Flexible Cryptanalysis in Java

Antonio Marcos de Oliveira Candia

Abstract - Cryptanalysis is an important tool in computer forensics. Various approaches have been used to implement high-performance computational cryptanalysis; however, they fail to provide an easy-to-use, portable and flexible solution. To achieve these goals we introduce the Quebra-pedra framework. This framework provides dictionary and brute-force attacks and uses standard Java features to provide these requirements simply and, at the same time, achieves performance levels similar to those of C/C++ programming languages by using native code.

Key Words: computer forensics, computational cryptanalysis, distributed computing, routing protocols.

I. INTRODUCTION

The search for evidence of criminal acts using computer forensics frequently includes the retrieval of encrypted information. When there is no cooperation in providing the key to decrypt the data, cryptanalysis techniques must be employed in the search.

Among the cryptanalysis techniques, known as attacks, some are for general-purpose use. The most generic techniques are, respectively, the complete key-space search – or brute-force – and the dictionary attack. These attacks demand enormous computational resources because of their extensive trial-and-error nature. Therefore, to be useful in the time frame of a criminal investigation, they require the use of high performance computing.

On the other hand, there are numerous algorithms used to encrypt data, some in public domain and highly documented, and some proprietary and protected by patents or restrictive use licences. Because of this, the cryptanalytic systems – at least those that are publicly available – are capable of attacking only a limited number. Thus a more flexible solution is needed.

To enable the attacks in these cases, a programming framework (the Quebra-pedra) was developed. This framework is designed to possess as its main characteristics flexibility, portability and performance. This paper shows how the Java programming language was used to obtain these characteristics.

II. Computational Cryptanalysis

Some applications that are available publicly provide computational cryptanalysis. L0phtCrack (LC) [1], and John The Ripper [2] are two of the
well-known applications that are capable of brute-force and dictionary attacks. However, by default, they are programmed to attack only a few cryptographic algorithms and are designed to search only for passwords to access protected operating systems. It is possible to extend them to any algorithm, but this extension requires some programming and access to the source code of those cryptographic algorithms. These problems render these applications inadequate to immediate use with unknown cryptographic algorithms. To be useful in computer forensics, a tool that is not bounded by these problems is needed.

The design of such a system must begin by the attacks that it supports. Below, we will describe some details of the general-purpose attacks cited previously.

The complete key-space search, or brute-force attack, has the main characteristic of always locating the key, using the cryptographic algorithms used today because it is based on the premise that all of the keys are tested. However, in most cases, the key-space is very large, making this attack a non-viable solution in a traditional computer system. For example, considering a key-space formed by the 26 letters of the alphabet, the 10 algarisms and the blank space character, the possible combinations for 1 to 8 characters would be about $361,000,000,000,000$. Assuming that we have a hypothetical system capable of testing 1000 keys per second, we would need about 1,022,778 hours or 114.5 years to search the entire key-space. It should be noted that the punctuation characters were not included in this example but in the real world they are frequently used in the keys and each character inserted in the former calculation would exponentially enlarge said key-space.

The dictionary attack is a variation on the brute-force attack that tries to reduce the total time of the search. In this attack, the cryptanalyst uses a dictionary containing words to be tested as keys to the encrypted data under attack. Some systems also implement a set of rules that modify the base dictionary being tested and, in this way, extend the dictionary to achieve a higher probability of finding the correct key. This attack is based on the premise that a large number of users chooses their keys among words of common use, or variations of these words. Obviously, the success of the attack depends on the correct key being present on the dictionary. To enhance the probability of success, a good dictionary is required. For instance, the dictionary available in a standard Unix system has approximately 235,000 words. Using the same parameters of the example cited previously, we can say that four minutes are necessary to test the entire dictionary. Each modification added to this dictionary will require four additional minutes to be tested. In this case, it is the cryptanalyst’s job to choose the enhanced modifications to improve the probability of success in finding the correct key, keeping in mind that each modification will be tested in a short time.

To allow a simplified cryptanalysis mechanism, the design and implementation of a extensible, portable, high-performance and low-cost framework was required. These are the basic premises upon which the Quebra-pedra framework has been built.

### III. Quebra-pedra: a general-purpose cryptanalysis framework

Quebra-pedra (qp) is a general-purpose cryptanalysis Java framework based on key-space search. This framework provides brute-force and dictionary attacks, and can be easily extended to any cryptanalysis problem that can be attacked in this way. A stand-alone application – qpsq – has been implemented using the framework to verify the potential of the platform. Figure 1 shows a basic diagram of this implementation.

![Figura 1 Class diagram of a stand-alone key-search application using the Quebra-pedra framework](image)

The framework is based on the following principles:

- execution platform independence (compatibility/interoperability);
- cryptographic algorithm independence (extensibility).
Platform independence is important because it allows any available processor to be used in the search for the key. Therefore, this independence allows the cryptanalyst to choose only those processors best suited to the job. To obtain this independence, the Java programming language was used. This language has been ported to virtually any processor available today and was, therefore, a natural alternative to the implementation of the framework.

In the early design stages it became clear that cryptographic independence would be a key characteristic of the framework. As stated previously, the existing systems are not easily adaptable to new algorithms. In computer forensics this is a highly desirable feature of a cryptanalytic system. In Quebra-pedra this was solved by using standard Java features.

The framework uses a simple mechanism based on two concepts introduced in the Java language, primarily in version 1.3: reflexive code and Java native interface [5].

Reflexive code is a concept that allows, among other things, the dynamic load of code in runtime. This allows, therefore, the modification of a system to enhance his capabilities on-the-fly, without the need to recompile it. The Java native interface (JNI) allows the use of native code written in C/C++ language inside a Java system. This code only needs to be in dinamic library (DLL in MS-Windows, .so in Unix systems) form.

The use of these concepts together allows, therefore, the dynamic load of native code without the need to recompile the cryptanalytic system. This can be better understood in the following hypothetical situation: a cryptanalyst comes across some encrypted data and, after an initial analysis, concludes that a proprietary algorithm has been used, and the method of encryption is unknown. In this situation the analyst can utilize the same libraries used by the encryption program, if they are available, to search for the key. It is only necessary that the analyst implement an adaptor, comprised of a few lines of Java code, and the libraries can be used directly in the cryptanalytic system. Therefore, this model of code reuse eliminates the reprogramming/recompiling of the application task, rendering the adaptation easy and rapid.

The adaptor is a small Java class that must implement a specific interface (qpAlgorithm in figure 1). Figure 1 also shows two such adaptors: qpcrypt and qpmd5. Some examples come with Quebra-pedra to make the implementation of the adaptors an easier task.

As a bonus, this method isolates the cryptanalyst from other details on the cryptanalytic system, as communication protocols and distribution of tasks mechanisms. Additionally, by design, the verification method is decoupled from the framework, and rather is in the body of the adaptors. This means that the analyst can use the same system to perform simple tasks such as password searches, or more complex tasks, as searching for keys to text decryption, zipped archives and data fragments.

The framework also provides some useful characteristics such as a successes cache and text recognitions, and will provide in the near future additional features including automatic checkpointing of execution states. The successes cache is a database of all the previous success key searches and is verified each time a search is initiated. If the key is already in the database, the work will not have to be conducted again, thus saving resources. The text recognition features are currently very simple, but very powerful. The framework does have a Sinkov log-likelihood [8] implementation. This particular implementation computes the digraphs occurrence in texts, and is the most powerful statistical test for this kind of problem.

IV. Performance of the system

Java has been categorized frequently as a low performance programming language, however, no study has been conducted to evaluate its performance as a cryptanalytic tool. To address this, some performance measures have been done with the qpseq implementation. These measures were based on the execution of key searches by brute-force and dictionary attacks with the DES algorithm and the results found are shown on Table 1.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time relative to C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java with Hotspot</td>
<td>~8</td>
</tr>
<tr>
<td>Java with JNI (C native code)</td>
<td>~1.15</td>
</tr>
</tbody>
</table>

Table 1: Comparison of execution times between the qpseq and a C implementation

The measures show to the DES algorithm that the JVM with Hotspot technology was about eight times slower than the reference implementation, and that the version that uses C code through JNI was only about 1.15 times slower. Preliminary results with other algorithms show the same
tendency. The system used in these tests was the Sun JRE 1.4.2, the gcc 2.8 C compiler, the Linux operating system, with the 2.4.22 kernel, all running in a Pentium IV 2.4GHz computer.

From these tests we concluded that the purely interpreted Java language is too slow to be useful as a cryptanalytic tool, but when the native code is used through JNI, it is an adequate alternative to other languages. If other characteristics such as ease of programming and dynamic load through reflexive code are taken into account, the little overhead demonstrated by Java becomes irrelevant.

V. High-performance cryptanalysis: distributed processing

A typical personal computer is underused because it remains in an idle state most of the time. These processors can be used to run parallel tasks that require a great deal of computing power. Some systems use this approach of sharing idle CPU time: SETI@Home [3] and its variations [4] are well-known examples. This system architecture is considered today to be one of the subcategories of grid computing. [1,4]

A distributed cryptanalytic application is being implemented using the Quebra-pedra framework and the ProActive middleware. This middleware is part of the ObjectWeb project, an actively developed open-source community hosted in INRIA, a french computing science research institute. ProActive is a GRID middleware (a Java library with an open source license) for parallel, distributed, and concurrent computing, also featuring mobility and security in a uniform framework [6]. Preliminary results show significant advantages in using this middleware. A better study is needed to quantify this advantage.

VI. Conclusions

Despite the performance issues, the Java language can be used to obtain a cryptanalytic system that is portable, extensible and can achieve good levels of performance through the use of native (non-Java) code. This paper demonstrates how simple concepts that are part of the language were used to obtain such characteristics in a cryptanalysis framework – the Quebra-pedra. A grid computing application is being developed using this framework and the ProActive library to further investigate whether Java can be of practical use in high-performance cryptanalysis.

References