Simseer and Bugwise - Web Services for Binary-level Software Similarity and Defect Detection

Silvio Cesare and Yang Xiang
School of Information Technology
Deakin University
Burwood, Victoria 3125, Australia

{scesare, yang}@deakin.edu.au

Abstract

Simseer and Bugwise are online web services that perform binary program analysis: 1) Simseer identifies similarity between submitted executables based on similarity in the control flow of each binary. A software similarity service provides benefit in identifying malware variants and families, discovering software theft, and revealing plagiarism of software programs. Simseer additionally performs code packing detection and automated unpacking of hidden code using application-level emulation. Finally, Simseer uses the similarity information from a sample set to identify program relationships and families through visualization of an evolutionary tree. 2) Bugwise is a service that identifies software bugs and defects. To achieve this end, it performs decompilation and data flow analysis. Bugwise can identify a subset of use-after-free bugs and has already found defects in Debian Linux. Bugwise and Simseer are both built on Malwise, a platform of binary analysis.

Keywords: computer security, software similarity, software theft detection, plagiarism detection, bug detection, could computing.

1 Introduction

Software similarity is an important topic with a number of applications. It can be used in the areas of malware detection, software theft detection and plagiarism detection. These are the applications for which Simseer was designed to address.

Software similarity analysis is built upon a platform of program analysis that performs the relevant aspects of feature extraction. This process of software analysis can be used not only for software similarity tasks, but also to detect software bugs and defects.

Defect Detection is the problem of finding software bugs. Examples of bugs that defect detection can identify are buffer overflows, divide-by-zeros, and dynamic memory management problems such as use-after-frees.

Malware variant detection is the problem of identifying malware that is a replicated, obfuscated, or an evolved copy of a known malicious sample. Malware variant detection can be used to attribute a sample to a particular author or family of malware. Malware variant detection is the problem of identifying similarity between known malware and unknown programs.

Software theft detection identifies the unauthorized duplication or copying of software. The purpose of this area is to have automated ways to discover or verify copyright infringement of software or intellectual property. Software theft detection is the problem of identifying unauthorized similar software.

Plagiarism detection detects student cheating in assignments where the submission is a piece of software. Students copying each other's work can be broken down into the problem of identifying similar copies of software in the students' submissions

1.1 Motivation

Defect Detection can reduce the cost of maintaining software by identifying problems during quality and assurance testing and not after the public software release is made. Identifying software defects that impact on the security of software means that producers of software can stay ahead of attackers who actively try to discover these defects themselves. Bug detection in binaries is important to external auditors who need to validate the security of software they are given. Binary auditing is also important to verify the compiler and linker are working properly without introducing new defects.

Malware detection is an important problem on the internet today. According to the Symantec Internet Threat Report (Symantec 2008), 499,811 new malware samples were received in the second half of 2007. The same vendor reported over 1.5 billion malicious code detections in 2010 (Symantec 2011). F-Secure published, "As much malware [was] produced in 2007 as in the previous 20 years altogether" (F-Secure 2007). This growth continues today.

Malware variant detection can be used to enhance the traditional approach of signature based malware detection by providing more predictive power to those signatures. Most malware today is a variant of existing malware, so identifying variants is effective in detecting a significant amount of malicious code that traditional approaches fail to identify.

Software theft is a problem with significant consequences. In 2005, a federal court determined that the independent software vendor Compuserve be paid $140 million by IBM to license its software or $260 million to purchase its services because it was discovered that IBM products had illegitimately used code from

Compuware without authorization (Wang et al. 2009). Software theft detection is an important area that helps protect the high worth of intellectual property.

Plagiarism detection is important to maintain integrity in educational environments. If students believe they will be caught if they cheat then they are unlikely to proceed with that unethical practice. If educators receive a high number of assignment submissions then it may be hard to recognize that cheating has occurred, so automated methods are an important tool.

1.2 Innovation

Simseer is a tool that can detect similar software and identify malware variants, discover software theft, and reveal plagiarism. Bugwise can detect some classes of software defects in binaries. The contributions of this paper are as follows:

- We propose an online web service to address the issues of malware variant detection, software theft detection, and plagiarism detection.
- We propose an online web service to address the issue of closed source software defect analysis.
- We use state-of-the-art algorithms in our novel service.
- We implement and make public our services.

1.3 Structure of the Paper

The structure of this paper is as follows: Section 2 examines related work in software similarity and bug detection. Section 3 describes a high level overview of our aims and approach. Section 4 discusses the design and implementation of our system as a cloud service. Section 5 evaluates different aspects of our system. Section 6 gives details on how to access our service. Section 7 looks at future work. Finally, Section 8 concludes the paper.

2 Related Work

Detecting defects in software has a long history in formal methods. Data flow analysis is used by compilers (Aho, Sethi & Ullman 1986) and is what Bugwise uses to perform binary analysis. Abstract interpretation, which formalizes data flow analysis was introduced in (Cousot & Cousot 1977). Theorem proving has been used to prove the absence of bugs (Dijkstra 1975; Hoare 1969). Satisfiability over Modulo Theories (SMT) extends SAT and has been used to perform bug detection (Cadar et al. 2008; Molnar & Wagner 2007) and symbolic execution (King 1976). Decompilation has been used to analyse binary programs in the past including work in (Cifuentes 1994; Van Emmerik 2007) which used compilation techniques to aid the decompilation process.

The areas relating to software similarity are malware variant detection, software theft detection, plagiarism detection, and code clone detection. A unified approach to the software similarity problem is to divide the problem into feature extraction to construct fingerprints, known as birthmarks, and then to calculate birthmark similarity using mathematical distance and similarity functions. Birthmarks can be considered as strings, vectors, sets, trees, graphs and other objects.

In malware variant detection, raw code has been used to construct string based signatures, which is common in Antivirus software (Griffin et al. 2009; Kephart & Arnold 1994). Kolmogorov complexity of raw code has been used in (Wicherski 2009). The Normalized Compression Distance was used in (Wehner 2007). Opcode distributions are another feature used in (Bilar 2007). N-grams were used on instructions in (Karim et al. 2005) and evolutionary trees were constructed. Static and dynamic API call features were used in (Ye et al. 2007) and (Kolbitsch et al. 2009) respectively. Control flow and data flow were used as a feature in (Christodorescu & Jha 2003; Christodorescu et al. 2005).

Control flow is the approach that Simseer uses to construct birthmarks. Interprocedural control flow was proposed as a feature in (Briones & Gomez 2008; Carrera & Erdélyi 2004; Dullien & Rolles 2005; Gerald & Lori 2007; Hu, Chiueh & Shin) . Simseer uses intraprocedural control flow of a program's procedures and similar techniques have been applied in (Cesare & Xiang 2010b) (Cesare & Xiang 2010a) (Bonfante, Kaczmarek & Marion 2008) (Kruegel et al. 2006).

In software theft detection, similar techniques have been used. Instruction sequences were used in (Park et al. 2008). K-grams of instruction sequences were used in (Myles & Collberg 2005). Control flow was used in (Lim et al. 2009a, 2009b) and static API calls used in (Choi et al. 2008, 2009).

In plagiarism detection systems such as JPlag (Prechelt, Malpohl & Philippsen 2002) and YAP3 (Wise 1996) have used the text of raw source code as a feature. Parse trees were used in (Son, Park & Park 2006) allowing tree based distances to calculate similarity. Program Dependence Graphs (PDGs) were used in (Liu et al. 2006).

Code clone techniques are based on the software similarity problem. It is the problem of identifying duplicate or similar fragments of code in a piece of software. Approaches have included using raw source code as a birthmark in (Ducasse, Rieger & Demeyer 1999) and for large scale applications in (Kamiya, Kusumoto & Inoue 2002; Livieri et al. 2007). Abstract Syntax Trees (ASTs) were used in (Baxter et al. 1998). PDGs were proposed in (Krinke 2001).

Birthmark similarity is the next step after feature extraction and birthmark creation. Distance metrics for strings, vectors, sets, trees, and graphs exist. For strings, the Levenshtein distance is the minimum number of insertions, deletions, and substitutions to transform one string to another. Sequence alignment is often used in bioinformatics including the optimal local sequence alignment, known as the Smith-Waterman algorithm. Vector distance metrics include the Manhattan distance or the classic Euclidean distance. Cosine similarity is a popular vector similarity measure. Set similarity includes the Dice Coefficient and the Jaccard Index. Trees and graphs have edit distances to describe the number of basic operations to transform one object to another. Maximum common subtrees or subgraphs are other measures used to identify similarity and distance.

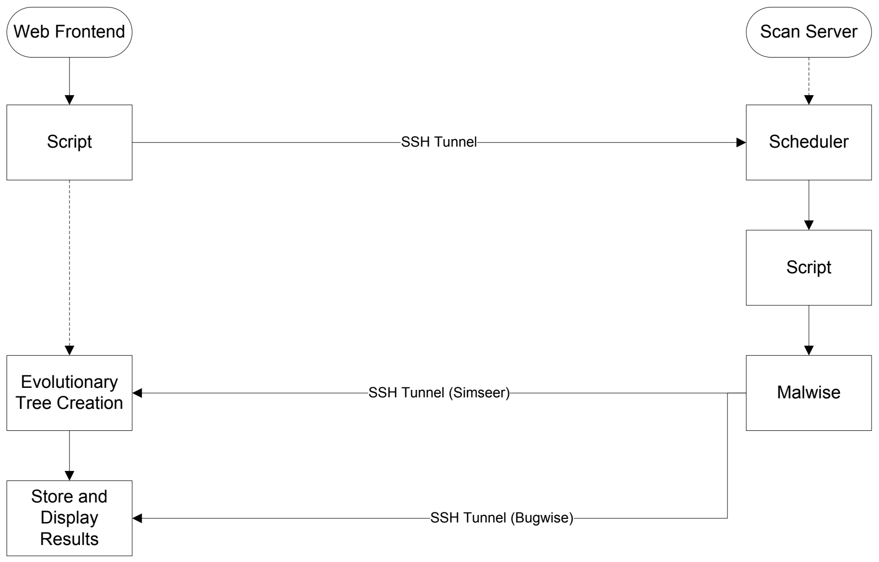


Figure 1. Web services work flow.

3 Our Approach

The aims of this work are to provide a web service to score and visualize similarity between executable binaries, and also to provide a web service to detect software defects in binaries. To perform software similarity scoring and defect detection, we employed some of our previous work, Malwise, to do the backend processing.

Malwise performs software similarity scoring by using control flow within a binary as a signature. Control flow is considered invariant under common program transformations and is effective at detecting program variants.

Malwise can also perform general static analysis of binaries. It does this by disassembling the binary, translating the disassembly to an intermediate language, and then performing decompilation, data flow, and other analyses. Data flow analysis combined with decompilation is capable at detecting some defects from some classes of bugs in binaries.

4 System Design and Implementation

The system uses two Virtual Private Servers (VPS) in the cloud and could potentially be scaled into larger server farms. One server is the web frontend and one server is the scan server. The servers have 1GB of memory each. The workflow for the web service involving all components is shown in Fig. 1. The user interface is a submission system that returns a results page.

4.1 The Web Frontend

Both Simseer and Bugwise are accessed by web frontends. Bugwise is almost functionally equivalent in its processing, so Simseer will be explained in depth.

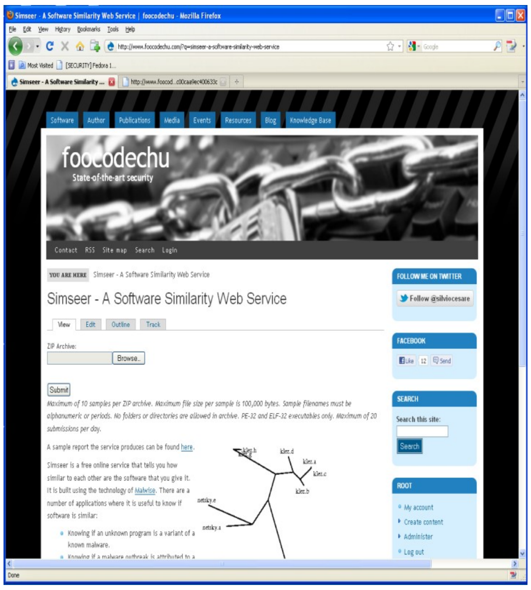
The web frontend is the user interface to the Simseer cloud service and the landing page and the final result is shown in Fig. 2. And Fig. 3. A user of the service can submit a ZIP archive of executables which are subsequently transferred to and processed by the server. Our implementation is coded in the server side PHP programming language. The PHP code is responsible for rate limiting the number of submission requests per IP address by maintaining a record of submissions to the system in a MySQL database.

The PHP code launches a shell script which takes over handling of the archive submission. The script checks that the ZIP archive is valid, does not contain an excessive number of samples, does not contain symbolic links as archive members, and does not contain archive member names using special characters.

The system logs that a submission to the system has been made and makes a copy of the submission content into storage. The script launches a C++ compiled program that acts as a client in a client-service protocol with the scan server. The protocol enables transmission of files that will be processed by the scan server. Communication with the scan server is performed over an SSH port forwarded tunnel which allows security in the client-server protocol.

4.2 The Scheduling Work Queue

The scan server listens locally on a TCP port which is connected via an SSH tunnel back to the web frontend. The C++ implemented server component launches the Malwise backend to process files received. However, scheduling must occur so that the server does not consume excessive resources. Thus receipt and processing of files is queued so that only 1 job is active at any given time. The number of parallel jobs can be arbitrary, however due to the single core nature of our Virtual Private Server (VPS) scan server, running jobs in parallel does not result in an increase in performance. Additionally, running multiple jobs in parallel places more restrictions on memory usage per instance which we wanted to avoid. Once a job has been scheduled and the ZIP archive or binary received from the web frontend host, a script is launched to process the file and launch the



YOU ARE HERE Simseer - A Software Similarity Web Service

Simseer - A Software Similarity Web Service

ZIP Archive:

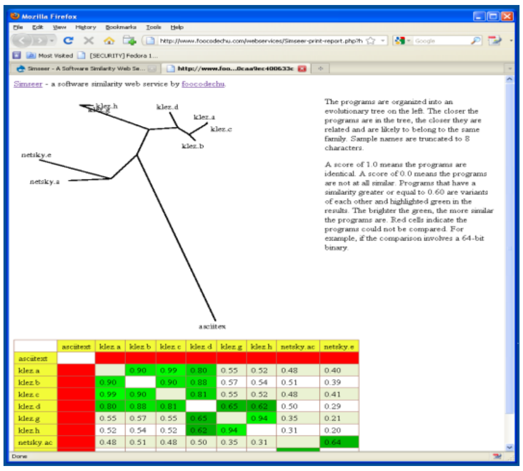
Maximum of 10 samples per ZIP archive. Maximum file size per sample is 100,000 bytes. Sample filenames must be alphanumeric or periods. No folders or directories are allowed in archive. PE-32 and ELF-32 executables only. Maximum of 20 submissions per day.

A sample report the service produces can be found here.

Simseer is a free online service that tells you how similar to each other are the software that you give it. It is built using the technology of Malwise. There are a number of applications where it is useful to know if software is similar:

- Knowing if an unknown program is a variant of a known malware.
- Knowing if a malware outbreak is attributed to a

Figure 2. The Simseer landing page.


Mozilla Firefox
http://www.foocodechu.com/webservices/Simseer-print-report.php?h
Simseer - a software similarity web service by foocodechu
klez.h
klez.g
klez.d
klez.a
klez.c
klez.b
netsky.e
netsky.a
asciitex
Done

The programs are organized into an evolutionary tree on the left. The closer the programs are in the tree, the closer they are related and are likely to belong to the same family. Sample names are truncated to 8 characters.

A score of 1.0 means the programs are identical. A score of 0.0 means the programs are not at all similar. Programs that have a similarity greater or equal to 0.60 are variants of each other and highlighted green in the results. The brighter the green, the more similar the programs are. Red cells indicate the programs could not be compared. For example, if the comparison involves a 64-bit binary.

	asciitext	klez.a	klez.b	klez.c	klez.d	klez.g	klez.h	netsky.ac	netsky.e
asciitext		🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
klez.a	🟥		0.90	0.99	0.80	0.55	0.52	0.48	0.40
klez.b	🟥	0.90		0.90	0.88	0.57	0.54	0.51	0.39
klez.c	🟥	0.99	0.90		0.81	0.55	0.52	0.48	0.41
klez.d	🟥	0.80	0.88	0.81		0.65	0.62	0.50	0.29
klez.g	🟥	0.55	0.57	0.55	0.65		0.94	0.35	0.21
klez.h	🟥	0.52	0.54	0.52	0.62	0.94		0.31	0.20
netsky.ac	🟥	0.48	0.51	0.48	0.50	0.35	0.31		0.64

Figure 3. Simseer results.

Malwise system. For Simseer, the script unpacks the archive ready for Malwise to process. For Bugwise, the binary is passed on directly.

4.3 Malwise Backend

Simseer and Bugwise both use the Malwise backend. The difference between Simseer and Bugwise is the module list that Malwise uses. The Malwise backend is coded in C++ and consists of 100,000 Lines of Code (LOC). Malwise is launched as a standalone program from the scheduler launched script. It is possible to use Malwise as a daemon and avoid the cost of repeated program loading when submitting jobs. However, the reliability of the system as a whole is increased when we launch Malwise as a standalone program for each job because if a scan then causes a crash it is contained to an individual job. If Malwise was run as a daemon and allowed jobs to be queued then all jobs would be lost if the program failed. Even though we launch jobs separately, the service allows for scalability because jobs could potentially be launched on server farms behind the interface. Likewise,

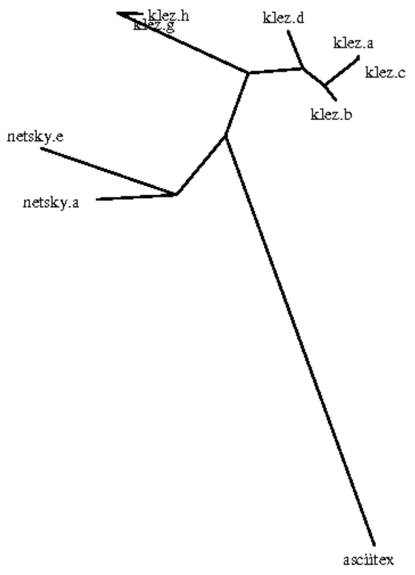


Figure 4. Program relationship visualization.

the system still maintains a global view of jobs being launched - it stores copies of the binaries submitted to the service. This allows us to perform offline analysis and correlation to determine if novel samples are being submitted to the service or if known samples are the primary source of submissions.

The backend is modular and allows for loading of modules at program startup defined by an XML configuration file. A sample of the differences between the configuration for Simseer and Bugwise is shown in Fig. 8 and Fig. 9. Malwise returns its results in XML. This XML is transferred back across the SSH tunnel to the client on the web frontend host where it is stored for processing.

4.3.1 Simseer

The modules we have deployed to implement Simseer are:

- Packer Detection using Entropy Analysis
- Automated Unpacking using Application-level Emulation.
- Control Flow Decompilation
- Software Similarity Detection using Q-Grams of Decompiled Control Flow Graphs

The automated unpacker is a module to remove obfuscations and encryptions by revealing the hidden code (Cesare & Xiang 2010a). Packing is common in most malware. To deploy the automated unpacker we needed to make available the Windows system libraries. The reason for this is that the emulator requires libraries to implement dynamic linking of the emulated guest programs.

We used two types of configurations to Malwise with the above module list. In the first configuration, processing executables creates a signature for the software similarity detection. In the second configuration, the signature database is assumed to be already filled.

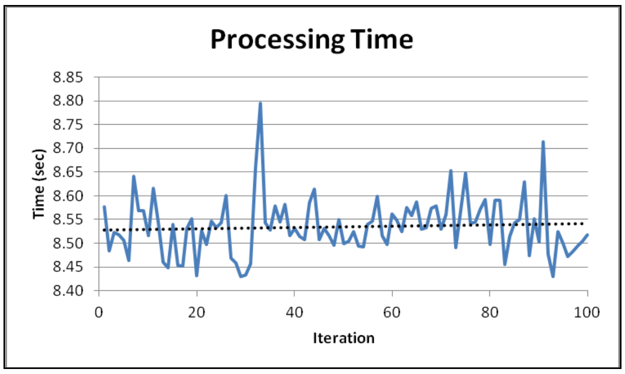


Figure 5. Simseer processing time.

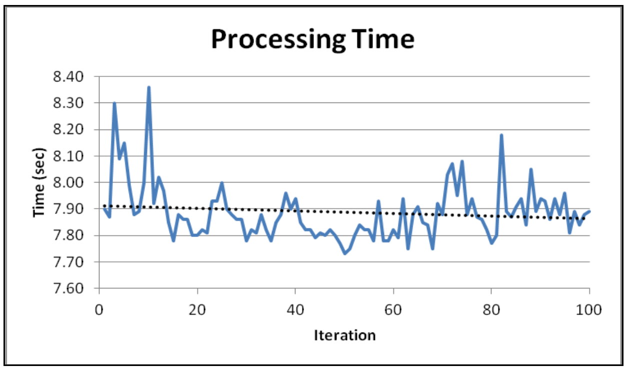


Figure 6. Malwise processing time.

Thus Simseer is split into two phases – signature database creation and software similarity detection. The script handling the launching of Malwise calls Malwise once for each phase, and therefore two times in total.

4.3.2 Bugwise

The modules we have deployed to implement Bugwise are:

- Intermediate Language Optimisation
- Decompilation Modules
- Linux
- Data Flow Analysis
- Double-free Detection

The intermediate language optimisations are a set of compiler style optimisations that operates over the intermediate language Malwise uses to represent x86 assembly code. The optimisations that are implemented are:

- Dead Code Elimination
- Copy propagation
- Constant folding
- Constant propagation

The decompilation modules translate stack based local variables to native variables in the intermediate language. This allows the data flow analysis to identify problems such as use-after-frees and double-frees.

A Linux specific module is used to identify the beginning of the main() function via the __libc_start_main library call.

The data flow analysis module enables a variety of analyses such as:

- Reaching Definitions
- Upwards Exposed Uses

klez.a
klez.b
klez.c
klez.d
klez.g
klez.h
netsky.aa
netsky.e
asciitext

Figure 7. Simseer samples.

```
<ModuleGroup>
    <Name>Scan</Name>
    <Run>Packer Detection Using Entropy</Run>
    <Run>Unpacker Using Application Level Emulation</Run>
    <Run>Structuring</Run>
    <Run>NGram Structuring</Run>
</ModuleGroup>
```

Figure 8. Simseer configuration.

```
<ModuleGroup>
    <Name>Scan</Name>
    <Run>Code Optimsation 1</Run>
    <Run>Linux Arch</Run>
    <Run>Pre Decompiler Data Flow Analysis</Run>
    <Run>X86 Decompiler Data Flow Analysis</Run>
    <Run>Decompiler Data Flow Analysis</Run>
    <Run>Code Optimsation 2</Run>
    <Run>IRDataFlowAnalysis</Run>
    <Run>Double Free Detection</Run>
</ModuleGroup>
```

Figure 9. Bugwise configuration.

- Reaching Copies

Finally, the double-free detection module uses the data flow analysis to discover use of the free pointer after a free() without a reassignment of the pointer. In practice, Bugwise has found software defects in Debian Linux given only access to the binary executables.

4.4 Simseer Evolutionary Tree Visualization

A phylogenetic or evolutionary tree is a visual representation of the evolutionary relationships between species based on similarity between features or characteristics. Species closer to the tree in relation to the number of branches or branch lengths are more closely related. Simseer uses evolutionary trees to visualize the relationships between programs and their variants. This visualization is useful because program variants are typically derivatives and modified versions of their upstream source.

The web frontend host is responsible for processing the XML results returned by Malwise. One of the responsibilities of the script launched on the web host is to create and render an evolutionary tree of the submissions. The XML returned by Malwise scores the similarity between each sample. The script transforms the XML into a distance matrix. Distance is calculated as 1 –

similarity. This distance matrix is then analysed to create an evolutionary tree using the PHYLIP software package (Felsenstein 2005). The PHYLIP package uses the neighbour joining method (Saitou & Nei 1987) to construct an evolutionary tree. The evolutionary tree is described by the Newick tree format which gives such information as branch lengths in the tree. The Newick tree file is processed to render a figure suitable for display. The figure is then transformed to a PNG image and stored on the web host. An example of the tree visualization is shown in Fig. 4.

4.5 Results Processing

The results shown to the user are different depending on whether Simseer or Bugwise is being used.

4.5.1 Simseer

To display the results, the Malwise XML similarity results are displayed as an HTML table. The background colour of the table cells are proportional to how similar the samples are. The lighter the colour, the more similar the programs are. If the programs are not variants of each other, the table cell is left unshaded. The evolutionary tree image of the programs is shown on the same page. The results processing is performed after submitting an archive to the system and may also be accessed at a later time. Later viewing of the results is achieved by accessing a PHP page to reprocess the Malwise XML results and displaying the permanently stored evolutionary tree image. To specify which archive is requested to be processed, an MD5 digest of the ZIP archive is passed as a parameter to the web page using the GET HTTP method.

4.5.2 Bugwise

Bugwise lists the double frees detection in a HTML table. The double free detector returns the address of the code in the disassembly for both frees that are involved in the bug. To be able to use the results effectively, an analyst must be familiar with reverse engineering. For people performing binary analysis without source this skill is expected.

5 Efficiency of Malwise as a Web Service

We performed an evaluation of the time it takes to process 9 samples using the Simseer web service. We did this by writing a Python script to submit the samples to the web service over HTTP and read the results. The samples we used to perform this test are shown in Fig. 9. Eight samples were malware and 1 sample was some ASCII text which should not be found similar to any of the executables. We submitted the 9 samples as a ZIP archive to a local machine running the Simseer web service. We performed this test 100 times. A mean time of 8.53 seconds was recorded with a standard deviation of 0.06 seconds. The results are shown graphically in Fig. 5.

We performed a similar evaluation on the samples, but this time we ran the tests by command line and without performing the program visualization using evolutionary trees. This test gives us a base line for Malwise, upon which Simseer is based. The comparison between Malwise (Fig. 9) and Simseer (Fig. 8) demonstrates how effective the web service is (Fig. 8) when compared to using the system without the web interface (Fig. 9). The mean processing time for 100 iterations was 7.89 seconds with a standard deviation of 0.11 seconds. The results are shown graphically in Fig. 6.

The overhead of Simseer as a web service, excluding varied upload times of different speed networks, is 0.64 seconds. These results show that providing Simseer as a web service is efficient and does not add significant overhead to Malwise.

We take the previous results into account when considering Bugwise. Bugwise is much slower than Simseer due to the data flow analysis that is required for bug detection. We see no significant overhead in launching Bugwise since it uses the same web frontend and scheduling code as Simseer.

6 Availability

The Simseer service is free to use. It can be accessed on the web at http://www.foocodechu.com/?q=simseer-a-software-similarity-web-service. The Bugwise service is also free to use and can be accessed on the web at http://www.foocodechu.com/?q=node/19. We have implemented rate limiting to restrict the number of scans per day per IP address. We have also limited the number of samples that can be submitted per ZIP archive to the Simseer, and limited the size of the binary that can be submitted to the Bugwise service. As the service grows, we may relax some of these constraints.

7 Future Work

One thing we would like to do is replace our custom scheduling work queue with an enterprise messaging system such as RabbitMQ. Enterprise-level messaging systems have guarantees on reliabilities in the case of transmission or network failures. Using such a system would improve our reliability. Enterprise messaging also leads to an easy solution to distributed scan servers as we can have a single producer of messages on the web front end, and consumers in multiple scan servers.

We would also like to implement more flexibility in which modules are used in launching Simseer and Bugwise. Malwise has many modules available, and multiple options are available for software similarity scoring and defect detection.

Another possibility is using any-time clustering on the stream of samples that are given to Simseer. In this approach, cluster analysis is performed incrementally as objects are given to the system sequentially. An any-time phylogenetic tree analysis could follow on from any-time clustering. Any-time clustering could provide intelligence into new families of malware that are given to Simseer. This could benefit analysts in determining if a new sample relates to an existing family is something never seen before or relatively new.

Bugwise could be extended by treating bug detection instead as bug management. An automated bug reporting system could be used to submit, remove, and verify bugs that it discovers. This type of approach has been used successfully in network vulnerability management and we think that there exists many parallels.

8 Conclusion

In this paper we have demonstrated novel services to 1) score and visualize the software similarity of executable binary programs 2) detect software defects in binaries. The Simseer and Bugwise services are deployed as cloud services and are free to use. Simseer can be used to identify malware variants, detect software theft, and reveal plagiarism of software programs. Bugwise has already found real defects in Debian Linux. Simseer and Bugwise are built as a modular extension to our Malwise binary analysis platform. It demonstrates the versatility of our system that we can launch both services using only slightly different parameters with separate configurations. We performed an evaluation on the overhead incurred by making our Malwise platform using web services. We found that such an overhead was minimal and not significant. We are the first to make a public service that analyses executable binaries in these contexts and see the area of cloud based software analysis and similarity detection as having future growth.

9 References

Aho, AV, Sethi, R & Ullman, JD 1986, *Compilers: principles, techniques, and tools*, Addison-Wesley, Reading, MA.

Baxter, ID, Yahin, A, Moura, L, Sant'Anna, M & Bier, L 1998, 'Clone detection using abstract syntax trees', in p. 368.

Bilar, D 2007, 'Opcodes as predictor for malware', *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156-68.

Bonfante, G, Kaczmarek, M & Marion, JY 2008, 'Morphological Detection of Malware', in *International Conference on Malicious and Unwanted Software, IEEE*, Alexendria VA, USA, pp. 1-8.

Briones, I & Gomez, A 2008, 'Graphs, Entropy and Grid Computing: Automatic Comparison of Malware', in *Virus Bulletin Conference*, pp. 1-12.

Cadar, C, Ganesh, V, Pawlowski, PM, Dill, DL & Engler, DR 2008, 'EXE: automatically generating inputs of death', *ACM Transactions on Information and System Security TISSEC (2008)*, vol. 12, no. 2, pp. 10:1-:38.

Carrera, E & Erdélyi, G 2004, 'Digital genome mapping–advanced binary malware analysis', in *Virus Bulletin Conference*, pp. 187-97.

Cesare, S & Xiang, Y 2010a, 'Classification of Malware Using Structured Control Flow', in *8th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010)*.

Cesare, S & Xiang, Y 2010b, 'A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost', in *IEEE 24th International Conference on Advanced Information Networking and Application (AINA 2010)*.

Choi, S, Park, H, Lim, H & Han, T 2008, 'A static birthmark of binary executables based on API call structure', *Advances in Computer Science–ASIAN 2007. Computer and Network Security*, pp. 2-16.

Choi, S, Park, H, Lim, H & Han, T 2009, 'A static API birthmark for Windows binary executables', *Journal of Systems and Software*, vol. 82, no. 5, pp. 862-73.

Christodorescu, M & Jha, S 2003, 'Static analysis of executables to detect malicious patterns', paper presented to Proceedings of the 12th USENIX Security Symposium.

Christodorescu, M, Jha, S, Seshia, SA, Song, D & Bryant, RE 2005, 'Semantics-aware malware detection', in *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P 2005)*, Oakland, California, USA.

Cifuentes, C 1994, 'Reverse compilation techniques', Queensland University of Technology.

Cousot, P & Cousot, R 1977, 'Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints', in *Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, California, pp. 238-52.

Dijkstra, EW 1975, 'Guarded commands, nondeterminacy and formal derivation of programs', *Communications of the ACM*, vol. 18, no. 8, pp. 453-7.

Ducasse, S, Rieger, M & Demeyer, S 1999, 'A language independent approach for detecting duplicated code', in p. 109.

Dullien, T & Rolles, R 2005, 'Graph-based comparison of Executable Objects (English Version)', in *SSTIC*.

F-Secure 2007, 'F-Secure Reports Amount of Malware Grew by 100% during 2007', retrieved 19 August 2009, <http://www.f-secure.com/en_EMEA/about-us/pressroom/news/2007/fs_news_20071204_1_eng.html>.

Felsenstein, J 2005, *PHYLIP (phylogeny inference package), version 3.6*, Joseph Felsenstein.

Gerald, RT & Lori, AF 2007, 'Polymorphic malware detection and identification via context-free grammar homomorphism', *Bell Labs Technical Journal*, vol. 12, no. 3, pp. 139-47.

Griffin, K, Schneider, S, Hu, X & Chiueh, T 2009, 'Automatic Generation of String Signatures for Malware Detection', in *Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009*, Saint-Malo, France.

Hoare, CAR 1969, 'An axiomatic basis for computer programming', *Communications of the ACM*, vol. 12, no. 10, pp. 576-80.

Hu, X, Chiueh, T & Shin, KG 'Large-Scale Malware Indexing Using Function-Call Graphs', in *Computer and Communications Security*, Chicago, Illinois, USA, pp. 611-20.

Kamiya, T, Kusumoto, S & Inoue, K 2002, 'CCFinder: a multilinguistic token-based code clone detection system for large scale source code', *IEEE Transactions on Software Engineering*, pp. 654-70.

Karim, ME, Walenstein, A, Lakhotia, A & Parida, L 2005, 'Malware phylogeny generation using permutations of code', *Journal in Computer Virology*, vol. 1, no. 1, pp. 13-23.

Kephart, JO & Arnold, WC 1994, 'Automatic extraction of computer virus signatures', in *4th Virus Bulletin International Conference*, pp. 178-84.

King, JC 1976, 'Symbolic execution and program testing', *Communications of the ACM*, vol. 19, no. 7, pp. 385-94.

Kolbitsch, C, Comparetti, PM, Kruegel, C, Kirda, E, Zhou, X, Wang, XF & Santa Barbara, UC 2009, 'Effective and efficient malware detection at the end host', in *18th USENIX Security Symposium*.

Krinke, J 2001, 'Identifying similar code with program dependence graphs', in p. 301.

Kruegel, C, Kirda, E, Mutz, D, Robertson, W & Vigna, G 2006, 'Polymorphic worm detection using structural information of executables', *Lecture notes in computer science*, vol. 3858, p. 207.

Lim, H, Park, H, Choi, S & Han, T 2009a, 'A method for detecting the theft of Java programs through analysis of the control flow information', *Information and Software Technology*, vol. 51, no. 9, pp. 1338-50.

Lim, H, Park, H, Choi, S & Han, T 2009b, 'A Static Java Birthmark Based on Control Flow Edges', in *Computer Software and Applications Conference (COMPSAC '09)*, pp. 413-20.

Liu, C, Chen, C, Han, J & Yu, PS 2006, 'GPLAG: detection of software plagiarism by program dependence graph analysis', paper presented to Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, Philadelphia, PA, USA.

Livieri, S, Higo, Y, Matushita, M & Inoue, K 2007, 'Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder', in *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, pp. 106-15.

Molnar, DA & Wagner, D 2007, *Catchconv: Symbolic execution and run-time type inference for integer conversion errors*, Technical Report UCB/EECS-2007-23, EECS Department, University of California, Berkeley.

Myles, G & Collberg, C 2005, 'K-gram based software birthmarks', paper presented to Proceedings of the 2005 ACM symposium on Applied computing, Santa Fe, New Mexico.

Park, H, Choi, S, Lim, H & Han, T 2008, 'Detecting code theft via a static instruction trace birthmark for Java methods', in pp. 551-6.

Prechelt, L, Malpohl, G & Philippsen, M 2002, 'Finding plagiarisms among a set of programs with JPlag', *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016-38.

Saitou, N & Nei, M 1987, 'The neighbor-joining method: a new method for reconstructing phylogenetic trees', *Molecular biology and evolution*, vol. 4, no. 4, pp. 406-25.

Son, J-W, Park, S-B & Park, S-Y 2006, 'Program Plagiarism Detection Using Parse Tree Kernels', in Q Yang & G Webb (eds), *PRICAI 2006: Trends in Artificial Intelligence*, Springer Berlin / Heidelberg, vol. 4099, pp. 1000-4.

Symantec 2008, *Symantec internet security threat report: Volume XII*, Symantec.

Symantec 2011, 'Internet Security Threat Report', vol. 16.

Van Emmerik, MJ 2007, 'Static Single Assignment for Decompilation', The University of Queensland.

Wang, X, Jhi, Y-C, Zhu, S & Liu, P 2009, 'Behavior based software theft detection', paper presented to Proceedings of the 16th ACM conference on Computer and communications security, Chicago, Illinois, USA.

Wehner, S 2007, 'Analyzing worms and network traffic using compression', *Journal of Computer Security*, vol. 15, no. 3, pp. 303-20.

Wicherski, G 2009, 'peHash: A Novel Approach to Fast Malware Clustering', in *Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*, Boston, MA, USA.

Wise, MJ 1996, 'YAP3: improved detection of similarities in computer program and other texts', *SIGCSE Bull.*, vol. 28, no. 1, pp. 130-4.

Ye, Y, Wang, D, Li, T & Ye, D 2007, 'IMDS: intelligent malware detection system', in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*.