Project Report Program Analysis

This project is motivated by the research presented in the paper

“Automated Vulnerability Detection in Source CodeUsing Deep Representation Learning”

The paper aims to do a binary classification between a vulnerable vs non vulnerable code.

The dataset for this paper was built in following manner:

	SATE IV	GitHub	Debian
Total	121,353	9,706,269	3,046,758
Passing curation	11,896	782,493	491,873
‘Not vulnerable’	6,503 (55%)	730,160 (93%)	461,795 (94%)
‘Vulnerable’	5,393 (45%)	52,333 (7%)	30,078 (6%)

There were three components to the dataset:

1. STATE IV dataset which is a toy dataset
2. Github
3. Debian

The paper applies post processing techniques to divide into 5 major vulnerabilities.

CWE ID	CWE Description	Frequency %
120/121/122	Buffer Overflow	38.2%
119	Improper Restriction of Operations within the Bounds of a Memory Buffer	18.9%
476	NULL Pointer Dereference	9.5%
469	Use of Pointer Subtraction to Determine Size	2.0%
20, 457, 805 etc.	Improper Input Validation, Use of Uninitialized Variable, Buffer Access with Incorrect Length Value, etc.	31.4%

In this report we

1. Discuss these vulnerabilities in detail.
2. Analysis Tools to manually find these vulnerabilities.
3. Recreating and benchmarking the paper’s machine learning results

The paper discusses about CWE

CWE is also termed as a Common Weakness Enumeration. It is considered as a common standard repository for source code vulnerabilities. It provides a common ground for both the developers, vendors, customers to discuss the vulnerabilities.

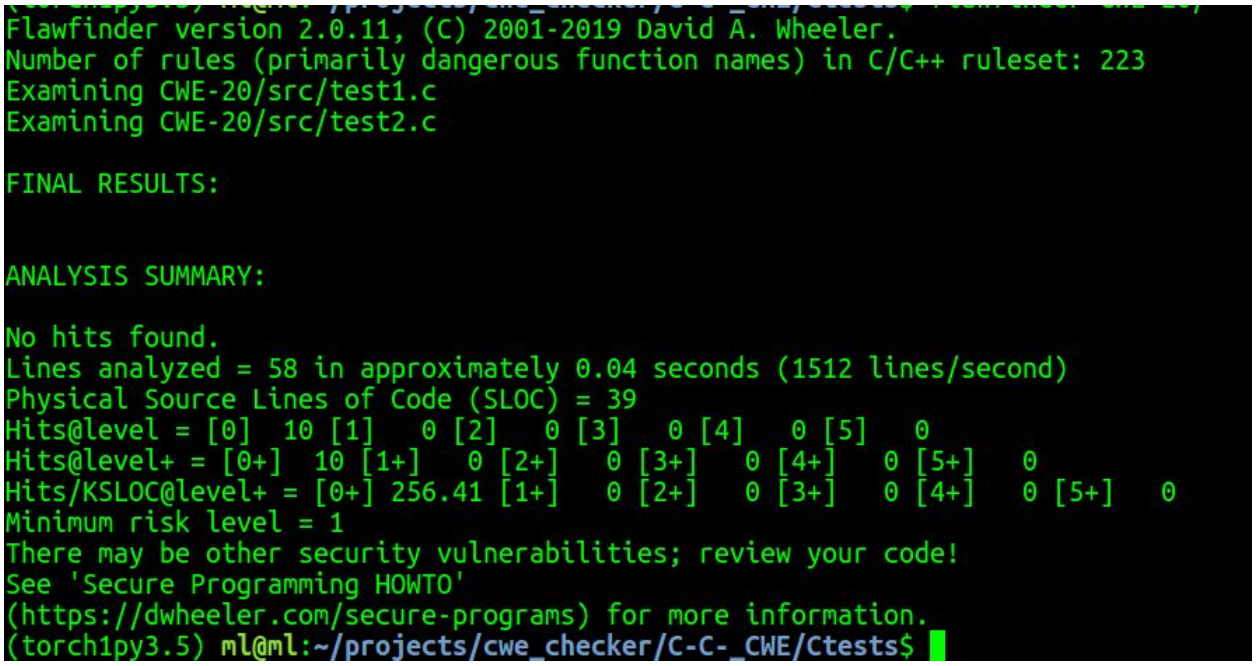
The evolution of vulnerability theory is mostly occurring within the context of the Common Weakness Enumeration (CWE), a classification of almost 800 types and categories of weaknesses that lead to vulnerabilities such as buffer overflows, XSS, insufficient randomness, and bad permissions. The theory is being used to explain classification problems and train non-expert analysts about the vulnerability researcher mindset.

Manual CWE analysis:

For manual extraction of CWE vulnerabilities we used:

1. CppCheck
2. Flawfinder

Flawfinder results:



```
(torch1py3.5) ml@ml:~/projects/cwe_checker/C-C-_CWE/Ctests$ flawfinder CWE-401
Flawfinder version 2.0.11, (C) 2001-2019 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining CWE-401/src/test1.c

FINAL RESULTS:

CWE-401/src/test1.c:10:  [1] (buffer) read:
  Check buffer boundaries if used in a loop including recursive loops
  (CWE-120, CWE-20).

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 20 in approximately 0.00 seconds (4636 lines/second)
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]   0 [1]   1 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   1 [1+]   1 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 55.5556 [1+] 55.5556 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming HOWTO'
(https://dwheeler.com/secure-programs) for more information.
```

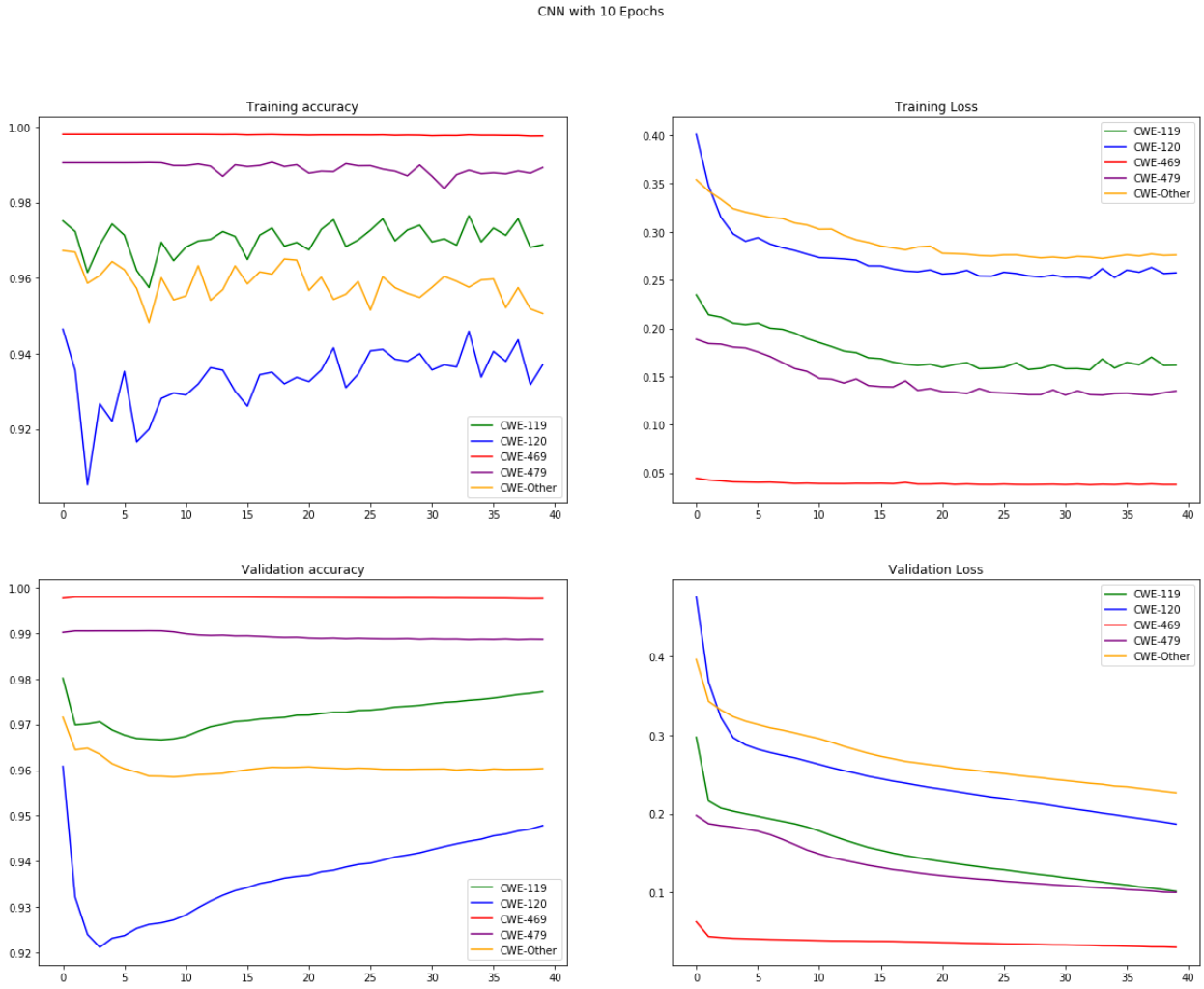
Cppcheck results:

```
(torch1py3.5) ml@ml:~/projects/cwe_checker/C-C-_CWE/Ctests$ cppcheck CWE-401/src/test1.c
Checking CWE-401/src/test1.c...
[CWE-401/src/test1.c:11]: (error) Memory leak: buf
```

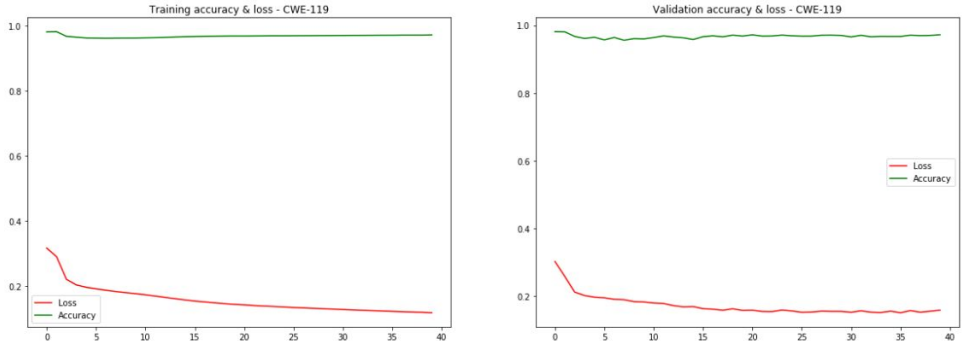
We tried experiments in two settings:

1. The experiments for multiple output multiple class vulnerability detection.
2. The experiments for single class (**CWE-119**) binary detection.

Results for multiple output multiple class vulnerability detection



Results for single class (CWE-119) binary detection



Loss and Accuracy Graph

No.	MaxPool	Weights (0 : 1)	TP	FP	TN	FN	Acc	Precision	Recall	PR-AUC	AUC	MCC	F1
1	4	1 : 50	2090	9883	115084	362	0.9195	0.1745	0.8523	0.3272	0.9433	0.3640	0.2897
2	4	1 : 35	2089	8480	116487	363	0.9305	0.1976	0.8519	0.3516	0.9450	0.3905	0.3208
3	4	1 : 25	2055	7442	117525	397	0.9384	0.2163	0.8380	0.3472	0.9455	0.4072	0.3439
4	4	1 : 20	2040	6989	117978	412	0.9419	0.2259	0.8319	0.3441	0.9459	0.4154	0.3553
5	4	1 : 15	1994	5937	119030	458	0.9498	0.2514	0.8132	0.3533	0.9455	0.4354	0.3840
6	4	1 : 10	1930	5257	119710	522	0.9546	0.2685	0.7871	0.3479	0.9463	0.4436	0.4004
7	4	1 : 5	1610	3240	121727	842	0.9679	0.3319	0.6566	0.3343	0.9430	0.4527	0.4409
8	3	1 : 5 (40ep)	1801	3780	121187	651	0.9652	0.3227	0.7345	0.3661	0.9429	0.4727	0.4484
9	4	1 : 5 (40ep)	1740	3417	121550	712	0.9675	0.3374	0.7096	0.3717	0.9436	0.4756	0.4573
10	5	1 : 5 (40ep)	1784	3591	121376	668	0.9665	0.3319	0.7275	0.3710	0.9437	0.4776	0.4558

CNN Performance at various accuracy Metrics

Conclusion:

From the above experiments we can conclude that there is not any one tool or approach which is perfect for program analysis. As we can see that cppcheck although finds the issue but that is different from flaw finder whereas our ml approach is good for binary classification but doesn't beat benchmark models for multi class classification.