



Introduction to SinJAR (a New Tool for Reverse Engineering Java Applications) and Tracing its Malicious Actions using Hidden Markov Models

Jaouhar FATTAHI, Mohamed MEJRI, Marwa ZIADIA, Emil PRICOP and Ouejdene SAMOUD

DRDC – Valcartier Research Centre

Frontiers in Artificial Intelligence and Applications

Volume 297

Pages 441 - 453

Date of Publication from Ext Publisher: November 2017

Defence Research and Development Canada

External Literature (P)

DRDC-RDDC-2017-P118

December 2017

CAN UNCLASSIFIED

IMPORTANT INFORMATIVE STATEMENTS

Disclaimer: This document is not published by the Editorial Office of Defence Research and Development Canada, an agency of the Department of National Defence of Canada, but is to be catalogued in the Canadian Defence Information System (CANDIS), the national repository for Defence S&T documents. Her Majesty the Queen in Right of Canada (Department of National Defence) makes no representations or warranties, expressed or implied, of any kind whatsoever, and assumes no liability for the accuracy, reliability, completeness, currency or usefulness of any information, product, process or material included in this document. Nothing in this document should be interpreted as an endorsement for the specific use of any tool, technique or process examined in it. Any reliance on, or use of, any information, product, process or material included in this document is at the sole risk of the person so using it or relying on it. Canada does not assume any liability in respect of any damages or losses arising out of or in connection with the use of, or reliance on, any information, product, process or material included in this document.

This document was reviewed for Controlled Goods by Defence Research and Development Canada (DRDC) using the Schedule to the *Defence Production Act*.

Endorsement statement: This publication has been peer-reviewed and published by the Editorial Office of Defence Research and Development Canada, an agency of the Department of National Defence of Canada. Inquiries can be sent to: Publications.DRDC-RDDC@drdc-rddc.gc.ca.

© Her Majesty the Queen in Right of Canada (Department of National Defence), 2017

© Sa Majesté la Reine en droit du Canada (Ministère de la Défense nationale), 2017

CAN UNCLASSIFIED

Introduction to SinJAR (a New Tool for Reverse Engineering Java Applications) and Tracing its Malicious Actions using Hidden Markov Models

Jaouhar FATTAHI ^{a,b,1}, Mohamed MEJRI ^b, Marwa ZIADIA ^b, Emil PRICOP ^c and Ouejdene SAMOUD ^b

^a*Valcartier Research Centre. Defence Research and Development Canada.*

^b*Department of Computer Science and Software Engineering. Laval University. Québec. Canada.*

^c*Automatic Control, Computers and Electronics Department. Petroleum-Gas University of Ploiesti, Romania.*

Abstract. In this paper, we are proposing a new tool for reversing Java applications called SinJAR. SinJAR is a lightweight software written in Java aiming at inspecting bytecode at compile time and producing the structure tree of a targeted application. Besides, it is able to detect vulnerabilities and security weaknesses inside the Java code. SinJAR can be used for two purposes. The first one is sane and consists in using it to verify whether or not an application is safe and compliant with its specification. The second one is malicious and consists in spying applications through their bytecode and exploiting vulnerabilities that they may enclose. In this paper, we will show how to detect SinJAR malicious actions after showing the capabilities of the tool through few ad hoc attack scenarios conducted in a real military context.

Keywords. Reverse engineering, java reflection, bytecode, vulnerability, anomaly detection, security, SinJAR, Hidden Markov Model.

Introduction

Reverse engineering of software [1–4] is the art of extracting knowledge from an existing software in order to rebuild it totally, partially, or to rebuild another knowledge based on the gathered information [5]. It is used in many fields for learning purposes only, or to reproduce the same product by avoiding copyright infringement [6], to improve documentation weaknesses when the original designers are no longer available, to understand a legacy software to update it and fix bugs in it, to add a security layer above a legacy

¹Jaouhar Fattahi: Valcartier Research Centre. Defence Research and Development Canada. 2459 de la Bravoure Road, Québec, Canada. G3J 1X5. E-mail: jaouhar.fattahi@drdc-rddc.gc.ca.

Laboratory of Computer Security. Department of Computer Science and Software Engineering. Pavillon Adrien-Pouliot, local 3770. Laval University. Québec, Canada, G1K 7P4. E-mail: jaouhar.fattahi.1@ulaval.ca.

system, etc. It is also used for espionage purposes in order to figure out, and then find out, what a competitor is carrying out. Reverse engineering gets another dimension when it comes to the military field because the security of a whole nation could be compromised. History tells us long about military software spy operations that had caused significant damages for a given nation or had changed a military balance in armed conflicts as a result of software cloning or reverse engineering.

The purpose behind the concept of reverse engineering is often to discover unavailable or incomplete information that could be for example program internal workings, operating principle, source code or design philosophy, etc. They all depend upon the system under study. Generally, the absence of this kind of information is for particular reasons; either it is the owner choice to not share it or that this information has been decayed or destroyed. In the two cases, reverser will be confronted with a poor or even non-existent documentation.

Cyber-security is one of reverse engineering categories' application beside software development. Reverse engineering is indeed widely used to audit and improve the security by searching: major vulnerabilities, poor coding practices and security flaws, reverse engineer built-in for malware detection [7] and analysis, viruses' studies and tests for eradicating through an anti-virus development, a lost source code retrieving for enhancement, cryptographic algorithms reversing and so forth. On the other hand, malicious reverse engineering attacks are widely increasing. Hackers use reverse engineering to locate software and programs weak points for malicious exploitation like piracy, crack and application resale, etc. As prevention, developers try to impede reverse engineer through obfuscation process [8], which aims to transform a program into an equivalent one that is harder to reverse engineer.

Reverse engineering process is accomplished through a set of tools that could be categorized on disassemblers, decompilers, debuggers, and monitoring. Disassembler is a program that ensures the conversion of machine language into assembly language. Disassembler output is more intelligible for human than its input, but differs from the decompiler output, which produces a readable high-level language from the same input.

The existing tools [9–13] play of course a key role in increasing program understanding for the reverser. However, although their effectiveness, their use remains somewhat limited as they cannot be considered [14, 15]. Thus, more effective tools for reverse engineering process are always needed and welcome, especially, those that help the reverser shorten the time spent on well understanding the program under study. However, it goes without saying that the tools' usage must always be accompanied by a minimum of skills, otherwise, whatever their effectiveness, they will not help a lot.

Vulnerability detection is the process of analyzing an existing software in order to know about the weaknesses that it involves and discover the weak points or poorly constructed parts in. Learning about vulnerabilities is paramount to avoid attacks and build a secure software. The process of finding out vulnerabilities is a very hard one seeing that there is no standard way to do so and vulnerabilities are of varied sorts. For that reason, tools for detecting vulnerabilities are continuously sought-after.

In this paper, we will introduce a new tool for reverse engineering and vulnerability detection that can be used to inspect any Java application throughout its Java archive file (JAR). We will see what API it implements, its architecture and the facilities it offers to inspect Java applications. Then, we will perform a real experiment on an application in a real military environment. We will show how powerful SinJAR was to extract sensitive

information from within the Java archive of the targeted application only. Then, we will recommend a sophisticated category of tools to detect SinJAR malicious actions based on stochastic models, in particular, the Hidden Markov Model. This is due to the difficulty of detecting SinJAR actions using conventional tools because it is a silent and stealthy tool.

1. Paper organization

This paper is organized as follows:

1. In section 2, we give an overview of the SinJAR tool. We point out in particular the API that it implements and we briefly cover its modules;
2. In section 3, we exhibit a test scenario of the Sinjar tool. We show how it succeeds in reverse engineering a military application and in discovering a serious vulnerability in;
3. In section 4, we show how to detect SinJAR malicious actions when it is malevolently used. The proposed detector is based on Hidden Markov Model (HMM). We give the steps to train and validate the HMM and then how to use it to detect the malicious actions of the SinJAR tool over the traces captured by adequate tools. We show how powerful the used HMM-based tool was to do so;
4. In section 5, we discuss the results that we obtain;
5. In section 6, we give a short conclusion and we introduce to a future extension of our new tool.

2. SinJAR tool

2.1. *What is SinJAR?*

SinJAR is a tool for reverse engineering and vulnerability detection intended to inspect Java applications by using the Java reflection mechanism. SinJAR accepts a Java archive file of a target application and returns its object oriented architecture in both XML and JSON format. This includes the class names used inside the application, attribute names, attribute types, attribute default values, method names, their argument names and types, their returned values and their types, class constructors, getters and setters, package names, method and attribute modifiers (public, private, protected, static, final, abstract, synchronized, etc.), inheritance between classes, implemented interfaces' names, annotations, etc.

2.2. *What is SinJAR for?*

Upon inspecting these elements, SinJAR is able not only to understand the application architecture but also to detect vulnerabilities like connection strings to servers, passwords, user names, and so on. SinJar helps the user find out these weaknesses and fix them. Besides, it can be used to verify whether or not an application is compliant with its specification, and if not, what the origin of the structural problem in a given component, be it related to the design or the implementation.

2.3. Java reflection mechanism

Reflection is an advanced mechanism and a strong concept of Java core [16]. It is not commonly used in conventional programming but it is the cornerstone in most of the major Java and J2EE frameworks like JUnit, Spring, Hibernate, Eclipse and Tomcat. By using Java reflection we can inspect classes, interfaces, enums, get their structure, or get methods and fields information at runtime or compile time. Besides, we can use reflection to instantiate objects, call methods or alter field values without even knowing the names of the classes, methods, fields, etc. Moreover, and in spite of the common visibility rules, by using reflection we can access private data, private methods or any other private object in a given Java class. The API that offers reflection services is implemented in the `java.lang.reflect` package. Hereafter, we give the main methods that can be used in a reflection-based Java programming.

1. `someObject.class.getMethods()`: returns an array of method names in `someObject`;
2. `someObject.class.getFields()`: returns an array of field names in `someObject`;
3. `someClass.getName()`: returns the class name of `someClass`;
4. `someClass.getModifiers()`: returns the modifiers of `someClass` (i.e. public, protected, private);
5. `someClass.getSuperclass()`: returns the superclass of `someClass`;
6. `someClass.getConstructors()`: returns the constructors of `someClass`;
7. `someClass.getInterfaces()`: returns the interfaces implemented by `someClass`;
8. `someClass.getPackage()`: returns the package name of `someClass`;
9. `someClass.getAnnotations()`: returns the annotations of `someClass`;
10. Etc.

For the full reflection API documentation, please refer to [17].

2.4. SinJAR Modules

SinJAR consists of three modules:

1. The HELPER module of SinJAR is implemented in the package `sinjar.reflection.helper`. This module provides services for reading class files through the JARs of the targeted application;
2. The ATTACKER module of SinJAR is implemented in the package `sinjar.reflection.attack`. This module provides services for penetrating a JAR file for a targeted application, inspecting the application classes, reconstructing the application tree including its packages, classes, methods, constructors, fields and their default values, modifiers, etc. Finally, it provides the complete application architecture in XML and JSON format;
3. The VIEWER module of SinJAR is implemented in the package `sinjar.viewer`. This module allows to visualize the tree of an inspected application.

3. Testing SinJAR in a military environment

3.1. Targeted application

The targeted application is a simple GUI written in Java that executes ordinary CRDU operations on a Pilot table in a MySQL database 5.5 running on Ubuntu 14.04 LTS. Through which, a user can insert a pilot record, delete it, update it or visualize it. The user launches the application via its JAR file PilotApp.jar without having access to its source code. An overview of this application is given by Figure 1.

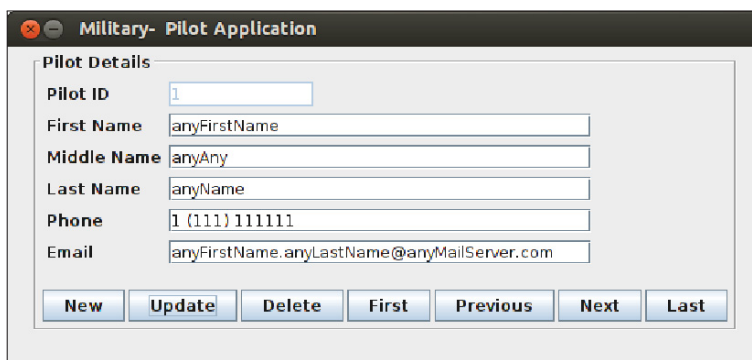


Figure 1. Application inspected by SinJAR

3.2. Attack with SinJAR

The attack is launched through a Linux shell calling the ATTACKER module of SinJAR on the Pilot application JAR file (i.e. PilotApp.jar) from within the same console of the user who is allowed to execute the targeted application. The attacker is hence assumed to be a regular user with the devil intention to spy the application he is using. Once launched, the ATTACKER module reproduces all the structure of the attacked application and returns it in both XML format and JSON format. Figure 2 shows the output of the attack returned by the ATTACKER module and visualized by the VIEWER module of SinJAR. Please notice that beside the fact that the application is completely reverse engineered, SinJAR succeeds to capture sensitive information included in the application JAR file which consists in: the connection string to the database (i.e. `mysql://192.168.1.107:3306/testdb`), a user name (i.e. `scott`) and his password (i.e. `iloveritta`). This information is perilous since the intruder is now empowered to further attack the database and acquire more sensitive information such as military maps, satellite photos, technical drawings, research papers, and all that can be seen or done by the user `scott` on the database.

4. Detecting SinJAR malicious actions

4.1. Hidden Markov Model-Based Detectors

A Hidden Markov Model (HMM) [18–22] is a stochastic model in which a system is assumed to satisfy the Markov property [23]. The Markov property supposes that the

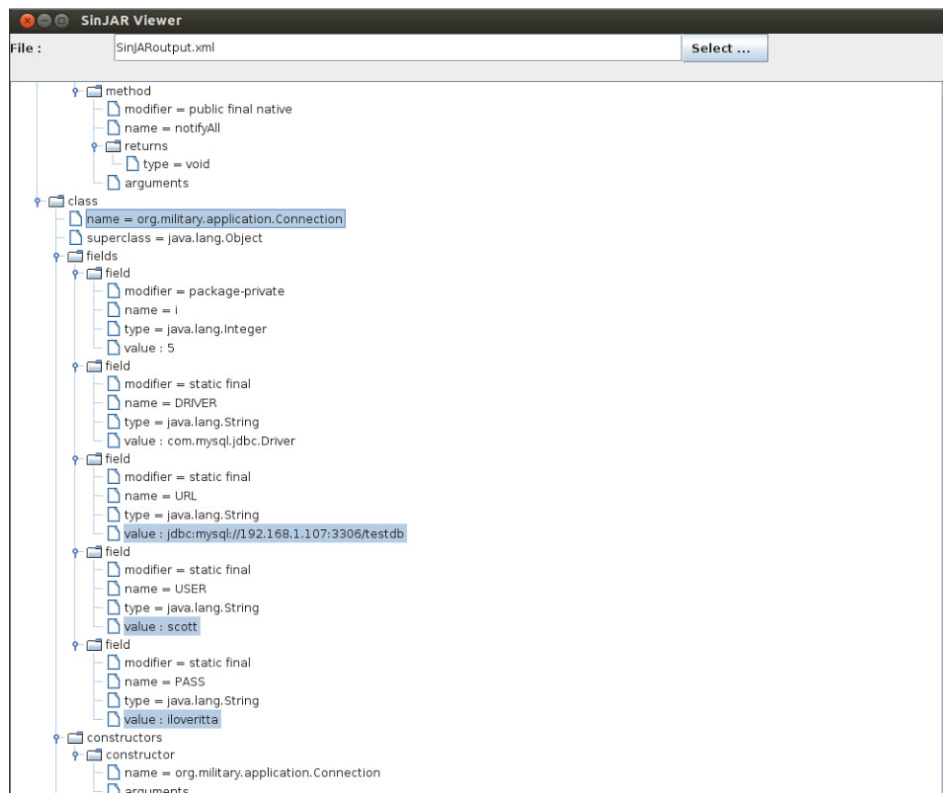


Figure 2. SinJAR reverses a military application and reports security vulnerabilities

conditional probability distribution of future states depends only upon the current state. Formally, a HMM is a quadruplet $\langle S, \Sigma, T, G \rangle$ such that:

- S : is a set of N hidden states including two special states, *start* and *end*. The state *start* launches the process and the state *end* terminates it;
- Σ : is a set of M symbols or alphabet;
- T : is a transition matrix that contains the probabilities of going from a state s_i de $S - \{end\}$ to a state s_j of $S - \{start\}$;
- G : is a matrix that contains the probabilities of emitting the symbol σ_i de Σ from s_j the state $S - \{start, end\}$.

A HMM is able to answer three questions:

1. Given an observable sequence (a string of Σ) of size T , what is the probability of occurrence of that sequence according to the predefined HMM? This question is answered by the *forward-backward* algorithm [24]. This algorithm is of a complexity $O(N^2T)$ where N is the number of states of the HMM;
2. Given an observable sequence of size T , what is the most probable sequence of states that produces it according to the predefined HMM? This question is answered by the *Viterbi* algorithm [25]. This algorithm is also of a complexity $O(N^2T)$ where N is the number of states of the HMM;

3. Given an observable sequence of size T , how to change a given HMM in a way that the probability of that sequence be maximum? This question is answered by the *Baum-Welch* algorithm [26]. This algorithm is also of a complexity $O(N^2T)$ where N is the number of states of the HMM.

For HMM-based detectors, an HMM should be trained first and then validated on normal sequences. The *Baum-Welch* algorithm is here used to perform a series of changes on the transition matrix until the probabilities of occurrence of these sequences be maximum. As for the validation step, another set of sequences is delivered to the HMM so that an acceptance threshold is fixed. The HMM knows so far how the normal sequences *look like* and how to decide if a sequence is normal or not. To do so, the HMM calculates the probability of occurrence of a new sequence using the *forward-backward* algorithm, if it is above the acceptance threshold the sequence is deemed normal, otherwise, it is deemed anomalous.

4.2. Environment

4.2.1. LTTng for recording traces

The Linux Trace Toolkit next generation (LTTng) [27] is an open source tracing framework for Linux. It can be used to trace the Linux kernel, user applications, and user libraries. It consists of:

1. Kernel modules to trace the Linux kernel;
2. Shared libraries to trace C or C++ applications;
3. A Java library to trace Java applications;
4. A Python library to trace Python applications;
5. Command-line tools to monitor the LTTng tracers.

LTTng is used to generate kernel traces [28] that consist of events (system calls) resulting from the target application when it is normally executed and when it is attacked. These traces are either in CTF or text format. The traces resulting from a normal execution are deemed normal traces, and the ones resulting from the attack on that application are deemed anomalous traces.

4.2.2. TotalADS and HMM implementation

TotalADS [29, 30] is a new open source framework for automated host-based anomaly detection developed at Concordia University as a plug-in for Eclipse. It encompasses varied anomaly detection techniques such as Hidden Markov Model (HMM), Kernel State Modeling (KSM), and Sequence Matching (SQM). It accepts several traces and logs formats such as the Common Trace Format (CTF) [31], XML and text format. It supports live streaming as well as stored traces. It is thought and designed to closely cooperate with the Trace Compass module [32], which is another Eclipse plug-in, that offers a rich environment for visualizing traces and anomalies. Using TotalADS, one can diagnose traces and detect anomalies. An anomaly is reported when a trace is *distant enough* from a normal behavior in the logic of the used technique. In this paper, we will

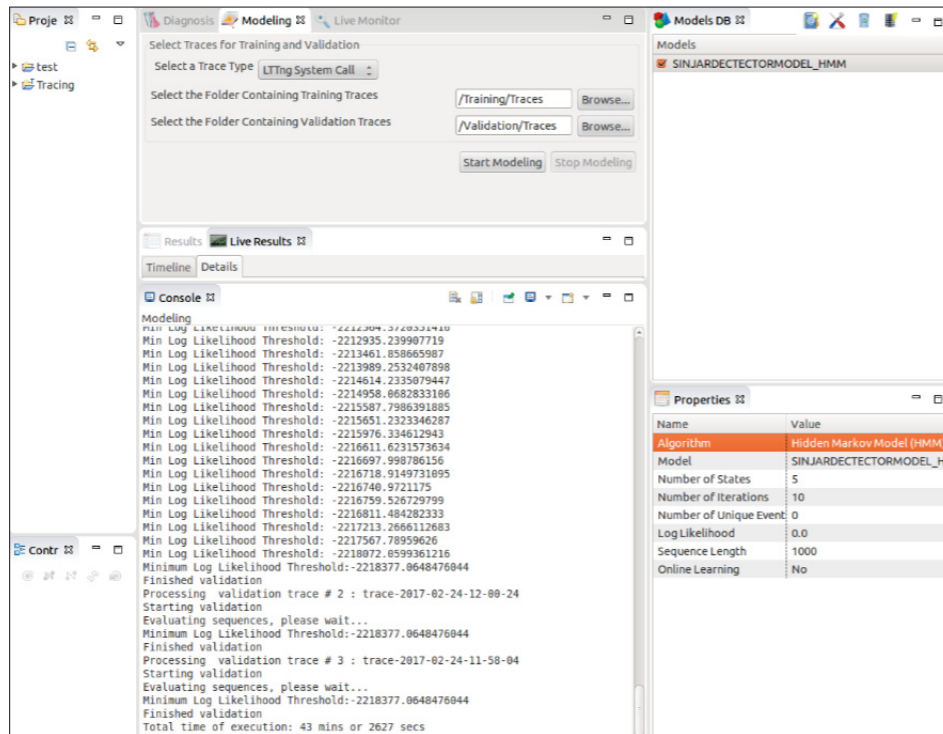


Figure 3. HMM training and validation

focus on the HMM algorithm as our preferred technique because we think it is the most suitable one to detect SinJAR anomalous traces.

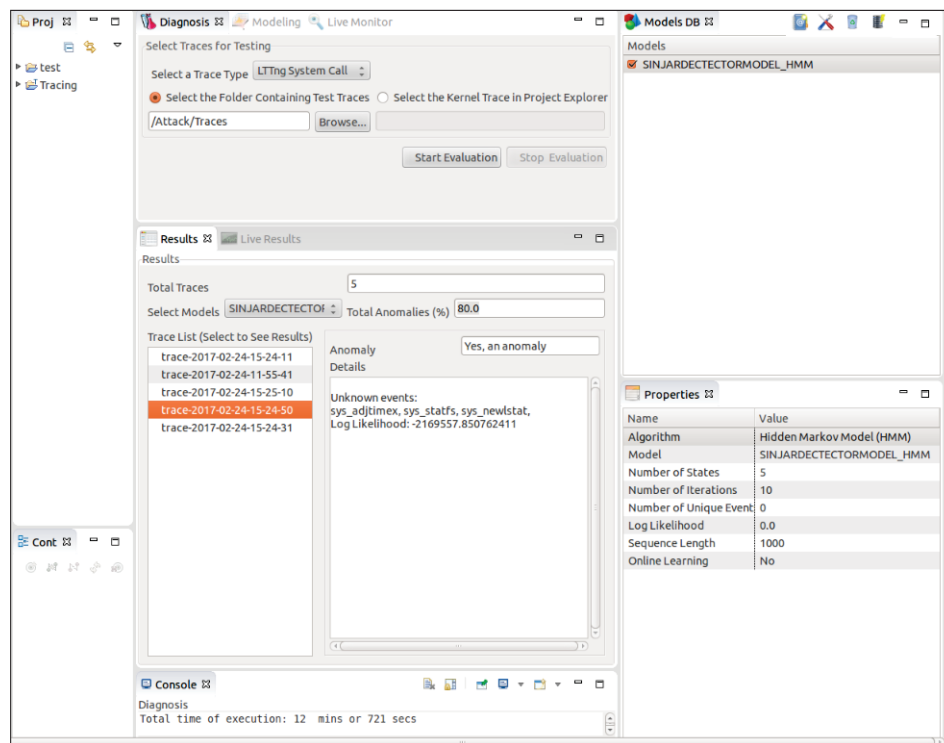
4.2.3. Attack scenario

Before launching the attack, we generate the HMM that reflects the system normal behavior. We set the HMM state number to 5 and the sequence length to 1000. We launch LTTng to record the kernel trace. Meanwhile, we execute the Pilot application as well as a set of usual Linux commands that a normal user is entitled to run in his daily work. Once done, the traces are stored in a directory. We then launch TotalADS to generate the corresponding HMM. We use two thirds of the traces to train the HMM, and the remaining third to validate it. TotalADS stores the constructed HMM in a JSON format in a noSQL database. Once done, we launch the attack using the SinJAR tool. In the meantime, we launch LTTng to record the anomalous traces of the Linux kernel. Once the Pilot application is hacked and reverse engineered, we store them in a directory. As for the analysis, we provide TotalADS with four long anomalous traces and one normal trace to analyze. The expected ratio of anomalous traces is hence 80%.

5. Discussion

As for the SinJAR tool, as we can see in Figure 2, it was successful reversing the Pilot application and returning its structure and all the expected packages, classes, methods, constructors, attributes, modifiers, super classes, etc. This is thanks to the powerful Java reflection API that it implements. SinJAR was particularly successful reporting a major vulnerability consisting of sensitive information hardcoded in the application.

Concerning SinJAR malicious traces detection, as we can clearly see in Figure 4, HMM succeeds in catching all the malicious traces resulting from the attack by SinJAR. In the same vein, HMM succeeds in detecting the normal sequences as given in Figure 5. The ratio of 80% reflects the accurate ratio of malicious traces we were expecting. We can conclude that HMMs are the most suitable techniques to detect stealthy actions that can easily escape signature-based detection tools. That is because they base their decision on the comparison between two states: a normal state resulting from a learning phase and another state to analyze. The decision is made upon the probability for a given trace to be generated by the HMM. If it is higher than the threshold set by the HMM during the model validation step, it is said to be normal. If not it is reputed anomalous.



The screenshot displays the SinJAR tool's interface, which is divided into several panes. The top-left pane, titled 'Select Traces for Testing', shows the 'LTng System Call' trace type selected and the folder path '/Attack/Traces' entered. The 'Start Evaluation' button is visible. The top-right pane, 'Models DB', lists the model 'SINJARDETECTORMODEL_HMM'. The central 'Results' pane shows 'Total Traces' as 5 and 'Total Anomalies (%)' as 80.0. A 'Trace List' on the left shows five traces, with the one from '2017-02-24-15-24-50' highlighted in orange. The 'Anomaly Details' pane for this trace shows 'Unknown events: sys_adjtimex, sys_stats, sys_newlstat, Log Likelihood: -2169557.850762411'. The bottom-right 'Properties' pane lists model parameters: Algorithm (Hidden Markov Model (HMM)), Model (SINJARDETECTORMODEL_HMM), Number of States (5), Number of Iterations (10), Number of Unique Events (0), Log Likelihood (0.0), Sequence Length (1000), and Online Learning (No). The bottom 'Console' pane shows the total execution time as 12 minutes or 721 seconds.

Figure 4. HMM detects SinJAR's malicious traces

Nevertheless, the learning phase must be carefully led to reflect the real normal behavior of the system which is not always an easy task.

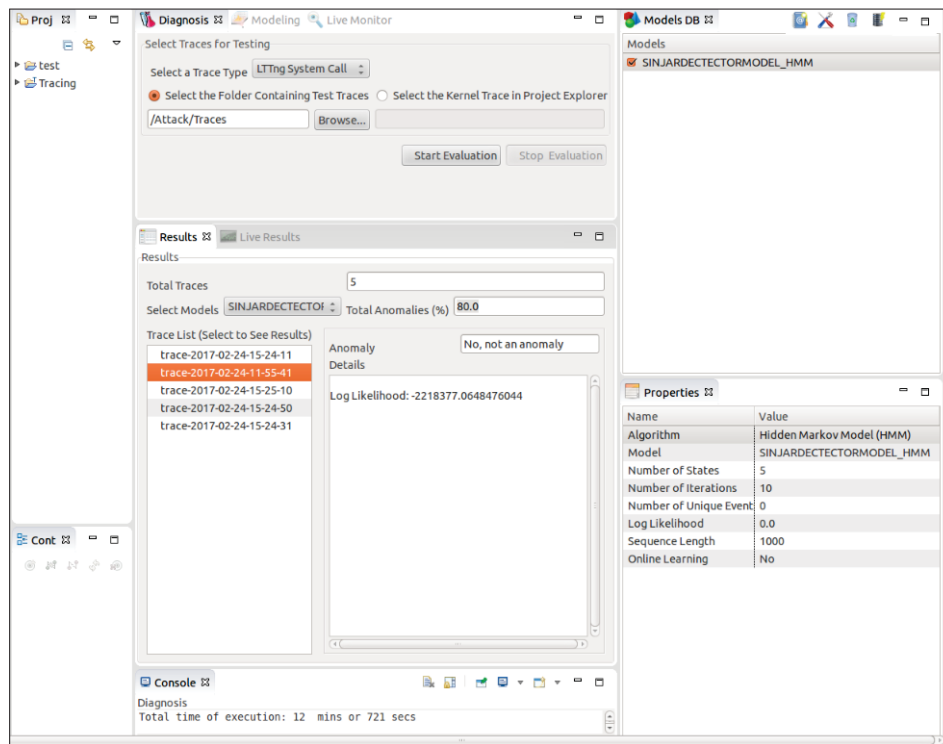


Figure 5. HMM reports normal traces

In the state-of-the-art we find other tools comparable to the SinJAR tool such as:

1. DiStorm3: it is a tool that comes with Kali Linux [33] and used in embedded or kernel modules to disassemble instructions in 16, 32 and 64-bit modes. It tightly depends on the C library;
2. Dex2jar: it is an API to read the Dalvik Executable format and convert it to ASM format. Some efforts are made to set about the deobfuscation of JAR files in Java applications but they are still limited and controversial;
3. Apktool: it is a tool for debugging smali code, a common human readable format of android applications written in Java;
4. InsDal [34]: it is a tool that inserts code in specific points of the Dalvik byte-code depending on the need of the user. It arranges the registers in a way that it protects the original code from illegal manipulation. It also optimizes the inserted code in order to save memory and reduce overhead;
5. DynStruct [35]: it is an open source tool that instruments dynamic binary to get sensitive information related to memory accesses. This information is then worked out to reconstruct structures created and used in the binary code.
6. Javasnnoop: it is an aspect security oriented tool. It comes with Kali Linux as well. It is mainly used to validate web applications [36], but it can be also used for other applications.

Compared to these tools and others such as Refine/C, Imagix4D, Sniff+, and Rigi [4, 37], the SinJAR tool is a multidisciplinary and a multi-objective one. Indeed, it can be used with any type of application and does not depend on any operating system. It is also flexible, modular, extensible, lightweight and can be used either as a standalone tool or attached to application debuggers.

6. Conclusion

In this paper, we have introduced a new tool for reverse engineering and detecting vulnerability inside Java applications, called SinJAR. This new tool was able to report serious vulnerabilities when tested in a military environment. We have also given recommendations about advanced detectors that should be used to trace the malicious actions of that tool when it is malevolently used. In a future work, we will add new functionalities to SinJAR so that it will be able to inspect applications dynamically. We also aspire to integrate it with other validation tools such as debuggers, refactoring tools, and software security checkers. Besides, we intend to use it for reinforcing security [38, 39] in legacy programs by adding and altering code in, in order to conform to a given security policy.

Warning

It is **strictly prohibited** to use the SinJAR tool for malicious purposes.

Acknowledgments

Warm thanks go to Mario Couture and Daniel Thibault, Valcartier Research Centre. DRDC-RDDC, Québec, Canada.

References

- [1] Paolo Tonella and Alessandra Potrich. *Reverse Engineering of Object Oriented Code (Monographs in Computer Science)*. Springer; Softcover reprint of hardcover 1st ed. 2005 édition (26 mai 2011), Secaucus, NJ, USA, 2011.
- [2] Eldad. Eilam and Elliot J. Chikofsky. *Reversing secrets of reverse engineering*. Wiley, Indianapolis, IN, 2005.
- [3] Juan Caballero and Dawn Song. Automatic protocol reverse-engineering: Message format extraction and field semantics inference. *Comput. Netw.*, 57(2):451–474, February 2013.
- [4] B. Adams, H. Tromp, K. de Schutter, and W. de Meuter. Design recovery and maintenance of build systems. In *2007 IEEE International Conference on Software Maintenance*, pages 114–123, Oct 2007.
- [5] Junghee Lim, Thomas W. Reps, and Ben Liblit. Extracting output formats from executables. In *13th Working Conference on Reverse Engineering (WCRE 2006), 23-27 October 2006, Benevento, Italy*, pages 167–178, 2006.
- [6] Pamela Samuelson. Reverse engineering under siege. *Commun. ACM*, 45(10):15–20, October 2002.
- [7] Monirul I. Sharif, Andrea Lanzi, Jonathon T. Giffin, and Wenke Lee. Automatic reverse engineering of malware emulators. In *IEEE Symposium on Security and Privacy*, pages 94–109. IEEE Computer Society, 2009.
- [8] Christian S. Collberg and Clark D. Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Trans. Software Eng.*, 28(8):735–746, 2002.

- [9] Mathieu Nayrolles and Abdelwahab Hamou-Lhadj. BUMPER: A tool for coping with natural language searches of millions of bugs and fixes. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, pages 649–652, 2016.
- [10] Robin David, Sébastien Bardin, Thanh Dinh Ta, Laurent Mounier, Josselin Feist, Marie-Laure Potet, and Jean-Yves Marion. BINSEC/SE: A dynamic symbolic execution toolkit for binary-level analysis. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, pages 653–656, 2016.
- [11] Angela Lozano, Carlos Noguera, and Viviane Jonckers. Managing traceability links with matraca. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*, pages 665–668, 2016.
- [12] Jason Raber. Columbo: High performance unpacking. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 507–510, 2017.
- [13] Xu Li and Laurie J. Hendren. Mc2for demo: A tool for automatically translating MATLAB to FORTRAN 95. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, pages 458–463, 2014.
- [14] Scott Tilley. Coming attractions in program understanding ii: Highlights of 1997 and opportunities in 1998. Technical Report CMU/SEI-98-TR-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1998.
- [15] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. A survey of automatic protocol reverse engineering tools. *ACM Comput. Surv.*, 48(3):40:1–40:26, December 2015.
- [16] Paulo Barros, René Just, Suzanne Millstein, Paul Vines, Werner Dietl, Marcelo d’Amorim, and Michael D. Ernst. Static analysis of implicit control flow: Resolving java reflection and android intents (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 669–679, 2015.
- [17] Trail: The Reflection API. <https://docs.oracle.com/javase/tutorial/reflect/>. Last accessed: 2017-05-05.
- [18] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37:1554–1563, 1966.
- [19] L. Rabiner and B. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, January 1986.
- [20] R. Elliott, L. Aggoun, and J. Moore. *Hidden Markov models: estimation and control*. Springer, 1995.
- [21] Zoubin Ghahramani. Hidden markov models. chapter An Introduction to Hidden Markov Models and Bayesian Networks, pages 9–42. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2002.
- [22] Olivier Cappé, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [23] Sean Meyn and Richard L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.
- [24] Lawrence R. Rabiner. Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [25] G. D. Forney. The viterbi algorithm. *Proc. of the IEEE*, 61:268–278, March 1973.
- [26] Lloyd R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4), December 2003.
- [27] The LTTng Documentation. <http://lttng.org/docs/v2.9/>. Last accessed: 2017-05-05.
- [28] S. S. Murtaza, A. Sultana, A. Hamou-Lhadj, and M. Couture. On the comparison of user space and kernel space traces in identification of software anomalies. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 127–136, March 2012.
- [29] Syed Shariyar Murtaza, Abdelwahab Hamou-Lhadj, Wael Khreich, and Mario Couture. Total ADS: automated software anomaly detection system. In *14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014, Victoria, BC, Canada, September 28-29, 2014*, pages 83–88, 2014.
- [30] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and Mario Couture. A host-based anomaly detection approach by representing system calls as states of kernel modules. In *IEEE 24th In-*

ternational Symposium on Software Reliability Engineering, ISSRE 2013, Pasadena, CA, USA, November 4-7, 2013, pages 431–440, 2013.

- [31] Common Trace Format. <http://diamon.org/ctf/>. Last accessed: 2017-05-05.
- [32] Trace compass. <http://tracecompass.org/>. Last accessed: 2017-05-05.
- [33] Robert W. Beggs. *Mastering Kali Linux for Advanced Penetration Testing*. Packt Publishing, 2014.
- [34] Jierui Liu, Tianyong Wu, Xi Deng, Jun Yan, and Jian Zhang. Insdal: A safe and extensible instrumentation tool on dalvik byte-code for android applications. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 502–506, 2017.
- [35] Daniel Mercier, Aziem Chawdhary, and Richard Jones. dynstruct: An automatic reverse engineering tool for structure recovery and memory use analysis. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 497–501, 2017.
- [36] Joseph Muniz and Aamir Lakhani. *Web Penetration Testing with Kali Linux*. Packt Publishing, 2013.
- [37] B. Bellay and H. Gall. A comparison of four reverse engineering tools. In *Proceedings of the Fourth Working Conference on Reverse Engineering*, pages 2–11, Oct 1997.
- [38] Mohamed Mejri and Hamido Fujita. Enforcing security policies using algebraic approach. In *New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Seventh SoMeT 2008, October 15-17, 2008, Sharjah, United Arab Emirates*, pages 84–98, 2008.
- [39] T. Mechri, Mahjoub Langar, Mohamed Mejri, Hamido Fujita, and Yutaka Funyu. Automatic enforcement of security in computer networks. In *New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Sixth SoMeT 2007, November 7-9, 2007, Rome, Italy*, pages 200–222, 2007.

DOCUMENT CONTROL DATA		
(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)		
1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g., Centre sponsoring a contractor's report, or tasking agency, are entered in Section 8.) DRDC – Valcartier Research Centre Defence Research and Development Canada 2459 route de la Bravoure Quebec (Quebec) G3J 1X5 Canada	2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.) CAN UNCLASSIFIED	
	2b. CONTROLLED GOODS NON-CONTROLLED GOODS DMC A	
3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.) Introduction to SinJAR (a New Tool for Reverse Engineering Java Applications) and Tracing its Malicious Actions using Hidden Markov Models		
4. AUTHORS (last name, followed by initials – ranks, titles, etc., not to be used) SAMOUD, JaouharFATTAHI, MohamedMEJRI, MarwaZIADIA, EmilPRICOPandOuejdene; Applications, FrontiersinArtificialIntelligenceand; 297, Volume; 453, Pages441-; 2017, DateofPublicationfromExtPublisher:November		
5. DATE OF PUBLICATION (Month and year of publication of document.) December 2017	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) 13	6b. NO. OF REFS (Total cited in document.) 39
7. DESCRIPTIVE NOTES (The category of the document, e.g., technical report, technical note or memorandum. If appropriate, enter the type of report, e.g., interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.) External Literature (P)		
8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.) DRDC – Valcartier Research Centre Defence Research and Development Canada 2459 route de la Bravoure Quebec (Quebec) G3J 1X5 Canada		
9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)	
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC-RDDC-2017-P118	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)	
11a. FUTURE DISTRIBUTION (Any limitations on further dissemination of the document, other than those imposed by security classification.) Public release		
11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Any limitations on further dissemination of the document, other than those imposed by security classification.)		

12. **ABSTRACT** (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

13. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g., Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Cyber threat; cyber attack; Java