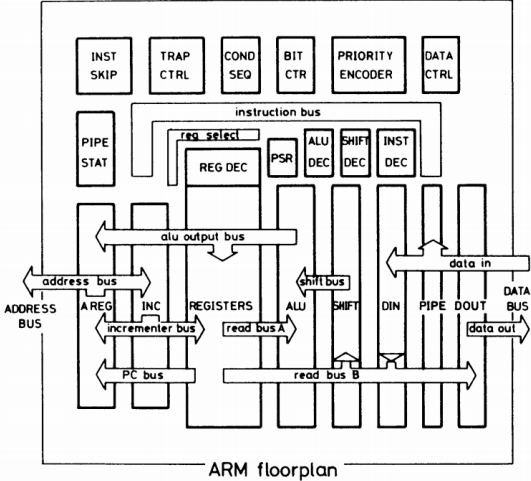
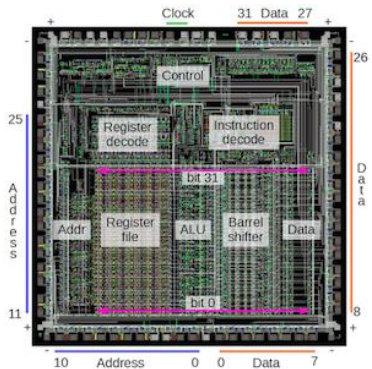


# The Case of the Leaky Winden

## Examination Log

Examiners: Luís Aguiar 80950, Jorge Pereira 81428, Filipe Azevedo 82468

Date / Time	Description
20-09-2017, 8h30	Both Jorge, Filipe and Luis downloaded the zip file from the course page, and extracted it, inside the Kali Linux operating system.
8h35	They calculated the md5 of each file, and verified that the files were not changed before they were delivered to them. They then started opening the files, including the zip file (without success, because it was protected by a password), and the python program.
8h45	Luis and Jorge started to investigate the python program ( <b>compress.py</b> ). Meanwhile Filipe was trying to find out the password of the zip file.
10h00	<p>Filipe was able to perform a dictionary attack using <b>fcrackzip</b>. This was done after arranging the words of the first book of Lord of the Rings (LOTR-Part1.txt, also included in the files that were delivered to us) one per line, ordering them alphabetically and removing duplicates.</p> <p>Below is the sequence of commands used:</p> <pre>grep -o -E '\w+' LOTR-Part1.txt &gt; dictionary.txt sort dictionary.txt   uniq &gt; passwords.txt fcrackzip -v -D -p passwords.txt online_banking.zip</pre> <p>After this sequence of commands, it was possible to find out that the password of the zip file was "Frodo" and that it contained the following files:</p> <ul style="list-style-type: none"><li>• <i>banking.docx</i> (md5: b70702822417bd39a7997a0f8c73941f);</li><li>• <i>arm1_floormap.bmp</i> (md5: f233a382d996fcc58601af79af167f2b).</li></ul>
13h00	<p>Luis decompiled the <b>compress.py</b> program using the <b>uncompyle2</b> tool (that can be found here: <a href="https://goo.gl/iyQLK1">https://goo.gl/iyQLK1</a>, or at the annexed files (inside the Auxiliary Items) with md5: d291885262b95f04082cf0fe868be87c) executing the following command:</p> <pre>uncompyle2 compress.py &gt; compress_decompiled.py</pre> <p>He commented line 91 (otherwise, the program would not run) and the result is the file named:</p> <ul style="list-style-type: none"><li>• <i>compress_decompiled.py</i> (md5: 6de42c79f6b476e8bfc8f25cac09b553).</li></ul> <p>Luis began studying its behavior.</p>
15h30	Jorge started helping Luis understanding the behavior of the program, and made the first version of the program that performs the inverse (that did not work).
15h35	Filipe ran the <b>file</b> tool to check if the type of files found inside the zip were according to their name. He concludes that the Microsoft Word document was actually a Microsoft Word document, however the image appeared as a data file.

15h40	<p>Filipe tried to open the image and realized that the header was destroyed, and after consulting a list of magic numbers online (<a href="https://goo.gl/yzGwwV">https://goo.gl/yzGwwV</a>) he noticed that in the place where it should be 89 50 4E, it was 00 00 00. And that image instead of being a BMP, as the name inferred, was a PNG. So he used <b>ghex</b> to replace the first 6 hexadecimal digits of the image and changed the name of the file to:</p> <ul style="list-style-type: none"> <li><i>arm1_floorplan.png</i> (md5: bf3ab0d53c35353b0d74eb7c2cc2e5de)</li> </ul> <p>The image appears to be the sketch of a processor. After that, Filipe decided to start analyzing the Microsoft Word document.</p>  <p>The diagram is a block diagram of an ARM processor architecture. At the top, there are control blocks: INST SKIP, TRAP CTRL, COND SEQ, BIT CTR, PRIORITY ENCODER, and DATA CTRL. Below these is the instruction bus, which connects to a REG DEC (register decoder) block. The REG DEC is connected to a set of REGISTERS. The REGISTERS are connected to an ALU (Arithmetic Logic Unit) and a SHIFT bus. The ALU is connected to an ALU output bus. The SHIFT bus is connected to a SHIFT DEC (shift decoder) block. The SHIFT DEC is connected to a DIN (data input) block. The DIN is connected to a PIPE (pipeline) block. The PIPE is connected to a DOUT (data output) block. The DOUT is connected to a DATA BUS. The DATA BUS is connected to a data in block and a data out block. The diagram also shows an address bus, a PC bus, and an incrementer bus. The entire diagram is labeled 'ARM floorplan' at the bottom.</p>
15h45	<p>Filipe did not find anything strange inside the word document (even after he extracted it). The file only contained the string "Password: 51782" written. So, he started helping Jorge and Luis trying to understand the <i>compress_decompiled.py</i> program.</p>
17h00	<p>Jorge finally made a version of the program to revert the <i>compress.py</i> that worked. It was called <b>obtain.py</b>. To run it is necessary to execute the following command:</p> <pre>python obtain.py name_of_image.png &gt; output</pre> <p>Then he decided to run it with the <i>rossio.png</i> file. He analyzed the output with <b>ghex</b> and concluded that the file was a PNG image. So, he changed the name of the file from <i>hidden_rossio.txt</i> to <i>hidden_rossio.png</i>, tried to open it and the image on the right was obtained.</p> <p>The final version of the tool used to recovery the hidden information in the <i>rossio.png</i> file was named <b>obtain.py</b> and her md5 is:</p> <ul style="list-style-type: none"> <li><i>obtain.py</i> (md5: 9dd531b3c4013274f3a05fa5ef2e94d6).</li> </ul> <p>The image recovered from the <i>rossio.png</i> file has the following name:</p> <ul style="list-style-type: none"> <li><i>hidden_rossio.png</i> (md5: 49f1a931bd2e74f24082cf997292cd27).</li> </ul>  <p>The image is a detailed, colorful floorplan of a processor. It shows various components like the Control unit, Register decode, Instruction decode, Addr, Register file, ALU, Barrel shifter, and Data. The floorplan is labeled with 'Clock' at the top, 'Address' on the left, and 'Data' on the right. The bottom of the floorplan is labeled with '10 Address 0 0 Data 7'.</p>

18h00	<p>Luis then decided to run the tool again upon the <i>lisbon.png</i> image. However, this time he didn't succeed. Then he tried running the tool one more time, now with the <i>jeronimos.png</i> image. And he obtained a picture of what looks like the "Torre de Belém".</p> <ul style="list-style-type: none"> <li>• <i>torre_belem.png</i> (md5: 155460b6fc3ee0a2c9ee63a3dc28b535).</li> </ul> <p>But he decided to run the tool again upon the <i>torre_belem.png</i> file, and he obtained a text file containing the following text:</p> <p>"ARM1 chip</p> <p>The ARM1 chip is built from functional blocks, each with a different purpose. Registers store data, the ALU (arithmetic-logic unit) performs simple arithmetic, instruction decoders determine how to handle each instruction, and so forth. Compared to most processors, the layout of the chip is simple, with each functional block clearly visible. (In comparison, the layout of chips such as the 6502 or Z-80 is highly hand-optimized to avoid any wasted space. In these chips, the functional blocks are squished together, making it harder to pick out the pieces.)</p> <p>The diagram below shows the most important functional blocks of the ARM chip.[2] The actual processing happens in the bottom half of the chip, which implements the data path. The chip operates on 32 bits at a time so it is structured as 32 horizontal layers: bit 31 at the top, down to bit 0 at the bottom. Several data buses run horizontally to connect different sections of the chip. The large register file, with 25 registers, stands out in the image. The Program Counter (register 15) is on the left of the register file and register 0 is on the right.</p> <p>Computation takes place in the ALU (arithmetic-logic unit), which is to the right of the registers. The ALU performs 16 different operations (add, add with carry, subtract, logical AND, logical OR, etc.) It takes two 32-bit inputs and produces a 32-bit output. The ALU is described in detail here.[4] To the right of the ALU is the 32-bit barrel shifter. This large component performs a binary shift or rotate operation on its input, and is described in more detail below. At the left is the address circuitry which provides an address to memory through the address pins. At the right data circuitry reads and writes data values to memory."</p> <p>The file extracted from <i>torre_belem.png</i>:</p> <ul style="list-style-type: none"> <li>• <i>hidden_belem.txt</i> (md5: e406087f8689534997055b2eb2a64808)</li> </ul>
18h05	<p>Filipe tried running the <b>obtain.py</b> tool on the rest of the images but without success.</p>

23-09-2017, 17h00	Jorge and Filipe remembered that the files generated by the <b>compress.py</b> all have an RGBA color scheme.
17h05	Jorge and Filipe ran the <b>pngcheck</b> tool over all the images and concluded that both <i>lisbon.png</i> , <i>jeronimos.png</i> and <i>rossio.png</i> have an RGBA color scheme while the <i>electrico.png</i> has an RGB color scheme. Therefore, they decided to focus their efforts on the <i>lisbon.png</i> file.
24-09-2017 15h00	Luis made a second version of the program <b>obtain.py</b> , which did not need to receive an image as input to run with the result of running the <b>obtain.py</b> tool with the <i>lisbon.png</i> image, but did not get any success.
15h30	<p>Jorge and Luis remembered the password discovered by Filipe within the Microsoft Word document, and remembered that the program used to hide data inside images, when started with a password, caused a jump in the length of module 13 of the same, multiplied by 2 (the last two bits of each color channel – i.e. 3, because of RGB). That is, in this case the password, as it was 51782 causes a jump of <math>(51782 \% 13) * 6</math>. So, they made a tool that allows you to automate getting the message hidden in the <i>lisbon.png</i> image, considering that password. And the following information was obtained.</p> <p>“ACCESS CODES</p> <p>SERVER 1: Sr!_01llxt</p> <p>SERVER 2: p_GETKI4dA”</p> <p>The complete file extracted from lisbon.png:</p> <ul style="list-style-type: none"> <li><i>hidden_lisbon.txt</i> (md5: 68a439fadc22bf501c21e01e4ae55100).</li> </ul> <p>And the tool:</p> <ul style="list-style-type: none"> <li><i>obtain_lisbon.py</i> (md5: 32ca4458829617dd81d7a06b0ea9cf90).</li> </ul>