The International Journal of Engineering and Science (IJES)
|| Volume || 8 || Issue || 11 || Series II || Pages || PP 87-90|| 2019 ||
ISSN (e): 2319 – 1813 ISSN (p): 23-19 – 1805

Conceptual Design of an Efficient Java Obfuscator Using Advances in Artificial Intelligence and Multicore Processors

Okah Paul-Kingsley Onyemaechi, Prof. Inyiama and E. U Randolph
[1]*Department of Electronic and Computer Engineering, NnamdiAzikiwe University Awka, Anambra State Nig.*
[2]*Department of Electronic and Computer Engineering, NnamdiAzikiwe University Awka, Anambra State Nig.*
[3]*Atlantis Research Center, G.R.A Enugu State Nigeria.*
Corresponding Author: Okah Paul-Kingsley Onyemaechi

ABSTRACT
The internet has brought stiff competition among business owners', organizations and governments. So the need for the protection of information has become very strategic to the survival of any institution or government. Software protection plays a significant role in this internet era. Software protection is a combination of principles and techniques to enhance software security. Most software programmes distributed on the internet today are Java-based. They suffer a lot of tampering and reverse engineering which has necessitated the need to proffer solutions. Current Java-based obfuscators do not take advantage of recent advances in multi-core microprocessor technology. Recent advances in Artificial Intelligence related to pattern recognition; facial, image and handwriting, have not been incorporated into any of the obfuscators. Though, the efficiency of obfuscators will greatly improve when these technologies are added. The aim of this paper is to investigate how these two developments can be integrated into obfuscators to provide greater obfuscation and runtime efficiency.

KEYWORDS; *pattern recognition, parallel processing, lexer/parser, machine learning*

Date of Submission: 07-11-2019 Date of acceptance: 27-11-2019

I. INTRODUCTION

Software obfuscation is a semantic-preserving transformation aimed at bringing software into a form which impedes the understanding of its algorithm and data structures to prevent the extraction of valuable information from the source code. Software obfuscation has wide usage in computer security, information hiding and cryptography. Security requirement for software has become a major focus of interest. The core interest of protection of software secrets is to make the work of reverse engineering very difficult, if not impossible. In the software industry today a recurring challenge has been software piracy, especially now that we have the Internet explosion. Recent development in the computer digital industry has shown that software represents a significant intellectual property [1]. The software developers and providers would be done great harm if through reverse engineering all the technology secrets they suffered to develop are exposed. Consequently, the whole concept of software obfuscation is to ensure the protection of the commercial and industry-level software.

II. BACKGROUND

Existing software obfuscation available in the market cannot provide the much needed protection as evident in the discussion by Barak et al. [2], on the notion of impossibility of software obfuscation. Most obfuscators like SourceFormax, SemanticDesign, and other byte code obfuscators like SandMark are variations of the same basic concepts. Software obfuscation is on two levels: source code and byte code obfuscation. The source code obfuscation consists of techniques to make source code less comprehensible and automatically transforms programmer's code into more complex and functionally equivalent one. These transformations are control transformation and data transformation. The control transformation involves the opaque variable, opaque predicate, code splitting, loop condition extension. Data obfuscation involves identifier re-naming, white space and comment removal, string encryption, and variable substitution. These code obfuscation transformation techniques have all been tried and found inadequate for the level of protection needed. The transformation capabilities of most obfuscators are known to the creators of de-obfuscators. They design automatic de-obfuscators with counter measures to defeat any attempt to protect materials of interest. The attacker know already that it is either of these mentioned transformation techniques. The need then arises to adopt a different approach to designing Java-based obfuscators. This paper seeks to integrate pattern recognition and advantages in processing power of multi-processor.

Advances in Microprocessor technology

Prior to 2005 central processing unit (CPU) manufacturers such as Intel and AMD produced only single core processors. The most common are the tri-core, quad-core, hexa-core and deca-core. Currently, AMD and INTEL have CPU'S with up to 12 cores for commercial use. Faster dedicated CPUs like Zeon processors have up to 64 cores. Though this may be an extreme application for obfuscators, companies able to afford servers with this type of processor can run company-wide obfuscation on a few of the cores of the main server efficiently.

As the speed limit of single processors became obvious because of heat dissipation and frequency constraints, faster processors could only be made by including multiple cores in a single housing. This made inter-processor communication faster. Also, the use of shared internal memory (Level2 Cache) made data exchange faster. Unfortunately, most software developers failed to make use of the parallel processing abilities of these new processors because software code has to be specifically refactored do this. Running a program not designed for parallel architecture does not create any efficiency improvement in the runtime behaviour of the software because the other CPU cores remain idle or unused at runtime. In programming, explicit use must be made of these functionalities in new processors. The programmers have to specifically use Threads and assign processor cores to specific processes in order to optimize the processors for efficiency.

Specifically, commercial obfuscators such as SourceFormax, SandMark, and ProGuard are not optimized for parallel processing. Though some apply multithreading for loading files, the do not assign CPU cores that will help in speed and efficiency improvement. This may not appear important for small sized programs, but for companies that need to obfuscate gigabytes of code this becomes a major bottle neck. This bottle neck is also aggravated by the fact that when modifications and software design cycles are considered, changes in software code will entail greater loads on the obfuscator. So, redesigning obfuscators to optimized CPU characteristics will provide the needed efficiency improvement in commercial obfuscators.

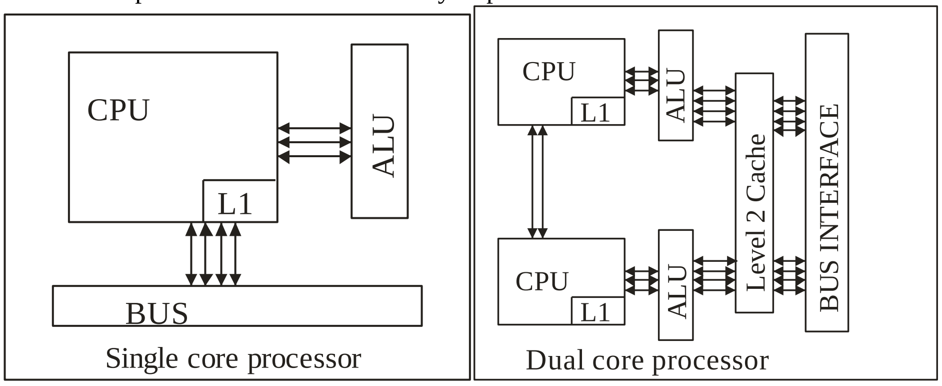

Fig. 1.1: Single Core and Dual core Processor

Artificial intelligence

The use of artificial intelligence in the software industry has increased over the years. Their applications has spread through virtually every part of the industry, from automotive, military, social media, security and space. Aside routing, AI is most used in finding patterns in large datasets in business modeling. Facial, Voice, Image and Handwriting recognition represent areas where AI has yielded great gains, particularly in all applications related to security.

The underlying architecture in all artificial intelligence systems is machine learning and pattern recognition. This enables the system train itself and adapt to unforeseen situations making it intelligent in an artificial sense. Machine learning involves an adaptive process where decisions based on the output are used to modify processing path of algorithms. Pattern recognition also employs complex heuristics and statistical analysis to identify obvious and elusive patterns in data sets [3].

Again, current obfuscators have not been adapted to these new computing frontiers where their adoption will yield tremendous efficiency advantages.

Integrating parallel processing and Artificial Intelligence to obfuscators

The basic workflow of an obfuscator involves the following:

The loader - loads the source files from storage

The Parser/Lexer – breaks down the language syntax, removes redundancies such as white spaces, comment tags, duplicate strings, and to applies the grammatical rules of the obfuscator to create an output suitable as an input to the Randomization Engine.

The Randomization Engine - handles data encryption, class hierarchy expansion or contraction, index tables, interface construction, and dead code pooling, The Substitution Engine - Takes care of code reassembly using the output of the randomization engine to create an obfuscated version of the code.

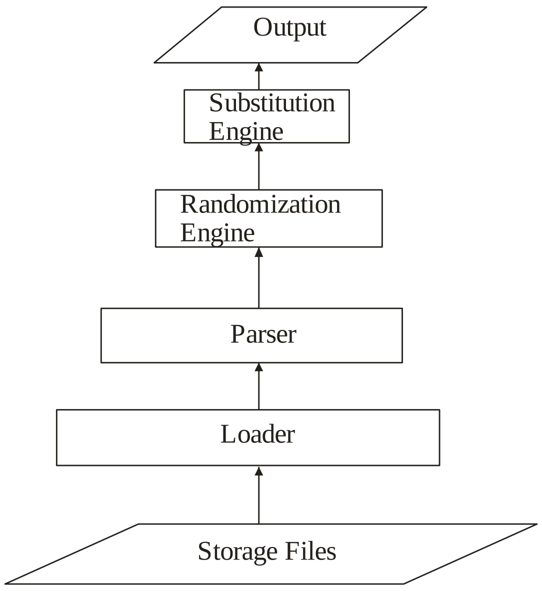

Fig. 1.2: Basic Obfuscator Schematic

Obfuscators can take advantage of recent multi-core processors to improve their runtime efficiency. It will allow classes to be analyzed concurrently and achieve a greater coupling between classes. More coding patterns that can be merged or isolated will show up when classes are processed in parallel by the obfuscator. Assigning different CPU cores to the class loaders, parser engine, and assigning multiple threads so that the randomization and substitution engine can process classes in parallel batches will result in greater overall performance.

This parallel processing approach allows for the coupling of an AI engine to provide runtime optimization of the obfuscated code. Incremental obfuscation of the code made possible by batch processing makes it possible for the profiler performance metrics to be used to choose different substitution paths for the generated code.

Parallel processing also allows for a pre structural analysis of the source code at the same time parser thread is running. This will provide information the AI engine will used to flatten class hierarchies for more effective obfuscation.

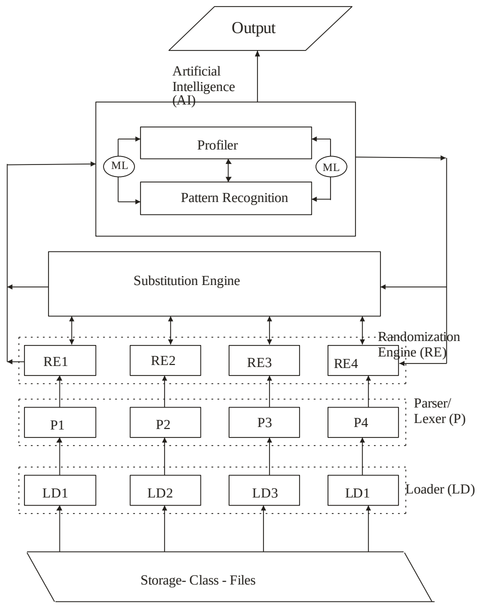

Fig.1.3: Artificial Intelligence/Multi-Core CPU Enhanced Obfuscator Schematic:

The addition of an AI engine as an optimizer hooked onto the substitution, randomization engine, and profiler gives an overview that can support machine learning algorithms. This allows for more effective obfuscation since the feedback provided by the AI can create more robust variations of obfuscations even for the same code. Also, because of similarities in programing constructs, pattern recognition can find classes that share similar internal structures. These structures can also be added to index tables and their former references substituted with references or method calls.

Since an AI engine can recognize patterns in the obfuscated code a human cannot, these patterns can be used to estimate performance metrics of the obfuscator. And provide the obfuscator with the logic of how to rearrange these patterns to produce more random distribution of these patterns.

III. CONCLUSION

This paper attempts to provide a more efficient conceptual design to existing obfuscators by adding advances in computing technology in recent years. It summarizes the major advance made in hardware CPU technology in the area of multi-core CPUs, and proffers a design that integrates the latest in Artificial Intelligence, Pattern Recognition, and Machine Learning. We believe integrating this design to current obfuscators will greatly improve both the runtime and obfuscation efficiency of current and future obfuscators.

REFERENCE

[1]. Xuesong Zhang, Fengling He and WanliZuo. (2010): “Theory and Practice of Program Obfuscation”. Computer Science Department, Jilin University, China.

[2]. Boaz Barak, Oded. Goldreich, RusselImpagliazzo, Steven Rudich, Amit. Sahai and Ke. Yang and SalilVadhan, (2012) “On the (IM) Possibility of Obfuscating Programs”, In proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, 2012.

[3]. Christopher M. Bishop (2006) PATTERN RECONGNITION AND MACHINE LEARNING Microsoft Research Ltd Cambridge CB3 0FB U.K

Okah Paul-Kingsley Onyemaechi "Conceptual Design of an Efficient Java Obfuscator Using Advances in Artificial Intelligence and Multicore Processors" The International Journal of Engineering and Science (IJES), 8.11 (2019): 87-90