Dan Grahn & Junjie Zhang, PhD | CAMLIS 2021

An Analysis of C/C++ Datasets for Machine Learning-Assisted Software Vulnerability Detection

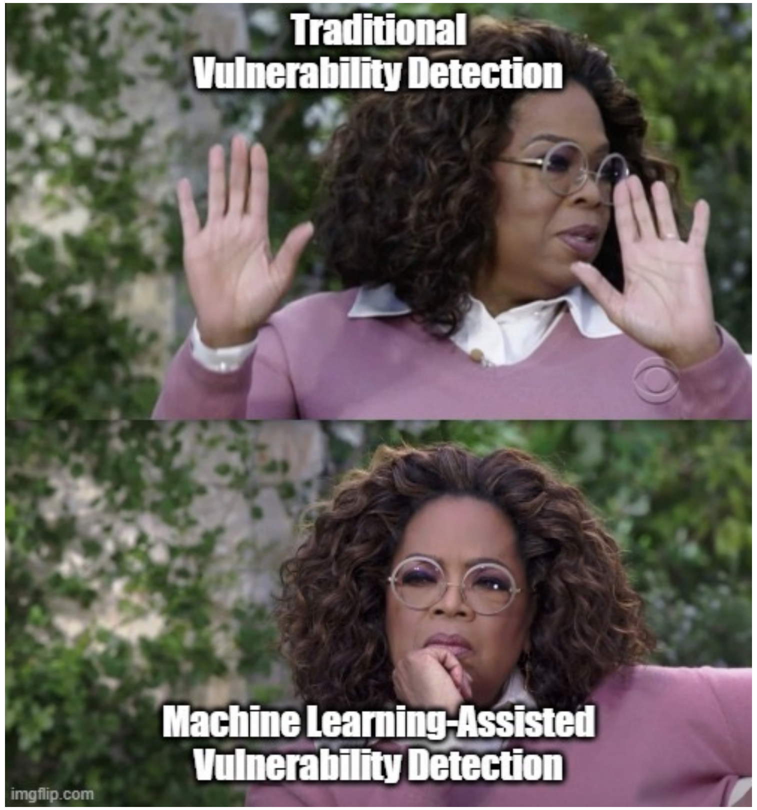
WRIGHT STATE UNIVERSITY

Outline

- Motivation
  - What makes MLAVD difficult?
  - State of MLAVD
- Research
  - Selected Datasets + Wild C
  - Results
  - Contribution
  - Questions / Contact

Motivation for MLAVD

- Vulnerability detection (VD) is a hands-on and resource-intensive process.
  - Manual code reviews divert programmers.
  - Static SCA is prone to false positives.
  - Fuzzing/Dynamic analysis takes a lot of compute.
- Machine Learning-Assisted Software Vulnerability Detection (MLAVD) offers the promise of accelerating the VD process.



What makes MLAVD difficult?

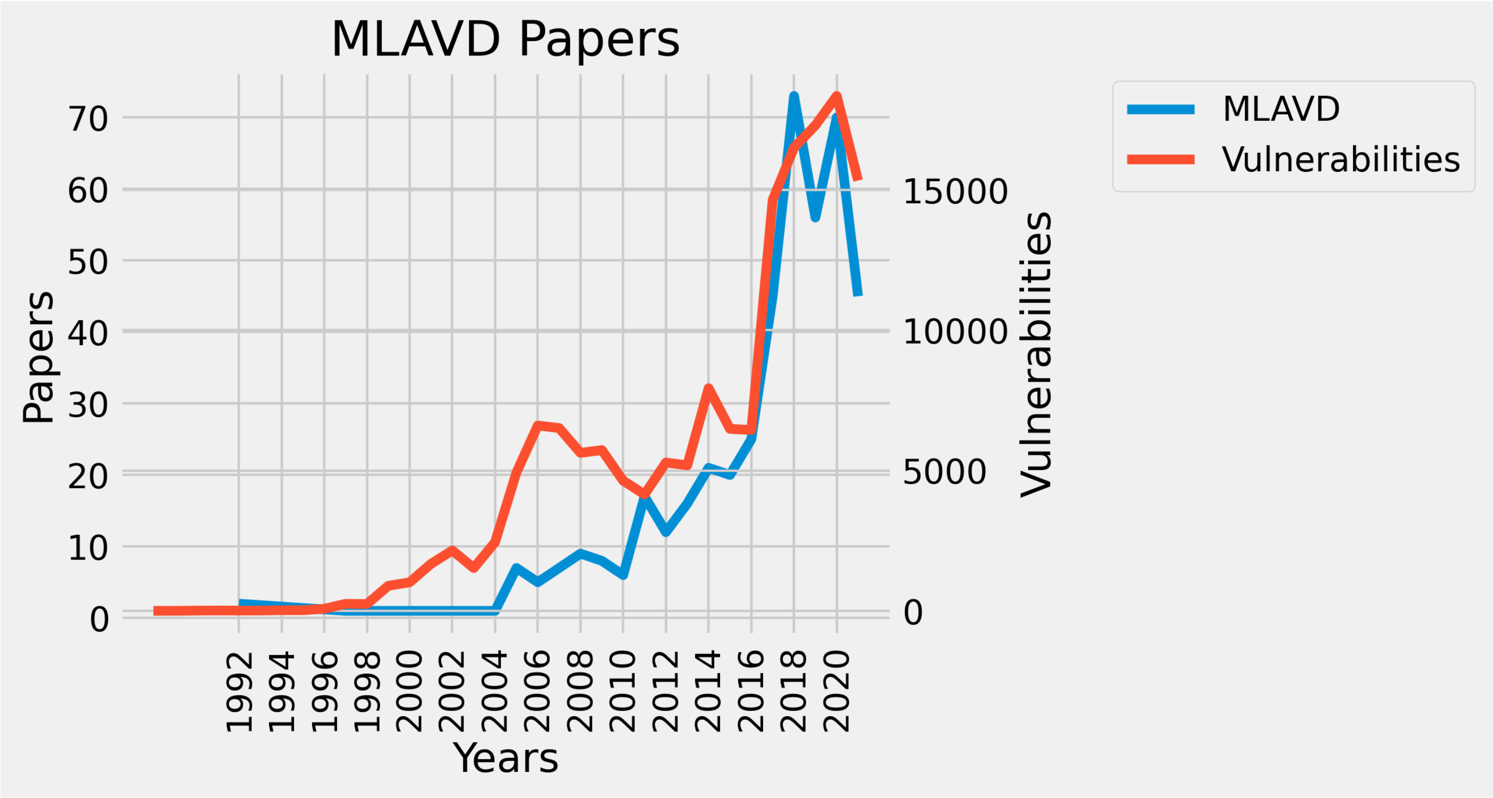
- The difference between safe and vulnerable code can be extremely subtle.
  - E.g., CWE-193 Off-by-one Error
  - This code inserts a null pointer to signify the last widget but fails to allocate space for it.

```
int i;
unsigned int num;
Widget **list;

num = GetUntrustedSizeValue();
if ((num == 0) || (num > MAX_NUM_WIDGETS)) {
  ExitError("Incorrect number of widgets requested!");
}

list = (Widget **) malloc(num * sizeof(Widget*));
printf("list ptr=%p\n", list);

for(i = 0; i < num; i++) {
  list[i] = InitializeWidget();
}

list[num] = NULL;
showWidgets(list);
```

Source: https://cwe.mitre.org/data/definitions/193.html

State of MLAVD

- Interest in MLAVD has increased dramatically in the past few years.
- *Advent of deep learning?*
- *Increase in cybercrime?*
- *Ubiquity of software?*



Source: NVD and Research Archives

State of MLAVD

- Bold claims regarding performance are being made regularly.
  - It's not uncommon to see Accuracy and F1 scores in the 90s.
- But much of the work is based on just a few datasets.

What about the datasets?

Research Goal

- Explore available datasets used for MLAVD to:
  - Determine how realistic their code is,
  - Uncover any hidden biases, and
  - Detect any additional shortcomings.

Selected Datasets: Big-Vul

- Collected by crawling the CVE database and linking CVEs with open-source GitHub projects.
- Labelled using CVE and commit information.

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	**MIT**	**Functions**	✘	**348 Projects**	**3,754**	**3**
Draper VDISC	CC-BY 4.0	Functions	✘	1.27M Funcs	87,704	5
IntroClass	BSD	Scripts	✓	6 Asgmts	998	9
Juliet 1.3	CC0 1.0	Scripts	✓	64,099 Cases	64,099	34
ManyBugs	BSD	Projects	✓	5.9M Lines	185	9
SVCP4C	GPLv3	Files	✘	2,378 Files	9,983	2
Taxonomy of Buffer Overflows	MIT	Scripts	✓	1,164 Cases	873	2
Wild C/C++	*CC-BY 4.0*	*Files*	✘	*12.1M Files*	*N/A*	*N/A*

Selected Datasets: Draper-VDISC

- Collected from Debian and public Git repositories, deduplicated.
- Labelled using combined predictions of Clang, Cppcheck, and Flawfinder.

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	MIT	Functions	✘	348 Projects	3,754	3
Draper VDISC	**CC-BY 4.0**	**Functions**	✘	**1.27M Funcs**	**87,704**	**5**
IntroClass	BSD	Scripts	✓	6 Asgmts	998	9
Juliet 1.3	CC0 1.0	Scripts	✓	64,099 Cases	64,099	34
ManyBugs	BSD	Projects	✓	5.9M Lines	185	9
SVCP4C	GPLv3	Files	✘	2,378 Files	9,983	2
Taxonomy of Buffer Overflows	MIT	Scripts	✓	1,164 Cases	873	2
Wild C/C++	*CC-BY 4.0*	*Files*	✘	*12.1M Files*	*N/A*	*N/A*

Selected Datasets: IntroClass

- Real submissions of six assignments from an introduction programming class.
- Includes expected and actual output for repair testing.
- Published with ManyBugs

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	MIT	Functions	✘	348 Projects	3,754	3
Draper VDISC	CC-BY 4.0	Functions	✘	1.27M Funcs	87,704	5
IntroClass	**BSD**	**Scripts**	✓	**6 Asgmts**	**998**	**9**
Juliet 1.3	CC0 1.0	Scripts	✓	64,099 Cases	64,099	34
ManyBugs	BSD	Projects	✓	5.9M Lines	185	9
SVCP4C	GPLv3	Files	✘	2,378 Files	9,983	2
Taxonomy of Buffer Overflows	MIT	Scripts	✓	1,164 Cases	873	2
Wild C/C++	*CC-BY 4.0*	*Files*	✘	*12.1M Files*	*N/A*	*N/A*

Selected Datasets: Juliet 1.3

- Hand-curated collection of vulnerabilities.
- The most frequently used MLAVD dataset.
- Part of the NIST Software Assurance Reference Dataset (SARD)

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	MIT	Functions	✘	348 Projects	3,754	3
Draper VDISC	CC-BY 4.0	Functions	✘	1.27M Funcs	87,704	5
IntroClass	BSD	Scripts	✓	6 Asgmts	998	9
Juliet 1.3	**CC0 1.0**	**Scripts**	✓	**64,099 Cases**	**64,099**	**34**
ManyBugs	BSD	Projects	✓	5.9M Lines	185	9
SVCP4C	GPLv3	Files	✘	2,378 Files	9,983	2
Taxonomy of Buffer Overflows	MIT	Scripts	✓	1,164 Cases	873	2
Wild C/C++	*CC-BY 4.0*	*Files*	✘	*12.1M Files*	*N/A*	*N/A*

Selected Datasets: ManyBugs

- Collected from 9 open-source programs.
- Labelled using commit information.
- Includes before/after patches for repair testing.
- Published with IntroClass

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	MIT	Functions	✘	348 Projects	3,754	3
Draper VDISC	CC-BY 4.0	Functions	✘	1.27M Funcs	87,704	5
IntroClass	BSD	Scripts	✓	6 Asgmts	998	9
Juliet 1.3	CC0 1.0	Scripts	✓	64,099 Cases	64,099	34
ManyBugs	**BSD**	**Projects**	✓	**5.9M Lines**	**185**	**9**
SVCP4C	GPLv3	Files	✘	2,378 Files	9,983	2
Taxonomy of Buffer Overflows	MIT	Scripts	✓	1,164 Cases	873	2
Wild C/C++	*CC-BY 4.0*	*Files*	✘	*12.1M Files*	*N/A*	*N/A*

Selected Datasets: SonarCloud Vulnerable Code Prospector 4 C (SVCP4C

- Method to collect vulnerable code from SonarCloud API.
- Paper also provides dataset.

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	MIT	Functions	✘	348 Projects	3,754	3
Draper VDISC	CC-BY 4.0	Functions	✘	1.27M Funcs	87,704	5
IntroClass	BSD	Scripts	✓	6 Asgmts	998	9
Juliet 1.3	CC0 1.0	Scripts	✓	64,099 Cases	64,099	34
ManyBugs	BSD	Projects	✓	5.9M Lines	185	9
SVCP4C	**GPLv3**	**Files**	✘	**2,378 Files**	**9,983**	**2**
Taxonomy of Buffer Overflows	MIT	Scripts	✓	1,164 Cases	873	2
Wild C/C++	*CC-BY 4.0*	*Files*	✘	*12.1M Files*	*N/A*	*N/A*

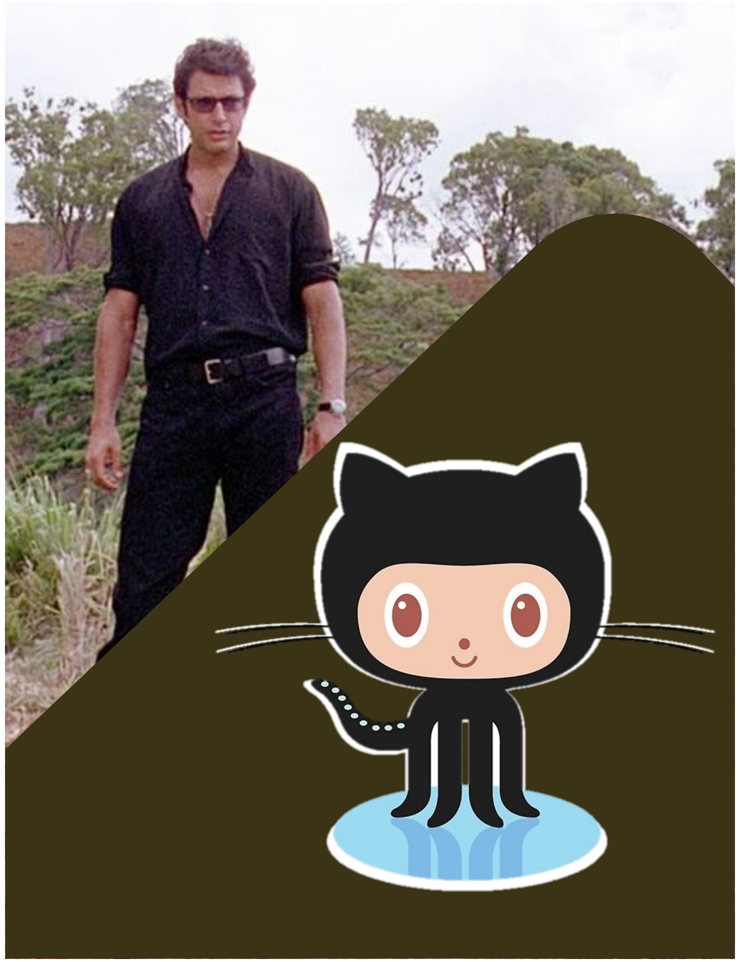
Selected Datasets: Taxonomy of Buffer Overflows

- A structured taxonomy of buffer overflows based on 22 attributes.
- Each of the 291 types has three vulnerable and one non-vulnerable example.
- Part of NIST SARD

Name	License	Granularity	Compiles?	Cases	# of Vulns	Relevant Citations
Big-Vul	MIT	Functions	✗	348 Projects	3,754	3
Draper VDISC	CC-BY 4.0	Functions	✗	1.27M Funcs	87,704	5
IntroClass	BSD	Scripts	✓	6 Asgmts	998	9
Juliet 1.3	CC0 1.0	Scripts	✓	64,099 Cases	64,099	34
ManyBugs	BSD	Projects	✓	5.9M Lines	185	9
SVCP4C	GPLv3	Files	✗	2,378 Files	9,983	2
Taxonomy of Buffer Overflows	**MIT**	**Scripts**	✓	**1,164 Cases**	**873**	**2**
Wild C/C++	*CC-BY 4.0*	*Files*	✗	*12.1M Files*	*N/A*	*N/A*

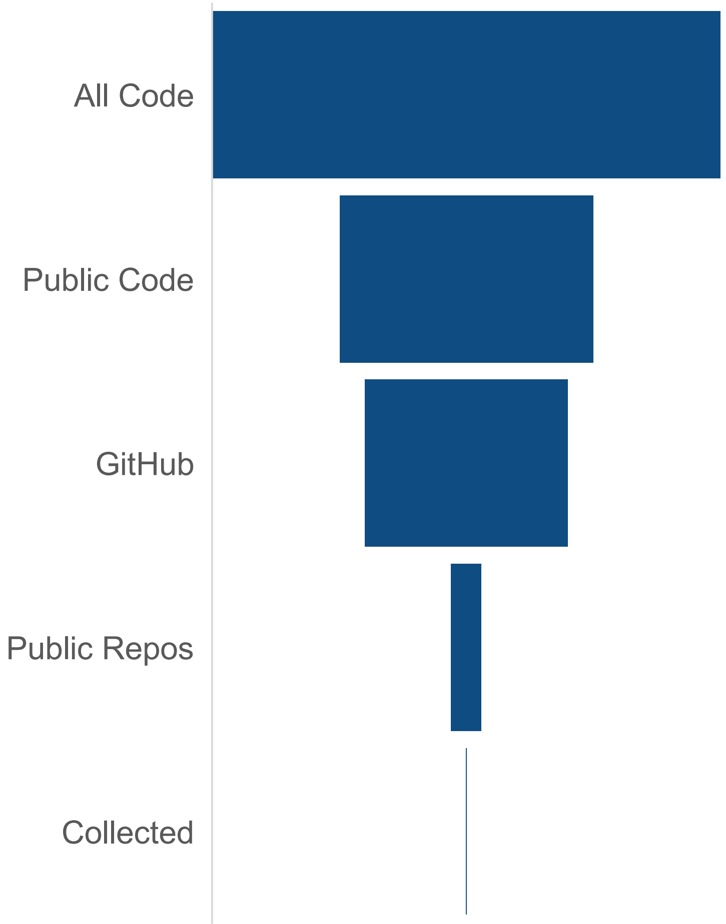
Introducing Wild C

- Before we can determine how realistic the datasets are, we need to know what "normal" C/C++ code looks like.
    - It's outside the scope of our paper (or any paper) to collect all C/C++ code.
- **Wild C is a large publicly available collection of C/C++ source code.**
    - All public C/C++ repositories on GitHub with 10+ stars
    - 36,568 repositories
    - 12M C/C++ files, 411M functions



Wild C: Rationale

- Why GitHub?
  - Private code isn't accessible.
  - GitHub is the largest public code host (by a huge margin).
- Why 10 stars?
  - The more similar code is written, the more likely it is to be put into a popular library.
  - Code which isn't popular is more likely to be one-off projects, programming assignments, and similar.
  - Resource constraints. 🤷‍♂️



Preprocessing

- Extract tokens from each file with ANTLR
- Convert token output to CSV files with listed columns.
- Aggregate various metrics based on files, tokens, and datasets.

Column	Description
`uuid`	Generated UUID for referencing
`dataset`	Source dataset
`file_name`	Source filename
`token_num`	Index of token in file
`char_start`	Character at which the token starts, relative to file
`char_end`	Character at which the token ends, relative to file
`token_text`	Raw text of the token
`token_type`	Type of token as specified by the grammar
`channel`	ANTLR internal for handling input categories
`line`	Line on which the token starts
`line_char`	Character on which the token starts, relative to line start

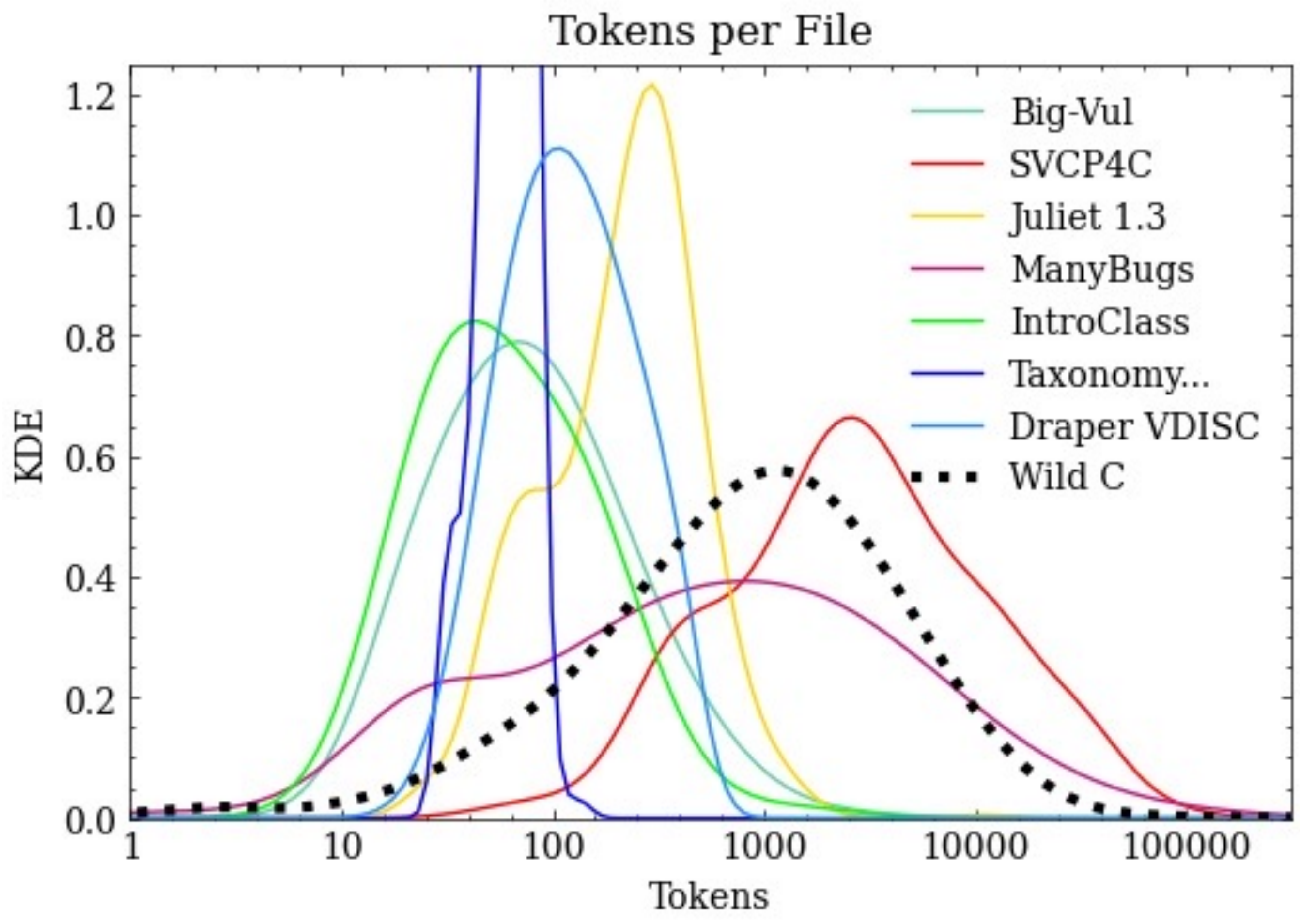
Results: Tokens per File / File Length

Plot
Histogram of tokens per file normalized using a kernel-density estimate with X-axis on a log scale.

Caution: Big-Vul and Draper VDISC contain functions not files.

Takeaways

- Most datasets are biased towards shorter files.
- Datasets drawn from existing repos perform better.



Results: Token Usage

Takeaways

- Each of the datasets has token-types which are missing.
- Because some tokens are used more than others, this has varying effect.
- Datasets drawn from existing repositories have significantly fewer missing tokens and less percent difference in usage.

Dataset	Missing Tokens			Usage % Difference	
	Count	%	Use %	Median	Mean
Big-Vul	8	6.1	0.002	**34.5**	**48.1**
Draper VDISC	**2**	**1.5**	**0.001**	41.5	49.0
IntroClass	92	70.8	11.547	81.4	316.1
Juliet	43	33.1	0.317	82.9	612.2
ManyBugs	11	8.5	0.018	50.0	86.0
SVCP4C	23	17.7	0.061	40.9	59.7
Taxonomy…	74	56.9	4.954	93.9	432.8
	130	**Total Token Types**			

Results: Token Usage Outliers

	Big-Vul		Draper VDISC		IntroClass		Juliet	
	Type	% Diff	Type	% Diff	Type	% Diff	Type	% Diff
1	explicit	425.1	register	255.9	%	3200.8	wchar_t	34435.5
2	char16_t	219.5	this	196.0	AndAnd	2505.7	namespace	3233.5
3	register	212.1	delete	140.6	/	1203.3	delete	2744.1
4	static_cast	179.4	double	111.1	else	645.2	using	2041.2
	ManyBugs		**SVCP4C**		**Taxonomy**		**Merged**	
	Type	% Diff	Type	% Diff	Type	% Diff	Type	% Diff
1	extern	2010.3	CharLiteral	406.0	do	5711.3	extern	1434.1
2	typedef	574.2	register	279.5	char	2373.4	wchar_t	926.7
3	wchar_t	332.0	char	190.7	<=	2293.0	typedef	397.2
4	CharLiteral	322.0	AndAnd	185.5	CharLiteral	2185.5	CharLiteral	284.5

Results: Token Bigram Usage

NLP uses N-grams to help bring context to word usage. We do the same for tokens.

Takeaways

- None of the datasets contain more than 42% of the bigrams in Wild C.
- Bigram usage "widens the gap" between hand-created and collected datasets.

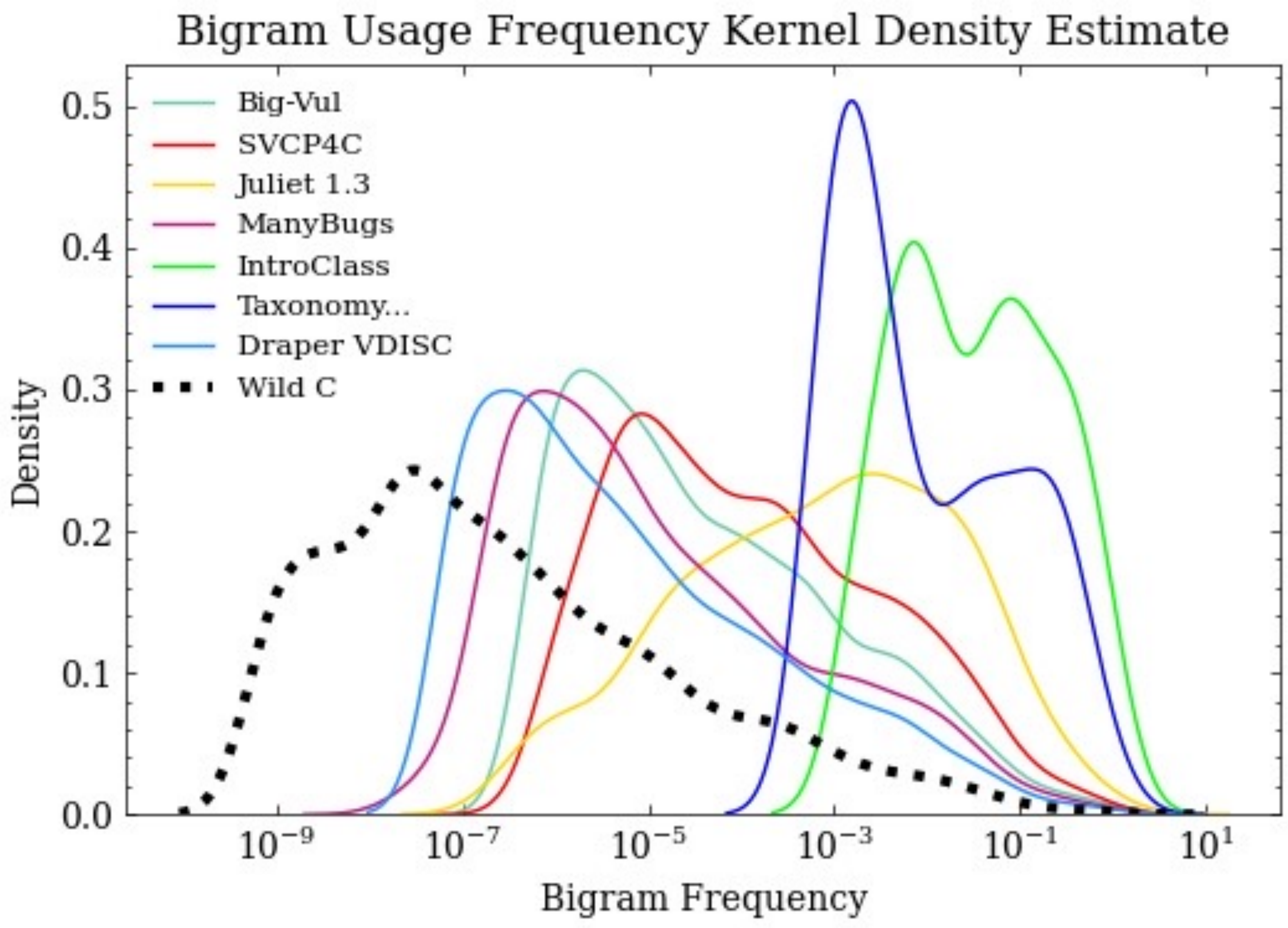
Dataset	Missing Bigrams		Usage % Difference		
	Count	%	Use %	Median	Mean
Big-Vul	6,063	74.0	0.055	56.3	244.1
Draper VDISC	**4,788**	**58.4**	**0.054**	**68.2**	**226.3**
IntroClass	8,066	98.4	42.051	94.7	734.9
Juliet	7,637	93.2	4.651	92.8	1,341.1
ManyBugs	5,408	66.0	0.106	89.6	2,363.6
SVCP4C	6,654	81.2	0.320	72.3	498.0
Taxonomy…	7,989	97.5	19.326	92.4	635.6
	8,274	**Total Bigrams in Wild C**			

Results: Token Bigram Usage Frequency

Plot
Histogram of token bigram usage frequencies by dataset normalized using a kernel-density estimate with X-axis on a log scale.

Takeaways

- Collected datasets are closer to Wild C.
- The larger the collected dataset, the closer to Wild C.
- Juliet exhibits a strange distribution compared to other datasets.



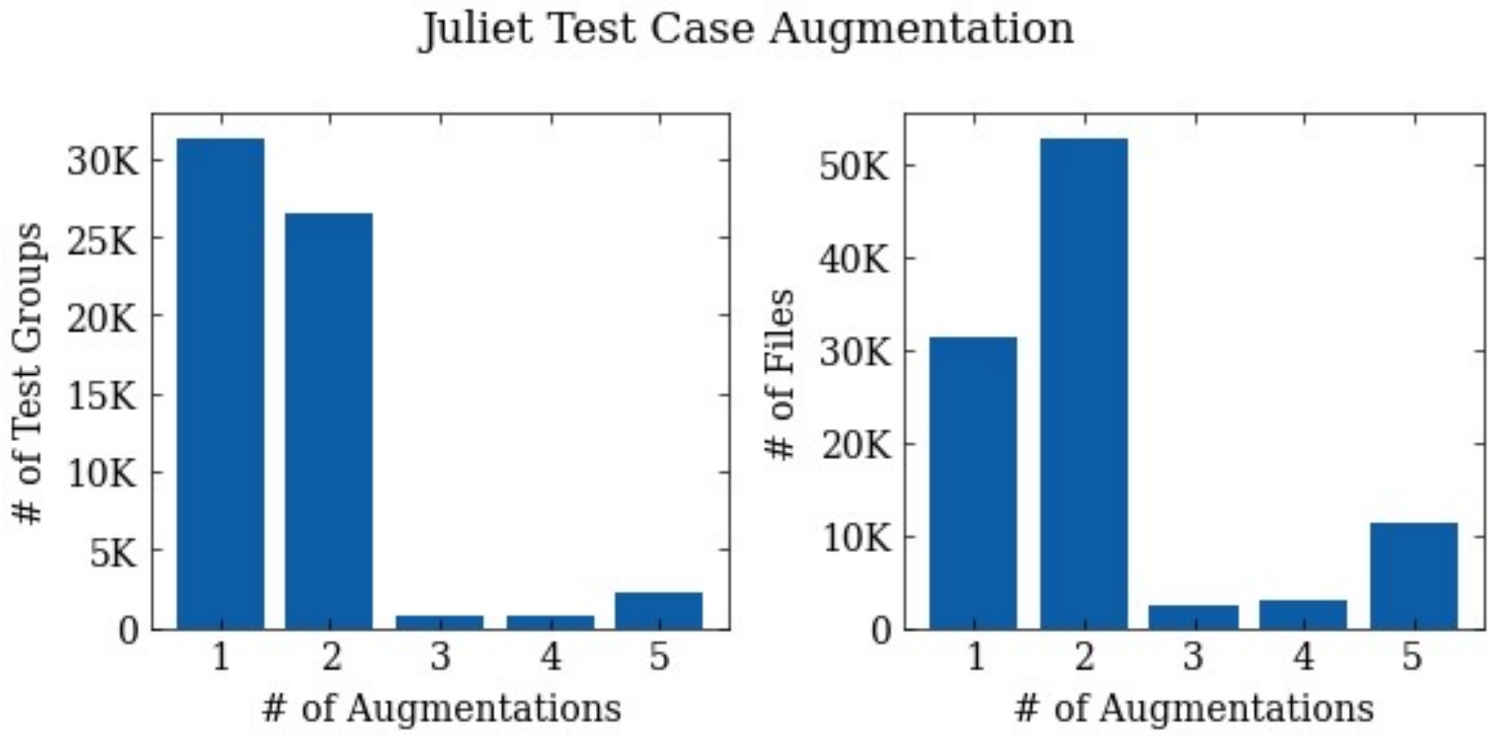
Near Duplicates: Juliet's Test Case Augmentations

Plot
Juliet augments tests by swapping datatypes in the vulnerabilities. Histogram of augmentations by number of test groups and number of files.

Takeaways

- Juliet contains pre-split augmentations.

Uh-oh!
Pre-split augmentations are bad for machine learning.



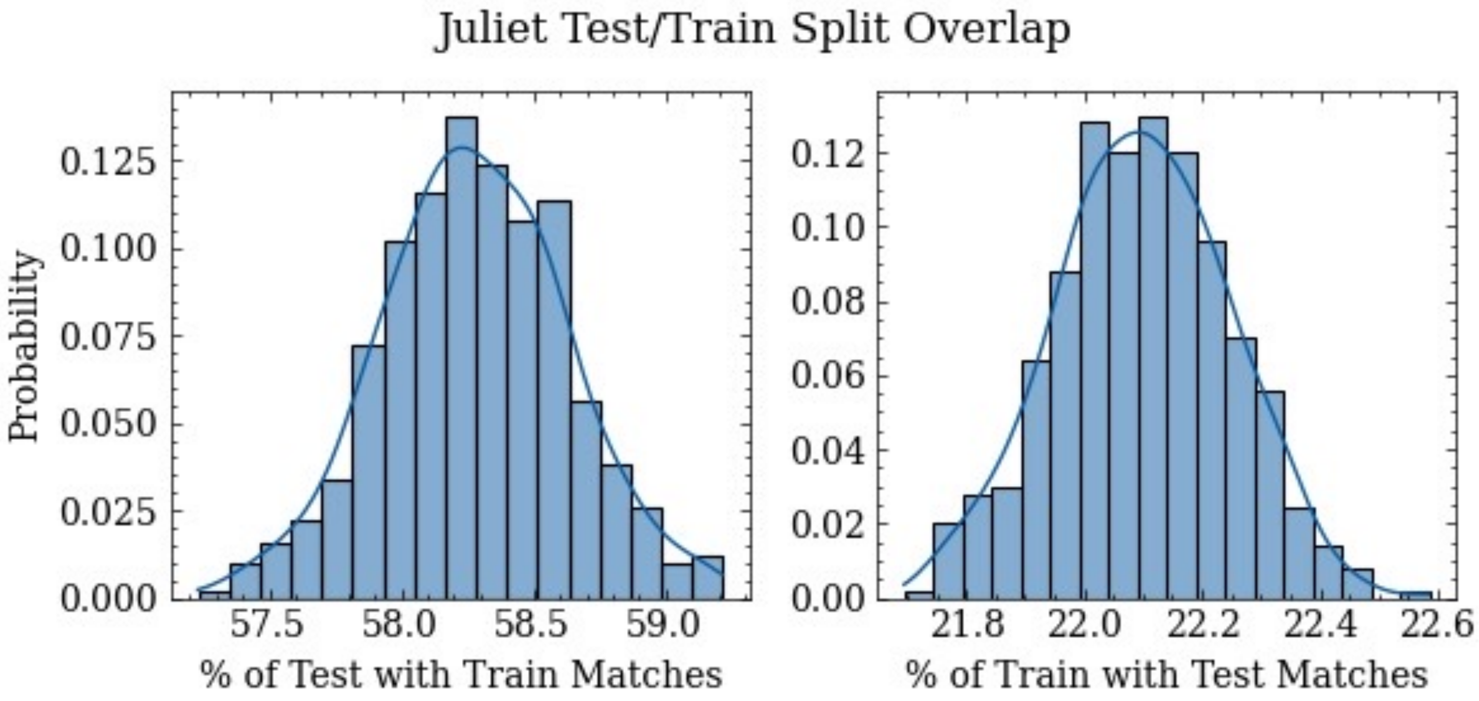
Near Duplicates: Juliet Data Leakage Analysis

Plot
Histogram of the percentage of test examples with matches in the training set and training examples with matches in the test set for random 80/20 splits.

Takeaways

- μ = 58.3% of test cases augmented in training data
- μ = 22.1% of training cases augmented in test data

At least 16 papers use Juliet w/o addressing this augmentation.



Near Duplicates: File Information

- MinHash with LSH to find near-duplicates with a Jacquard similarity of >0.99.
- All datasets exhibit duplication and suffer from data leakage between random test/train splits.

Name	Unique Groups	Unique % of Total Dataset	Test Split	% Test w/Train Match	% Train w/Test Match
Big-Vul	91,300	63.87%	0.10	45.84%	23.01%
Draper VDISC	931,804	**73.12%**	0.01	**36.10%**	**5.29%**
IntroClass	28	45.16%	0.20	70.14%	43.27%
Juliet 1.3	7,933	7.84%	0.10	98.00%	82.60%
ManyBugs	8,197	3.67%	0.10	99.70%	91.19%
SVCP4C	1,104	9.71%	0.20	99.77%	86.05%
Taxonomy of Buffer Overflows	61	5.24%	0.20	99.91%	93.66%
Wild C/C++	*2,343,364*	*21.97%*	*0.10*	*85.16%*	*36.25%*

Contributions

1. **Analysis of Representivity**
2. **Analysis of Duplicativeness**
3. Availability of Wild C

	Dataset	Notes
!	Big-Vul	High duplication. May be suitable for testing.
✓	Draper VDISC	Lower duplication. Biased towards tool-detectable vulnerabilities. “Most promising dataset.”
✗	IntroClass	Insufficient diversity of C/C++ used.
✗	Juliet	Contains vulnerability augmentation. Hand-generated. Use with extreme caution.
!	ManyBugs	High duplication. Few unique vulnerabilities. May be suitable for “whole project” testing.
✗	SVCP4C	High duplication. Biased towards vulnerabilities detected by *SonarCloud*. Use with caution.
✗	Taxonomy…	Insufficient diversity of C/C++ used.

Contributions

1. Analysis of Representivity
2. Analysis of Duplicativeness
3. **Availability of Wild C**

- Largest public dataset of C/C++ code (to the best of our knowledge)
- "ready to apply" to
  - Comment prediction
  - Function name recommendation
  - Code completion
  - Variable name recommendation
  - Etc.
- Potential to use automatic bug insertion to provide expanded vulnerability detection dataset.

Future Work

1. **Analyze datasets for other languages**
2. Analyze difference between safe and vulnerable subsets of the data.
3. Build a better dataset.

Future Work

1. Analyze datasets for other languages
2. **Analyze difference between safe and vulnerable subsets of the data.**
3. Build a better dataset.

Future Work

1. Analyze datasets for other languages
2. Analyze difference between safe and vulnerable subsets of the data.
3. **Build a better dataset.**

5 Future Dataset Recommendations

1. It must be drawn from *real* code.
2. It must exercise a sufficient diversity of C/C++.
3. It should be compilable.
4. It should be deduplicated.
5. It should be difficult enough to act as a viable benchmark.

Thanks for listening!

Feel free to reach out. ☺

Dan Grahn
dan.grahn@wright.edu
@realDanGrahn

WRIGHT STATE
UNIVERSITY

{DG}