OJIT: A Novel Secure Remote Execution Technology By Obfuscated Just-In-Time Compilation

M. Hataba[1]*, A. El-Mahdy*, A. Shoukry*, E. Rohou**

*Parallel Computing Lab, E-JUST **INRIA,Rennes

Vision

- Our work introduces OJIT (Obfuscated Just-In-Time compilation technique); a technique inspired by the “security by obscurity” principles adopted in foist viruses and surreptitious malware design.
- Target securing remote computation platforms such as those in cloud computing.

Introduction

- The world of computing is now undergoing a major paradigm shift. The unlimited potentials of a connected world have allowed a new form of “remote execution” of programs, where processing of one’s data is done on a physically out of reach computing premise. For example we have cloud computing platforms bundled with thin clients like smartphones.
- Unfortunately, a new breed of cyber threats appeared such as side-channel attacks and cartography. Relying solely on data encryption is not enough, as the decryption software itself runs remotely on the cloud, and therefore security can be compromised.

The Security Problem In The Cloud

- The dilemma here is how to trust a computing environment that one cannot control.
- We have to invent new security measures to trust that execution will be private and the outcomes are integral.
- Depending solely on cryptography is not sufficient because the decryption itself will be done on untrusted platforms.

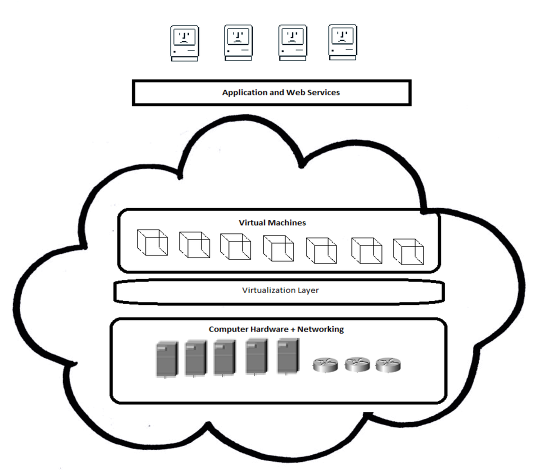

Figure 1: Cloud Computing Architecture

- Side-Channel attacks are one of the major threats in the cloud environment.
- The idea behind them is to analyze usage patterns and/or their timing to get information about code behavior.
- Utilize this information to reverse engineer or tamper with the code.
- This attack could be launched by a malicious insider or even a third party impersonator.

Current Approaches

Title	Basic Idea	Critique
“Twin Clouds “ by Bugiel, et al	• The Trusted Private Cloud : for evaluating encrypted security critical data. • Commodity public cloud :for computing time critical queries in parallel under encryption in the query phase.	• Increased the cost of infrastructure. • Hardware cost for encrypted execution “garbled circuit”. • More communication cost among the clouds.
“Hypervisor Security “ by McCune, et al	• RTM “Root Trust Management”. • Chain of trust. • CA “Certificate Authority”.	• Costly start overhead. • Side-channels attacks. • Certificate authority as a central point of failure. • Sabotage attacks via buffer and memory overflow.
“Secure Virtual Architecture” by Criswell, et al	• Add instructions for memory safety, type safety and control flow integrity. • Monitor all privileged operations. • Control physical resources.	• Eavesdropping types of attack. • Focus only on the instruction set beyond the code-generation phases- not utilizing LLVM’s JIT compiler.

Our Approach: Security By Obscurity

- Hiding the purpose, meaning, and operation of the code from attackers either humans or reverse engineering software.
- Gaming security by obscurity utilizing the information imbalance between the end-user and the service provider.
- Compilers offer a vast amount of semantic information which can be utilized for security objectives.
- Fortification and logical complexity together with the dynamic nature of the JIT compiler covers a wide range of attack vectors .
- Continually changing “What”, “When” and perhaps “Where it’s done too”.
- Our focus is on the execution phase against side-channel attacks; we are not currently concerned with securing the JIT compiler itself.

System Operation

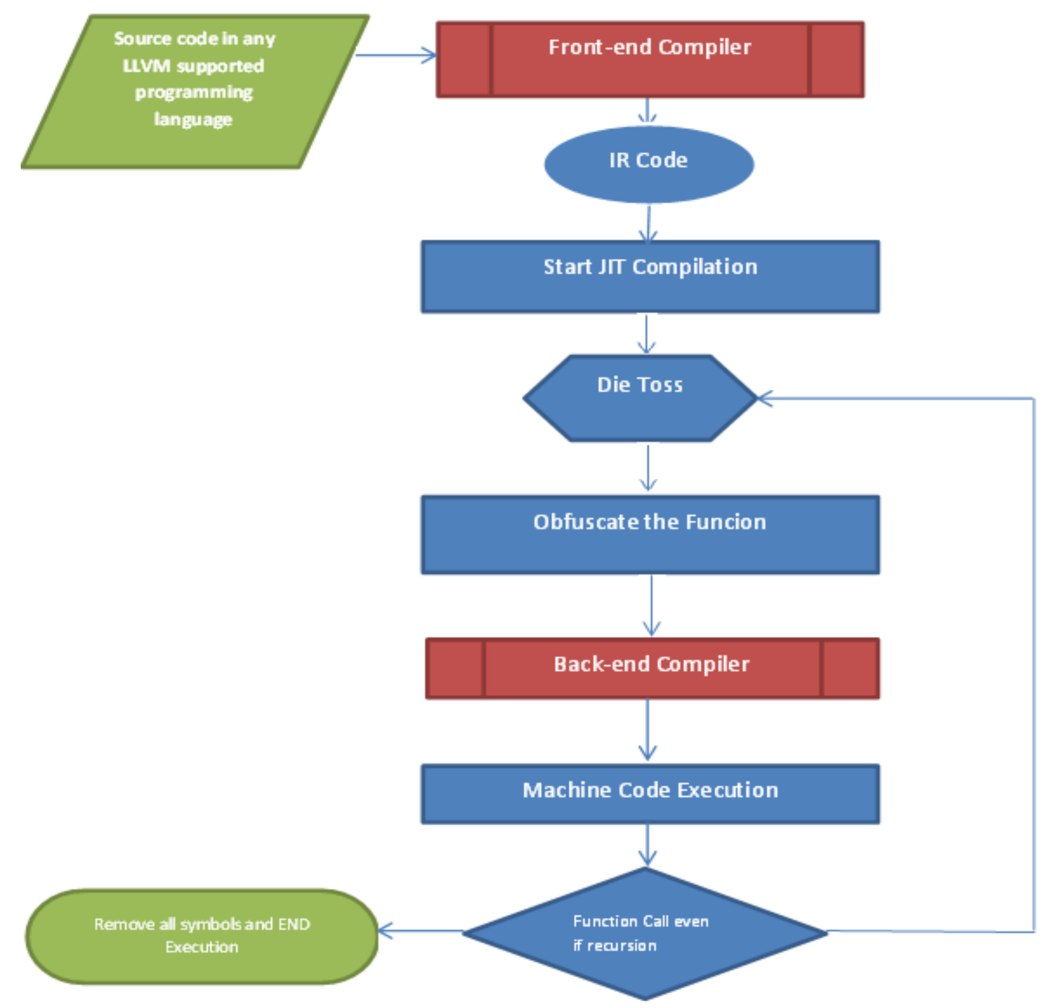

Figure 2: Flow Chart of OJIT System Operation.

- We are currently focusing on JITed code obfuscation.
- *Code morphing* – i.e multi-versioning of the same code with same functionality
- Dynamic switching - Jump around between these ever changing versions
- We modified the Execution Engine of LLVM forcing it to lazily call the JIT compiler every time a function is invoked.
- Every case of recursion is treated as a new function call.
- OJIT mainly works on a function call passes (Trampoline Call) as a trigger for recompiling a piece of code.
- Every function call results in a random order set of transformation passes applied to it.
- We can also extract loops as recursive function calls.
- Thereby we made the entire program as a series of function calls
- A strong random number generator forces unexpected code version “O(N^K) ”, N is the no. of transformation, size of pass set.

Evaluation Metrics

- We selected a concrete set of metrics to evaluate and assess the obfuscation strength of the system.
- We collected information about the number of instructions before and after every obfuscation step.
- We also deduce the cyclomatic number and the knot count to measure the complexity in the Control Flow Graph.
- We introduced a new obfuscation metric to measure how different is every code version as compared to its predecessor during the dynamic code-morphing/obfuscation phase.
- This is expressed in terms of a similarity percentage of the Longest Common Subsequence between the two code versions.

Experimental Analysis

We tested our system on a recursion intensive program that is the Bzip2 benchmark available in the SPEC CPU 2006 suite.

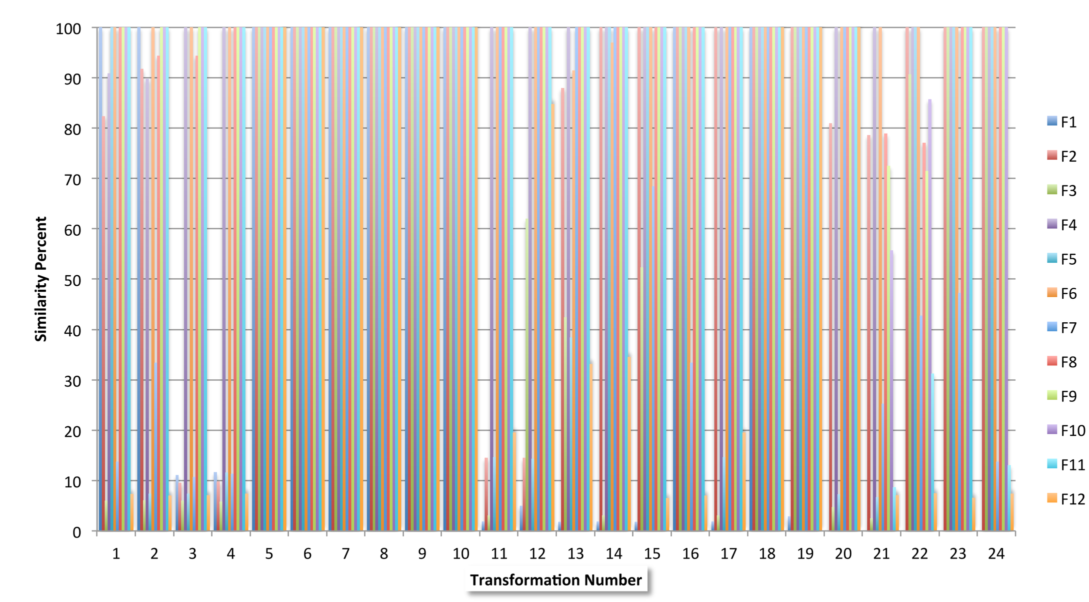

Figure 3: LCS between successive versions of the various functions of the Bzip2 benchmark.

Conclusions

- Cloud Computing is now facing what the first banking systems faced: trust issues, privacy concerns and reasonable security doubts.
- Security by Obscurity has been around and misused for some time – we hope to bring the right side.
- By utilizing the dynamic nature of JITed compiler we only hope that we added an insufferable burden on a reverse engineering or tampering malicious insiders.

[1] mohamed.hataba@ejust.edu.eg