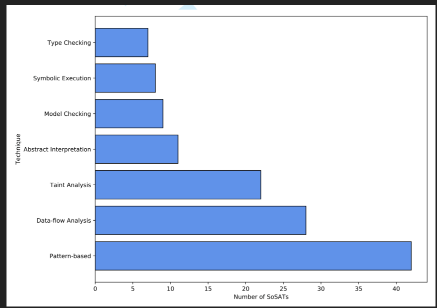
RQ3: Are the research outputs and codebases of SASTs publicly available?

RQ4: What conclusions can we draw on the performance of SASTs from the results presented in the selected work?

@rmaranhao

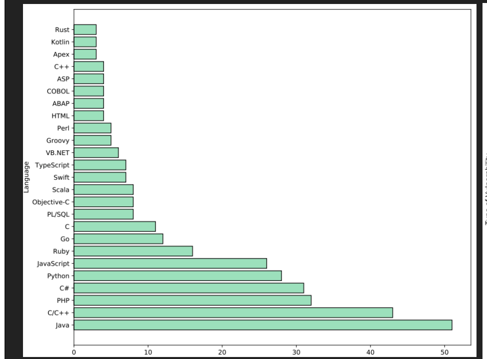
RQ1: What are the underlying techniques used by SASTs?

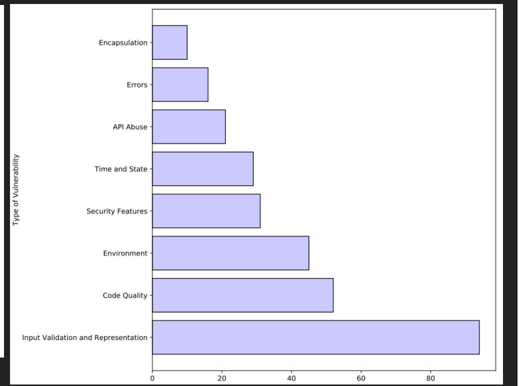
- Pattern-based//AST Matchers
  RATS, Flawfinder, ITS4, Bandit, SLIC/ACID (> 40 tools)
- Flow Analysis (Control-, Data- and Taint analysis)
  Checkmarx, FindBugs, Polyspace Bug Finder, WAP, Pixy (> 40 tools)
- Abstract Interpretation
  Astrée PolySpace Code Prover, Polyspace for Ada (> 10 tools)
- Model Checking
  MOPS, ESBMC, CBMC, JBMC (approx. 10 tools)
- Symbolic Execution
  Infer, PVS-Studio (approx. 10 tools)
- Hybrid Solutions (Static and Dynamic Analysis)
  appScreener, CodeDX, PT Application Inspector, Veracode, Sparrow, thunderscan (11 tools)
- Machine Learning
  Static Reviewer, VulDeePecker, DeepCode, TAP (4 tools)
- Source Code Query Tools
  CodeQL, CppDepend (2 tools)



@rmaranhao

RQ2: Which classes of vulnerabilities and programming languages are covered by the existing SASTs?





Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors

RQ3: Are the research outputs and codebases of SASTs publicly available?

For approximately 40% (58/145) of the SASTs, the codebase is not available which makes their understanding, usage and extensibility more difficult.

We also collected the license of each tool. Only 6 tools did not have any type of license. More than 50% of the tools have an open-source license, i.e., tools than can usually be used freely in research and in the industry

RQ4: What conclusions can we draw on the performance of SASTs from the results presented in the selected work?

Only 23 empirical validations were found for 23 tools. Overall, all tools reported False Positives.

There is a preference for validating the tools with real data instead of artificial. However, empirical validations with real vulnerabilities are rare due to the low amounts of data—which sometimes may not be enough to assess the real performance of the tool.
Datasets with more real data is needed to fairly assess the performance of SASTs.

@rmaranhao

Table 7. SoSATs Empirical Validation Results

Name	S/U	Type	Year	Technique(s)	Vul. Taxonomy (7+1)	PL	Real	Artificial	#TP	#FP	FPR	FNR	Prec.	Recall	F-score	Acc.	Size	Time	Refs
ESBMC	S	🏛	2009	Model Checking; SMT Solver	(3) Input Validation and Representation; Time and State; Code Quality	C/C++	✓ (9523 tasks)	✗	4316	24	-	-	-	-	-	-	-	-	[62–64, 100]
Flowdroid	U	🏛	2013	Taint Tracking	(1) Encapsulation	Java	✓ (500 apps)	✗	117	9	-	-	86%	96%	-	-	-	-	[13]
HCL/IBM AppScan	S	©	1998	Abstract Interpretation	(1) Input Validation and Representation	Multi: C, C++, Java, C#, JavaScript	✓ (60 websites)	✗	33	4	-	-	-	-	-	-	-	-	[247]
IKOS	S	💻	2012	Abstract Interpretation	(2) Input Validation and Representation; Code Quality	C/C++	✓ (479 KLOC)	✗	-	-	-	-	91%-99%	-	-	-	-	18h25s	[35]
JBMC	S	🏛	2018	Model Checking	(1) Errors	Java	✗	✓ (368 tests)	327	14	-	-	-	-	-	-	-	-	[65]
JuliaSoft	S	©	2019	Abstract Interpretation	(1) Input Validation and Representation	Java	✗	✓ (2740 programs)	1415	117	8%	-	-	92%	-	-	-	-	[229]
KINT	U	🏛	2012	Taint Analysis; Range Analysis	(1) Input Validation and Representation	C	✓ (1 app)	✗	1	42	-	-	-	-	-	-	8916 files	160m	[268]
MOPS	S	🏛	2002	Model Checking	(1) Time and State	C	✓ (9 packages)	✗	2333	108	-	-	-	-	-	-	-	-	[45, 46, 217]
phpSAFE	U	🏛	2015	Taint Analysis	(1) Input Validation and Representation	PHP	✓ (35 plugins)	✗	387	62	-	-	86%	66%	75%	-	-	-	[180]
Pixy	U	🏛	2006	Taint Analysis	(1) Input Validation and Representation	PHP	✓ (6 apps)	✗	51	47	50%	-	-	-	-	-	50605 LOC	-	[135]
Pysa/Pyre	U	💻	2020	Pattern-based; Taint Analysis	(1) Input Validation and Representation	Python	✓ (1app)	✗	180	150	45%	-	-	55%	-	-	-	-	
SABER/SVF	U	🏛	2015	Data-flow analysis; Pointer analysis	(1) Code Quality	C/C++	✓ (5 apps)	✓ (15 programs)	211	48	-	-	-	18.5%	-	-	2324.1 KLOC	522.58s	[233, 235]
Saturn	U	🏛	2005	Control-flow Analysis; Alias Analysis	(1) Code Quality	C	✓ (1 program)	✗	179	121	-	-	-	-	-	60%	12455 files (4.8M LOC)	19h40m	[116, 277, 278]
SLIC/ACID	U	🏛	2018	Pattern-based	(1) Environment	Multi: Ansible, Puppet	✓ (140 scripts)	✗	-	-	-	-	99%	99%	-	-	-	-	[201]
Splint/LCLint	U	🏛	2002	Annotations-	(2) Input Validation	C/C++	✓ (1	✗	25	76	-	-	-	-	-	-	-	-	[89]

Table 7. SoSATs Empirical Validation Results

Name	S/U	Type	Year	Technique(s)	Vul. Taxonomy (7+1)
ESBMC	S	🏛	2009	Model Checking; SMT Solver	(3) Input Validation and Representation; Time and State; Code Quality
Flowdroid	U	🏛	2013	Taint Tracking	(1) Encapsulation
HCL/IBM AppScan	S	©	1998	Abstract Interpretation	(1) Input Validation and Representation
IKOS	S	💻	2012	Abstract Interpretation	(2) Input Validation and Representation; Code Quality
JBMC	S	🏛	2018	Model Checking	(1) Errors
JuliaSoft	S	©	2019	Abstract Interpretation	(1) Input Validation and Representation
KINT	U	🏛	2012	Taint Analysis; Range Analysis	(1) Input Validation and Representation
MOPS	S	🏛	2002	Model Checking	(1) Time and State
phpSAFE	U	🏛	2015	Taint Analysis	(1) Input Validation and Representation
Pixy	U	🏛	2006	Taint Analysis	(1) Input Validation and Representation
Pysa/Pyre	U	💻	2020	Pattern-based; Taint Analysis	(1) Input Validation and Representation
SABER/SVF	U	🏛	2015	Data-flow analysis; Pointer analysis	(1) Code Quality
Saturn	U	🏛	2005	Control-flow Analysis; Alias Analysis	(1) Code Quality
SLIC/ACID	U	🏛	2018	Pattern-based	(1) Environment
Splint/LCLint	U	🏛	2002	Annotations-based; Type...	(2) Input Validation and Representation; ...

Table 9. **Unsound** Static Analysis Tools (117 SoSATs were found)

Name	Type	Year	Input	Technique(s)	Vul. Taxonomy (7+1)	PL	Code Available	License	EMS (Perf.)	Popularity	Institute	Refs
(H) AMNESIA	🏛	2005	Source Code	Data-flow Analysis	**(1)** Input Validation and Representation	Java	✓ (bin)	-	✓	G 35.4k	University of Southern California	[117, 118]
Android Lint	💻	2013	Source Code	Pattern-based	**(2)** API Abuse; Security Features	Java	✓	Apachev2	-	G 151	Google, JetBrains	[106]
▭ Androwam	💻	2012	Smali Code	Data-flow Analysis	**(2)** Code Quality; Encapsulation	Java	✓ (Python)	LGPLv3	✓	★ 289, ⑂ 104	University of Lion	[245]
ApexSec	©	2010	Source Code	N/A	**(2)** Input Validation and Representation; Environment	PL/SQL	✗	Paid; Trial Avail.	-	G 4	Recx Ltd.	[152]
AppChecker	©	2007	Source Code	Data-flow Analysis	**(6)** Input Validation and Representation; API Abuse; Time and State; Errors; Code Quality; Environment	***Multi:*** C/C++, C#, PHP, Java	✗	Avail. for Russian Companies	-	G 3	Echelon	[85]
▭ AppCodeScan	💻	2007	Source Code	Pattern-based; Data-flow Analysis	**(1)** Input Validation and Representation	***Multi:*** C#, Java	✓ (bin)	CC BY-NC-SA 3.0	-	G 2	Blueinfy	[34]
Application Inspector	💻	2019	Source Code	Pattern-based	**(2)** API Abuse; Security Features	***Multi:*** C/C++, C#, Java, JavaScript, HTML, Python, Objective-C, Go, Ruby, PowerShell	✓ (C#)	MIT	-	★ 3.4k, ⑂ 277	Microsoft	[172]
(H) appScreener	©	2015	Source Code; Binary Code	N/A	**(2)** Input Validation and Representation; Environment	***Multi:*** C/C++, Go, Groovy, Java, JavaScript, Kotlin, PHP, Python, Ruby	✗	Paid; Trial Avail.	✓ (10%–20%)	G 5	Solar app-Screener	[10]
AttackFlow	©	2016	Source Code	Taint-flow Analysis	**(4)** Input Validation and Representation; Security Features; Time and State; Code Quality	***Multi:*** .Net, Java, JavaScript, TypeScript, HTML	✗	Paid	-	G 12	AttackFlow	[17]
bandit	💻	2015	Source Code	Pattern-based; Control-flow Analysis	**(3)** Input Validation and Representation; Security Features; Environment	Python	✓ (Python)	Apachev2	✓ (#FPs may exist)	★ 2.8k, ⑂ 278	Beyond Security	[27, 198]
▭ BOON	🏛	2000	Source Code	Integer range	**(1)** Input Validation and Representation	C	✓ (C)	BSD	✓ (FPR=12.5%)	G 788k	University of California,Berkeley	[263, 290]
Brakeman Pro	©	2020	Source Code	Data-flow Analysis	**(3)** Input Validation and Representation; Security Features; Environment	Ruby	✓ (Ruby)	Paid;	✓	★ 5.9k, ⑂ 616	Brakeman, Inc (Acquired by Synopsys)	[193]
Brakeman/Railroader	💻	2010	Source Code	Data-flow Analysis	**(3)** Input Validation and Representation; Security Features; Environment	Ruby	✓ (Ruby)	MIT	✓	★ 40, ⑂ 1	Railroader	[202]
C/C++ test	©	1996	Source Code	Pattern-based; ...	**(2)** Input Validation ...	C/C++	✗	Paid; Trial ...	-	G 29	Parasoft	[186]

44 lines (44 sloc) | 806 Bytes

```yaml
- tool: codeql
- description: TODO
- soundness: unsound
- hybrid: no
- deprecated: no
- dev_env:
    acquired_by: GitHub
    acquired_in: 2019
    company: Semmle
    type: Open-Source
- release_year: 2006
- code:
    available: yes
    license: MIT
    type: binary
- popularity:
    forks: 881
    stars: 4108
- techniques:
  - Pattern-based
  - Control-flow Analysis
  - Data-flow Analysis
  - Taint Tracking
  - Range Analysis
  - Variant Analysis
- programming_languages:
  - C/C++
  - C#
  - Go
  - Java
  - JavaScript
  - Python
  - TypeScript
- kingdoms:
  - Input Validation and Representation
  - API Abuse
  - Errors
  - Code Quality
  - Environment
- input: Source Code
- resources:
  - https://securitylab.github.com/tools/codeql
  - https://lgtm.com/help/lgtm/about-lgtm
- observations: TODO
```


Systematic Survey on SAST Tools

(On the road to improve static analyzers adoption for security)

amnesia.yaml	tools data:
android_lint.yaml	tools data:
androwarn.yaml	tools data:
apexsec.yaml	tools data:
appchecker.yaml	tools data:
appcodescan.yaml	tools data:
application_inspector.yaml	tools data:
appscreener.yaml	tools data:
astree.yaml	tools data:
attackflow.yaml	tools data:
bandit.yaml	tools data:
boon.yaml	tools data:
brakeman_pro.yaml	tools data:
brakeman_railroader.yaml	tools data:
c_c++_test.yaml	tools data:
cargo_audit.yaml	tools data:
cast_application_intelligence_platform_aip.yaml	tools data:
cat_net.yaml	tools data:
cbmc.yaml	tools data:

145 SASTs

231 academic papers

Systematic Survey on SAST Tools

(On the road to improve static analyzers adoption for security)

- ⚠️ SASTs may disrupt the team's productivity.
- ⚠️ Not Actionable.
- ⚠️ Poor Usability.
- ⚠️ Lack of structured and organised knowledge.
- ⚠️ Difficult to measure the coverage of the field.
- ⚠️ Scalability Issues.
- ⚠️ Language and pattern dependent.
- ⚠️ Better tools for security are wanted.
- 🆕 Artificial Intelligence has the potential to shift security left but still provides untrustworthy results.
- 🆕 Risk Analysis based on code changes (use static analysis to locate the problem and collect features).
- 🆕 New types of software (quantum, blockchain, infrastructure-as-code scripts).

The problem we are trying to fix with this collection of tools and papers.

@rmaranhao

Outline

- About me
- Research Overview
- Collection of SAST Tools
- **Software Vulnerability Detection + AI**
- Best Practices For Patch Documentation
- Alert Prioritisation
- Infrastructure-as-code (IaC) scripts
- Fixing Vulnerabilities Potentially Hinders Software Maintainability

Software Vulnerability Detection + AI

(Promises to shift security left in the SDLC)

We spent months performing experiments with deep learning algorithms like **Code2vec** and **CodeBERT** for vulnerabilities in **JavaScript** code (collected from advisory databases such as OSV and NVD).

Scrappers: https://github.com/TQRG/security-patches-dataset

Many studies between 2017 and 2021 reported **accuracy > 90%** for the software vulnerability task with AI.

But the reality for us was a bit different. **We could not even reach an accuracy of 70%.**

Software Vulnerability Detection + AI

(Promises to shift security left in the SDLC)

Microsoft maintains a leaderboard with results for different tasks and different models trained and test on CodeXGLUE (a C/C++ dataset).

Our results with the new dataset were again below 70%.

We submitted our results with code2vec which were validated by the microsoft team.

Defect Detection (Code-Code)

Rank	Model	Organization	Date	Accuracy
5	VulBERTa-CNN	Imperial College ...	2021-11-10	64.42
6	ContraBERT_C	Anonymous	2022-11-17	64.17
7	ContraBERT_G	Anonymous	2022-11-17	63.32
8	PLBART	UCLA & Columbi...	2021-04-02	63.18
9	code2vec	SecurityAware T...	2021-06-09	62.48
10	CodeBERT	CodeXGLUE Team	2020-08-30	62.08

D Coimbra, S Reis, R Abreu, C Păsăreanu, H Erdogmus. On using distributed representations of source code for the detection of C security vulnerabilities. International Workshop on Principles of Diagnosis (DX)

Software Vulnerability Detection + AI

(Promises to shift security left in the SDLC)

We looked into the datasets of papers published in the software vulnerability + AI scope and we started to see a trend:

!! Lots of duplicates between the training and testing datasets that led to inflated results.

Which was later reported in the paper "Deep learning based vulnerability detection: Are we there yet?" by S. Chakraborty et al.

Software Vulnerability Detection + AI

(Promises to shift security left in the SDLC)

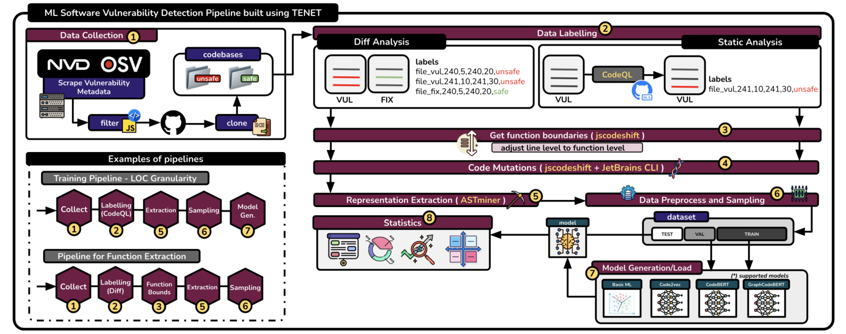


Fig. 1. Tenet architecture with two short pipeline examples. The first pipeline uses *lines of code* as the granularity of a vulnerability and labels the samples using static analysis with CodeQL. The second pipeline uses *functions* as the granularity of a vulnerability and generates labels using diff analysis.

https://github.com/TQRG/tenet

Software Vulnerability Detection + AI

(Promises to shift security left in the SDLC)

After spending time trying to fix the core problem with AI, we shifted our efforts to explainability and probing analysis.

"How different data attributes impact traditional machine learning classifiers?" Sampling Strategy, Distribution between classes, Granularity, Project Diversity, Multiplicity of Software Vulnerabilities

"BERT-based Models for Vulnerability Detection: Looking Beyond Validation Metrics" Probing analysis to check if BERT-models encode semantic (unused vars, tainted vars, vuln code) syntax (function, loop, conditional) and structural (complexity) information in code samples at function level for different CWEs.

@rmaranhao

Outline

- About me
- Research Overview
- Collection of SAST Tools
- Software Vulnerability Detection + AI
- **Best Practices For Patch Documentation**
- Alert Prioritisation
- Infrastructure-as-code (IaC) scripts
- Fixing Vulnerabilities Potentially Hinders Software Maintainability

Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

Many works have reported that **commit metadata** (including commit messages) **are not enough** to classify security-related commits.

One study reported that it could only extract security-related words from 38% of the commit messages; however, it uses a dataset of silent fixes (which naturally have more cryptic messages).

Yet, none of the approaches looked carefully into the key information that could be extracted from commit messages.

Therefore, we performed an analysis of security commit messages and best practices application by security engineers.

@rmaranhao

Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

We used Named Entity Recognition (NER), a natural language processing approach, to identify and extract key information, called *entities*, from unstructured data (in this case, text).

An entity can be any word or bag of words that refers to the same entity category. For instance, different names of companies "Netflix", "Google" or "Apple" are entities that belong to the Company category.

We designed a set of category entities that we tried to extract from commit messages.

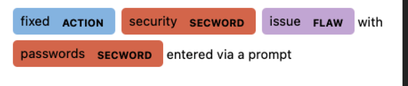


Figure 3: Named Entity Recognition (NER) application example for a security commit message.

Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

RQ1. What information is being mentioned in public security patches?

Table 1: Entity category names, rationale, and entity examples.

Type	Category	Rationale	Entity Examples	Rules
SEC	SECWORD	Security-relevant words are usually used to describe the vulnerability and respective fix (we used a large set of security-relevant words collected in previous work [18, 41]).	ldap injection, crlf injection, improper validation, command injection, cross-site scripting, sanitize, bypass	1719
	VULNID	Vulnerability IDs are used to identify vulnerabilities for different ecosystems in commit messages: CVE, GHSA, OSV, PyPI, etc. We crafted rules for the different IDs patterns.	GHSA-269q-hmxg-m83q, CVE-2016-2512, CVE-2015-8309, GHSA-9x4c-63pf-525f, OSV-2016-1	9
	CWEID	Vulnerabilities usually belong to a weakness type. One common taxonomy used to classify security weaknesses is the Common Weakness Enumeration (CWE) one. Therefore, we crafted rules to detect CWE IDs.	CWE-119, CWE-20, CWE-79, CWE-189	2
	SEVERITY	Vulnerabilities usually have a severity assigned.	low, medium, high, critical	4
	DETECTION	Vulnerabilities are detected manually or using specific tools.	Manual, CodeQL, Coverity, OSS-Fuzz, libfuzzer	8
COM	SHA	Commit hashes that reference older versions where the vulnerability was introduced (OSV Schema [42]).	f8d773084564, 228a782c2dd0	2
	ACTION	A commit usually implies an action, in the case of security, fixing a vulnerability (corrective maintenance).	fix, patch, change, add, remove, found, protect, update, optimize, mitigate	18
	FLAW	Fixing a security vulnerability usually implies fixing a flaw.	defect, weakness, flaw, fault, bug, issue	10
	ISSUE	The GitHub issue/pull request number is sometimes referenced in the message and can provide more information on the vulnerability.	#2, #13245	1
	EMAIL	Contact e-mails of reviewers and authors usually appear after tags such as 'Reported-by' and are important to know who to contact.	johndoe123@gmail.com,catlover@yahooo.com,adventuretime@hotmail.com,supercool@outlook.com[1]	1
	URL	Links to reports, blog posts, and bug-trackers references provide more information about the vulnerability.	https://www.htbridge.ch/advisory/multiple_vulnerabilities_in_mantisbt.html	1
	VERSION	Software versions are commonly referenced in commit messages.	3.1.0, v3.2, v2.6.28, 1.6.3, 2.1.395	4

Type SEC: Security specific entity categories; **Type COM**: Commit specific entity categories.
[1] Artificial e-mails generated automatically with ChatGPT for compliance with General Data Protection Regulation (GDPR).

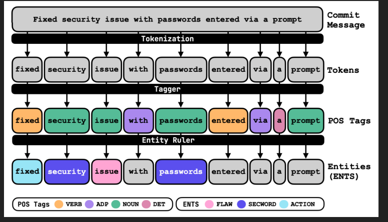
Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

RQ1. What information is being mentioned in public security patches?

To extract the entities for each category, we used a Python library called Spacy—which provides end-to-end pipelines for several natural language processing tasks (e.g., NER).

We built our own customized NER pipeline for security commit messages.



Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

RQ1. What information is being mentioned in public security patches? Analysis of 11036 commit messages.

Finding 1. Security engineers use security-related words in 61.2% of the security commit messages used to patch software vulnerabilities.

Finding 2. Vulnerability IDs, Weakness IDs and Severity are rarely mentioned in security commit messages—although important for manual and automated detection and prioritization.

Finding 3. No extraction of entities was performed from 8% of security commit messages mainly due to poorly written messages, misspelling issues and no clear connection with security.

Table 4: Extraction Results

Category	#Entities	#Commits	%Commits
SECWORD	16126	6749	61.2%
ACTION	10364	6409	58.1%
EMAIL	4738	2086	18.9%
SHA	4943	1467	13.3%
FLAW	4402	2843	25.8%
ISSUE	3561	2805	25.4%
URL	1175	929	8.4%
VULNID	1799	1330	12.1%
VERSION	658	571	5.2%
DETECTION	629	374	3.4%
SEVERITY	142	118	1.1%
CWEID	25	23	0.2%
Total	48562	10168	92.1%

Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

Do security engineers follow best practices to write security commit messages?

C1	4.10% of commit messages follow the conventional commits convention "(scope):" using prefixes such as "patch" or "fix"
C2	100% of commit messages have a one line subject/header. But only 4288 out of 11036 (38.85%) headers have security-related words (SECWORD) and reflect an action (ACTION).
C3	59.91% of commit messages have a body but only 36.53% have SECWORDS.
C4	8.4% of commit messages were signed-off-by.
C5	3.33% of commit messages include the reviewer contact.
C6	25.42% of commit messages include references to issues.
C7	1.78% of commit messages have references to bug trackers.

Table 3: Best Practices to Write Generic Commit Messages

ID	Best Practice	Standard
C1	The header should be prefixed with a type.	[21]
C2	The message should have a one-line header/subject.	[21, 36, 38]
C3	The message should have a body.	[36, 38]
C4	The message should mention the contact of the author (signed-off-by and authored-by).	[36, 38]
C5	The message should mention the contact of the reviewer (reviewed-by).	[36, 38]
C6	The message should mention references to issues or pull requests.	[38]
C7	The message should mention references bug trackers.	[38]

Best Practices For Patch Documentation

(Aiming to improve patch management triage systems and gather more data through for Vuln. Detection with AI and SASTs validation and comparison.)

Do security engineers follow best practices to write security commit messages?

C1	4.10% of commit messages follow the conventional commits convention “(scope):” using prefixes such as “patch” or “fix”
C2	100% of commit messages have a one line subject/header. But only 4288 out of 11036 (38.85%) headers have security-related words (SECWORD) and reflect an action (ACTION).
C3	59.91% of commit messages have a body but only 36.53% have SECWORDS.
C4	8.4% of commit messages were signed-off-by.
C5	3.33% of commit messages include the reviewer contact.
C6	25.42% of commit messages inc
C7	1.78% of commit messages hav

Table 3: Best Practices to Write Generic Commit Messages

ID	Best Practice	Standard
C1	The header should be prefixed with a type.	[21]
C2	The message should have a one-line header/subject.	[21, 36, 38]
C3	The message should have a body.	[36, 38]
C4	The message should mention the contact of the author (signed-off-by and authored-by).	[36, 38]
C5	The message should mention the contact of the reviewer	[36, 38]
	mention references to issues or pull	[38]
	ıention references bug trackers.	[38]

Finding 4. Security engineers, do not follow best practices to write security commit messages in general. Even when it seems they are, we concluded that key information is missing—which indicates we need best practices for writing better security commit messages.

SECOM

A convention for security commit messages

Validated with the Open-Source Security Foundation (OpenSSF)

Feedback received from the security community suggests that they see value in SECOM and would like to see it evolve into a standard practice—5 out of the 7 participants responded “Yes” to “Would you use this or a similar convention as standard practice in your own work or advocate its use in your team?”, the remaining two participants answered “Unsure”.

Convention has been mentioned at BlackHat and Defcon by a security researcher that is already using it to patch thousands of vulnerabilities.

https://tqrg.github.io/secom/

```
<type>: <header/subject> (<Vuln-ID>)

<body>
# (what) describe the vulnerability/problem
# (why) describe its impact
# (how) describe the patch/fix

Weakness: <Weakness Name or CWE-ID>
Severity: <Low, Medium, High and Critical>
CVSS: <Numerical representation (0-10) of severity>
Detection: <Detection Method>
Report: <Report Link>
Introduced in: <Commit Hash>

Reported-by: <Name> (<Contact>)
Reviewed-by: <Name> (<Contact>)
Co-authored-by: <Name> (<Contact>)
Signed-off-by: <Name> (<Contact>)

Bug-tracker: <Bug-tracker Link>
OR
Resolves: <Issue/PR No.>
See also: <Issue/PR No.>
```

SECOM

(Fields)

Field	Description	Rationale
type	Usage of `vuln-fix` at the beginning of the header/subject to specify the fix is related to a vulnerability.	A **type** should be assigned to each commit [21]—which will make the identification of vulnerability fixes easier. The `vuln-fix` value was proposed by the Google OSV team during the feedback collection **(F)** phase. In addition, 4.10% of commits follow the conventional commits convention “<type>(scope):”.
Header/Subject	It should be approximately 50 chars (max 72 chars), capitalized with no period in the end and in the imperative form.	According to the common best practices for commit messages, it is important to summarize the purpose of the commit in one line [36, 38]. In our best practices analysis, we observed that 100% of commit messages had an header but only 38.85% had security-related words and represented an action.
Vuln-ID	When available, e.g., CVE, OSV, GHSA, and other formats.	Adding the vulnerability ID to the header/subject can help to localize the commit responsible for patching the vulnerability faster using features like reflog or shortlog. Only 12.1% of commit messages included mentions of the vulnerability ID, but 4 out of the 7 participants in **(F)** phase found including the vulnerability ID in the message important.
Body	Describe the vulnerability (what), its impact (why), and the patch to fix the vulnerability (how) in approximately 75 words (25 words per point).	The body is the most important part of the commit message since it provides space to add details on the problem, impact, and solution [27]. In our empirical analysis, we observed that 59.91% commit messages have a body. However, only 4031 out of those 6875 cases included security related words or had meaningful information.
Weakness	Common Weakness Enumeration ID or name.	The weakness ID provides information on which type of vulnerability can exist in the software. Software patch management teams may proceed differently according to the type of weakness. However, only 0.2% messages included this type of information.
Severity	Severity of the issue (Low, Medium, High, Critical).	Severity can motivate software users to perform patch management faster (in case, of critical vulnerabilities) [17]. Again, only 1.1% of commit messages mentioned severity levels.
CVSS	Numerical (0-10) representation of severity of a security vulnerability (Common Vulnerability Scoring System).	CVSS allows users to make better sense of the vulnerability severity and can motivate software users to perform patch management faster [17]. This field was proposed by a security engineer at OpenSSF that mentioned that sometimes is possible to calculate the score by following the CVSS questionnaire.
Detection	Detection method (Tool, Manual, etc).	It can be interesting to help future researchers with replication. 4 out of the 7 participants in the **(F)** phase sees value in adding this field (Table 6, RQ2).
Report	Link for vulnerability report which can back up the lack of information provided in commit messages. Our tool extracted	It usually provides more information on the vulnerability exploit or proof-of-concept. We observed that 3 out of the 7 participants would like to see links to reports, **(F)** phase (RQ1).
Introduced in	Commit hash from the commit that introduced the vulnerability.	Suggested by a survey participant of the **(F)** phase and used in the OSV Schema [42]. In addition, we found SHA keys in 1467 commits messages.
Signed-off by	Name and contact of the person that reported the issue.	To provide credit to the person that found the problem and ask for more details when necessary. However, only 8.4% commit messages were signed-off by the respective authors.
Reviewed-by	Name and contact of the person that reviewed and closed the issue.	Reviewers are usually the internal developers or senior developers that review and approve the issues. Only 3.33% of messages have the reviewers contact.
Bug-tracker	Link to the issue in an external bug-tracker or `Resolves... See also:` when GitHub is used to manage issues.	Important to document and discuss the problem, its impact, and patch with people involved. In our empirical analysis, we extracted URLs from a total of 929 commits.

Table 5: Fields description and rationale.

SECOM

(Compliance Checklist)

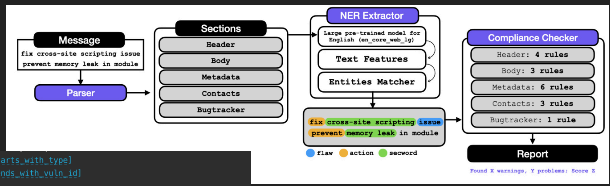
Header	type	Did you set the type of the commit as "vuln-fix" at the beginning of the header?	**M**
	header/subject	Did you summarize the patch changes?	**M**
	header/subject	Did you summarize the patch changes within ~50 chars?	**O**
	Vuln-ID	Is there a vulnerability ID available? Did you include it between parentheses at the end of the header?	**M**
Body	what	Did you describe the vulnerability or problem in the first sentence of the body?	**M**
	why	Did you describe the impact of the vulnerability in the second sentence of the body?	**M**
	how	Did you describe how the vulnerability was fixed in the third sentence?	**M**
	*	Did you describe the what, why, and how within ~75 words (~25 words per section)?	**O**
Metadata	Weakness	Can this vulnerability be classified with a type? If so, add it to the metadata section.	**M**
	Severity	Can infer severity (Low, Medium, High, Critical) for this vulnerability? If so, add it to the metadata section.	**M**
	CVSS	Can you calculate the numerical representation of the severity through the Common Vulnerability Scoring System calculator (https://www.first.org/cvss/calculator/3.0)?	**M**
	Detection	How did you find this vulnerability? (e.g., Tool, Manual, etc)	**O**
	Report	Is there a link for the vulnerability report available? If so, include it.	**O**
	Introduced in	Include the commit hash from the commit where the vulnerability was introduced.	**O**
Contacts	Reviewed-by	Include the name and/or contact of the person that reviewed and accepted the patch.	**O**
	Signed-off-by	Include the name and/or contact of the person that authored the patch.	**M**
Bug-Tracker	External	Include the link to the issues or pull requests in the external bug-tracker.	**O**
	GitHub	Include the links for the issues and pull-requests related to the patch (`Resolves.. See also:`).	**O**

Table 7: SECOM Compliance Checklist. [M-Mandatory; O-Optional; *-All fields in the section.]

SECOMlint

(Compliance Checker)

https://tqrg.github.io/secomlint




```
❌ Header is missing the vuln-fix type at the start. [header_starts_with_type]
🟡 Header is missing the vulnerability ID at the end. [header_ends_with_vuln_id]
🟡 Body has more than 75 words. [body_max_length]
🟡 Contacts section includes tag for reported-by but email is missing. [contacts_has_reported_by]
🟡 Contacts section includes tag or mention for co-authored-by but email is missing. [contacts_has_co_authored_by]
✅ Header size is within the max length (50 chars). [header_max_length]
✅ Header is not empty. [header_is_not_empty]
✅ Body is not empty. [body_is_not_empty]
✅ Body follows the what, why and how structure (three paragraphs). [body_has_three_paragraphs]
✅ Metadata mentions a weakness (CWE) id. [metadata_has_weakness]
✅ Metadata mentions severity. [metadata_has_severity]
✅ Metadata mentions report. [metadata_has_report]
✅ Metadata mentions sha where vulnerability was introduced in. [metadata_has_introduced_in]
✅ Contacts section includes signed-off-by info. [contact_has_signed_off_by]
✅ Bug tracker section includes references to issues. [bugtracker_has_reference]

found 1 problem(s), 4 warning(s); 🎯 Commit message is 70.59% in compliance with [SECOM] convention.
```

Future Work

- Explore GPT-3 to produce suggestions/recommendations based on the code—to shorten the burden of a new best practice.

Work in Progress

Improved annotation with an annotation tool for natural language called Prodigy.

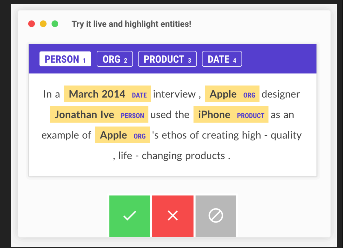
Trained a transformed based model for named entity recognition based on the data we extracted. Initial acc = 79%

Prodigy can access the uncertainty of each prediction. When it finds a case with high uncertainty, it presents the message and entities to the user for validation.

Active Learning - Different iterations of the model with new data (imp. of 5% after 4 iterations of 100 messages each)

prodigy

Radically efficient machine teaching. An annotation tool powered by active learning.



Not a real case; just a use case provided by the website

Future Work

Text Classification + NER

Outline

About me

Research Overview

Collection of SAST Tools

Software Vulnerability Detection + AI

Best Practices For Patch Documentation

Alert Prioritisation

Infrastructure-as-code (IaC) scripts

Fixing Vulnerabilities Potentially Hinders Software Maintainability

---

False Positives Prioritisation and Filtration

(Helping with triage of the alerts outputted by SASTs tools)

🐞🔒 Nowadays, many companies use static analysis tools (SASTs) to automate the detection of bugs and potential security violations.

😕 SASTs are known for their **high false positive rates** — general problem!

🤯 Extensive lists of warnings disrupt the developers' productivity since they are expected to judge each warning on their own, many times with poor knowledge and experience — time waster!

🚩 But, given that verification problems are undecidable, reporting false positive warnings is inevitable.

@rmaranhao

False Positives Prioritisation and Filtration

(Helping with triage of the alerts provided by SASTs tools)

😳 Infer produces a list of warnings without any specific order or priority assigned. Alert prioritisation or post processing may soften the impact of false positives in tool adoption.

@rmaranhao

False Positives Prioritisation and Filtration

(Helping with triage of the alerts provided by SASTs tools)

✅ Our approach orders the list of warnings by the probability of being a False Positive.

@rmaranhao

False Positives Prioritisation and Filtration

(Helping with triage of the alerts provided by SASTs tools)

Collection

Table 1: Alerts distribution per type of bug

Project	Resource Leak	Null Deref.	Alerts
apache-tomcat-9.0.50	66	230	296
avrora-0.1.52	20	28	48
joda-time-2.10.6	2	10	12
jython-2.7.2.2b3	62	118	180
xalan-j-2.7.1	10	38	48
jackrabbit-2.21.7	98	91	189
apollo-1.8.2	7	22	29
biojava-5.4.0	186	121	307
h2database-1.4.200	83	74	157
susi_server-230d679	58	39	97
Total	592	771	1363

Classification

Table 2: Alerts classification distribution per label (13 alerts were removed due to an Infer bug)

Project	True Positive	False Positive	Alerts
apache-tomcat-9.0.50	225	69	294
avrora-0.1.52	36	12	48
joda-time-2.10.6	11	1	12
jython-2.7.2.2b3	88	91	180
xalan-j-2.7.1	27	21	48
jackrabbit-2.21.7	89	100	189
apollo-1.8.2	16	13	29
biojava-5.4.0	203	104	307
h2database-1.4.200	121	31	152
susi_server-230d679	57	34	91
Total	874	476	1350

@rmaranhao

False Positives Prioritisation and Filtration

(Helping with triage of the alerts provided by SASTs tools)

We compared different deep learning architectures (LSTM, BERT, CodeBERT and GraphCodeBERT).

Model	Acc
LSTM	60.23
BERT	70.20
CodeBERT	74.26
GraphCodeBERT	77.23

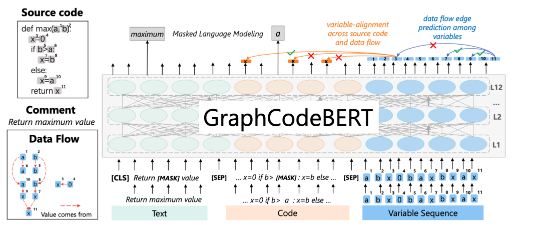


Figure 2: An illustration about GraphCodeBERT pre-training. The model takes source code paired with comment and the corresponding data flow as the input, and is pre-trained using standard masked language modeling (Devlin et al., 2018) and two structure-aware tasks. One structure-aware task is to predict where a variable is identified from (marked with orange lines) and the other is data flow edges prediction between variables (marked with blue lines).

@rmaranhao

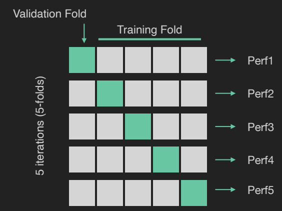
Training Configuration (k-fold cross validation)

Evaluating machine learning algorithms requires data separation into a:
Training set, used to estimate model parameters;
Test set, used to evaluate the classifier's performance.

We use the k-fold cross-validation technique:
- The dataset is split in k sets.
- One by one, is used for testing and the remaining k-1 other sets are used for training. This process is repeated k times for each set.

We performed a 5-fold cross validation for both scenarios. Each execution was performed 5 times with different random seeds (5-fold cross validation x 5 random seeds = 25 runs).

Why 25 runs? To check the consistency of the results.

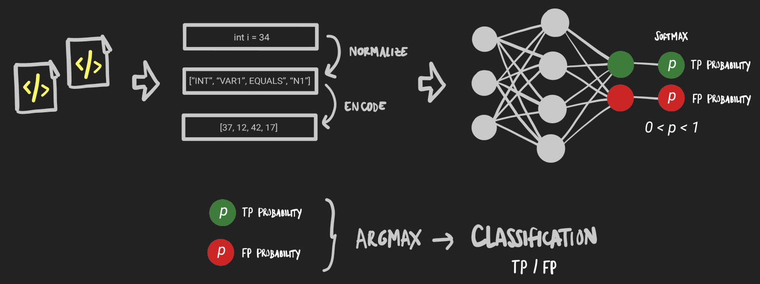


@rmaranhao

False Positives Prioritisation and Filtration

(Helping with triage of the alerts provided by SASTs tools)

We use a softmax layer to calculates the likelihood of a sample being a true positive or false positive [x, y] where x is the likelihood of being a true positive and y the likelihood of being a false positive — we use y to organize the list of warnings.



@rmaranhao

False Positive Probability Prediction

Infer's original output (First 10 warnings)

```
src/org/apache/xalan/xsltc/runtime/output/WriterOutputBuffer.java:38: error: NULL_DEREFERENCE
src/org/apache/xalan/extensions/XPathFunctionResolverImpl.java:61: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/util/JavaCupRedirect.java:63: error: RESOURCE_LEAK
src/org/apache/xalan/xsltc/compiler/FormatNumberCall.java:59: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:65: error: NULL_DEREFERENCE
src/org/apache/xml/serializer/SerializerBase.java:71: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:79: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:83: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/Key.java:90: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/trax/TrAXFilter.java:116: error: NULL_DEREFERENCE
```

Output Prioritized (First 10 warnings)

```
src/org/apache/xalan/xsltc/util/JavaCupRedirect.java:63: error: RESOURCE_LEAK
Probability of being a False Positive: 0.06495786
src/org/apache/xalan/xslt/EnvironmentCheck.java:134: error: RESOURCE_LEAK
Probability of being a False Positive: 0.09818842
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.java:1305: error: RESOURCE_LEAK
Probability of being a False Positive: 0.10622824
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.java:1312: error: RESOURCE_LEAK
Probability of being a False Positive: 0.10622824
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.java:1209: error: RESOURCE_LEAK
Probability of being a False Positive: 0.11100773
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.java:1164: error: RESOURCE_LEAK
Probability of being a False Positive: 0.11100773
src/org/apache/xalan/xsltc/runtime/AbstractTranslet.java:561: error: RESOURCE_LEAK
Probability of being a False Positive: 0.16833092
src/org/apache/xalan/xsltc/compiler/Key.java:90: error: NULL_DEREFERENCE
Probability of being a False Positive: 0.2923071
src/org/apache/xalan/xsltc/dom/DOMAdapter.java:184: error: NULL_DEREFERENCE
Probability of being a False Positive: 0.2968096
src/org/apache/xalan/xsltc/dom/DOMAdapter.java:249: error: NULL_DEREFERENCE
Probability of being a False Positive: 0.3001589
```


False Positive Probability Prediction

Infer's original output (First 10 warnings)

```
src/org/apache/xalan/xsltc/runtime/output/WriterOutputBuffer.java:38: error: NULL_DEREFERENCE
src/org/apache/xalan/extensions/XPathFunctionResolverImpl.java:61: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/util/JavaCupRedirect.java:63: error: RESOURCE_LEAK
src/org/apache/xalan/xsltc/compiler/FormatNumberCall.java:59: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:65: error: NULL_DEREFERENCE
src/org/apache/xml/serializer/SerializerBase.java:71: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:79: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:83: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/Key.java:90: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/trax/TrAXFilter.java:116: error: NULL_DEREFERENCE
```

Output Prioritized (First 10 warnings)


```
src/org/apache/xalan/xsltc/util/JavaCupRedirect.java:63: e
Probability of being a False Positive: 0.06495786
src/org/apache/xalan/xslt/EnvironmentCheck.java:134: erro
Probability of being a False Positive: 0.09818842
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.jav
Probability of being a False Positive: 0.10622824
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.jav
Probability of being a False Positive: 0.10622824
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.jav
Probability of being a False Positive: 0.11100773
src/org/apache/xalan/xsltc/trax/TransformerFactoryImpl.jav
Probability of being a False Positive: 0.11100773
src/org/apache/xalan/xsltc/runtime/AbstractTranslet.java:5
Probability of being a False Positive: 0.16833092
src/org/apache/xalan/xsltc/compiler/Key.java:90: error: NU
Probability of being a False Positive: 0.2923071
src/org/apache/xalan/xsltc/dom/DOMAdapter.java:184: error:
Probability of being a False Positive: 0.2968096
src/org/apache/xalan/xsltc/dom/DOMAdapter.java:249: error: NULL_DEREFERENCE
Probability of being a False Positive: 0.3001589
```

List is in ascending order of being a false positive, i.e., true positives appear in the top of the list.

False Positive Probability Prediction

Infer's original output (First 10 warnings)

```
src/org/apache/xalan/xsltc/runtime/output/WriterOutputBuffer.java:38: error: NULL_DEREFERENCE
src/org/apache/xalan/extensions/XPathFunctionResolverImpl.java:61: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/util/JavaCupRedirect.java:63: error: RESOURCE_LEAK
src/org/apache/xalan/xsltc/compiler/FormatNumberCall.java:59: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:65: error: NULL_DEREFERENCE
src/org/apache/xml/serializer/SerializerBase.java:71: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:79: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/ApplyImports.java:83: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/compiler/Key.java:90: error: NULL_DEREFERENCE
src/org/apache/xalan/xsltc/trax/TrAXFilter.java:116: error: NULL_DEREFERENCE
```

Output Prioritized (First 10 warnings)

```
src/org/apache/xalan/xsltc/util/JavaCupRedirect.java:63: e
Probability of being a False Positive: 0.06495786
/xalan/xslt/EnvironmentCheck.java:134: erro
being a False Positive: 0.09818842
ormerFactoryImpl.jav
being a False Positive: 0.10622824
/xalan/xsltc/trax/TransformerFactoryImpl.jav
being a False Positive: 0.10622824
/xalan/xsltc/trax/TransformerFactoryImpl.jav
being a False Positive: 0.11100773
/xalan/xsltc/trax/TransformerFactoryImpl.jav
being a False Positive: 0.11100773
/xalan/xsltc/runtime/AbstractTranslet.java:5
being a False Positive: 0.16833092
/xalan/xsltc/compiler/Key.java:90: error: NU
being a False Positive: 0.2923071
/xalan/xsltc/dom/DOMAdapter.java:184: error:
being a False Positive: 0.2968096
/xalan/xsltc/dom/DOMAdapter.java:249: error: NULL_DEREFERENCE
Probability of being a False Positive: 0.3001589
```

If we wanted to make FP filtration, could we simply remove false positives from the list? **Not quite because of misclassifications.**

List is in ascending order of being a false positive, i.e., true positives appear in the top of the list.

Can we use uncertainty to remove false positives?

Uncertainty refers to the lack of confidence for each output of a machine learning algorithm.

How do we calculate it so far?
Using a MonteCarlo dropout approach.

- Analyze the different outputs generated by the T forward passes.
- The higher the value, the more uncertain the model is.

Uncertainty (MonteCarlo Dropout) — T=5

1 means False Positive; 0 means True Positive; Pred means prediction.

Uncertainty distribution for the false alarms detected correctly (Label: 1, Pred: 1)

```
count    47.000000
mean      0.121971
std       0.095684
min       0.016707
25%       0.041606
50%       0.099058
75%       0.175737
max       0.366078
Name: predictive_unc_out, dtype: float64
```

Uncertainty distribution for the real alarms predicted as false alarms (Label: 0, Pred: 1)

```
count    31.000000
mean      0.300449
std       0.082733
min       0.095695
25%       0.284798
50%       0.343593
75%       0.354827
max       0.367706
Name: predictive_unc_out, dtype: float64
```

Uncertainty distribution for the false alarms predicted as real alarms (Label: 1, Pred: 0)

```
count    24.000000
mean      0.254796
std       0.105699
min       0.077910
25%       0.160091
50%       0.297044
75%       0.352268
max       0.367724
Name: predictive_unc_out, dtype: float64
```

Can we use uncertainty to remove false positives?

Uncertainty (MonteCarlo Dropout) — T=5

1 means False Positive; 0 means True Positive; Pred means prediction.

Uncertainty distribution for the false alarms detected correctly (Label: 1, Pred: 1)

Uncertainty distribution for the real alarms predicted as false alarms (Label: 0, Pred: 1)

Uncertainty distribution for the false alarms predicted as real alarms (Label: 1, Pred: 0)

The next question is **"how to use these uncertainty values to fix the false positive filtration issue"?**

```
1189.0 1.0 1.0 ( 0.9808582663536072 , 0.0167069364060149 )
660.0 1.0 1.0 ( 0.9598968625068665 , 0.0200239749043437 )
559.0 1.0 1.0 ( 0.972375988960266 , 0.0251415324887267 )
577.0 1.0 1.0 ( 0.9828186631202698 , 0.0266482426567068 )
500.0 1.0 1.0 ( 0.9800212979316713 , 0.0271233179453896 )
770.0 1.0 1.0 ( 0.9784590005874634 , 0.0272263334816144 )
236.0 1.0 1.0 ( 0.9633317589759828 , 0.0280264249865703 )
597.0 1.0 1.0 ( 0.974230170249939 , 0.0301422853449366 )
238.0 1.0 1.0 ( 0.9778905510902404 , 0.030943173810184 )
573.0 1.0 1.0 ( 0.9571858048439026 , 0.0322401585060153 )
742.0 1.0 1.0 ( 0.9547353982925416 , 0.0367057303005549 )
1054.0 1.0 1.0 ( 0.974616289138794 , 0.0398652718347465 )
1331.0 1.0 1.0 ( 0.9473590850830078 , 0.0433458915435773 )
579.0 1.0 1.0 ( 0.968421757221222 , 0.0476049305169327 )
1157.0 1.0 1.0 ( 0.9297662973403932 , 0.0489453594022135 )
94.0 1.0 1.0 ( 0.9841968417167664 , 0.0510306827585265 )
99.0 1.0 1.0 ( 0.967397689819336 , 0.0535892701838277 )
1187.0 1.0 1.0 ( 0.9521318078041076 , 0.0636437384511407 )
528.0 1.0 1.0 ( 0.9391631484031676 , 0.066620227034892 )
1121.0 1.0 1.0 ( 0.9164852499961852 , 0.0705890700726293 )
1057.0 1.0 1.0 ( 0.9609205722808838 , 0.0802505084974802 )
414.0 1.0 1.0 ( 0.812978982925415 , 0.081460339762227 )
467.0 1.0 1.0 ( 0.953866720199585 , 0.0844247870401106 )
1155.0 1.0 1.0 ( 0.7676697373390198 , 0.0990576612115126 )
20.0 1.0 1.0 ( 0.9529691338539124 , 0.0993099416542683 )
1107.0 1.0 1.0 ( 0.8964836597442627 , 0.1094783417498151 )
569.0 1.0 1.0 ( 0.923535704612732 , 0.1126134460854173 )
224.0 1.0 1.0 ( 0.8990851044654846 , 0.1189801196792885 )
333.0 1.0 1.0 ( 0.9325357675552368 , 0.1193484347387036 )
323.0 1.0 1.0 ( 0.9021071791648864 , 0.1356616176612773 )
563.0 1.0 1.0 ( 0.8537898063659668 , 0.1390569551607128 )
688.0 1.0 1.0 ( 0.8205998539924622 , 0.1506043838108364 )
310.0 1.0 1.0 ( 0.8858692646026611 , 0.1548434509196032 )
632.0 1.0 1.0 ( 0.7768675684928894 , 0.1657737582232674 )
89.0 1.0 1.0 ( 0.873923659324646 , 0.1729220861542172 )
567.0 1.0 1.0 ( 0.8919172883033752 , 0.1785520589750137 )
509.0 1.0 1.0 ( 0.7880885601043701 , 0.1961830362571335 )
716.0 1.0 1.0 ( 0.7272935509681702 , 0.1965394816717129 )
```

Can we use uncertainty to remove false positives?

Uncertainty (MonteCarlo Dropout) — T=5

1 means False Positive; 0 means True Positive; Pred means prediction.

The next question is "how to use these uncertainty values to fix the false positive filtration issue"?

This list is ordered by the descending order of being a False Positive.
idx_alert, Label, Pred, (P_fp, Unc)

Can we use uncertainty to remove false positives and reduce de list of alerts?

Uncertainty (MonteCarlo Dropout) — T=5

1 means False Positive; 0 means True Positive, Pred means prediction

	Uncertainty distribution for the false alarms detected correctly (Label: 1, Pred: 1)	Uncertainty distribution for the real alarms predicted as false alarms (Label: 0, Pred: 1)	Uncertainty distribution for the false alarms predicted as real alarms (Label: 1, Pred: 0)
count	47.000000	31.000000	24.000000
mean	0.121971	0.300449	0.254796
std	0.095684	0.082733	0.105699
min	0.016707	0.095695	0.077910
25%	0.041606	0.284798	0.160091
50%	0.099058	0.343593	0.297044
75%	0.175737	0.354827	0.352268
max	0.366078	0.367706	0.367724
Name: predictive_unc_out, dtype: float64			

The next question is **"how to use these uncertainty values to fix the false positive filtration issue"?**
idx_alert, Label, Pred, (P_fp, Unc)

```
1189.0 1.0 1.0 ( 0.9808582663536072 , 0.0167069364060149 )
660.0 1.0 1.0 ( 0.9598968625068665 , 0.0200239749043437 )
559.0 1.0 1.0 ( 0.972375988960266 , 0.025141532488726 7 )
577.0 1.0 1.0 ( 0.9828186631202698 , 0.0266482426567068 )
500.0 1.0 1.0 ( 0.9800212979316713 , 0.0271233179453896 )
770.0 1.0 1.0 ( 0.9784590005874634 , 0.0272263334816144 )
236.0 1.0 1.0 ( 0.9633317589759828 , 0.0280264249865703 )
597.0 1.0 1.0 ( 0.974230170249939 , 0.0301422853449366 )
238.0 1.0 1.0 ( 0.9778905510902404 , 0.030943173810184 )
573.0 1.0 1.0 ( 0.9571858048439026 , 0.0322401585060153 )
742.0 1.0 1.0 ( 0.9547353982925416 , 0.0367057303005549 )
1054.0 1.0 1.0 ( 0.974616289138794 , 0.0398652718347465 )
1331.0 1.0 1.0 ( 0.9473590850830078 , 0.0433458915435773 )
579.0 1.0 1.0 ( 0.968421757221222 , 0.0476049305169327 )
1157.0 1.0 1.0 ( 0.9297662973403932 , 0.0489453594022135 )
94.0 1.0 1.0 ( 0.9841968417167664 , 0.0510306827585265 )
99.0 1.0 1.0 ( 0.967397689819336 , 0.0535892701838277 )
1187.0 1.0 1.0 ( 0.9521318078041076 , 0.0636437384511407 )
528.0 1.0 1.0 ( 0.9391631484031676 , 0.066620227034892 )
1121.0 1.0 1.0 ( 0.9164852499961852 , 0.0705890700726293 )
1057.0 1.0 1.0 ( 0.9609205722808838 , 0.0802505084974802 )
414.0 1.0 1.0 ( 0.812978982925415 , 0.081460339762227 )
467.0 1.0 1.0 ( 0.953866720199585 , 0.0844247870401106 )
1155.0 1.0 1.0 ( 0.7676697373390198 , 0.0990576612115126 )
20.0 1.0 1.0 ( 0.9529691338539124 , 0.0993099416542683 )
1107.0 1.0 1.0 ( 0.8964836597442627 , 0.1094783417498151 )
569.0 1.0 1.0 ( 0.923535704612732 , 0.1126134460854173 )
224.0 1.0 1.0 ( 0.8990851044654846 , 0.1189801196792885 )
333.0 1.0 1.0 ( 0.9325357675552368 , 0.1193484347387036 )
323.0 1.0 1.0 ( 0.902107179164886 4 , 0.1356616176612773 )
563.0 1.0 1.0 ( 0.8537898063659668 , 0.1390569551607128 )
688.0 1.0 1.0 ( 0.8205998539924622 , 0.1506043838108364 )
310.0 1.0 1.0 ( 0.885869264602661 1 , 0.1548434509196032 )
632.0 1.0 1.0 ( 0.7768675684928894 , 0.1657737582232674 )
89.0 1.0 1.0 ( 0.873923659324646 , 0.1729220861542172 )
567.0 1.0 1.0 ( 0.8919172883033752 , 0.1785520589750137 )
509.0 1.0 1.0 ( 0.7880885601043701 , 0.1961830362571335 )
716.0 1.0 1.0 ( 0.7272935509681702 , 0.1965394816717129 )
```


Can we use uncertainty to remove false positives and reduce de list of alerts?

Uncertainty (MonteCarlo Dropout) — T=5

1 means False Positive; 0 means True Positive, Pred means prediction

	Uncertainty distribution for the false alarms detected correctly (Label: 1, Pred: 1)	Uncertainty distribution for the real alarms predicted as false alarms (Label: 0, Pred: 1)	Uncertainty distribution for the false alarms predicted as real alarms (Label: 1, Pred: 0)
count	47.000000	31.000000	24.000000
mean	0.121971	0.300449	0.254796
std	0.095684	0.082733	0.105699
min	0.016707	0.095695	0.077910
25%	0.041606	0.284798	0.160091
50%	0.099058	0.343593	0.297044
75%	0.175737	0.354827	0.352268
max	0.366078	0.367706	0.367724
Name: predictive_unc_out, dtype: float64			

The next question is **"how to use these uncertainty values to fix the false positive filtration issue"?**
idx_alert, Label, Pred, (P_fp, Unc)

One way is to simply output the prediction, prob_fp and uncertainty together with the alert information and leave to the user to make a decision (but now with more information).


```
1189.0 1.0 1.0 ( 0.9808582663536072 , 0.0167069364060149 )
660.0 1.0 1.0 ( 0.9598968625068665 , 0.0200239749043437 )
559.0 1.0 1.0 ( 0.972375988960266 , 0.025141532488726 7 )
577.0 1.0 1.0 ( 0.9828186631202698 , 0.0266482426567068 )
500.0 1.0 1.0 ( 0.9800212979316713 , 0.0271233179453896 )
770.0 1.0 1.0 ( 0.9784590005874634 , 0.0272263334816144 )
236.0 1.0 1.0 ( 0.9633317589759828 , 0.0280264249865703 )
597.0 1.0 1.0 ( 0.974230170249939 , 0.0301422853449366 )
238.0 1.0 1.0 ( 0.9778905510902404 , 0.030943173810184 )
573.0 1.0 1.0 ( 0.9571858048439026 , 0.0322401585060153 )
742.0 1.0 1.0 ( 0.9547353982925416 , 0.0367057303005549 )
1054.0 1.0 1.0 ( 0.974616289138794 , 0.0398652718347465 )
1331.0 1.0 1.0 ( 0.9473590850830078 , 0.0433458915435773 )
579.0 1.0 1.0 ( 0.968421757221222 , 0.0476049305169327 )
1157.0 1.0 1.0 ( 0.9297662973403932 , 0.0489453594022135 )
94.0 1.0 1.0 ( 0.9841968417167664 , 0.0510306827585265 )
99.0 1.0 1.0 ( 0.967397689819336 , 0.0535892701838277 )
1187.0 1.0 1.0 ( 0.9521318078041076 , 0.0636437384511407 )
528.0 1.0 1.0 ( 0.9391631484031676 , 0.066620227034892 )
1121.0 1.0 1.0 ( 0.9164852499961852 , 0.0705890700726293 )
1057.0 1.0 1.0 ( 0.9609205722808838 , 0.0802505084974802 )
414.0 1.0 1.0 ( 0.812978982925415 , 0.081460339762227 )
467.0 1.0 1.0 ( 0.953866720199585 , 0.0844247870401106 )
1155.0 1.0 1.0 ( 0.7676697373390198 , 0.0990576612115126 )
20.0 1.0 1.0 ( 0.9529691338539124 , 0.0993099416542683 )
1107.0 1.0 1.0 ( 0.8964836597442627 , 0.1094783417498151 )
569.0 1.0 1.0 ( 0.923535704612732 , 0.1126134460854173 )
224.0 1.0 1.0 ( 0.8990851044654846 , 0.1189801196792885 )
333.0 1.0 1.0 ( 0.9325357675552368 , 0.1193484347387036 )
323.0 1.0 1.0 ( 0.902107179164886 4 , 0.1356616176612773 )
563.0 1.0 1.0 ( 0.8537898063659668 , 0.1390569551607128 )
688.0 1.0 1.0 ( 0.8205998539924622 , 0.1506043838108364 )
310.0 1.0 1.0 ( 0.885869264602661 1 , 0.1548434509196032 )
632.0 1.0 1.0 ( 0.7768675684928894 , 0.1657737582232674 )
89.0 1.0 1.0 ( 0.873923659324646 , 0.1729220861542172 )
567.0 1.0 1.0 ( 0.8919172883033752 , 0.1785520589750137 )
509.0 1.0 1.0 ( 0.7880885601043701 , 0.1961830362571335 )
716.0 1.0 1.0 ( 0.7272935509681702 , 0.1965394816717129 )
```

Can we use uncertainty to remove false positives and reduce de list of alerts?

Uncertainty (MonteCarlo Dropout) — T=5

1 means False Positive; 0 means True Positive, Pred means prediction

	Uncertainty distribution for the false alarms detected correctly (Label: 1, Pred: 1)	Uncertainty distribution for the real alarms predicted as false alarms (Label: 0, Pred: 1)	Uncertainty distribution for the false alarms predicted as real alarms (Label: 1, Pred: 0)
count	47.000000	31.000000	24.000000
mean	0.121971	0.300449	0.254796
std	0.095684	0.082733	0.105699
min	0.016707	0.095695	0.077910
25%	0.041606	0.284798	0.160091
50%	0.099058	0.343593	0.297044
75%	0.175737	0.354827	0.352268
max	0.366078	0.367706	0.367724
Name: predictive_unc_out, dtype: float64			

The next question is **"how to use these uncertainty values to fix the false positive filtration issue"?**
idx_alert, Label, Pred, (P_fp, Unc)

One way is to simply output the prediction, prob_fp and uncertainty together with the alert information and leave to the user to make a decision (but now with more information).

The other is to use descriptive statistics to find a threshold. For instance, the min values for misclassifications are **0.095695** and **0.077910**. Therefore, if we pick a threshold of **0.075** (which is smaller than both min values), we can achieve a reduction of 20 out of 71 FPs — a reduction of 28% of false alarms in the actual list of alerts provided by Infer.

Work in Progress Exploring Confidence Intervals Theory for Deep Learning to find the misclassified correctly


```
1189.0 1.0 1.0 ( 0.9808582663536072 , 0.0167069364060149 )
660.0 1.0 1.0 ( 0.9598968625068665 , 0.0200239749043437 )
559.0 1.0 1.0 ( 0.972375988960266 , 0.025141532488726 7 )
577.0 1.0 1.0 ( 0.9828186631202698 , 0.0266482426567068 )
500.0 1.0 1.0 ( 0.9800212979316713 , 0.0271233179453896 )
770.0 1.0 1.0 ( 0.9784590005874634 , 0.0272263334816144 )
236.0 1.0 1.0 ( 0.9633317589759828 , 0.0280264249865703 )
597.0 1.0 1.0 ( 0.974230170249939 , 0.0301422853449366 )
238.0 1.0 1.0 ( 0.9778905510902404 , 0.030943173810184 )
573.0 1.0 1.0 ( 0.9571858048439026 , 0.0322401585060153 )
742.0 1.0 1.0 ( 0.9547353982925416 , 0.0367057303005549 )
1054.0 1.0 1.0 ( 0.974616289138794 , 0.0398652718347465 )
1331.0 1.0 1.0 ( 0.9473590850830078 , 0.0433458915435773 )
579.0 1.0 1.0 ( 0.968421757221222 , 0.0476049305169327 )
1157.0 1.0 1.0 ( 0.9297662973403932 , 0.0489453594022135 )
94.0 1.0 1.0 ( 0.9841968417167664 , 0.0510306827585265 )
99.0 1.0 1.0 ( 0.967397689819336 , 0.0535892701838277 )
1187.0 1.0 1.0 ( 0.9521318078041076 , 0.0636437384511407 )
528.0 1.0 1.0 ( 0.9391631484031676 , 0.066620227034892 )
1121.0 1.0 1.0 ( 0.9164852499961852 , 0.0705890700726293 )
1057.0 1.0 1.0 ( 0.9609205722808838 , 0.0802505084974802 )
414.0 1.0 1.0 ( 0.812978982925415 , 0.081460339762227 )
467.0 1.0 1.0 ( 0.953866720199585 , 0.0844247870401106 )
1155.0 1.0 1.0 ( 0.7676697373390198 , 0.0990576612115126 )
20.0 1.0 1.0 ( 0.9529691338539124 , 0.0993099416542683 )
1107.0 1.0 1.0 ( 0.8964836597442627 , 0.1094783417498151 )
569.0 1.0 1.0 ( 0.923535704612732 , 0.1126134460854173 )
224.0 1.0 1.0 ( 0.8990851044654846 , 0.1189801196792885 )
333.0 1.0 1.0 ( 0.9325357675552368 , 0.1193484347387036 )
323.0 1.0 1.0 ( 0.902107179164886 4 , 0.1356616176612773 )
563.0 1.0 1.0 ( 0.8537898063659668 , 0.1390569551607128 )
688.0 1.0 1.0 ( 0.8205998539924622 , 0.1506043838108364 )
310.0 1.0 1.0 ( 0.885869264602661 1 , 0.1548434509196032 )
632.0 1.0 1.0 ( 0.7768675684928894 , 0.1657737582232674 )
89.0 1.0 1.0 ( 0.873923659324646 , 0.1729220861542172 )
567.0 1.0 1.0 ( 0.8919172883033752 , 0.1785520589750137 )
509.0 1.0 1.0 ( 0.7880885601043701 , 0.1961830362571335 )
716.0 1.0 1.0 ( 0.7272935509681702 , 0.1965394816717129 )
```


Outline

- About me
- Research Overview
- Collection of SAST Tools
- Software Vulnerability Detection + AI
- Best Practices For Patch Documentation
- Alert Prioritisation
- **Infrastructure-as-code (IaC) scripts**
- Fixing Vulnerabilities Potentially Hinders Software Maintainability

Infrastructure-as-code (IaC) scripts in Puppet Manifests

(New technology based on scripts that are prone to security vulnerabilities [1])

- Software configuration management and deployment tools like **Puppet** became popular amongst software development warehouses.
- These tools help infrastructure teams **increase productivity** by automating various config tasks (e.g., server setup) through scripts that can be *reused* and *versioned*.
- As with any piece of code, IaC scripts are also prone to defects such as **security vulnerabilities**.

Oh gosh!

199K vulnerable
IaC templates

paloalto

67k potential
Security Smells in IaC

Rahman et al. [ICSE'19; TSE'20]

[1] Akond Rahman, Chris Parnin, Laurie Williams. The Seven Sins: Security Smells in Infrastructure as Code Scripts. ICSE'19

@rmaranhao

Assessment > 12 types of weaknesses

Weakness	Name	Example
CWE-798	Use of Hard Coded Credentials	$username = "mariadb"
CWE-269	Use of Hard Coded Password	$password = "!TQ23Rg"
CWE-321	Use of Hard Coded Cryptographic Key	$key = "A67ANBD7"
CWE-319	Use of HTTP without TLS	$req = "http://www.domain.org/secret"
CWE-546	Suspicious Comment	#https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=538392
CWE-326	Use of Weak Crypto Algorithms	password => md5($debian_password)
CWE-284	Invalid IP address Binding	$bind_host = "0.0.0.0"
CWE-258	Empty Password in Configuration File	$rabbitmq_pwd = ""
CWE-250	Admin by default	$user = "admin"
CWE-521	Weak Password	pwd => "12345"
CWE-1007	Homoglyphs Detection (typo-squatting attacks)	$source = "http://deb.debiαn.org/debiαn"
CWE-829	Malicious Dependencies	$postgresql_version = 8.4

@rmaranhao

Motivation > Automated Security Weakness Detection in Puppet

- Focus on **Puppet**
- Lightweight Solution Available (called **SLIC**) [Rahman et al., ICSE'19]
  99% of precision and accuracy in an oracle dataset
- **SLIC** detects 7 types of weaknesses.

1st question: How does **SLIC** perform on a new dataset?

@rmaranhao

Study 1 > Validation with Students

Research Team

1419 GitHub repositories (~34k Puppet Scripts).

Found **31990 security warnings** on 9144 of Puppet scripts.

Table 2: Breakdown of warnings reported by SLIC.

Rule	#	%
Hard-coded secrets	22365	69.9
Use of HTTP without TLS	3757	11.7
Suspicious comments	2780	8.7
Use of Weak Crypto. Algos.	1489	4.7
Invalid IP Address Binding	769	2.4
Empty Password	684	2.1
Admin by default	146	0.5
Total	31990	100

@rmaranhao

Study 1 > Validation with Students

Research Team

2 authors validated a total of 502 warnings.

Two samples: **proportional** and **uniform**.

Table 3: Performance of SLIC. (Validation with Students)

SLIC	*proportional*			*uniform*		
Rule	#TP	#FP	Pr.	#TP	#FP	Pr.
Hard-coded secrets	122	52	0.70	26	10	0.72
Use of HTTP without TLS	9	20	0.31	10	26	0.28
Suspicious comments	10	12	0.45	8	28	0.22
Use of Weak Crypto. Algorithms	7	4	0.64	25	11	0.69
Invalid IP Address Binding	6	0	1.00	28	8	0.78
Empty Password	4	2	0.67	21	15	0.58
Admin by default	1	1	0.50	21	15	0.58
Total	159	91	0.64	139	113	0.55

Precision decreased from 99% to 64%. 🙁

Study 1 > Validation with Students

Research Team

2 authors validated a total of 502 warnings.

Two samples: **proportional** and **uniform**.

Table 3: Performance of SLIC. (Validation with Students)

SLIC	*proportional*			*uniform*		
Rule	#TP	#FP	Pr.	#TP	#FP	Pr.
Hard-coded secrets	122	52	0.70	26	10	0.72
Use of HTTP without TLS	9	20	0.31	10	26	0.28
Suspicious comments	10	12	0.45	8	28	0.22
Use of Weak Crypto. Algorithms	7	4	0.64	25	11	0.69
Invalid IP Address Binding	6	0	1.00	28	8	0.78
Empty Password	4	2	0.67	21	15	0.58
Admin by default	1	1	0.50	21	15	0.58
Total	159	91	0.64	139	113	0.55

Precision decreased from 99% to 64%. 🙁

Maybe we don't have enough context?! 🤔

Study 2 > Validation with OSS Maintainers

Maintainers

📧 Issued alerts to projects maintainers involved in the slack puppet community.

💀 Issues included the code sample, issues description and links to more information.

commented 6 days ago

The following script seems to have a hard-coded secret `cron_user=root`:

puppet-apt_mirror/manifests/init.pp
Line 191 in 2d0e6bb

```
191      $cron_user                = 'root',
```

A secret can be a password, user name, or private cryptographic key.

This type of smell can lead to well-known types of vulnerabilities, as documented by CWE (CWE-798 and CWE-259). Hard-coded secrets can be used to bypass protection mechanisms, gain privileges on applications and access to sensitive data.

Storing secrets in **Puppet** configuration files is considered to be a security smell (cf. [icse20]).

Recommendation
To protect/manage your secrets, it is recommended to use a vault (e.g., https://www.vaultproject.io/). After configuring the vault, you can replace your secrets by variables from the vault. For instance, replace `$password = '12345'` by `$password = $vault::password`. Thus, your secrets will no longer be disclosed publicly.

Location
Description
Assessment
Actionability

Study 2 > Validation with OSS Maintainers

Maintainers

Got 51 answers to the 228 issues submitted; but only 33 were **clearly validated**.

❌ "N/A";":thumbs_down:"

✅ "These todos's shouldn't be there, I agree…"

Study 2 > Validation with OSS Maintainers

Got 51 answers to the 228 issues submitted; but only 33 were **clearly validated**.

❌ "N/A";":thumbs_down"

✅ "These todos's shouldn't be there, I agree…"

Maintainers

Table 4: Performance of SLIC. (Validation with Owners)

Rule	#TP	#FP	Precision
Hard-coded secrets	77	119	0.39
Use of HTTP without TLS	1	72	0.01
Suspicious comments	3	15	0.17
Use of Weak Crypto. Algos.	0	3	0.00
Invalid IP Address Binding	0	1	0.00
Empty Password	1	5	0.17
Admin by default	1	0	1.00
Total	83	215	0.28

Ups! Precision is even worse.

Precision decreased to 28%,

1st question: How does **SLIC** perform on a new dataset?

Not great!

1st question: How does **SLIC** perform on a new dataset?

Not great!

Problem > Puppet IaC Security Linters are not reliable yet!

Precision is even lower when evaluated by maintainers—developers with more knowledge and context of the applications.

1st question: How does **SLIC** perform on a new dataset?

Not great!

Problem > Puppet IaC Security Linters are not reliable yet!

Precision is even lower when evaluated by maintainers—developers with more knowledge and context of the applications.

During study 1 and study 2, we were able to list several problems in the tool weakness- and analysis-related.

`if has_key($userdata, 'env')` SLIC found a hard coded secret in this logical condition

1st question: How does **SLIC** perform on a new dataset?

🙁 *Not great!*

Problem > Puppet IaC Security Linters are not reliable yet!

- Precision is even lower when evaluated by maintainers—developers with more knowledge and context of the applications.
- During study 1 and study 2, we were able to list several problems in the tool weakness- and analysis-related.

  if has_key($userdata, 'env')   SLIC found a hard coded secret in this logical condition

- Static analysis tools can be iteratively improved and extended by incorporating feedback from the developer community [Sadowski, ACM Commun.'18]

@rmaranhao

Methodology > Improve the linter with Practitioners' Feedback

SLIC
Rahman et al.

Phase 1

Students | Owners of OSS Projects
Meeting w/ Puppet Labs + Vox Pupuli

InfraSecure v0.1.0

Phase 2

Practitioners
Puppet Community | Prolific

InfraSecure v1.0.0

InfraSecure v1.1.0

@rmaranhao

InfraSecure v0.1.0 > Design Choices

Variable/Attribute Assignments (VASS)   Reduce the number of incorrect predictions
isVarAssign(token) ∧ isAtrAssign(token)

❌ if has_key($userdata, 'env')   SLIC found a hard coded secret in this logical condition

@rmaranhao

InfraSecure v0.1.0 > Design Choices

Variable/Attribute Assignments (VASS)   Reduce the number of incorrect predictions
isVarAssign(token) ∧ isAtrAssign(token)

❌ if has_key($userdata, 'env')   SLIC found a hard coded secret in this logical condition

Reasoning about the token value (TOKVAL)   Some of the rules did not reason about *token.value*

❌ aws_admin_username = downcase($::operatingsystem)   No secret is stored

@rmaranhao

InfraSecure v0.1.0 > Design Choices

Variable/Attribute Assignments (VASS) Reduce the number of incorrect predictions
isVarAssign(token) ∧ isAtrAssign(token)

✗ `if has_key($userdata, 'env')` SLIC found a hard coded secret in this logical condition

Reasoning about the token value (TOKVAL) Some of the rules did not reason about *token.value*

✗ `aws_admin_username = downcase($::operatingsystem)` No secret is stored

Credentials that are not consider secrets by the community **isUserDefault(token.value)**

[Maintainer] *"The names of these UNIX accounts are not considered to be secret. They are published openly as part of the PE documentation: https://puppet.com/docs/pe/ 2019.8/what_gets_installed_and_where.html#user_and_group_accounts_installed"*

@rmaranhao

InfraSecure v0.1.0 > Rule Improvements

Usage of Weak Crypto Algorithms Search for in calls to functions
isFunctionCall()

✗ `md5checksum = '07bd73571b7028b73fc8ed19bc85226d'` Not a call to the md5() function

Invalid IP address binding IPs follow dot-decimal notation
isInvalidIPBind(token.value)

✗ `description => 'Open up postgresql for access to sensu from 0.0.0.0/0'` Description != IP

Check our paper for more! **Section 4.3**

@rmaranhao

InfraSecure v0.1.0 > Design Choices

Table 6: Performance of InfraSecure v0.1.0.

InfraSecure v0.1.0	*proportional*			*uniform*		
Rule	#TP	#FP	Pr.	#TP	#FP	Pr.
Hard-coded secrets	118	22	0.84	24	4	0.86
Use of HTTP without TLS	8	17	0.32	9	23	0.28
Suspicious comments	5	2	0.71	6	10	0.38
Use of Weak Crypto. Algorithms	5	2	0.71	23	2	0.92
Invalid IP Address Binding	6	0	1.00	28	1	0.97
Empty Password	4	2	0.67	21	15	0.58
Admin by default	1	1	0.50	20	15	0.57
Total	147	46	0.76	131	70	0.65

Precision increased!

Can we improve even more?
Let's ask ***practitioners!***

@rmaranhao

Methodology > Improve the linter with Practitioners' Feedback

SLIC
Rahman et al.

Phase 1

Students | Owners of OSS Projects
Meeting w/ Puppet Labs + Vox Pupuli

InfraSecure v0.1.0

Phase 2

Practitioners
Puppet Community | Prolific

InfraSecure v1.0.0

InfraSecure v1.1.0

@rmaranhao

Study 3 > Validation with Practitioners

Practitioners

Validate InfraSecure v0.1.0 alerts

Experiment shared with the Puppet communities on Slack (puppet.community.slack.com) and Reddit (r/puppet).

14 participants

Prolific
117 participants

Validation of ⚠️
339 warnings

Pre-screening: Specific Industries (e.g., Computer and Electronics), experience with configuration management tools, security and infrastructure as a service; and, a quizz of three programming questions about different puppet configurations. (**check the replication package**)

InfraSecure v1.0.0 > More feedback and improvements

Use of HTTP without TLS is fine sometimes Customizable rule (whitelist with credible sources)
inWhitelist(token.value)

❌ Apturl => "http://deb.debian.org/debian SLIC reports every single occurence of http:// as unsafe.

[Practitioner] *"I think it is fine if localhost is used. Otherwise TLS should be mandatory. All the big financial organizations will not use this check because they cannot create internal certs or use letsencrypt."*

[Practitioner] *"By default, it's unsafe to not use HTTPS. But for internal testing/development it is acceptable to me to not use HTTPS all the time."*

InfraSecure v1.1.0 > New Patterns (Extension)

Weak Password — **isStrongPwd()** — Uses PHP algorithm developed by Thomas Hruska.

Homograph Attacks *supply chain attack* — **hasCyrillic()** — Social engineering attack that purposely uses misspelt domains for malicious purposes.

Malicious Dependencies *supply chain attack* — **isResource()** **isMalicious()** — Our database integrates malicious versions of software for 33 different packages used by the Puppet community (e.g., rabbitmq, apt, cassandra, postgresql, etc).

CWE-521	Weak Password	pwd => "12345"
CWE-1007	Homoglyphs Detection (typo-squatting attacks)	$source = "http://deb.debian.org/debian"
CWE-829	Malicious Dependencies	$postgresql_version = 8.4

Study 3 > Validation with Practitioners

Practitioners

Table 8: Performance of InfraSecure (v1.1.0). (Validation with Practitioners)

Rule	#TP	#FP	#Unsure	Precision
Hard-coded secrets	28	8	3	0.78
Use of HTTP without TLS	32	3	2	0.91
Suspicious Comments	16	15	7	0.52
Use of Weak Crypto. Algo.	33	3	6	0.92
Invalid IP Address Binding	26	8	6	0.77
Empty Password	33	3	1	0.92
Admin by default	30	6	6	0.83
Malicious Dependencies	25	6	3	0.81
Weak Password	32	2	0	0.94
Total	255	54	34	0.83

Table 9: Precision obtained in different cycles of feedback collection for InfraSecure.

Participants	version	Precision
Research Team, Owners of OSS Projects, PuppetLabs, Voxpupuli	v0.1.0	76%
Practitioners (cycle 1)	v1.0.0	79%
Practitioners (cycle 2)	v1.1.0	83%

Precision increased
between iterations (28% -> 76% -> 79% -> 83%)

More Anti-Patterns
Malicious dependencies, Homograph Attacks and Weak Passwords

More Customisation
Whitelist

🤩 Rules

Table 7: InfraSecure rules to detect security smells.

CWE	Weakness Name	Rule
CWE-321	Hard-coded Key	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isKey($t.prev\_code\_token$) ∧ isNonSecret($t.prev\_code\_token$) ∧ !isPlaceholder($t.next\_code\_token$)
CWE-259	Hard-coded Password	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isPassword($t.prev\_code\_token$) ∧ isNonSecret($t.prev\_code\_token$) ∧ !isPlaceholder($t.next\_code\_token$) ∧ !isUserDefault($t.next\_code\_token$) ∧ !invalidSecret($t.next\_code\_token$)
CWE-798	Hard-coded Usernames	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isUser($t.prev\_code\_token$) ∧ isNonSecret($t.prev\_code\_token$) ∧ !isPlaceholder($t.next\_code\_token$) ∧ !isUserDefault($t.next\_code\_token$) ∧ !invalidSecret($t.next\_code\_token$)
	ts	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ (isKey($t.prev\_code\_token$) ∨ isPassword($t.prev\_code\_token$) ∨ isUser($t.prev\_code\_token$)) ∧ !isPlaceholder($t.next\_code\_token$) ∧ !isUserDefault($t.next\_code\_token$) ∧ !invalidSecret($t.next\_code\_token$)
	out TLS	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isHTTP($t.next\_code\_token$) ∧ !inWhitelist($t.next\_code\_token$)
	ents	isComment(t) ∧ isSuspiciousWord(t)
	to. Algo.	(isVarAssign($t.prev\_code\_token$) ∨ isAtrAssign($t.prev\_code\_token$) ∨ isFunctionCall($t.next\_code\_token$)) ∧ !isCheckSum($t.prev\_code\_token$) ∧ isWeakCrypto($t.next\_code\_token$)
	Binding	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isInvalidIPBind($t.next\_code\_token$)
		(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isPassword($t.prev\_code\_token$) ∧ isEmptyPassword(t.prev_code_token)
		(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isNonSecret($t.prev\_code\_token$) ∧ isUser($t.prev\_code\_token$) ∧ !isPlaceholder($t.next\_code\_token$) ∧ isAdmin($t.next\_code\_token$)
	ks	(isVarAssign(t) ∨ isAtrAssign(t)) ∧ hasCyrillic($t.next\_code\_token$)
		(isVarAssign(t) ∨ isAtrAssign(t)) ∧ isPassword($t.prev\_code\_token$) ∧ isStrongPwd($t.next\_code\_token$)
	lencies	isResource(t) ∧ isVersion($t.prev\_code\_token$) ∧ isMalicious($t.next\_code\_token$)

...s if the URL is in the list of configurable safe domains/whitelist. If the URL is in the whitelist, an alert should not be raised. ...pt is in the database of malicious dependencies.

Table 5: InfraSecure's list of string and AST patterns.

<table>
<tr><th>Rule</th><th>String Pattern</th></tr>
<tr><td>isAdmin(t.value)</td><td>root|admin</td></tr>
<tr><td>isNonSecret(t.value)</td><td>gpg|path|type|buff|zone|mode|tag|header|scheme|length|guid</td></tr>
<tr><td>isPassword(t.value)</td><td>pass(word|_|$)|pwd</td></tr>
<tr><td>isUser(t.value)</td><td>user|usr</td></tr>
<tr><td>isKey(t.value)</td><td>(pvt|priv)+.*(cert|key|rsa|secret|ssl)+</td></tr>
<tr><td>isPlaceholder(t.value)</td><td>${.*}|($)?.*::.*(::)?</td></tr>
<tr><td>hasCyrillic(t.value)</td><td>^(http(s)?://)?.*\p{Cyrillic}+</td></tr>
<tr><td>isInvalidIPBind(t.value)</td><td>^((http(s)?://)?0.0.0.0(:\d{1,5})?)$</td></tr>
<tr><td>isSuspiciousWord(t.value)</td><td>hack|fixme|ticket|bug|checkme|secur|debug|defect|weak</td></tr>
<tr><td>isWeakCrypto(t.value)</td><td>^(sha1|md5)</td></tr>
<tr><td>isCheckSum(t.value)</td><td>checksum|gpg</td></tr>
<tr><td>isHTTP(t.value)</td><td>^http://.+</td></tr>
<tr><td>isUserDefault(t.value)</td><td>pe-puppet|pe-webserver|pe-puppe... postgres|pe-console-services|pe-orchestration-services|pe-ace-serv... bolt-server</td></tr>
<tr><td>invalidSecret(t.value)</td><td>undefined|unset|www-data|wwwrun|www|no|yes|[]|undef|true|false|changeit|changeme|none</td></tr>
<tr><td>isStrongPwd(t.value) [24]</td><td>StrongPassword::StrengthChecker(t.value)</td></tr>
<tr><td>isEmptyPassword(t.value)</td><td>t.value == “”</td></tr>
</table>

Check our paper for more! **Tables 5 & 7**

🐦@rmaranhao

Main Conclusions

👍 ***(1) It is feasible to tune security linters to produce acceptable precision.***

🦸 ***(2) Involving practitioners in discussions is an effective way to guide the improvement of those linters.***

💯 *In the process of feedback collection, tool owners can learn more on how to extend the **anti-patterns coverage** and how to better **customise** the tool!*

https://github.com/TQRG/puppet-lint-infrasecure

🐦@rmaranhao

Work in Progress

Exploring dynamic taint analysis to keep track of vaults (storage where secrets can be stored to not be hard-coded in the scripts).

Taint Analysis (Weak Password)

```
$password = ''                              Tainted Variable (source)
…
$password = $::vault:password
…
$user = do_something($password)             (sink)
```

The question is: what's inside **$::vault:password?** Is it safe?

By monitoring the code execution and interaction between Puppet and vaults (i.e., **dynamic analysis**), we could check the value stored in the $::vault:password variable.

if $::vault:password is non-weak, then it's safe

```
$password = ''                              Tainted Variable (source)
…
$password = $::vault:password               Loaded by DA (=  #!1mAn57)
…
$user = do_something($username, $password)  (sink)
```

if $::vault:password is weak, then it's not safe

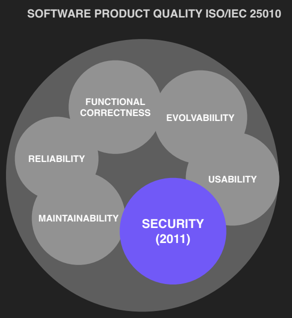
```
$password = ''                              Tainted Variable (source)
…
$password = $::vault:password               Loaded by DA (= 123456)
…
$user = do_something($username, $password)  (sink)
```

https://github.com/TQRG/puppet-lint-infrasecure

🐦@rmaranhao

Outline

Fixing Vulnerabilities Potentially Hinders Software Maintainability
(Do security patches have a negative impact in software maintainability?)



- Complex code is difficult to ***understand***, ***maintain*** and ***test***.
- Complexity hides bugs -> security vulnerabilities
- Software vulnerabilities ~ code complexity

The risk of software vulnerabilities can be minimised by writing clean and maintainable code.

@rmaranhao

Fixing Vulnerabilities Potentially Hinders Software Maintainability
(Do security patches have a negative impact in software maintainability?)

MAINTAINABLE SECURITY

The degree of effectiveness and efficiency with which software can be changed to mitigate a security vulnerability — *corrective maintenance*.

THE PROBLEM

- Many developers still lack knowledge on best practices to deliver and maintain secure and high-quality software.
- Software maintainability is ~75% of the cost related to a project.
- Available tooling does not provide any information on the quality of a patch.

WHY IS IT IMPORTANT?

- In a world where zero day vulnerabilities are constantly emerging, mitigation needs to be fast and efficient.

Therefore, it is important to write maintainable code to support the production of **more secure software** and, **prevent the introduction of new vulnerabilities**.

@rmaranhao

Fixing Vulnerabilities Potentially Hinders Software Maintainability
(Do security patches have a negative impact in software maintainability?)

BUT improving software security is not a trivial task and requires implementing patches that might **affect** software maintainability.

PREVIOUS RESEARCH

34% of security patches performed introduce new problems and 52% are incomplete and do not fully secure systems.

OUR HYPOTHESIS

Some of these patches may have a negative impact on the software maintainability and, possibly, even be the cause of the introduction of new vulnerabilities — **harming software reliability** and **introducing technical debt**.

MAIN CONTRIBUTION TO THE SE COMMUNITY

Evidence that supports the trade-off between security and maintainability: developers may be hindering software maintainability while patching vulnerabilities.

@rmaranhao

Fixing Vulnerabilities Potentially Hinders Software Maintainability
(Do security patches have a negative impact in software maintainability?)

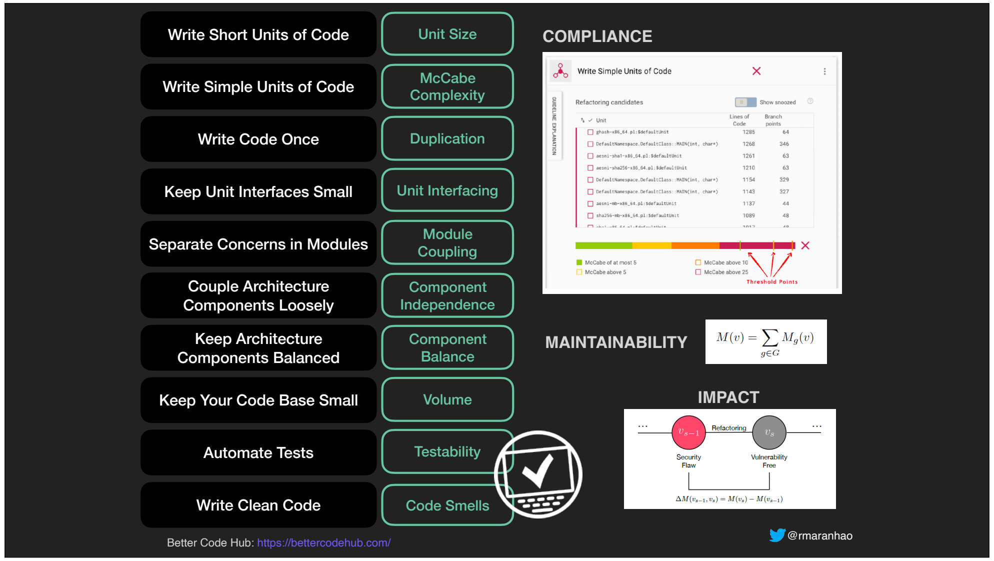
MOTIVATION

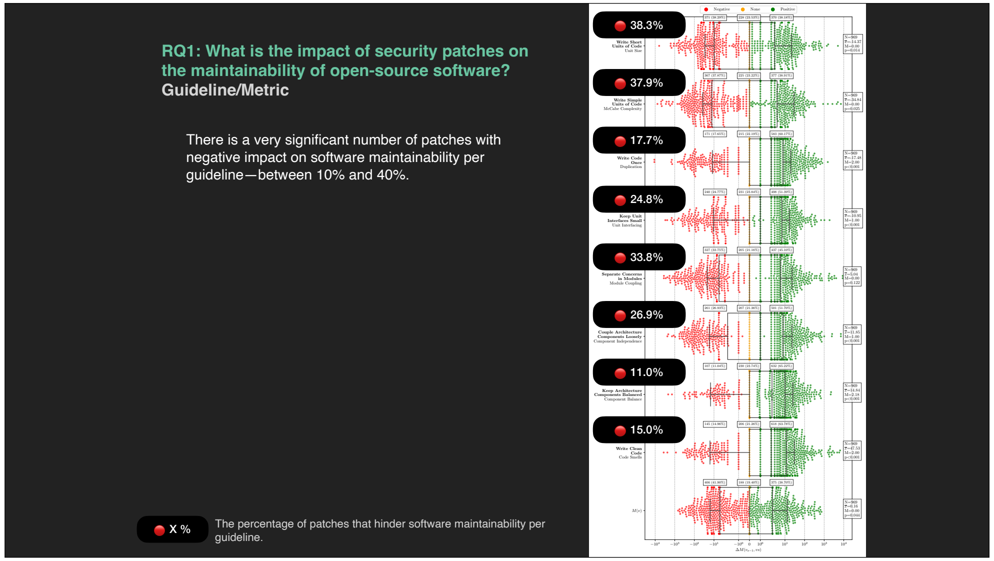
More lines of code
More cyclomatic complexity

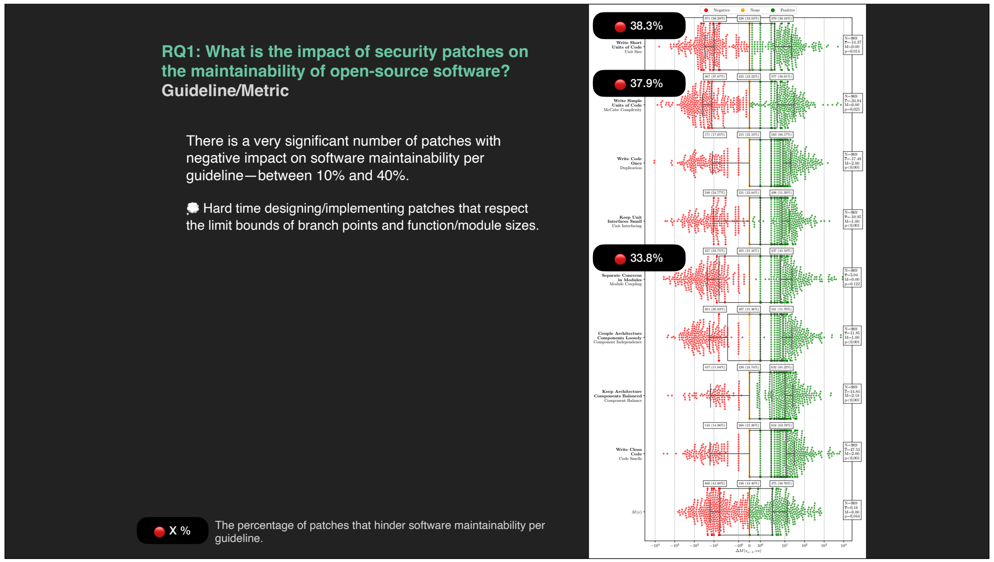
```
static int ssl_scan_clienthello_tlsext(SSL *s, PACKET *pkt,
    int *al){
 // [snip]
+  sk_OCSP_RESPID_pop_free(s->tlsext_ocsp_ids,
    OCSP_RESPID_free); ❶
+  if (PACKET_remaining(&responder_id_list) > 0) {
+      s->tlsext_ocsp_ids = sk_OCSP_RESPID_new_null();
+      if (s->tlsext_ocsp_ids == NULL) { ❷
+          *al = SSL_AD_INTERNAL_ERROR;
+          return 0;
+      }
+  } else {
+      s->tlsext_ocsp_ids = NULL;
+  }

   while (PACKET_remaining(&responder_id_list) > 0) {
     OCSP_RESPID *id;
     PACKET responder_id;
     const unsigned char *id_data;
     if (!PACKET_get_length_prefixed_2(&responder_id_list, &
         responder_id) || PACKET_remaining(&responder_id) ==
         0) {
         return 0;
     }

-  if (s->tlsext_ocsp_ids == NULL
-      && (s->tlsext_ocsp_ids =
-      sk_OCSP_RESPID_new_null()) == NULL) { ❸
-    *al = SSL_AD_INTERNAL_ERROR;
-    return 0;
-  }

 // [snip]
 }
```

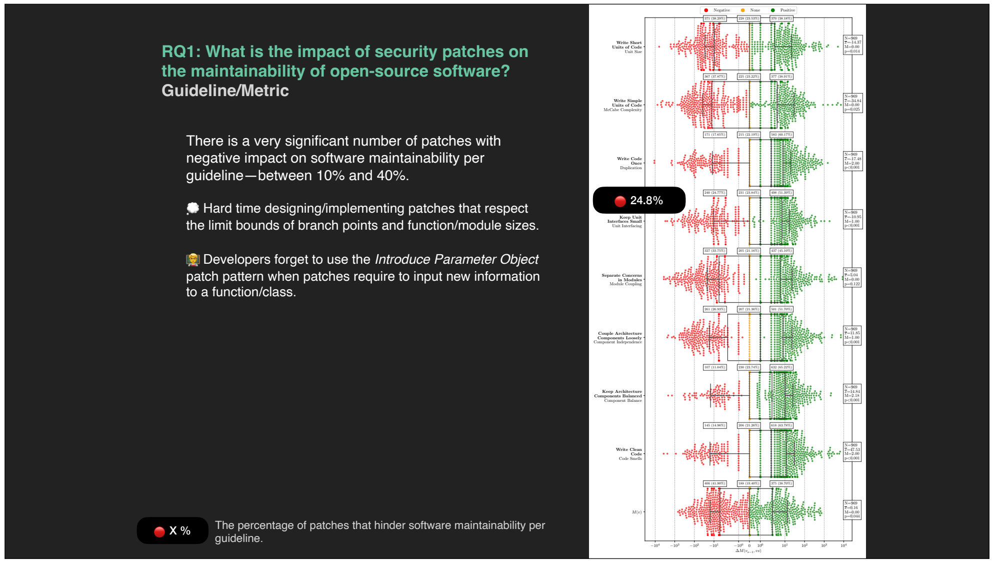
Listing 1 Patch provided by OpenSSL developers to the CVE-2016-6304 vulnerability on file ssl/t1_lib.c

@rmaranhao


Write Short Units of Code
Unit Size
Write Simple Units of Code
McCabe Complexity
Write Code Once
Duplication
Keep Unit Interfaces Small
Unit Interfacing
Separate Concerns in Modules
Module Coupling
Couple Architecture Components Loosely
Component Independence
Keep Architecture Components Balanced
Component Balance
Keep Your Code Base Small
Volume
Automate Tests
Testability
Write Clean Code
Code Smells
COMPLIANCE
Write Simple Units of Code
Refactoring candidates
MAINTAINABILITY
$M(v) = \sum_{g \in G} M_g(v)$
IMPACT
Refactoring
Security Flaw
Vulnerability Free
$\Delta M(v_{s-1}, v_s) = M(v_s) - M(v_{s-1})$
Threshold Points
Better Code Hub: https://bettercodehub.com/
@rmaranhao


RQ1: What is the impact of security patches on the maintainability of open-source software?
Guideline/Metric
There is a very significant number of patches with negative impact on software maintainability per guideline—between 10% and 40%.
38.3%
37.9%
17.7%
24.8%
33.8%
26.9%
11.0%
15.0%
X %
The percentage of patches that hinder software maintainability per guideline.


RQ1: What is the impact of security patches on the maintainability of open-source software?
Guideline/Metric
There is a very significant number of patches with negative impact on software maintainability per guideline—between 10% and 40%.
Hard time designing/implementing patches that respect the limit bounds of branch points and function/module sizes.
38.3%
37.9%
33.8%
X %
The percentage of patches that hinder software maintainability per guideline.


RQ1: What is the impact of security patches on the maintainability of open-source software?
Guideline/Metric
There is a very significant number of patches with negative impact on software maintainability per guideline—between 10% and 40%.
Hard time designing/implementing patches that respect the limit bounds of branch points and function/module sizes.
Developers forget to use the Introduce Parameter Object patch pattern when patches require to input new information to a function/class.
24.8%
X %
The percentage of patches that hinder software maintainability per guideline.

RQ1: What is the impact of security patches on the maintainability of open-source software?
Guideline/Metric

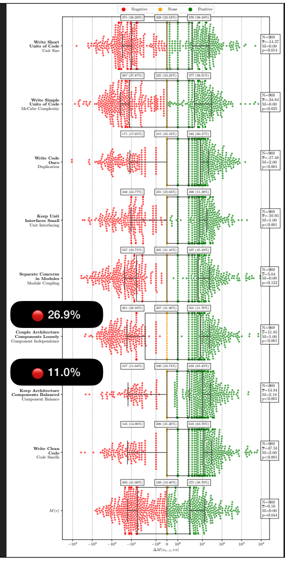
There is a very significant number of patches with negative impact on software maintainability per guideline—between 10% and 40%.

💭 Hard time designing/implementing patches that respect the limit bounds of branch points and function/module sizes.

👩‍🔧 Developers forget to use the *Introduce Parameter Object* patch pattern when patches require to input new information to a function/class.

❗ Lack of encapsulation to hide implementation details and make the system more modular.

🔴 X % The percentage of patches that hinder software maintainability per guideline.



RQ1: What is the impact of security patches on the maintainability of open-source software?
Guideline/Metric

There is a very significant number of patches with negative impact on software maintainability per guideline—between 10% and 40%.

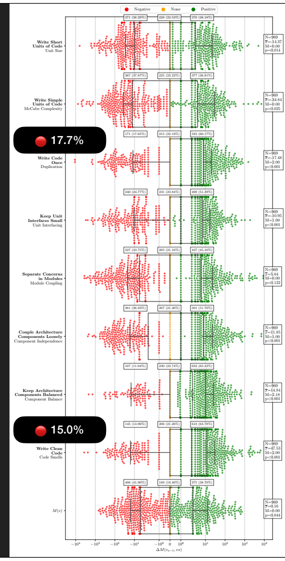
💭 Hard time designing/implementing patches that respect the limit bounds of branch points and function/module sizes.

👩‍🔧 Developers forget to use the *Introduce Parameter Object* patch pattern when patches require to input new information to a function/class.

❗ Lack of encapsulation to hide implementation details and make the system more modular.

👩‍🔧 Developers reuse code by copying and pasting existing code fragments instead of using the Extract method refactoring technique. Clone detection tools may help with this problem.

🔴 X % The percentage of patches that hinder software maintainability per guideline.



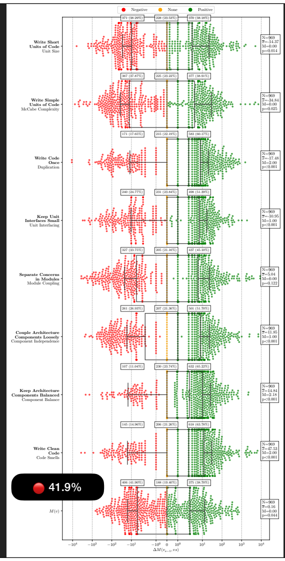
RQ1: What is the impact of security patches on the maintainability of open-source software?
Overall Score - M(v)

The larger number of negative cases may be explained by guidelines with higher concentrations of negative cases with higher amplitudes.

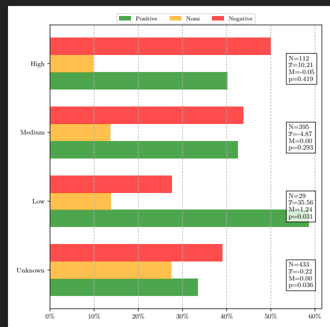
🔴 406 patches (41.9%)
🟡 188 patches (19.4%)
🟢 375 patches (38.7%)

🚀 Security patches may have a negative impact on the maintainability of open-source software.

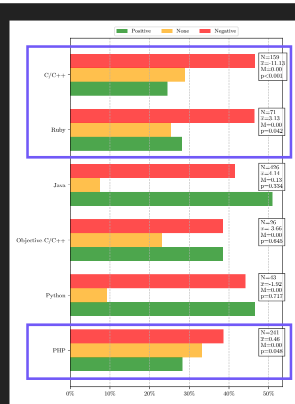
p-value = 0.044 < 0.05



RQ1: What is the impact of security patches on the maintainability of open-source software?
Severity, Programming Language



💭 Higher severity vulnerabilities patches may have a more negative impact on maintainability — **high/medium severity vulnerabilities may need more attention than low severity** while patching.



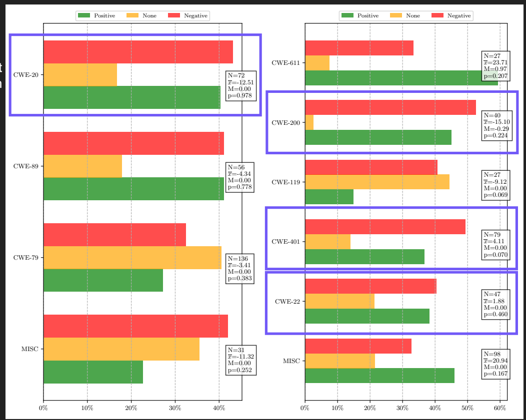
💡 Overall languages have a considerable amount of cases that negatively impact maintainability—between 35% to 50%—which confirms the **need for better/more secure programming languages**.

@rmaranhao

RQ2: Which weakness are more likely to affect open-source maintainability?

CWE-20, CWE-200, CWE-401, CWE-22

Improper Input Validation



Information Exposure

Missing Release of Memory after Effective Lifetime

Path Traversal

Common Weakness Enumeration (CWE) Website: https://cwe.mitre.org/

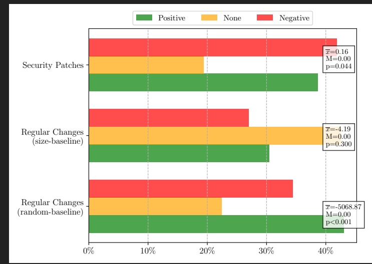
@rmaranhao

RQ3: What is the impact of security patches versus regular changes on the maintainability of open-source software?

📈 Results for both baselines show that **regular changes are less prone** to hinder the software maintainability of open-source software.

👀 Security-related commits are observed to harm software maintainability, while regular changes are less prone to harm software maintainability.

🚀 Thus, we urge the **importance of adopting maintainability practices** while applying security patches.



size-baseline: a dataset of random regular changes with the same size as security patches; **random-baseline**: a dataset of random changes.

@rmaranhao

END. WHAT SHOULD YOU DO NEXT?

🚶 Follow the best practices. Developers harm software maintainability because they still not consider some quality aspects in their solutions/patches.

⚠️ Prioritise high and medium severity vulnerabilities.

🙆 Pay special attention to the types of software vulnerabilities that are more prone to have an impact on software vulnerability.

🛠 Build tools for Patch Risk Assessment Bases on Source Code Metrics, Static Analysis features and Software Vulnerability Metadata.

👨‍🏫 Make maintainable security part of the CS curricula.

🤟 Build better and more secure programming languages.

🔗 Replication package available: https://github.com/TQRG/maintainable-security

@rmaranhao

That's it, folks!

Any questions? Ask now.

In the future, we can get in touch by email: rui@computer.org

@rmaranhao