

The use of machine learning with signal- and NLP processing of source code to fingerprint, detect, and classify vulnerabilities and weaknesses with MARFCAT

Serguei A. Mokhov
 Concordia University
 Montreal, QC, Canada
 mokhov@cse.concordia.ca

Abstract

We present a machine learning approach to static code analysis and fingerprinting for weaknesses related to security, software engineering, and others using the open-source MARF framework and the MARFCAT application based on it for the NIST's SATE 2010 static analysis tool exposition workshop.

Contents

1	Introduction	2
2	Related Work	3
3	Methodology	4
3.1	Core principles	4
3.2	CVEs – the “Knowledge Base”	4
3.3	Categories for Machine Learning	5
3.4	Basic Methodology	5
3.5	Line Numbers	6
3.5.1	Line Number Estimation Methodology	6
3.5.2	Classes of Functions	8
4	Results	8
4.1	Preliminary Results Summary	9
4.2	Version SATE.4	9
4.2.1	Wireshark 1.2.0	9
4.2.2	Wireshark 1.2.9	10
4.2.3	Chrome 5.0.375.54	11
4.2.4	Chrome 5.0.375.70	11
4.3	Version SATE.5	12
4.3.1	Chrome 5.0.375.54	12
4.3.2	Tomcat 5.5.13	12
4.3.3	Pebble 2.5-M2	14
4.3.4	Tomcat and Pebble Testing Results Summary	15
4.4	Version SATE.6	15
4.4.1	Dovecot 2.0.beta6	15
4.4.2	Tomcat 5.5.29	16
4.5	Version SATE.7	17

5 Conclusion	17
5.1 Shortcomings	17
5.2 Advantages	18
5.3 Practical Implications	18
5.4 Future Work	18
A Classification Result Tables	20

List of Tables

1 CVE Stats for Wireshark 1.2.0, Quick Enriched, version SATE.4	21
2 CVE NLP Stats for Wireshark 1.2.0, Quick Enriched, version SATE.4	22
3 CVE NLP Stats for Wireshark 1.2.0, Quick Enriched, version SATE.4	23
4 CVE Stats for Chrome 5.0.375.54, Quick Enriched, (clean CVEs) version SATE.4	24
5 CWE Stats for Chrome 5.0.375.54, (clean CVEs) version SATE.5	25
6 CVE Stats for Tomcat 5.5.13, version SATE.5	26
7 CWE Stats for Tomcat 5.5.13, version SATE.5	27
8 CVE NLP Stats for Tomcat 5.5.13, version SATE.5	28
9 CWE NLP Stats for Tomcat 5.5.13, version SATE.5	29
10 CVE NLP Stats for Chrome 5.0.375.54, version SATE.7	29
11 CWE NLP Stats for Chrome 5.0.375.54, version SATE.7	30

1 Introduction

This paper elaborates on the details of the methodology and the corresponding results of application of the machine learning techniques along with signal processing and NLP alike to static source code analysis in search for weaknesses and vulnerabilities in such a code. This work resulted in a proof-of-concept tool, code-named *MARFCAT*, a MARF-based Code Analysis Tool [Mok11], presented at the Static Analysis Tool Exposition (SATE) workshop 2010 [ODBN10] collocated with the Software Assurance Forum on October 1, 2010.

This paper is a “rolling draft” with several updates expected to be made before it reaches more complete final-like version as well as combined with the open-source release of the MARFCAT tool itself [Mok11]. As-is it may contain inaccuracies and incomplete information.

At the core of the workshop there were C/C++-language and Java language tracks comprising CVE-selected cases as well as stand-alone cases. The CVE-selected cases had a vulnerable version of a software in question with a list of CVEs attached to it, as well as the most know fixed version within the minor revision number. One of the goals for the CVE-based cases is to detect the known weaknesses outlined in CVEs using static code analysis and also to verify if they were really fixed in the “fixed version” [ODBN10].

The test cases at the time included CVE-selected:

- C: Wireshark 1.2.0 (vulnerable) and Wireshark 1.2.9 (fixed)
- C++: Chrome 5.0.375.54 (vulnerable) and Chrome 5.0.375.70 (fixed)
- Java: Tomcat 5.5.13 (vulnerable) and Tomcat 5.5.29 (fixed)

and non-CVE selected:

- C: Dovecot 2.0-beta6
- Java: Pebble 2.5-M2

We develop MARFCAT to machine-learn from the CVE-based vulnerable cases and verify the fixed versions as well as non-CVE based cases from similar programming languages.

Organization

We develop this “running” article gradually. The related work, some of the present methodology is based on, is referenced in Section 2. The methodology summary is in Section 3. We present the results, most of which were reported at SATE2010, in Section 4. We then describe the machine learning aspects as well as mathematical estimates of functions of how to determine line numbers of unknown potentially weak code fragments in Section 3.5. (The latter is necessary since during the representation of the code a wave form (i.e. signal) with current processing techniques the line information is lost (e.g. filtered out as noise) making reports less informative, so we either machine-learn the line numbers or provide a mathematical estimate and that section describes the proposed methodology to do so, some of which was implemented.) Then we present a brief summary, description of the limitations of the current realization of the approach and concluding remarks in Section 5.

2 Related Work

Related work (to various degree of relevance) can be found below (this list is not exhaustive):

- Taxonomy of Linux kernel vulnerability solutions in terms of patches and source code as well as categories for both are found in [MLB07].
- The core ideas and principles behind the MARF’s pipeline and testing methodology for various algorithms in the pipeline adapted to this case are found in [Mok08]. There also one can find the core options used to set the configuration for the pipeline in terms of algorithms used.
- A binary analysis using machine learning approach for quick scans for files of known types in a large collection of files is described in [MD08].
- The primary approach here is similar in a way that was done for DEFT2010 [Mok10b, Mok10a] with the corresponding DEFT2010App and its predecessor WriterIdentApp [MSS09].
- Tlili’s 2009 PhD thesis covers topics on automatic detection of safety and security vulnerabilities in open source software [Tli09].
- Statistical analysis, ranking, approximation, dealing with uncertainty, and specification inference in static code analysis are found in the works of Engler’s team [KTB⁺06, KAYE04, KE03].
- Kong et al. further advance static analysis (using parsing, etc.) and specifications to eliminate human specification from the static code analysis in [KZL10].
- Spectral techniques are used for pattern scanning in malware detection by Eto et al. in [ESI⁺09].

- Researchers propose a general data mining system for incident analysis with data mining engines in [IYE⁺09].
- Hanna et al. describe a synergy between static and dynamic analysis for the detection of software security vulnerabilities in [HLYD09] paving the way to unify the two analysis methods.
- The researchers propose a MEDUSA system for metamorphic malware dynamic analysis using API signatures in [NJG⁺10].

3 Methodology

Here we briefly outline the methodology of our approach to static source code analysis in its core principles in Section 3.1, the knowledge base in Section 3.2, machine learning categories in Section 3.3, and the high-level step-wise description in Section 3.4.

3.1 Core principles

The core methodology principles include:

- Machine learning
- Spectral and NLP techniques

We use signal processing techniques, i.e. presently we do not parse or otherwise work at the syntax and semantics levels. We treat the source code as a “signal”, equivalent to binary, where each n -gram ($n = 2$ presently, i.e. two consecutive characters or, more generally, bytes) are used to construct a sample amplitude value in the signal.

We show the system examples of files with weaknesses and MARFCAT learns them by computing spectral signatures using signal processing techniques from CVE-selected test cases. When some of the mentioned techniques are applied (e.g. filters, silence/noise removal, other preprocessing and feature extraction techniques), the line number information is lost as a part of this process.

When we test, we compute how similar or distant each file is from the known trained-on weakness-laden files. In part, the methodology can approximately be seen as some signature-based antivirus or IDS software systems detect bad signature, except that with a large number of machine learning and signal processing algorithms, we test to find out which combination gives the highest precision and best run-time.

At the present, however, we are looking at the files overall instead of parsing the fine-grained details of patches and weak code fragments, which lowers the precision, but is fast to scan all the files.

3.2 CVEs – the “Knowledge Base”

The CVE-selected test cases serve as a source of the knowledge base to gather information of how known weak code “looks like” in the signal form, which we store as spectral signatures clustered per CVE or CWE. Thus, we:

- Teach the system from the CVE-based cases

- Test on the CVE-based cases
- Test on the non-CVE-based cases

3.3 Categories for Machine Learning

The two primary groups of classes we train and test on include:

- CVEs [NIS11a, NIS11b]
- CWEs [VM10] and/or our custom-made, e.g. per our classification methodology in [MLB07]

The advantages of CVEs is the precision and the associated meta knowledge from [NIS11a, NIS11b] can be all aggregated and used to scan successive versions of the the same software or derived products. CVEs are also generally uniquely mapped to CWEs. The CWEs as a primary class, however, offer broader categories, of kinds of weaknesses there may be, but are not yet well assigned and associated with CVEs, so we observe the loss of precision.

Since we do not parse, we generally cannot deduce weakness types or even simple-looking aspects like line numbers where the weak code may be. So we resort to the secondary categories, that are usually tied into the first two, which we also machine-learn along, shown below:

- Types (*sink*, *path*, *fix*)
- Line numbers

3.4 Basic Methodology

Algorithmically-speaking, MARFCAT performs the following steps to do its learning analysis:

1. Compile meta-XML files from the CVE reports (line numbers, CVE, CWE, fragment size, etc.). Partly done by a Perl script and partly manually. This becomes an index mapping CVEs to files and locations within files.
2. Train the system based on the meta files to build the knowledge base (learn). Presently in these experiments we use simple mean clusters of feature vectors per default MARF specification ([Mok08, The11]).
3. Test on the training data for the same case (e.g. Tomcat 5.5.13 on Tomcat 5.5.13) with the same annotations to make sure the results make sense by being high and deduce the best algorithm combinations for the task.
4. Test on the testing data for the same case (e.g. Tomcat 5.5.13 on Tomcat 5.5.13) without the annotations as a sanity check.
5. Test on the testing data for the fixed case of the same software (e.g. Tomcat 5.5.13 on Tomcat 5.5.29).
6. Test on the testing data for the general non-CVE case (e.g. Tomcat 5.5.13 on Pebble).

3.5 Line Numbers

As was earlier mentioned, line number reporting with MARFCAT is an issue because the source text is essentially lost without line information preserved (filtered out as noise or silence or mixed in with another signal sample). Therefore, some conceptual ideas were put forward to either derive a heuristic, a function of a line number based on typical file attributes as described below, or learn the line numbers as a part of the machine learning process. While the methodology of the line numbers discussed more complete scenarios and examples, only an approximation subset was actually implemented in MARFCAT.

3.5.1 Line Number Estimation Methodology

Line number is a function of the file's dimensions in terms of line numbers, size in bytes, and words. The meaning of W may vary. The implementations of f may vary and can be purely mathematical or relativistic and with side effects. These dimensions were recorded in the meta XML files along with the other indexing information. This gives us the basic Equation 1.

$$l = f(L_T, B, W) \quad (1)$$

where

- L_T – number of lines of text in a file
- B – the size of the file in bytes
- W – number of *words* per `wc` [Fre09], but can be any blank delimited printable character sequence; can also be an n -gram of n characters.

The function should be additive to allow certain components to be zero if the information is not available or not needed, in particular $f(B)$ and $f(W)$ may fall into this category. The ceiling $\lceil \dots \rceil$ is required when functions return fractions, as shown in Equation 2.

$$f(L_T, B, W) = \lceil f(L_T) + f(B) + f(W) \rceil \quad (2)$$

Constraints on parameters:

- $l \in [1, \dots, L_T]$ – the line number must be somewhere within the lines of text.
- $f(L_T) > 0$ – the component dependent on the lines of text L_T should never be zero or less.
- $EOL = \{\backslash\mathbf{n}, \backslash\mathbf{r}, \backslash\mathbf{r}\backslash\mathbf{n}, \mathbf{EOF}\}$. The inclusion of **EOF** accounts for the last line of text missing the traditional line endings, but is non-zero.
- $L_T > 0 \implies B > 0$
- $B > 0 \implies L_T > 0$ under the above definition of **EOL**; if **EOF** is excluded this implication would not be true
- $B = 0 \implies L_T = 0, W = 0$

Affine combination is in Equation 3:

$$f(L_T, B, W) = \lceil k_L \cdot f(L_T) + k_B \cdot f(B) + k_W \cdot f(W) \rceil \quad (3)$$

- $k_L + k_B + k_W < 1 \implies$ the line is within the triangle

Affine combination with context is in Equation 4:

$$f(L_T, B, W) = [k_L \cdot f(L_T) + k_B \cdot f(B) + k_W \cdot f(W)] \pm \Delta c \quad (4)$$

where $\pm \Delta c$ is the amount of context surrounding the line, like in `diff` [MES02]; with $c = 0$ we are back to the original affine combination.

Learning approach with matrices and probabilities from examples. This case of the line number determination must follow the preliminary positive test with some certainty that a give source code file contains weaknesses and vulnerabilities. This methodology in itself would be next to useless if this preliminary step is not performed.

In a simple case a line number is a cell in the 3D matrix M given the file dimensions alone, as in Equation 5. The matrix is sparse and unknown entries are 0 by default. Non-zero entries are learned from the examples of files with weaknesses. This matrix is capable of encoding a single line location per file of the same dimensions. As such it can't handle multiple locations per file or two or more distinct unrelated files with different line numbers for a single location. However, it serves as a starting point to develop a further and better model.

$$l = f(L_T, B, W) = M[L_T, B, W] \quad (5)$$

To allow multiple locations per file we either replace the W dimension with the locations dimension N if W is not needed, as e.g. in Equation 6, or make the matrix 4D by adding N to it, as in Equation 7. This will take care of the multiple locations issue mentioned earlier. N is not known at the classification stage, but the coordinates L_T, B, W will give a value in the 3D matrix, which is a vector of locations \vec{n} . At the reporting stage we simply report all of the elements in \vec{n} .

$$\vec{l} = f(L_T, B, W) = M[L_T, B, N] \quad (6)$$

$$\vec{l} = f(L_T, B, W) = M[L_T, B, W, N] \quad (7)$$

In the above matrices M , the returned values are either a line number l or a collection of line numbers \vec{l} that were learned from examples for the files of those dimensions. However, if we discovered a file tested positive to contain a weakness, but we have never seen its dimensions (even taking into the account we can sometimes ignore W), we'll get a zero. This zero presents a problem: we can either (a) rely on one of the math functions described earlier to fill in that zero with a non-zero line number or (b) use probability values, and convert M to M_p , as shown in Equation 8.

The M_p matrix would contain a vector value \vec{n}_p of probabilities a given line number is a line number of a weakness.

$$\vec{l}_p = f(L_T, B, W) = M_p[L_T, B, W, N] \quad (8)$$

We then select the most probable ones from the list with the highest probabilities. The index i within \vec{l}_p represents the line number and the value at that index is the probability $p = \vec{l}_p[i]$.

Needless to say this 4D matrix is quite sparse and takes a while to learn. The learning is performed by counting occurrences of line numbers of weaknesses in the training data over total

of entries. To be better usable for the unseen cases the matrix needs to be smoothed using any of the statistical estimators available, e.g. from NLP, such as add-delta, ELE, MLE, Good-Turing, etc. by spreading the probabilities over to the zero-value cells from the non-zero ones. This is promising to be the slowest but the most accurate method.

In MARF, M is implemented using `marf.util.Matrix`, a free-form matrix that grows upon the need lazily and allows querying beyond physical dimensions when needed.

3.5.2 Classes of Functions

Define is the meaning of:

- $k_? = \frac{L_T}{B}$
- $k_? = \frac{W}{B}$

Non-learning:

1.
 - $k_* = 1$
 - $f(L_T) = L_T/2$
 - $f(B) = 0$
 - $f(W) = 0$
2.
 - $k_L = \frac{W}{B}$
 - $f(L_T) = L_T/2$
 - $f(B) = 0$
 - $f(W) = 0$
3.
 - $k_L = \frac{L_T}{B}$
 - $f(L_T) = L_T/2$
 - $f(B) = 0$
 - $f(W) = 0$
4.
 - $k_* = 1$
 - $f(L_T) = \text{random}(L_T)$
 - $f(B) = 0$
 - $f(W) = 0$

4 Results

The preliminary results of application of our methodology are outlined in this section. We summarize the top precisions per test case using either signal-processing or NLP-processing of the CVE-based cases and their application to the general cases. Subsequent sections detail some of the findings and issues of MARFCAT's result releases with different versions.

The results currently are being gradually released in the iterative manner that were obtained through the corresponding versions of MARFCAT as it was being designed and developed.

4.1 Preliminary Results Summary

Current top precision at the SATE2010 timeframe:

- Wireshark:
 - CVEs (signal): 92.68%, CWEs (signal): 86.11%,
 - CVEs (NLP): 83.33%, CWEs (NLP): 58.33%
- Tomcat:
 - CVEs (signal): 83.72%, CWEs (signal): 81.82%,
 - CVEs (NLP): 87.88%, CWEs (NLP): 39.39%
- Chrome:
 - CVEs (signal): 90.91%, CWEs (signal): 100.00%,
 - CVEs (NLP): 100.00%, CWEs (NLP): 88.89%
- Dovecot:
 - 14 warnings; but it appears all quality or false positive
 - (very hard to follow the code, severely undocumented)
- Pebble:
 - none found during quick testing

What follows are some select statistical measurements of the precision in recognizing CVEs and CWEs under different configurations using the signal processing and NLP processing techniques.

“Second guess” statistics provided to see if the hypothesis that if our first estimate of a CVE/CWE is incorrect, the next one in line is probably the correct one. Both are counted if the first guess is correct.

4.2 Version SATE.4

4.2.1 Wireshark 1.2.0

Typical quick run on the enriched Wireshark 1.2.0 on CVEs is in Table 1. All 22 CVEs are reported. Pretty good precision for options `-diff` and `-cheb` (Diff and Chebyshev distance classifiers, respectively [Mok08]). In Unigram, Add-Delta NLP results on Wireshark 1.2.0’s training file for CVEs, the precision seems to be overall degraded compared to the classical signal processing pipeline. Only 20 out of 22 CVEs are reported, as shown in Table 2. CWE-based testing on Wireshark 1.2.0 (also with some basic line heuristics that does not impact the precision) is in Table 3.

The following select reports are about Wireshark 1.2.0 using a small subset of algorithms. There are line numbers that were machine-learned from the `_train.xml` file. The two XML report files are the best ones we have chosen among several of them. Their precision rate using machine learning techniques is 92.68% after several bug corrections done. All CVEs are reported making recall 100%. The `stats-*.txt` files are there summarizing the evaluation precision. The

results are as good as the training data given; if there are mistakes in the data selection and annotation XML files, then the results will also have mistakes accordingly.

The best reports are:

```
report-noprepreprawfftcheb-wireshark-1.2.0-train.xml
report-noprepreprawfftdiff-wireshark-1.2.0-train.xml
```

The first one validates with both sate2010 schemas, but the latter has problems with the exponential -E notation.

Files. The corresponding *.log files are there for references, but contain a lot of debug information from the tool. The tool is using thresholding to reduce the amount of noise going into the reports.

```
marfcat-nopreprep-raw-fft-cheb.log
marfcat-nopreprep-raw-fft-diff.log
marfcat--super-fast.log (primarily training log)
report-noprepreprawfftcheb-wireshark-1.2.0-train.xml
report-noprepreprawfftdiff-wireshark-1.2.0-train.xml
stats--super-fast.txt
wireshark-1.2.0_train.xml
```

4.2.2 Wireshark 1.2.9

The following analysis reports are about Wireshark 1.2.9 using a small subset of MARF's algorithms. The system correctly does *not* report the fixed CVEs (currently, the primary class), so most of the reports come up empty (no noise). All example reports (one per configuration) validate with the schemas `sate_2010.xsd` and `sate_2010.pathcheck.xsd`.

The best (empty) reports are:

```
report-noprepreprawfftcheb-wireshark-1.2.9-test.xml
report-noprepreprawfftdiff-wireshark-1.2.9-test.xml
report-noprepreprawffteucl-wireshark-1.2.9-test.xml
report-noprepreprawffthamming-wireshark-1.2.9-test.xml
```

The below particular report shows the Minkowski distance classifier (-mink) was not perhaps the best choice, as it mistakingly reported a known CVE that was in fact fixed, this is an example of machine learning "red herring":

```
report-noprepreprawfftmink-wireshark-1.2.9-test.xml
```

Files. All the corresponding tool-specific *.log files are there for reference.

```
marfcat-nopreprep-raw-fft-cheb.log
marfcat-nopreprep-raw-fft-diff.log
marfcat-nopreprep-raw-fft-eucl.log
marfcat-nopreprep-raw-fft-hamming.log
marfcat-nopreprep-raw-fft-mink.log
marfcat--super-fast-wireshark.log (training log)
report-noprepreprawfftcheb-wireshark-1.2.9-test.xml
report-noprepreprawfftdiff-wireshark-1.2.9-test.xml
report-noprepreprawffteucl-wireshark-1.2.9-test.xml
report-noprepreprawffthamming-wireshark-1.2.9-test.xml
report-noprepreprawfftmink-wireshark-1.2.9-test.xml
```

4.2.3 Chrome 5.0.375.54

This version's CVE testing result of Chrome 5.0.375.54 (after updates and removal unrelated CVEs per SATE organizers) is in Table 4. The corresponding select reports produced below are about Chrome 5.0.375.54 using a small subset of algorithms. There are line numbers that were machine-learned from the *_train.xml file. The two report-*.xml files are ones of the best ones we have picked. Their precision rate using machine learning techniques is 90.91% after all the corrections done. The stats-*.txt file is there summarizing the evaluation precision in the end of that file. Again, the results are as good as the training data given; if there are mistakes in the data selection and annotation XML files, then the results will also have mistakes accordingly.

The best reports are:

report-noprepreprawfftcheb-chrome-5.0.375.54-train.xml

report-noprepreprawfftdiff-chrome-5.0.375.54-train.xml

Both validate with both sate2010 schemas.

Files. The corresponding *.log files are there for references, but contain A LOT of debug info from the tool. The tool is using thresholding to reduce the amount of noise going into the reports, but if you are curious to examine the logs, they are included.

```
chrome-5.0.375.54_train.xml
marfcat-nopreprep-raw-fft-cheb.log
marfcat-nopreprep-raw-fft-diff.log
marfcat--super-fast-chrome.log
README.txt
report-noprepreprawfftcheb-chrome-5.0.375.54-train.xml
report-noprepreprawfftdiff-chrome-5.0.375.54-train.xml
stats--super-fast.txt
```

4.2.4 Chrome 5.0.375.70

The following reports are about Chrome 5.0.375.70 using a small subset of algorithms. The system correctly does *not* report the fixed CVEs, so most of the reports come up empty (no noise) as they are expected to be for known CVE-selected weaknesses. All example reports (one per configuration) validate with the schema sate_2010.xsd and sate_2010.pathcheck.xsd.

The best (empty) reports are:

```
report-noprepreprawfftcheb-chrome-5.0.375.70-test.xml
report-noprepreprawfftdiff-chrome-5.0.375.70-test.xml
report-noprepreprawffteucl-chrome-5.0.375.70-test.xml
report-noprepreprawffthamming-chrome-5.0.375.70-test.xml
report-noprepreprawfftmink-chrome-5.0.375.70-test.xml
```

Files. All the corresponding tool-specific *.log files are there for reference.

```
chrome-5.0.375.70_test.xml
marfcat-nopreprep-raw-fft-cheb.log
marfcat-nopreprep-raw-fft-diff.log
marfcat-nopreprep-raw-fft-eucl.log
marfcat-nopreprep-raw-fft-hamming.log
```

```

marfcat-nopreprep-raw-fft-mink.log
marfcat--super-fast-chrome.log
report-noprepreprawfftcheb-chrome-5.0.375.70-test.xml
report-noprepreprawfftdiff-chrome-5.0.375.70-test.xml
report-noprepreprawffteucl-chrome-5.0.375.70-test.xml
report-noprepreprawffthamming-chrome-5.0.375.70-test.xml
report-noprepreprawfftmink-chrome-5.0.375.70-test.xml

```

4.3 Version SATE.5

4.3.1 Chrome 5.0.375.54

Here we complete the CVE results from the MARFCAT SATE.5 version by using Chrome 5.0.375.54 training on Chrome 5.0.375.54 with classical CWEs as opposed to CVEs. The result summary is in Table 5.

4.3.2 Tomcat 5.5.13

With this MARFCAT version we did first CVE-based testing on training for Tomcat 5.5.13. Classifiers corresponding to `-cheb` (Chebyshev distance) and `-diff` (Diff distance) continue to dominate as in the other test cases. An observation: for some reason, `-cos` (cosine similarity classifier) with the same settings as for the C/C++ projects (Wireshark and Chrome) actually performs well and `*_report.xml` is not as noisy; in fact comparable to `-cheb` and `-diff`. These CVE-based results are summarized in Table 6. Further, we perform quick CWE-based testing on Tomcat 5.5.13. Reports are quite larger for `-cheb`, `-diff`, and `-cos`, but not for other classifiers. The precision results are illustrated in Table 7. Then, in SATE.5, quick Tomcat 5.5.13 CVE NLP testing shows higher precision of 87.88%, but the recall is poor, 25/31 – 6 CVEs are missing out (see Table 8). Subsequent, quick Tomcat 5.5.13 CWE NLP testing was surprisingly poor topping at 39.39% (see Table 9). The resulting select reports about this Apache Tomcat 5.5.13 test case using a small subset of algorithms are mentioned below with some commentary.

CVE-based training and reporting: As before, there are line numbers that were machine-learned from the `_train.xml` file as well as the types of locations and descriptions provided by the SATE organizers and incorporated into the reports via machine learning. This includes the types of locations, such as “fix”, “sink”, or “path” learned from the organizers-provided XML/spreadsheet as well as the source code files. Two of all the produced XML reports are the best ones. The macro precision rate in there using machine learning techniques is 83.72%. The `stats-*.txt` files are there summarizing the evaluation precision.

The best reports are:

```

report-noprepreprawfftcheb-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawfftdiff-apache-tomcat-5.5.13-train-cve.xml
(does not validate three tool-specific lines)

```

Other reports are, to a various degree of detail and noise:

```

report-noprepreprawfftcos-apache-tomcat-5.5.13-train-cve.xml
(does not validate two lines)
report-noprepreprawffteucl-apache-tomcat-5.5.13-train-cve.xml
(does not validate three tool-specific lines)
report-noprepreprawffthamming-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawfftmink-apache-tomcat-5.5.13-train-cve.xml

```

`report-nopreprepcharunigramadddelta-apache-tomcat-5.5.13-train-cve-nlp.xml`

The `--nlp` version reports use the NLP techniques with the machine learning instead of signal processing techniques. Those reports are largely comparable, but have smaller recall, i.e. some CVEs are completely missing out from the reports in this version. Some reports have problems with tool-specific ranks like: $4.199735736674989E - 4$, which we will have to see how to reduce these.

CWE-based training and reporting: The CWE-based reports use the CWE as a primary class instead of CVE for training and reporting, and as such currently do not report on CVEs directly (i.e. no direct mapping from CWE to CVE exists unlike in the opposite direction); however, their recognition rates are not very low either in the same spots, types, etc. In the future version of MARFCAT the plan is to combine the two machine learning pipeline runs of CVE and CWE together to improve mutual classification, but right now it is not available. The CWE-based training is also used on the testing files say of Pebble to see if there are any similar weaknesses to that of Tomcat found, again e.g. in Pebble. CWEs, unlike CVEs for most projects, represent better cross-project classes as they are largely project-independent. Both CVE-based and CWE-base methods use the same data for training. CWEs are recognized correctly 81.82% for Tomcat. NLP-based CWE testing is not included as its precision was quite low ($\approx 39\%$).

The best reports are:

`report-cweidnoprepreprawfftcheb-apache-tomcat-5.5.13-train-cwe.xml`

(does not validate)

`report-cweidnoprepreprawfftdiff-apache-tomcat-5.5.13-train-cwe.xml`

(does not validate)

Other reports are, to a various degree of detail and noise:

`report-cweidnoprepreprawfftcos-apache-tomcat-5.5.13-train-cwe.xml`

`report-cweidnoprepreprawffteucl-apache-tomcat-5.5.13-train-cwe.xml`

(does not validate)

`report-cweidnoprepreprawffthamming-apache-tomcat-5.5.13-train-cwe.xml`

`report-cweidnoprepreprawfftmink-apache-tomcat-5.5.13-train-cwe.xml`

Files. The corresponding *.log files are there for references, but contain A LOT of debug info from the tool. The tool is using thresholding to reduce the amount of noise going into the reports, but if you are curious to examine the logs, they are included.

`apache-tomcat-5.5.13-src_train.xml` (meta training file)

`marfcat-cweid-nopreprep-raw-fft-cheb.log`

`marfcat-cweid-nopreprep-raw-fft-cos.log`

`marfcat-cweid-nopreprep-raw-fft-diff.log`

`marfcat-cweid-nopreprep-raw-fft-eucl.log`

`marfcat-cweid-nopreprep-raw-fft-hamming.log`

`marfcat-cweid-nopreprep-raw-fft-mink.log`

`marfcat-nopreprep-char-unigram-add-delta.log`

`marfcat-nopreprep-raw-fft-cheb.log`

`marfcat-nopreprep-raw-fft-cos.log`

`marfcat-nopreprep-raw-fft-diff.log`

`marfcat-nopreprep-raw-fft-eucl.log`

`marfcat-nopreprep-raw-fft-hamming.log`

`marfcat-nopreprep-raw-fft-mink.log`

```

marfcat--super-fast-tomcat-train-cve.log
marfcat--super-fast-tomcat-train-cve-nlp.log
marfcat--super-fast-tomcat-train-cwe.log
report-cweidnoprepreprawfftcheb-apache-tomcat-5.5.13-train-cwe.xml
report-cweidnoprepreprawfftcos-apache-tomcat-5.5.13-train-cwe.xml
report-cweidnoprepreprawfftdiff-apache-tomcat-5.5.13-train-cwe.xml
report-cweidnoprepreprawffteucl-apache-tomcat-5.5.13-train-cwe.xml
report-cweidnoprepreprawffthamming-apache-tomcat-5.5.13-train-cwe.xml
report-cweidnoprepreprawfftmink-apache-tomcat-5.5.13-train-cwe.xml
report-nopreprepcharunigramadddelta-apache-tomcat-5.5.13-train-cve-nlp.xml
report-noprepreprawfftcheb-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawfftcos-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawfftdiff-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawffteucl-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawffthamming-apache-tomcat-5.5.13-train-cve.xml
report-noprepreprawfftmink-apache-tomcat-5.5.13-train-cve.xml
stats-per-cve-nlp.txt
stats-per-cve.txt
stats-per-cwe.txt

```

4.3.3 Pebble 2.5-M2

Using the machine learning approach of MARF by using the Tomcat 5.5.13 as a source of training on a Java project with known weaknesses, we used that (rather small) “knowledge base” to test if anything weak similar to the weaknesses in Tomcat are also present in the supplied version of Pebble 2.5-M2. The current result is that under the version of MARFCAT SATE.5 all reports come up empty under the current thresholding rules meaning the tool was not able to identify similar weaknesses in files in Pebble. The corresponding tool-specific log files are also provided if of interest, but the volume of data in them is typically large. It is planned to lower the thresholds after reviewing logs in detail to see if anything interesting comes up that we missed otherwise.

Files.

```

marfcat--super-fast-tomcat13-pebble-cwe.log
marfcat-cweid-nopreprep-raw-fft-cheb.log
marfcat-cweid-nopreprep-raw-fft-cos.log
marfcat-cweid-nopreprep-raw-fft-diff.log
marfcat-cweid-nopreprep-raw-fft-eucl.log
marfcat-cweid-nopreprep-raw-fft-hamming.log
marfcat-cweid-nopreprep-raw-fft-mink.log
report-cweidnoprepreprawfftcheb-pebble-test-cwe.xml
report-cweidnoprepreprawfftcos-pebble-test-cwe.xml
report-cweidnoprepreprawfftdiff-pebble-test-cwe.xml
report-cweidnoprepreprawffteucl-pebble-test-cwe.xml
report-cweidnoprepreprawffthamming-pebble-test-cwe.xml
report-cweidnoprepreprawfftmink-pebble-test-cwe.xml

```

4.3.4 Tomcat and Pebble Testing Results Summary

- Tomcat 5.5.13 on Tomcat 5.5.29 classical CVE testing produced only report with `-cos` with 10 weaknesses, some correspond to the files in training. However, the line numbers reported are midline, so next to meaningless.
- Tomcat 5.5.13 on Tomcat 5.5.29 classical CWE testing also report with `-cos` with 2 weaknesses.
- Tomcat 5.5.13 on Tomcat 5.5.29 NLP CVE testing single report (quick testing only does add-delta, unigram) came up empty.
- Tomcat 5.5.13 on Tomcat 5.5.29 NLP CWE testing, also with a single report (quick testing only does add-delta, unigram) came up empty.
- Tomcat 5.5.13 on Pebble classical CVE reports are empty.
- Tomcat 5.5.13 on Pebble NLP CVE report is not empty, but reports wrongly on `blank.html` (empty HTML file) on multiple CVEs. The probability $P = 0.0$ for all in this case CVEs, not sure why it is at all reported. A red herring.
- Tomcat 5.5.13 on Pebble classical CWE reports are empty.
- Tomcat 5.5.13 on Pebble NLP CWE is similar to the Pebble NLP CVE report on `blank.html` entries, but fewer of them. All the other symptoms are the same.

4.4 Version SATE.6

4.4.1 Dovecot 2.0.beta6

This is a quick test and a report for Dovecot 2.0.beta6, with line numbers and other information. The report is ‘raw’, without our manual evaluation and generated as-is at this point.

The report of interest:

```
report-cweidnoprepreprawfftcos-dovecot-2.0.beta6-wireshark-test-cwe.xml
```

It appears though from the first glance most of the are warnings are ‘bogus’ or ‘buggy’, but could indicate potential presence of weaknesses in the flagged files. One thing is for sure the Dovecode’s source code’s main weakness is a near chronic lack of comments, which is also a weakness of a kind. Other reports came up empty. The source for learning was Wireshark 1.2.0.

Files.

```
dovecot-2.0.beta6_test.xml
marfcatsuperfastdovecotwiresharktestcwe.log
marfcatcweidnopreprep-raw-fft-cheb.log
marfcatcweidnopreprep-raw-fft-cos.log
marfcatcweidnopreprep-raw-fft-diff.log
marfcatcweidnopreprep-raw-fft-eucl.log
marfcatcweidnopreprep-raw-fft-hamming.log
marfcatcweidnopreprep-raw-fft-mink.log
report-cweidnoprepreprawfftcheb-dovecot-2.0.beta6-wireshark-test-cwe.xml
report-cweidnoprepreprawfftcheb-wireshark-1.2.0_train.xml.xml
report-cweidnoprepreprawfftcos-dovecot-2.0.beta6-wireshark-test-cwe.xml
```

```
report-cweidnoprepreprawfftdiff-dovecot-2.0.beta6-wireshark-test-cwe.xml
report-cweidnoprepreprawffteucl-dovecot-2.0.beta6-wireshark-test-cwe.xml
report-cweidnoprepreprawffthamming-dovecot-2.0.beta6-wireshark-test-cwe.xml
report-cweidnoprepreprawfftmink-dovecot-2.0.beta6-wireshark-test-cwe.xml
```

4.4.2 Tomcat 5.5.29

This is another quick CVE-based evaluation of Tomcat 5.5.29, with line numbers, etc. They are 'raw', without our manual evaluation and generated as-is.

The reports of interest:

```
report-noprepreprawfftcos-apache-tomcat-5.5.29-test-cve.xml
report-cweidnoprepreprawfftcos-apache-tomcat-5.5.29-test-cwe.xml
```

As for the Dovecot case, it appears though from the first glance most of the warnings are either 'bogus' or 'buggy', but could indicate potential presence of weaknesses in the flagged files or fixed as such. Need more manual inspection to be sure. Other XML reports came up empty. The source for learning was Tomcat 5.5.13.

Files.

```
marfcatsuperfasttomcat13tomcat29cve.log
marfcatsuperfasttomcat13tomcat29cwe.log
marfcatscweidnopreprep-raw-fft-cheb.log
marfcatscweidnopreprep-raw-fft-cos.log
marfcatscweidnopreprep-raw-fft-diff.log
marfcatscweidnopreprep-raw-fft-eucl.log
marfcatscweidnopreprep-raw-fft-hamming.log
marfcatscweidnopreprep-raw-fft-mink.log
marfcatsnopreprep-raw-fft-cheb.log
marfcatsnopreprep-raw-fft-cos.log
marfcatsnopreprep-raw-fft-diff.log
marfcatsnopreprep-raw-fft-eucl.log
marfcatsnopreprep-raw-fft-hamming.log
marfcatsnopreprep-raw-fft-mink.log
report-cweidnoprepreprawfftcheb-apache-tomcat-5.5.29-test-cwe.xml
report-cweidnoprepreprawfftcos-apache-tomcat-5.5.29-test-cwe.xml
report-cweidnoprepreprawfftdiff-apache-tomcat-5.5.29-test-cwe.xml
report-cweidnoprepreprawffteucl-apache-tomcat-5.5.29-test-cwe.xml
report-cweidnoprepreprawffthamming-apache-tomcat-5.5.29-test-cwe.xml
report-cweidnoprepreprawfftmink-apache-tomcat-5.5.29-test-cwe.xml
report-noprepreprawfftcheb-apache-tomcat-5.5.29-test-cve.xml
report-noprepreprawfftcos-apache-tomcat-5.5.29-test-cve.xml
report-noprepreprawfftdiff-apache-tomcat-5.5.29-test-cve.xml
report-noprepreprawffteucl-apache-tomcat-5.5.29-test-cve.xml
report-noprepreprawffthamming-apache-tomcat-5.5.29-test-cve.xml
report-noprepreprawfftmink-apache-tomcat-5.5.29-test-cve.xml
```


4.5 Version SATE.7

Up until this version NLP processing of Chrome was not successful. Errors related to the number of file descriptors opened and “mark invalid” for NLP processing of Chrome 5.0.375.54 for both CVEs and CWEs have been corrected, so we have produced the results for these cases. CVEs are reported in Table 10. CWEs are further reported in Table 11.

5 Conclusion

We review the current results of this experimental work, its current shortcomings, advantages, and practical implications. We also release MARFCAT Alpha version as open-source that can be found at [Mok11]. This is following the open-source philosophy of greater good (MARF itself has been open-source from the very beginning [The11]).

5.1 Shortcomings

The below is a list of most prominent issues with the presented approach. Some of them are more “permanent”, while others are solvable and intended to be addressed in the future work. Specifically:

- Looking at a signal is less intuitive visually for code analysis by humans.
- Line numbers are a problem (easily “filtered out” as high-frequency “noise”, etc.). A whole “relativistic” and machine learning methodology developed for the line numbers in Section 3.5 to compensate for that. Generally, when CVEs is the primary class, by accurately identifying the CVE number one can get all the other pertinent details from the CVE database, including patches and line numbers.
- Accuracy depends on the quality of the knowledge base (see Section 3.2) collected. “Garbage in – garbage out.”
- To detect CVE or CWE signatures in non-CVE cases requires large knowledge bases (human-intensive to collect).
- No path tracing (since no parsing is present); no slicing, semantic annotations, context, locality of reference, etc. The “sink”, “path”, and “fix” results in the reports also have to be machine-learned.
- A lot of algorithms and their combinations to try (currently ≈ 1800 permutations) to get the best top N. This is, however, also an advantage of the approach as the underlying framework can quickly allow for such testing.
- File-level training vs. fragment-level training – presently the classes are trained based on the entire file where weaknesses are found instead of the known fragments from CVE-reported patches. The latter would be more fine-grained and precise than whole-file classification, but slower. However, overall the file-level processing is a man-hour limitation than a technological one.
- No nice GUI. Presently the application is script/command-line based.

5.2 Advantages

There are some key advantages of the approach presented. Some of them follow:

- Relatively fast (e.g. Wireshark's \approx 2400 files train and test in about 3 minutes) on a now-commodity desktop.
- Language-independent (no parsing) – given enough examples can apply to any language, i.e. methodology is the same no matter C, C++, Java or any other source or binary languages (PHP, C#, VB, Perl, bytecode, assembly, etc.).
- Can automatically learn a large knowledge base to test on known and unknown cases.
- Can be used to quickly pre-scan projects for further analysis by humans and other tools that do in-depth semantic analysis.
- Can learn from other SATE'10 reports.
- Can learn from SATE'09 and SATE'08 reports.
- High precision in CVEs and CWE detection.
- Lots of algorithms and their combinations to select the best for a particular task or class (see Section 3.3).

5.3 Practical Implications

Most practical implications of all static code analyzers are obvious – to detect and report source code weaknesses and report them appropriately to the developers. We outline additional implications this approach brings to the arsenal below:

- The approach can be used on any target language without modifications to the methodology or knowing the syntax of the language. Thus, it scales to any popular and new language analysis with a very small amount of effort.
- The approach can nearly identically be transposed onto the compiled binaries and bytecode, detecting vulnerable deployments and installations – sort of like virus scanning of binaries, but instead scanning for infected binaries, one would scan for security-weak binaries on site deployments to alert system administrators to upgrade their packages.
- Can learn from binary signatures from other tools like Snort [Sou10].

5.4 Future Work

There is a great number of possibilities in the future work. This includes improvements to the code base of MARFCAT as well as resolving unfinished scenarios and results, addressing shortcomings in Section 5.1, testing more algorithms and combinations from the related work, and moving onto other programming languages (e.g. PHP, ASP, C#). Furthermore, plan to conceive collaboration with vendors such as VeraCode, Coverity, and others who have vast data sets to test the full potential of the approach with the others and a community as a whole. Then move on to dynamic code analysis as well applying similar techniques there.

References

- [ESI⁺09] Masashi Eto, Kotaro Sonoda, Daisuke Inoue, Katsunari Yoshioka, and Koji Nakao. A proposal of malware distinction method based on scan patterns using spectrum analysis. In *Proceedings of the 16th International Conference on Neural Information Processing: Part II*, ICONIP'09, pages 565–572, Berlin, Heidelberg, 2009. Springer-Verlag.
- [Fre09] Free Software Foundation, Inc. `wc` – print newline, word, and byte counts for each file. GNU coreutils 6.10, 2009. `man 1 wc`.
- [HLYD09] Aiman Hanna, Hai Zhou Ling, Xiaochun Yang, and Mourad Debbabi. A synergy between static and dynamic analysis for the detection of software security vulnerabilities. In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *OTM Conferences (2)*, volume 5871 of *Lecture Notes in Computer Science*, pages 815–832. Springer, 2009.
- [IYE⁺09] Daisuke Inoue, Katsunari Yoshioka, Masashi Eto, Masaya Yamagata, Eisuke Nishino, Jun'ichi Takeuchi, Kazuya Ohkouchi, and Koji Nakao. An incident analysis system NICTER and its analysis engines based on data mining techniques. In *Proceedings of the 15th International Conference on Advances in Neuro-Information Processing – Volume Part I*, ICONIP'08, pages 579–586, Berlin, Heidelberg, 2009. Springer-Verlag.
- [KAYE04] Ted Kremenek, Ken Ashcraft, Junfeng Yang, and Dawson Engler. Correlation exploitation in error ranking. In *Foundations of Software Engineering (FSE)*, 2004.
- [KE03] Ted Kremenek and Dawson Engler. Z-ranking: Using statistical analysis to counter the impact of static analysis approximations. In *SAS 2003*, 2003.
- [KTB⁺06] Ted Kremenek, Paul Twohey, Godmar Back, Andrew Ng, and Dawson Engler. From uncertainty to belief: Inferring the specification within. In *Proceedings of the 7th Symposium on Operating System Design and Implementation*, 2006.
- [KZL10] Ying Kong, Yuqing Zhang, and Qixu Liu. Eliminating human specification in static analysis. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 494–495, Berlin, Heidelberg, 2010. Springer-Verlag.
- [MD08] Serguei A. Mokhov and Mourad Debbabi. File type analysis using signal processing techniques and machine learning vs. `file` unix utility for forensic analysis. In Oliver Goebel, Sandra Frings, Detlef Guenther, Jens Nedon, and Dirk Schadt, editors, *Proceedings of the IT Incident Management and IT Forensics (IMF'08)*, LNI140, pages 73–85. GI, September 2008.
- [MES02] D. Mackenzie, P. Eggert, and R. Stallman. Comparing and merging files. [online], 2002. <http://www.gnu.org/software/diffutils/manual/ps/diff.ps.gz>.
- [MLB07] Serguei A. Mokhov, Marc-André Laverdière, and Djamel Benredjem. Taxonomy of linux kernel vulnerability solutions. In *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, pages 485–493, University of Bridgeport, U.S.A., 2007. Proceedings of CISSE/SCSS'07.
- [Mok07] Serguei A. Mokhov. Introducing MARF: a modular audio recognition framework and its applications for scientific and software engineering research. In *Advances in Computer and Information Sciences and Engineering*, pages 473–478, University of Bridgeport, U.S.A., December 2007. Springer Netherlands. Proceedings of CISSE/SCSS'07.
- [Mok08] Serguei A. Mokhov. Study of best algorithm combinations for speech processing tasks in machine learning using median vs. mean clusters in MARF. In Bipin C. Desai, editor, *Proceedings of C3S2E'08*, pages 29–43, Montreal, Quebec, Canada, May 2008. ACM.
- [Mok10a] Serguei A. Mokhov. Complete complimentary results report of the MARF's NLP approach to the DEFT 2010 competition. [online], June 2010. <http://arxiv.org/abs/1006.3787>.
- [Mok10b] Serguei A. Mokhov. L'approche MARF à DEFT 2010: A MARF approach to DEFT 2010. In *Proceedings of TALN'10*, July 2010. To appear in DEFT 2010 System competition at TALN 2010.
- [Mok11] Serguei A. Mokhov. MARFCAT – MARF-based Code Analysis Tool. Published electronically within the MARF project,

- <http://sourceforge.net/projects/marf/files/Applications/MARFCAT/>, 2010–2011. Last viewed February 2011.
- [MSS09] Serguei A. Mokhov, Miao Song, and Ching Y. Suen. Writer identification using inexpensive signal processing techniques. In Tarek Sobh and Khaled Elleithy, editors, *Innovations in Computing Sciences and Software Engineering; Proceedings of CISSE'09*, pages 437–441. Springer, December 2009. ISBN: 978-90-481-9111-6, online at: <http://arxiv.org/abs/0912.5502>.
- [NIS11a] NIST. National Vulnerability Database. [online], 2005–2011. <http://nvd.nist.gov/>.
- [NIS11b] NIST. National Vulnerability Database statistics. [online], 2005–2011. <http://web.nvd.nist.gov/view/vuln/statistics>.
- [NJG⁺10] Vinod P. Nair, Harshit Jain, Yashwant K. Golecha, Manoj Singh Gaur, and Vijay Laxmi. MEDUSA: METamorphic malware dynamic analysis using signature from API. In *Proceedings of the 3rd International Conference on Security of Information and Networks, SIN'10*, pages 263–269, New York, NY, USA, 2010. ACM.
- [ODBN10] Vadim Okun, Aurelien Delaitre, Paul E. Black, and NIST SAMATE. Static Analysis Tool Exposition (SATE) 2010. [online], 2010. See <http://samate.nist.gov/SATE.html> and <http://samate.nist.gov/SATE2010Workshop.html>.
- [Sou10] Sourcefire. Snort: Open-source network intrusion prevention and detection system (IDS/IPS). [online], 2010. <http://www.snort.org/>.
- [The11] The MARF Research and Development Group. The Modular Audio Recognition Framework and its Applications. [online], 2002–2011. <http://marf.sf.net> and <http://arxiv.org/abs/0905.1235>, last viewed April 2010.
- [Tli09] Syrine Tlili. *Automatic detection of safety and security vulnerabilities in open source software*. PhD thesis, Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada, 2009. ISBN: 9780494634165.
- [VM10] Various contributors and MITRE. Common Weakness Enumeration (CWE) – a community-developed dictionary of software weakness types. [online], 2010. See <http://cwe.mitre.org>.

A Classification Result Tables

What follows are result tables with top classification results ranked from most precise at the top. This include the configuration settings for MARF by the means of options (the algorithm implementations are at their defaults [Mok07]).

Table 1: CVE Stats for Wireshark 1.2.0, Quick Enriched, version SATE.4

guess	run	algorithms	good	bad	%
1st	1	-nopreprep -raw -fft -diff	38	3	92.68
1st	2	-nopreprep -raw -fft -cheb	38	3	92.68
1st	3	-nopreprep -raw -fft -eucl	29	12	70.73
1st	4	-nopreprep -raw -fft -hamming	26	15	63.41
1st	5	-nopreprep -raw -fft -mink	23	18	56.10
1st	6	-nopreprep -raw -fft -cos	37	51	42.05
2nd	1	-nopreprep -raw -fft -diff	39	2	95.12
2nd	2	-nopreprep -raw -fft -cheb	39	2	95.12
2nd	3	-nopreprep -raw -fft -eucl	34	7	82.93
2nd	4	-nopreprep -raw -fft -hamming	28	13	68.29
2nd	5	-nopreprep -raw -fft -mink	31	10	75.61
2nd	6	-nopreprep -raw -fft -cos	38	50	43.18
guess	run	class	good	bad	%
1st	1	CVE-2009-3829	6	0	100.00
1st	2	CVE-2009-2563	6	0	100.00
1st	3	CVE-2009-2562	6	0	100.00
1st	4	CVE-2009-4378	6	0	100.00
1st	5	CVE-2009-4376	6	0	100.00
1st	6	CVE-2010-0304	6	0	100.00
1st	7	CVE-2010-2286	6	0	100.00
1st	8	CVE-2010-2283	6	0	100.00
1st	9	CVE-2009-3551	6	0	100.00
1st	10	CVE-2009-3550	6	0	100.00
1st	11	CVE-2009-3549	6	0	100.00
1st	12	CVE-2009-3241	16	8	66.67
1st	13	CVE-2010-1455	34	20	62.96
1st	14	CVE-2009-3243	18	11	62.07
1st	15	CVE-2009-2560	8	6	57.14
1st	16	CVE-2009-2561	6	5	54.55
1st	17	CVE-2010-2285	6	5	54.55
1st	18	CVE-2009-2559	6	5	54.55
1st	19	CVE-2010-2287	6	6	50.00
1st	20	CVE-2009-4377	12	15	44.44
1st	21	CVE-2010-2284	6	9	40.00
1st	22	CVE-2009-3242	7	12	36.84
2nd	1	CVE-2009-3829	6	0	100.00
2nd	2	CVE-2009-2563	6	0	100.00
2nd	3	CVE-2009-2562	6	0	100.00
2nd	4	CVE-2009-4378	6	0	100.00
2nd	5	CVE-2009-4376	6	0	100.00
2nd	6	CVE-2010-0304	6	0	100.00
2nd	7	CVE-2010-2286	6	0	100.00
2nd	8	CVE-2010-2283	6	0	100.00
2nd	9	CVE-2009-3551	6	0	100.00
2nd	10	CVE-2009-3550	6	0	100.00
2nd	11	CVE-2009-3549	6	0	100.00
2nd	12	CVE-2009-3241	17	7	70.83
2nd	13	CVE-2010-1455	44	10	81.48
2nd	14	CVE-2009-3243	18	11	62.07
2nd	15	CVE-2009-2560	9	5	64.29
2nd	16	CVE-2009-2561	6	5	54.55
2nd	17	CVE-2010-2285	6	5	54.55
2nd	18	CVE-2009-2559	6	5	54.55
2nd	19	CVE-2010-2287	12	0	100.00
2nd	20	CVE-2009-4377	12	15	44.44
2nd	21	CVE-2010-2284	6	9	40.00
2nd	22	CVE-2009-3242	7	12	36.84

Table 2: CVE NLP Stats for Wireshark 1.2.0, Quick Enriched, version SATE.4

guess	run	algorithms	good	bad	%
1st	1	-nopreprep -char -unigram -add-delta	30	6	83.33
2nd	1	-nopreprep -char -unigram -add-delta	31	5	86.11
guess	run	class	good	bad	%
1st	1	CVE-2009-3829	1	0	100.00
1st	2	CVE-2009-2563	1	0	100.00
1st	3	CVE-2009-2562	1	0	100.00
1st	4	CVE-2009-4378	1	0	100.00
1st	5	CVE-2009-2561	1	0	100.00
1st	6	CVE-2009-4377	1	0	100.00
1st	7	CVE-2009-4376	1	0	100.00
1st	8	CVE-2010-2286	1	0	100.00
1st	9	CVE-2010-0304	1	0	100.00
1st	10	CVE-2010-2285	1	0	100.00
1st	11	CVE-2010-2284	1	0	100.00
1st	12	CVE-2010-2283	1	0	100.00
1st	13	CVE-2009-2559	1	0	100.00
1st	14	CVE-2009-3550	1	0	100.00
1st	15	CVE-2009-3549	1	0	100.00
1st	16	CVE-2010-1455	8	1	88.89
1st	17	CVE-2009-3243	3	1	75.00
1st	18	CVE-2009-3241	2	2	50.00
1st	19	CVE-2009-2560	1	1	50.00
1st	20	CVE-2009-3242	1	1	50.00
2nd	1	CVE-2009-3829	1	0	100.00
2nd	2	CVE-2009-2563	1	0	100.00
2nd	3	CVE-2009-2562	1	0	100.00
2nd	4	CVE-2009-4378	1	0	100.00
2nd	5	CVE-2009-2561	1	0	100.00
2nd	6	CVE-2009-4377	1	0	100.00
2nd	7	CVE-2009-4376	1	0	100.00
2nd	8	CVE-2010-2286	1	0	100.00
2nd	9	CVE-2010-0304	1	0	100.00
2nd	10	CVE-2010-2285	1	0	100.00
2nd	11	CVE-2010-2284	1	0	100.00
2nd	12	CVE-2010-2283	1	0	100.00
2nd	13	CVE-2009-2559	1	0	100.00
2nd	14	CVE-2009-3550	1	0	100.00
2nd	15	CVE-2009-3549	1	0	100.00
2nd	16	CVE-2010-1455	8	1	88.89
2nd	17	CVE-2009-3243	3	1	75.00
2nd	18	CVE-2009-3241	3	1	75.00
2nd	19	CVE-2009-2560	1	1	50.00
2nd	20	CVE-2009-3242	1	1	50.00

Table 3: CVE NLP Stats for Wireshark 1.2.0, Quick Enriched, version SATE.4

guess	run	algorithms	good	bad	%
1st	1	-cweid -nopreprep -raw -fft -cheb	31	5	86.11
1st	2	-cweid -nopreprep -raw -fft -diff	31	5	86.11
1st	3	-cweid -nopreprep -raw -fft -eucl	29	7	80.56
1st	4	-cweid -nopreprep -raw -fft -hamming	22	14	61.11
1st	5	-cweid -nopreprep -raw -fft -cos	33	25	56.90
1st	6	-cweid -nopreprep -raw -fft -mink	20	16	55.56
2nd	1	-cweid -nopreprep -raw -fft -cheb	33	3	91.67
2nd	2	-cweid -nopreprep -raw -fft -diff	33	3	91.67
2nd	3	-cweid -nopreprep -raw -fft -eucl	33	3	91.67
2nd	4	-cweid -nopreprep -raw -fft -hamming	27	9	75.00
2nd	5	-cweid -nopreprep -raw -fft -cos	41	17	70.69
2nd	6	-cweid -nopreprep -raw -fft -mink	22	14	61.11
guess	run	class	good	bad	%
1st	1	CWE-399	6	0	100.00
1st	2	NVD-CWE-Other	17	3	85.00
1st	3	CWE-20	50	10	83.33
1st	4	CWE-189	8	2	80.00
1st	5	NVD-CWE-noinfo	72	40	64.29
1st	6	CWE-119	13	17	43.33
2nd	1	CWE-399	6	0	100.00
2nd	2	NVD-CWE-Other	17	3	85.00
2nd	3	CWE-20	52	8	86.67
2nd	4	CWE-189	8	2	80.00
2nd	5	NVD-CWE-noinfo	83	29	74.11
2nd	6	CWE-119	23	7	76.67

Table 4: CVE Stats for Chrome 5.0.375.54, Quick Enriched, (clean CVEs) version SATE.4

guess	run	algorithms	good	bad	%
1st	1	-nopreprep -raw -fft -eucl	10	1	90.91
1st	2	-nopreprep -raw -fft -cos	10	1	90.91
1st	3	-nopreprep -raw -fft -diff	10	1	90.91
1st	4	-nopreprep -raw -fft -cheb	10	1	90.91
1st	5	-nopreprep -raw -fft -mink	9	2	81.82
1st	6	-nopreprep -raw -fft -hamming	9	2	81.82
2nd	1	-nopreprep -raw -fft -eucl	11	0	100.00
2nd	2	-nopreprep -raw -fft -cos	11	0	100.00
2nd	3	-nopreprep -raw -fft -diff	11	0	100.00
2nd	4	-nopreprep -raw -fft -cheb	11	0	100.00
2nd	5	-nopreprep -raw -fft -mink	10	1	90.91
2nd	6	-nopreprep -raw -fft -hamming	10	1	90.91
guess	run	class	good	bad	%
1st	1	CVE-2010-2301	6	0	100.00
1st	2	CVE-2010-2300	6	0	100.00
1st	3	CVE-2010-2299	6	0	100.00
1st	4	CVE-2010-2298	6	0	100.00
1st	5	CVE-2010-2297	6	0	100.00
1st	6	CVE-2010-2304	6	0	100.00
1st	7	CVE-2010-2303	6	0	100.00
1st	8	CVE-2010-2295	10	2	83.33
1st	9	CVE-2010-2302	6	6	50.00
2nd	1	CVE-2010-2301	6	0	100.00
2nd	2	CVE-2010-2300	6	0	100.00
2nd	3	CVE-2010-2299	6	0	100.00
2nd	4	CVE-2010-2298	6	0	100.00
2nd	5	CVE-2010-2297	6	0	100.00
2nd	6	CVE-2010-2304	6	0	100.00
2nd	7	CVE-2010-2303	6	0	100.00
2nd	8	CVE-2010-2295	10	2	83.33
2nd	9	CVE-2010-2302	12	0	100.00

Table 5: CWE Stats for Chrome 5.0.375.54, (clean CVEs) version SATE.5

guess	run	algorithms	good	bad	%
1st	1	-cweid -nopreprep -raw -fft -cheb	9	0	100.00
1st	2	-cweid -nopreprep -raw -fft -cos	9	0	100.00
1st	3	-cweid -nopreprep -raw -fft -diff	9	0	100.00
1st	4	-cweid -nopreprep -raw -fft -eucl	8	1	88.89
1st	5	-cweid -nopreprep -raw -fft -hamming	8	1	88.89
1st	6	-cweid -nopreprep -raw -fft -mink	6	3	66.67
2nd	1	-cweid -nopreprep -raw -fft -cheb	9	0	100.00
2nd	2	-cweid -nopreprep -raw -fft -cos	9	0	100.00
2nd	3	-cweid -nopreprep -raw -fft -diff	9	0	100.00
2nd	4	-cweid -nopreprep -raw -fft -eucl	8	1	88.89
2nd	5	-cweid -nopreprep -raw -fft -hamming	8	1	88.89
2nd	6	-cweid -nopreprep -raw -fft -mink	8	1	88.89
guess	run	class	good	bad	%
1st	1	CWE-79	6	0	100.00
1st	2	NVD-CWE-noinfo	6	0	100.00
1st	3	CWE-399	6	0	100.00
1st	4	CWE-119	6	0	100.00
1st	5	CWE-20	6	0	100.00
1st	6	NVD-CWE-Other	10	2	83.33
1st	7	CWE-94	9	3	75.00
2nd	1	CWE-79	6	0	100.00
2nd	2	NVD-CWE-noinfo	6	0	100.00
2nd	3	CWE-399	6	0	100.00
2nd	4	CWE-119	6	0	100.00
2nd	5	CWE-20	6	0	100.00
2nd	6	NVD-CWE-Other	11	1	91.67
2nd	7	CWE-94	10	2	83.33

Table 6: CVE Stats for Tomcat 5.5.13, version SATE.5

1st	1	-nopreprep -raw -fft -diff	36	7	83.72
1st	2	-nopreprep -raw -fft -cheb	36	7	83.72
1st	3	-nopreprep -raw -fft -cos	37	9	80.43
1st	4	-nopreprep -raw -fft -eucl	34	9	79.07
1st	5	-nopreprep -raw -fft -mink	28	15	65.12
1st	6	-nopreprep -raw -fft -hamming	26	17	60.47
2nd	1	-nopreprep -raw -fft -diff	40	3	93.02
2nd	2	-nopreprep -raw -fft -cheb	40	3	93.02
2nd	3	-nopreprep -raw -fft -cos	40	6	86.96
2nd	4	-nopreprep -raw -fft -eucl	36	7	83.72
2nd	5	-nopreprep -raw -fft -mink	31	12	72.09
2nd	6	-nopreprep -raw -fft -hamming	29	14	67.44
guess	run	algorithms	good	bad	%
1st	1	CVE-2006-7197	6	0	100.00
1st	2	CVE-2006-7196	6	0	100.00
1st	3	CVE-2006-7195	6	0	100.00
1st	4	CVE-2009-0033	6	0	100.00
1st	5	CVE-2007-3386	6	0	100.00
1st	6	CVE-2009-2901	3	0	100.00
1st	7	CVE-2007-3385	6	0	100.00
1st	8	CVE-2008-2938	6	0	100.00
1st	9	CVE-2007-3382	6	0	100.00
1st	10	CVE-2007-5461	6	0	100.00
1st	11	CVE-2007-6286	6	0	100.00
1st	12	CVE-2007-1858	6	0	100.00
1st	13	CVE-2008-0128	6	0	100.00
1st	14	CVE-2007-2450	6	0	100.00
1st	15	CVE-2009-3548	6	0	100.00
1st	16	CVE-2009-0580	6	0	100.00
1st	17	CVE-2007-1355	6	0	100.00
1st	18	CVE-2008-2370	6	0	100.00
1st	19	CVE-2008-4308	6	0	100.00
1st	20	CVE-2007-5342	6	0	100.00
1st	21	CVE-2008-5515	19	5	79.17
1st	22	CVE-2009-0783	11	4	73.33
1st	23	CVE-2008-1232	13	5	72.22
1st	24	CVE-2008-5519	6	6	50.00
1st	25	CVE-2007-5333	6	6	50.00
1st	26	CVE-2008-1947	6	6	50.00
1st	27	CVE-2009-0781	6	6	50.00
1st	28	CVE-2007-0450	5	7	41.67
1st	29	CVE-2007-2449	6	12	33.33
1st	30	CVE-2009-2693	2	6	25.00
1st	31	CVE-2009-2902	0	1	0.00
2nd	1	CVE-2006-7197	6	0	100.00
2nd	2	CVE-2006-7196	6	0	100.00
2nd	3	CVE-2006-7195	6	0	100.00
2nd	4	CVE-2009-0033	6	0	100.00
2nd	5	CVE-2007-3386	6	0	100.00
2nd	6	CVE-2009-2901	3	0	100.00
2nd	7	CVE-2007-3385	6	0	100.00
2nd	8	CVE-2008-2938	6	0	100.00
2nd	9	CVE-2007-3382	6	0	100.00
2nd	10	CVE-2007-5461	6	0	100.00
2nd	11	CVE-2007-6286	6	0	100.00
2nd	12	CVE-2007-1858	6	0	100.00
2nd	13	CVE-2008-0128	6	0	100.00
2nd	14	CVE-2007-2450	6	0	100.00
2nd	15	CVE-2009-3548	6	0	100.00
2nd	16	CVE-2009-0580	6	0	100.00
2nd	17	CVE-2007-1355	6	0	100.00
2nd	18	CVE-2008-2370	6	0	100.00
2nd	19	CVE-2008-4308	6	0	100.00
2nd	20	CVE-2007-5342	6	0	100.00
2nd	21	CVE-2008-5515	19	5	79.17
2nd	22	CVE-2009-0783	12	3	80.00
2nd	23	CVE-2008-1232	13	5	72.22
2nd	24	CVE-2008-5519	12	0	100.00
2nd	25	CVE-2007-5333	6	6	50.00
2nd	26	CVE-2008-1947	6	6	50.00
2nd	27	CVE-2009-0781	12	0	100.00
2nd	28	CVE-2007-0450	7	5	58.33
2nd	29	CVE-2007-2449	8	10	44.44
2nd	30	CVE-2009-2693	4	4	50.00
2nd	31	CVE-2009-2902	0	1	0.00

Table 7: CWE Stats for Tomcat 5.5.13, version SATE.5

guess	run	algorithms	good	bad	%
1st	1	-cweid -nopreprep -raw -fft -cheb	27	6	81.82
1st	2	-cweid -nopreprep -raw -fft -diff	27	6	81.82
1st	3	-cweid -nopreprep -raw -fft -cos	24	9	72.73
1st	4	-cweid -nopreprep -raw -fft -eucl	13	20	39.39
1st	5	-cweid -nopreprep -raw -fft -hamming	12	21	36.36
1st	6	-cweid -nopreprep -raw -fft -mink	9	24	27.27
2nd	1	-cweid -nopreprep -raw -fft -cheb	32	1	96.97
2nd	2	-cweid -nopreprep -raw -fft -diff	32	1	96.97
2nd	3	-cweid -nopreprep -raw -fft -cos	29	4	87.88
2nd	4	-cweid -nopreprep -raw -fft -eucl	17	16	51.52
2nd	5	-cweid -nopreprep -raw -fft -hamming	18	15	54.55
2nd	6	-cweid -nopreprep -raw -fft -mink	13	20	39.39
guess	run	class	good	bad	%
1st	1	CWE-264	7	0	100.00
1st	2	CWE-255	6	0	100.00
1st	3	CWE-16	6	0	100.00
1st	4	CWE-119	6	0	100.00
1st	5	CWE-20	6	0	100.00
1st	6	CWE-200	22	4	84.62
1st	7	CWE-79	24	21	53.33
1st	8	CWE-22	35	61	36.46
2nd	1	CWE-264	7	0	100.00
2nd	2	CWE-255	6	0	100.00
2nd	3	CWE-16	6	0	100.00
2nd	4	CWE-119	6	0	100.00
2nd	5	CWE-20	6	0	100.00
2nd	6	CWE-200	23	3	88.46
2nd	7	CWE-79	30	15	66.67
2nd	8	CWE-22	57	39	59.38

Table 8: CVE NLP Stats for Tomcat 5.5.13, version SATE.5

guess	run	algorithms	good	bad	%
1st	1	-nopreprep -char -unigram -add-delta	29	4	87.88
2nd	1	-nopreprep -char -unigram -add-delta	29	4	87.88
guess	run	class	good	bad	%
1st	1	CVE-2006-7197	1	0	100.00
1st	2	CVE-2006-7196	1	0	100.00
1st	3	CVE-2009-2901	1	0	100.00
1st	4	CVE-2006-7195	1	0	100.00
1st	5	CVE-2009-0033	1	0	100.00
1st	6	CVE-2007-1355	1	0	100.00
1st	7	CVE-2007-5342	1	0	100.00
1st	8	CVE-2009-2693	1	0	100.00
1st	9	CVE-2009-0783	1	0	100.00
1st	10	CVE-2008-2370	1	0	100.00
1st	11	CVE-2007-2450	1	0	100.00
1st	12	CVE-2008-2938	1	0	100.00
1st	13	CVE-2007-2449	3	0	100.00
1st	14	CVE-2007-1858	1	0	100.00
1st	15	CVE-2008-4308	1	0	100.00
1st	16	CVE-2008-0128	1	0	100.00
1st	17	CVE-2009-3548	1	0	100.00
1st	18	CVE-2007-5461	1	0	100.00
1st	19	CVE-2007-3382	1	0	100.00
1st	20	CVE-2007-0450	2	0	100.00
1st	21	CVE-2009-0580	1	0	100.00
1st	22	CVE-2007-6286	1	0	100.00
1st	23	CVE-2008-5515	3	1	75.00
1st	24	CVE-2008-1232	1	2	33.33
1st	25	CVE-2009-2902	0	1	0.00
2nd	1	CVE-2006-7197	1	0	100.00
2nd	2	CVE-2006-7196	1	0	100.00
2nd	3	CVE-2009-2901	1	0	100.00
2nd	4	CVE-2006-7195	1	0	100.00
2nd	5	CVE-2009-0033	1	0	100.00
2nd	6	CVE-2007-1355	1	0	100.00
2nd	7	CVE-2007-5342	1	0	100.00
2nd	8	CVE-2009-2693	1	0	100.00
2nd	9	CVE-2009-0783	1	0	100.00
2nd	10	CVE-2008-2370	1	0	100.00
2nd	11	CVE-2007-2450	1	0	100.00
2nd	12	CVE-2008-2938	1	0	100.00
2nd	13	CVE-2007-2449	3	0	100.00
2nd	14	CVE-2007-1858	1	0	100.00
2nd	15	CVE-2008-4308	1	0	100.00
2nd	16	CVE-2008-0128	1	0	100.00
2nd	17	CVE-2009-3548	1	0	100.00
2nd	18	CVE-2007-5461	1	0	100.00
2nd	19	CVE-2007-3382	1	0	100.00
2nd	20	CVE-2007-0450	2	0	100.00
2nd	21	CVE-2009-0580	1	0	100.00
2nd	22	CVE-2007-6286	1	0	100.00
2nd	23	CVE-2008-5515	3	1	75.00
2nd	24	CVE-2008-1232	1	2	33.33
2nd	25	CVE-2009-2902	0	1	0.00

Table 9: CWE NLP Stats for Tomcat 5.5.13, version SATE.5

guess	run	algorithms	good	bad	%
1st	1	-cweid -nopreprep -char -unigram -add-delta	13	20	39.39
2nd	1	-cweid -nopreprep -char -unigram -add-delta	17	16	51.52
guess	run	class	good	bad	%
1st	1	CWE-16	1	0	100.00
1st	2	CWE-255	1	0	100.00
1st	3	CWE-264	2	0	100.00
1st	4	CWE-119	1	0	100.00
1st	5	CWE-20	1	0	100.00
1st	6	CWE-200	3	1	75.00
1st	7	CWE-22	3	13	18.75
1st	8	CWE-79	1	6	14.29
2nd	1	CWE-16	1	0	100.00
2nd	2	CWE-255	1	0	100.00
2nd	3	CWE-264	2	0	100.00
2nd	4	CWE-119	1	0	100.00
2nd	5	CWE-20	1	0	100.00
2nd	6	CWE-200	4	0	100.00
2nd	7	CWE-22	5	11	31.25
2nd	8	CWE-79	2	5	28.57

Table 10: CVE NLP Stats for Chrome 5.0.375.54, version SATE.7

guess	run	algorithms	good	bad	%
1st	1	-nopreprep -char -unigram -add-delta	9	0	100.00
2nd	1	-nopreprep -char -unigram -add-delta	9	0	100.00
guess	run	class	good	bad	%
1st	1	CVE-2010-2304	1	0	100.00
1st	2	CVE-2010-2298	1	0	100.00
1st	3	CVE-2010-2301	1	0	100.00
1st	4	CVE-2010-2295	2	0	100.00
1st	5	CVE-2010-2300	1	0	100.00
1st	6	CVE-2010-2303	1	0	100.00
1st	7	CVE-2010-2297	1	0	100.00
1st	8	CVE-2010-2299	1	0	100.00
2nd	1	CVE-2010-2304	1	0	100.00
2nd	2	CVE-2010-2298	1	0	100.00
2nd	3	CVE-2010-2301	1	0	100.00
2nd	4	CVE-2010-2295	2	0	100.00
2nd	5	CVE-2010-2300	1	0	100.00
2nd	6	CVE-2010-2303	1	0	100.00
2nd	7	CVE-2010-2297	1	0	100.00
2nd	8	CVE-2010-2299	1	0	100.00

Table 11: CWE NLP Stats for Chrome 5.0.375.54, version SATE.7

guess	run	algorithms	good	bad	%
1st	1	-cweid -nopreprep -char -unigram -add-delta	8	1	88.89
2nd	1	-cweid -nopreprep -char -unigram -add-delta	8	1	88.89
guess	run	class	good	bad	%
1st	1	CWE-399	1	0	100.00
1st	2	NVD-CWE-noinfo	1	0	100.00
1st	3	CWE-79	1	0	100.00
1st	4	NVD-CWE-Other	2	0	100.00
1st	5	CWE-119	1	0	100.00
1st	6	CWE-20	1	0	100.00
1st	7	CWE-94	1	1	50.00
2nd	1	CWE-399	1	0	100.00
2nd	2	NVD-CWE-noinfo	1	0	100.00
2nd	3	CWE-79	1	0	100.00
2nd	4	NVD-CWE-Other	2	0	100.00
2nd	5	CWE-119	1	0	100.00
2nd	6	CWE-20	1	0	100.00
2nd	7	CWE-94	1	1	50.00

Index

API

- DEFT2010App, 3
- marf.util.Matrix, 8
- WriterIdentApp, 3

C, 2, 3, 12, 18

C++, 2, 12, 18

Chrome

- 5.0.375.54, 2, 11, 12, 17, 24, 25, 29, 30
- 5.0.375.70, 2, 11

CVE

- CVE-2006-7195, 26, 28
- CVE-2006-7196, 26, 28
- CVE-2006-7197, 26, 28
- CVE-2007-0450, 26, 28
- CVE-2007-1355, 26, 28
- CVE-2007-1858, 26, 28
- CVE-2007-2449, 26, 28
- CVE-2007-2450, 26, 28
- CVE-2007-3382, 26, 28
- CVE-2007-3385, 26
- CVE-2007-3386, 26
- CVE-2007-5333, 26
- CVE-2007-5342, 26, 28
- CVE-2007-5461, 26, 28
- CVE-2007-6286, 26, 28
- CVE-2008-0128, 26, 28
- CVE-2008-1232, 26, 28
- CVE-2008-1947, 26
- CVE-2008-2370, 26, 28
- CVE-2008-2938, 26, 28
- CVE-2008-4308, 26, 28
- CVE-2008-5515, 26, 28
- CVE-2008-5519, 26
- CVE-2009-0033, 26, 28
- CVE-2009-0580, 26, 28
- CVE-2009-0781, 26
- CVE-2009-0783, 26, 28
- CVE-2009-2559, 21, 22
- CVE-2009-2560, 21, 22
- CVE-2009-2561, 21, 22
- CVE-2009-2562, 21, 22
- CVE-2009-2563, 21, 22
- CVE-2009-2693, 26, 28
- CVE-2009-2901, 26, 28

- CVE-2009-2902, 26, 28

- CVE-2009-3241, 21, 22

- CVE-2009-3242, 21, 22

- CVE-2009-3243, 21, 22

- CVE-2009-3548, 26, 28

- CVE-2009-3549, 21, 22

- CVE-2009-3550, 21, 22

- CVE-2009-3551, 21

- CVE-2009-3829, 21, 22

- CVE-2009-4376, 21, 22

- CVE-2009-4377, 21, 22

- CVE-2009-4378, 21, 22

- CVE-2010-0304, 21, 22

- CVE-2010-1455, 21, 22

- CVE-2010-2283, 21, 22

- CVE-2010-2284, 21, 22

- CVE-2010-2285, 21, 22

- CVE-2010-2286, 21, 22

- CVE-2010-2287, 21

- CVE-2010-2295, 24, 29

- CVE-2010-2297, 24, 29

- CVE-2010-2298, 24, 29

- CVE-2010-2299, 24, 29

- CVE-2010-2300, 24, 29

- CVE-2010-2301, 24, 29

- CVE-2010-2302, 24

- CVE-2010-2303, 24, 29

- CVE-2010-2304, 24, 29

CWE

- CWE-119, 23, 25, 27, 29, 30

- CWE-16, 27, 29

- CWE-189, 23

- CWE-20, 23, 25, 27, 29, 30

- CWE-200, 27, 29

- CWE-22, 27, 29

- CWE-255, 27, 29

- CWE-264, 27, 29

- CWE-399, 23, 25, 30

- CWE-79, 25, 27, 29, 30

- CWE-94, 25, 30

- NVD-CWE-noinfo, 23, 25, 30

- NVD-CWE-Other, 23, 25, 30

Dovecot, 3, 15

Files

report-cweidnoprepreprawfftcheb-apache-tomcat-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.13-train-cwe.xml, 13	5.5.13-train-cve.xml, 12	5.5.13-train-cve.xml, 12	5.5.13-train-cve.xml, 12
report-cweidnoprepreprawfftcos-apache-tomcat-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.13-train-cwe.xml, 13	test.xml, 11	test.xml, 11	test.xml, 11
report-cweidnoprepreprawfftcos-apache-tomcat-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.29-test-cwe.xml, 16	1.2.9-test.xml, 10	1.2.9-test.xml, 10	1.2.9-test.xml, 10
report-cweidnoprepreprawfftcos-dovecot-2.0.beta6-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
wireshark-test-cwe.xml, 15	5.5.13-train-cve.xml, 12	5.5.13-train-cve.xml, 12	5.5.13-train-cve.xml, 12
report-cweidnoprepreprawfftdiff-apache-tomcat-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.13-train-cwe.xml, 13	test.xml, 11	test.xml, 11	test.xml, 11
report-cweidnoprepreprawffteucl-apache-tomcat-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.13-train-cwe.xml, 13	test.xml, 10	test.xml, 10	test.xml, 10
report-cweidnoprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.13-train-cwe.xml, 13	sate_2010.pathcheck.xsd, 10, 11	sate_2010.pathcheck.xsd, 10, 11	sate_2010.pathcheck.xsd, 10, 11
report-cweidnoprepreprawfftmink-apache-tomcat-report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft	report-noprepreprawfft
5.5.13-train-cwe.xml, 13	sate_2010.xsd, 10, 11	sate_2010.xsd, 10, 11	sate_2010.xsd, 10, 11
report-nopreprepcharunigramadddelta-apache-tomcat-5.5.13-train-cve-nlp.xml, 13	Java, 2, 3, 14, 18	Java, 2, 3, 14, 18	Java, 2, 3, 14, 18
report-noprepreprawfftcheb-apache-tomcat-5.5.13-train-cve.xml, 12	Libraries	Libraries	Libraries
report-noprepreprawfftcheb-chrome-5.0.375.54-train.xml, 11	MARF, 1–3, 5, 8, 10, 14, 17, 20	MARF, 1–3, 5, 8, 10, 14, 17, 20	MARF, 1–3, 5, 8, 10, 14, 17, 20
report-noprepreprawfftcheb-chrome-5.0.375.70-test.xml, 11	MARF, 1–3, 5, 8, 10, 14, 17, 20	MARF, 1–3, 5, 8, 10, 14, 17, 20	MARF, 1–3, 5, 8, 10, 14, 17, 20
report-noprepreprawfftcheb-wireshark-1.2.0-train.xml, 10	Applications	Applications	Applications
report-noprepreprawfftcheb-wireshark-1.2.9-test.xml, 10	MARFCAT, 1–6, 8, 12–14, 17, 18	MARFCAT, 1–6, 8, 12–14, 17, 18	MARFCAT, 1–6, 8, 12–14, 17, 18
report-noprepreprawfftcos-apache-tomcat-5.5.13-train-cve.xml, 12	Options	Options	Options
report-noprepreprawfftcos-apache-tomcat-5.5.29-test-cve.xml, 16	-nlp, 13	-nlp, 13	-nlp, 13
report-noprepreprawfftdiff-apache-tomcat-5.5.13-train-cve.xml, 12	-char, 22, 28–30	-char, 22, 28–30	-char, 22, 28–30
report-noprepreprawfftdiff-apache-tomcat-5.5.13-train-cve.xml, 12	-cheb, 9, 12, 21, 23–27	-cheb, 9, 12, 21, 23–27	-cheb, 9, 12, 21, 23–27
report-noprepreprawfftdiff-chrome-5.0.375.54-train.xml, 11	-cos, 12, 15, 21, 23–27	-cos, 12, 15, 21, 23–27	-cos, 12, 15, 21, 23–27
report-noprepreprawfftdiff-chrome-5.0.375.70-test.xml, 11	-cweid, 23, 25, 27, 29, 30	-cweid, 23, 25, 27, 29, 30	-cweid, 23, 25, 27, 29, 30
report-noprepreprawfftdiff-wireshark-1.2.0-train.xml, 10	-diff, 9, 12, 21, 23–27	-diff, 9, 12, 21, 23–27	-diff, 9, 12, 21, 23–27
report-noprepreprawfftdiff-wireshark-1.2.9-test.xml, 10	-eucl, 21, 23–27	-eucl, 21, 23–27	-eucl, 21, 23–27
report-noprepreprawffteucl-apache-tomcat-5.5.13-train-cve.xml, 12	-fft, 21, 23–27	-fft, 21, 23–27	-fft, 21, 23–27
report-noprepreprawffteucl-chrome-5.0.375.70-test.xml, 11	-hamming, 21, 23–27	-hamming, 21, 23–27	-hamming, 21, 23–27
report-noprepreprawffteucl-wireshark-1.2.9-test.xml, 10	-mink, 10, 21, 23–27	-mink, 10, 21, 23–27	-mink, 10, 21, 23–27
	-nopreprep, 21–30	-nopreprep, 21–30	-nopreprep, 21–30
	-raw, 21, 23–27	-raw, 21, 23–27	-raw, 21, 23–27
	-unigram, 22, 28–30	-unigram, 22, 28–30	-unigram, 22, 28–30
	Pebble, 3, 5, 13–15	Pebble, 3, 5, 13–15	Pebble, 3, 5, 13–15
	Test cases	Test cases	Test cases
	Chome 5.0.375.54, 2, 11, 12, 17, 24, 25, 29, 30	Chome 5.0.375.54, 2, 11, 12, 17, 24, 25, 29, 30	Chome 5.0.375.54, 2, 11, 12, 17, 24, 25, 29, 30
	Chome 5.0.375.70, 2, 11	Chome 5.0.375.70, 2, 11	Chome 5.0.375.70, 2, 11
	Dovecot 2.0.beta6.20100626, 3, 15	Dovecot 2.0.beta6.20100626, 3, 15	Dovecot 2.0.beta6.20100626, 3, 15

Pebble 2.5-M2, 3, 5, 13–15

Tomcat 5.5.13, 2, 5, 12, 14–16, 26–29

Tomcat 5.5.29, 2, 5, 15, 16

Wireshark 1.2.0, 2, 9, 15, 21–23

Wireshark 1.2.9, 2, 10

Tomcat

5.5.13, 2, 5, 12, 14–16, 26–29

5.5.29, 2, 5, 15, 16

Tools

diff, 7

wc, 6

Wireshark

1.2.0, 2, 9, 15, 21–23

1.2.9, 2, 10