WhatsApp monitoring for Fun and Stalking

by Flavio Giobergia

The idea behind the project

If you have used WhatsApp at least once in your life, you know that there is a really lovely/shitty (depending on the situation) feature that comes with it: you can see whether the person you are talking to is typing, online or offline and, in the latter case, when this person was last seen online.

There is something really interesting about this feature: you can see such information even though the other person does not know you and has never talked to you before. All you need is this person's phone number. This happens because the value of the “Last seen” option (Settings – Account – Privacy – Last seen) is defaulted to “Everyone”. Bear in mind that, even if you change this option, others will still be able to know whether you are online or not.

Once you know someone's activity on WhatsApp you may start to deduce a lot about this person, assuming that the target uses WhatsApp on a daily basis: an intensive use of WhatsApp can be usually associated with leisure time, while sporadic connections are usually due to work/school or whatever keeps one busy – but not too much. A complete absence of WhatsApp activity can be either due to a task that requires one's every attention, like driving, or maybe the target is sleeping. Many other things can be deduced assuming that enough logs are collected.

Please keep in mind that the data that can be collected about somebody might be insufficient to draw any conclusion: this might be because of a lack of data (i.e. the person does not use WhatsApp frequently enough) or one's time spent on WhatsApp does not follow any precise pattern (i.e. the person uses WhatsApp only when needed, and does not spend hours and hours talking rubbish).

This paper will explain how to create a Proof of Concept that will collect someone's WhatsApp logs.

Collecting the data

There is a number of different ways of collecting the logs. Since all I am talking about in this paper is a Proof of Concept, I chose to follow the easiest way possible. This results in a raw, inelegant solution.

At first I did not know much about how WhatsApp worked, but I was already expecting that I would have never found any official API. I thought of using the unofficial API, but that would have meant reading some documentation, and we all know that reading documentation is boring. The other options were to use a packet sniffer to read the communications between server and client, but it seemed to be encrypted. Learning how XMPP works was off the table: why spending time reading documentation for a protocol that, although related to, is different from the one used by WhatsApp? I could decompile the apk, but I was looking for something more immediate: if I were to decompile it, I would have had to a) understand the code, b) locate the part of interest, c) adapt said part to my purposes and d) recompile it. Sounds like a lot of work, does it not?

The number of possible ways of doing it was getting smaller, but I eventually came up with the laziest and easiest to implement solution. Since all I needed was a short text displayed on the screen, why not taking a screenshot, and then reading the part of interest? I have already worked a few times with OCR applied to captchas, so I already knew pretty much everything that needed to be done. It is actually easier to read a screenshot, rather than a captcha: when working with the latter, you also have to mind the noise introduced to make it hard for bots to read the images.

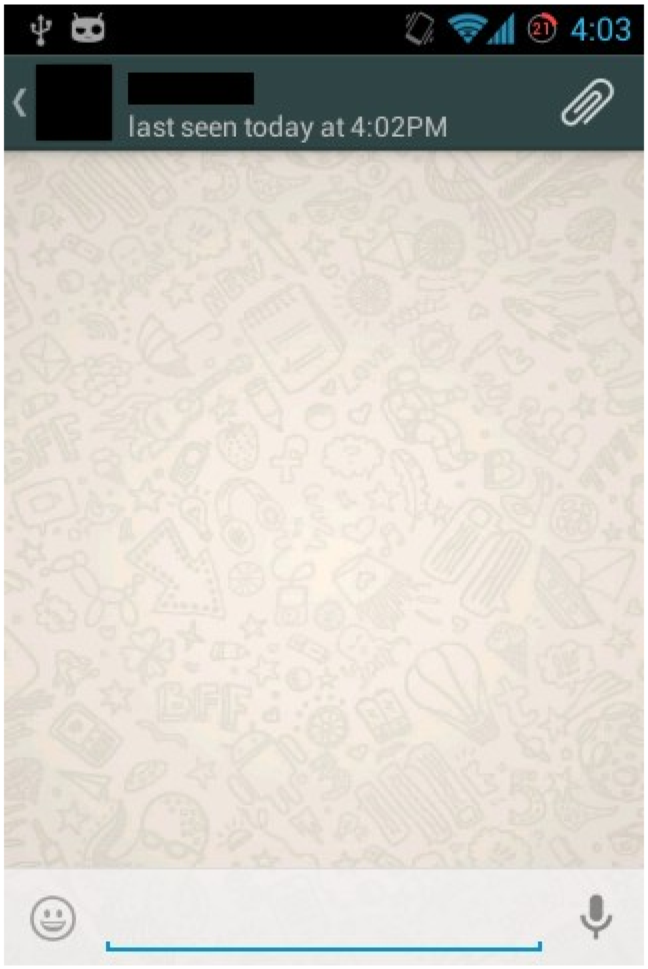
Capturing the screenshot

There are a few ways to take a screenshot from an android device. Some of them require the device to be rooted. Although this is not a problem in my case, I decided to use another way: since I needed an easy-to-write way for taking screenshots for this PoC, I decided to use `adb`, directly from a computer.

The command needed to save the screenshot on a computer is rather simple, and was found on this website.

```
adb shell screencap -p | sed 's/\r$//' > screen.png
```

One small detail that is actually important for this command to work is that the device has not to be sleeping. Luckily enough, among the Developer options there is the “Stay awake” option which, if enabled, will not let the screen sleep while charging. Since the device will be connected to the computer via an USB cable, this is exactly what we are going to need. This is what the screenshot will look like:



From PNG to PPM

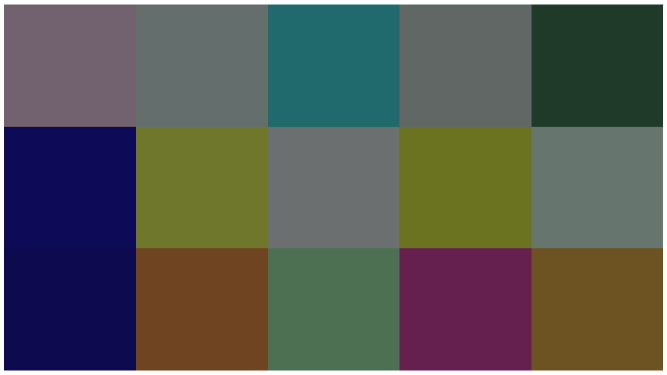
The screenshot is now available for manipulations. The format it is stored in is PNG: although PNG can be handled rather easily, I decided to convert the image to an even easier format: PPM. For those of you who have never heard of it, PPM is a lossless format, which can be easily read by humans, although absolutely not efficient space-wise. For the conversion, `convert` is used.

```
convert screen.png screen.ppm
```

Now, a brief explaination of how the PPM format (P6) works. The first line of the file contains the “P6” string, which is used to identify the format. The following line contains the width and the height of the image, and then a following line indicates the max color (in our case, 255). The rest of the file contains the triplets R, G B for every pixel in the image. R, G and B are 1 byte each, and are expressed using ASCII characters.

```
P6
5 3
255
random image :)
Wow, looks fun
OnE MoRe PlS!
```

Will produce this 5x3 pixels image (here enhanced for your viewing pleasure)



This is because the hex for “`random image :)\r\nWow, looks fun\r\nOnE MoRe PlS!`” is

```
72 61 6e 64 6f 6d 20 69 6d 61 67 65 20 3a 29 0d 0a 57 6f 77 2c 20 6c 6f 6f
6b 73 20 66 75 6e 0d 0a 4f 6e 45 20 4d 6f 52 65 20 50 6c 53 21
```

That is `#72616E` (first row, first column), `#646F6D` (first row, second column), `#20696D` (first row, third column) and so on.

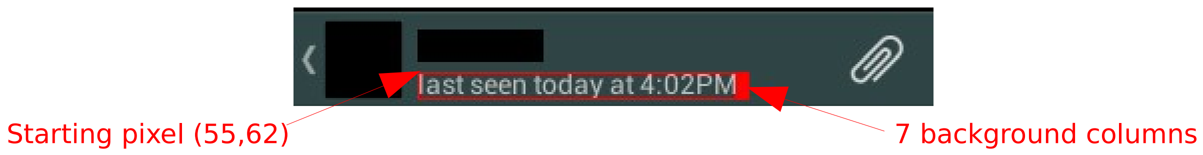
PPM Manipulation

Now that the PPM image is available, it can be read straightforwardly. The device I am using is a Samsung Galaxy SII mini, the resolution of the screenshot will always be of 320x480; therefore every screenshot, converted to PPM, will start with

```
P6\n
320 480\n
255\n
IMAGE
```

This means that the actual image will start 15 bytes after the beginning of the file. If we read the content of the file and store it in an array `a`, the first pixel's R, G and B bytes are contained in `a[15]`,`a[16]` and `a[17]` and, in general, the pixel located at the y-th row and x-th column is described by a[y*WIDTH+x+15], `a[y*WIDTH+x+16]`,`a[y*WIDTH+x+17]`, `WIDTH` being the width of the image, i.e. 320 pixels.

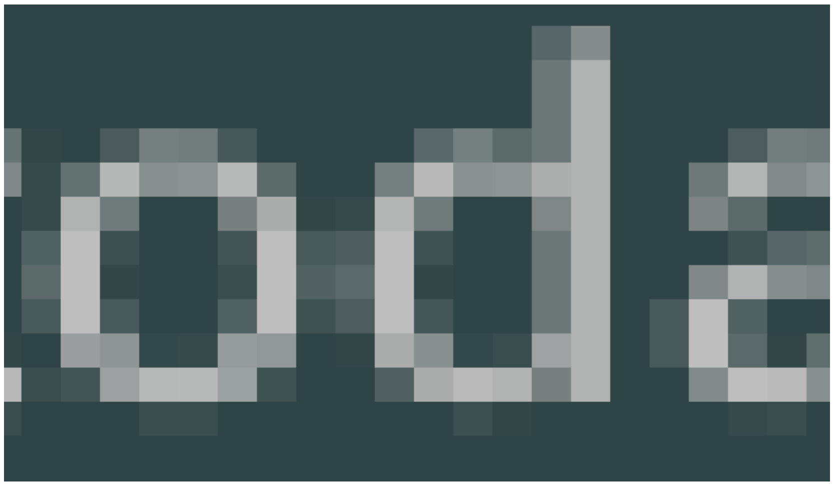
The next step consists in finding out where the text of interest (i.e. the “last seen” part) is kept. After thorough investigation, I came up with the conclusion that the text is stored in a rectangle of pixels having fixed height, but variable width (depending on the length of the text): the top left corner is at the 55th column, 62nd row, and the height is of 14 pixels. The width can be found out from time to time. The background color is `#2F4444`. When a fixed number of columns only containing background pixels is found, the text can be considered as ended. This fixed number of columns can be decided so that it is larger than the distance between adjacent words. I decided to use 7, and it seems to work just fine.



Each of the pixels can be accessed the way it was shown before.

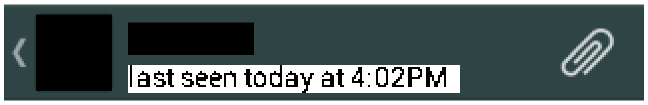
Reading the image

Before actually reading the image, there is one more step to do: it may appear that every letter in the text is written using a single color: this is actually not true. When zooming the image, it is clear that different, although not that much, colors are used:



Before reading the image, it would be great if we could convert it to b/w. Since we know both the background color (`#2F4444`) and the text's brightest color (`#B2B4B4`) we can find the color "in the middle": (2F+B2)/2, (44+B4)/2, (44+B4)/2 = `#707C7C`. A pixel P (R_P, G_P, B_P) can be considered as part of the background if $R_P \leq 70$ && $G_P \leq 7C$ && $B_P \leq 7C$; it is part of the text otherwise. It basically means that if a pixel is darker than the average of the background and the text colors, it is considered as part of the background.

Based on this assumption, the image is interpreted as follows:


[redacted]
[redacted]
last seen today at 4:02PM

This might seem poorly readable by any human being (at least, it is when compared to the original text), but let's keep in mind that this text will be read by a computer.

Now, we are almost ready. The last thing we need to do is deciding how to know when one letter ends and another begins. Based on the font, it is fairly safe to assume that a blank column is a good separator between characters. There are some cases that do not follow this rule, such as the 'st' sequence. Let's not give too much weight to this now; we will see later how to solve this problem.

We are now capable of taking a screenshot, finding the text we are interested in, converting it to b/w and splitting the rectangle in characters. What now?

To be honest, we are almost done: in case you did not think of it, every single character can be read as a matrix of 0s and 1s (0 being white and 1 black for instance). The fact that the text does not contain any noise leads to the fact that every single matrix representing the same character will be exactly the same, no matter what. This may seem obvious, but it is not: when working with captchas, this assumption can not be made, thus making captchas harder to read.

So, all that is left to be done is taking a few screenshots, reading them and saving the matrix for every new character found. Only a few characters are actually needed (mostly letters and digits). The following if the list of those characters:

```
a d e g i l n o p s t y A M P 0 1 2 3 4 5 6 7 8 9 / : . st ty
```

As mentioned before, there are some problems when splitting some characters. Since the problem is only with the sequences 'st' and 'ty' I decided to consider those chunks as if they were ordinary characters, rather than using some other more complicated splitting technique to separate them.

The following Ruby code can read a screenshot's "last seen" text using an hash containing the matrix for every character needed. The usage is straightforward, since you only need to place a "screen.ppm" image in the same directory as the script, and it will do the rest.

```
$ ruby script.rb
lastseentodayat4:02PM
$
```

The spaces are not recognized: it would not be hard to detect them (the script would only need to find multiple blank columns to know where a space is), but since it is no use for the script, all spaces are ignored.

```
WIDTH =  320
```

```
hash={
      'l'=>[[1,1,1,1,1,1,1,1,1,1,1,0,0,0]],
      'i'=>[[1,1,0,0,1,1,1,1,1,1,1,0,0,0]],
      'ty'=>[[0,0,0,0,1,0,0,0,0,0,0,0,0,0],
             [0,0,0,1,1,1,1,1,1,1,0,0,0,0],
             [0,0,1,1,1,1,1,1,1,1,1,0,0,0],
             [0,0,0,0,1,0,0,0,0,0,1,0,0,0],
             [0,0,0,0,1,0,0,0,0,0,0,0,0,0],
             [0,0,0,0,1,1,1,0,0,0,0,0,0,1],
             [0,0,0,0,0,0,1,1,1,1,0,0,1,1],
             [0,0,0,0,0,0,0,0,1,1,1,1,1,0],
             [0,0,0,0,0,0,1,1,1,1,0,0,0,0],
             [0,0,0,0,1,1,1,0,0,0,0,0,0,0],
             [0,0,0,0,1,0,0,0,0,0,0,0,0,0]],
      'st'=>[[0,0,0,0,1,1,1,0,0,1,0,0,0,0],
             [0,0,0,0,1,0,1,0,0,0,1,0,0,0],
             [0,0,0,1,1,0,0,1,0,0,1,0,0,0],
             [0,0,0,0,1,0,0,1,0,0,1,0,0,0],
             [0,0,0,0,1,1,0,1,1,1,1,0,0,0],
             [0,0,0,0,0,0,0,0,1,1,0,0,0,0],
             [0,0,0,0,1,0,0,0,0,0,0,0,0,0],
             [0,0,1,1,1,1,1,1,1,1,0,0,0,0],
             [0,0,1,1,1,1,1,1,1,1,1,0,0,0],
             [0,0,0,0,1,0,0,0,0,0,1,0,0,0]],
      's'=>[[0,0,0,0,1,1,1,0,0,1,0,0,0,0],
            [0,0,0,0,1,0,1,0,0,0,1,0,0,0],
            [0,0,0,1,1,0,0,1,0,0,1,0,0,0],
            [0,0,0,0,1,0,0,1,0,0,1,0,0,0],
            [0,0,0,0,1,1,0,1,1,1,1,0,0,0],
            [0,0,0,0,0,0,0,0,1,1,0,0,0,0]],
      'e'=>[[0,0,0,0,0,1,1,1,1,1,0,0,0,0],
            [0,0,0,0,1,0,1,1,0,1,1,0,0,0],
            [0,0,0,1,1,0,1,1,0,0,1,0,0,0],
            [0,0,0,0,1,0,1,1,0,0,1,0,0,0],
            [0,0,0,0,1,1,1,1,0,0,1,0,0,0],
            [0,0,0,0,0,1,1,0,0,0,0,0,0,0]],
      'n'=>[[0,0,0,0,1,1,1,1,1,1,1,0,0,0],
            [0,0,0,0,1,0,0,0,0,0,0,0,0,0],
            [0,0,0,0,1,0,0,0,0,0,0,0,0,0],
            [0,0,0,1,1,0,0,0,0,0,0,0,0,0],
            [0,0,0,0,1,1,0,0,0,0,0,0,0,0],
            [0,0,0,0,1,1,1,1,1,1,1,0,0,0]],
      't'=>[[0,0,0,0,1,0,0,0,0,0,0,0,0,0],
            [0,0,1,1,1,1,1,1,1,1,0,0,0,0],
            [0,0,1,1,1,1,1,1,1,1,1,0,0,0],
            [0,0,0,0,1,0,0,0,0,0,1,0,0,0]],
      'o'=>[[0,0,0,0,0,1,1,1,1,1,0,0,0,0],
            [0,0,0,0,1,0,0,0,0,1,1,0,0,0],
            [0,0,0,1,1,0,0,0,0,0,1,0,0,0],
            [0,0,0,1,1,0,0,0,0,0,1,0,0,0],
            [0,0,0,0,1,1,0,0,0,1,1,0,0,0],
            [0,0,0,0,0,1,1,1,1,1,0,0,0,0]],
'd'=>[[0,0,0,0,1,1,1,1,1,1,0,0,0,0],
       [0,0,0,0,1,0,0,0,0,1,1,0,0,0],
       [0,0,0,1,1,0,0,0,0,0,1,0,0,0],
       [0,0,0,0,1,0,0,0,0,0,1,0,0,0],
       [0,0,0,0,1,1,0,0,0,1,1,0,0,0],
       [1,1,1,1,1,1,1,1,1,1,1,0,0,0]],
'a'=>[[0,0,0,0,0,1,0,1,1,1,1,0,0,0],
       [0,0,0,0,1,0,0,1,0,0,1,0,0,0],
       [0,0,0,1,1,0,0,1,0,0,1,0,0,0],
       [0,0,0,0,1,0,0,1,0,0,1,0,0,0],
       [0,0,0,0,1,1,1,1,1,1,1,0,0,0],
       [0,0,0,0,0,1,1,1,1,1,1,0,0,0]],
'y'=>[[0,0,0,0,1,0,0,0,0,0,0,0,0,0],
       [0,0,0,0,1,1,1,0,0,0,0,0,0,1],
       [0,0,0,0,0,0,1,1,1,1,0,0,1,1],
       [0,0,0,0,0,0,0,0,1,1,1,1,1,0],
       [0,0,0,0,0,0,1,1,1,1,0,0,0,0],
       [0,0,0,0,1,1,1,0,0,0,0,0,0,0],
       [0,0,0,0,1,0,0,0,0,0,0,0,0,0]],
'9'=>[[0,0,1,1,1,1,1,0,0,0,1,0,0,0],
       [0,1,1,0,0,0,1,0,0,0,1,0,0,0],
       [0,1,0,0,0,0,0,1,0,0,1,0,0,0],
       [0,1,0,0,0,0,1,1,0,0,1,0,0,0],
       [0,1,1,1,0,1,1,0,1,1,1,0,0,0],
       [0,0,0,1,1,1,1,1,1,0,0,0,0,0]],
':'=>[[0,0,0,0,1,0,0,0,0,0,1,0,0,0]],
'1'=>[[0,1,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,1,0,0,0,0,0,0,0,0,0,0,0,0],
       [0,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,1,1,1,1,1,1,1,1,1,1,0,0,0]],
'4'=>[[0,0,0,0,0,0,1,1,1,0,0,0,0,0],
       [0,0,0,0,0,1,1,0,1,0,0,0,0,0],
       [0,0,0,1,1,0,0,0,1,0,0,0,0,0],
       [0,1,1,1,0,0,0,1,1,0,0,0,0,0],
       [0,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,0,1,0,0,0,0,0]],
'P'=>[[0,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,1,0,0,0,0,1,0,0,0,0,0,0,0],
       [0,1,0,0,0,0,1,0,0,0,0,0,0,0],
       [0,1,0,0,0,0,1,0,0,0,0,0,0,0],
       [0,1,1,0,0,1,1,0,0,0,0,0,0,0],
       [0,0,1,1,1,1,0,0,0,0,0,0,0,0]],
'M'=>[[0,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,1,1,1,1,1,1,1,1,1,1,0,0,0],
       [0,0,0,1,1,1,0,0,0,0,0,0,0,0],
       [0,0,0,0,0,1,1,1,0,0,0,0,0,0],
       [0,0,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,0,0,0,1,1,1,0,0,0],
       [0,0,0,0,0,1,1,1,1,0,0,0,0,0],
       [0,0,0,1,1,1,0,0,0,0,0,0,0,0],
       [0,1,1,1,1,0,0,0,0,0,0,0,0,0],
       [0,1,1,1,1,1,1,1,1,1,1,0,0,0]],
'5'=>[[0,0,0,1,1,1,1,0,1,1,0,0,0,0],
      [0,1,1,1,1,1,0,0,0,1,1,0,0,0],
      [0,1,0,0,0,1,0,0,0,0,1,0,0,0],
      [0,1,0,0,0,1,0,0,0,0,1,0,0,0],
      [0,1,0,0,0,1,1,0,0,1,1,0,0,0],
      [0,1,0,0,0,0,1,1,1,1,0,0,0,0]],
'0'=>[[0,0,1,1,1,1,1,1,1,1,0,0,0,0],
      [0,1,1,0,0,0,0,0,0,1,1,0,0,0],
      [0,1,0,0,0,0,0,0,0,0,1,0,0,0],
      [0,1,0,0,0,0,0,0,0,0,1,0,0,0],
      [0,1,1,0,0,0,0,0,0,1,1,0,0,0],
      [0,0,1,1,1,1,1,1,1,1,0,0,0,0]],
'8'=>[[0,0,1,1,1,0,0,1,1,1,0,0,0,0],
      [0,1,1,0,1,1,1,0,0,1,1,0,0,0],
      [0,1,0,0,0,1,1,0,0,0,1,0,0,0],
      [0,1,0,0,0,1,1,0,0,0,1,0,0,0],
      [0,1,1,1,1,1,1,0,0,1,1,0,0,0],
      [0,0,1,1,1,0,0,1,1,1,0,0,0,0]],
'3'=>[[0,0,1,1,0,0,0,0,1,1,0,0,0,0],
      [0,1,1,0,0,0,0,0,0,1,1,0,0,0],
      [0,1,0,0,0,1,0,0,0,0,1,0,0,0],
      [0,1,0,0,0,1,1,0,0,0,1,0,0,0],
      [0,1,1,0,1,1,1,0,0,1,1,0,0,0],
      [0,0,1,1,1,0,0,1,1,1,0,0,0,0]],
'7'=>[[0,1,0,0,0,0,0,0,0,0,0,0,0,0],
      [0,1,0,0,0,0,0,0,0,0,0,0,0,0],
      [0,1,0,0,0,0,1,1,1,1,1,0,0,0],
      [0,1,0,0,1,1,1,0,0,0,0,0,0,0],
      [0,1,0,1,1,0,0,0,0,0,0,0,0,0],
      [0,1,1,0,0,0,0,0,0,0,0,0,0,0]],
'/'=>[[0,0,0,0,0,0,0,0,0,0,1,0,0,0],
      [0,0,0,0,0,0,0,0,1,1,1,0,0,0],
      [0,0,0,0,0,0,1,1,1,0,0,0,0,0],
      [0,0,0,1,1,1,0,0,0,0,0,0,0,0],
      [0,1,1,1,0,0,0,0,0,0,0,0,0,0]],
'2'=>[[0,0,1,1,0,0,0,0,0,1,1,0,0,0],
      [0,1,1,0,0,0,0,0,1,1,1,0,0,0],
      [0,1,0,0,0,0,0,1,1,0,1,0,0,0],
      [0,1,0,0,0,0,1,1,0,0,1,0,0,0],
      [0,1,1,0,1,1,1,0,0,0,1,0,0,0],
      [0,0,1,1,1,0,0,0,0,0,1,0,0,0]],
'6'=>[[0,0,0,1,1,1,1,1,1,1,0,0,0,0],
      [0,0,1,1,0,1,1,0,0,1,1,0,0,0],
      [0,1,0,0,0,1,0,0,0,0,1,0,0,0],
      [0,1,0,0,0,1,0,0,0,0,1,0,0,0],
      [0,1,0,0,0,1,0,0,0,1,1,0,0,0],
      [0,0,0,0,0,0,1,1,1,1,0,0,0,0]],
'A'=>[[0,0,0,0,0,0,0,0,0,0,1,0,0,0],
      [0,0,0,0,0,0,0,0,1,1,1,0,0,0],
      [0,0,0,0,0,1,1,1,1,0,0,0,0,0],
      [0,0,1,1,1,1,0,1,0,0,0,0,0,0],
      [0,1,1,1,0,0,0,1,0,0,0,0,0,0],
      [0,0,1,1,1,1,0,1,0,0,0,0,0,0],
            [0,0,0,0,0,1,1,1,1,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,1,1,1,0,0,0]],
      'p'=>[[0,0,0,0,1,1,1,1,1,1,1,1,1,1],
            [0,0,0,0,1,0,0,0,0,1,1,0,0,0],
            [0,0,0,0,1,0,0,0,0,0,1,0,0,0],
            [0,0,0,1,1,0,0,0,0,0,1,0,0,0],
            [0,0,0,0,1,1,0,0,0,1,1,0,0,0],
            [0,0,0,0,0,1,1,1,1,1,0,0,0,0]],
      'g'=>[[0,0,0,0,0,1,1,1,1,1,0,0,0,0],
            [0,0,0,0,1,1,0,0,0,1,1,0,0,1],
            [0,0,0,1,1,0,0,0,0,0,1,0,0,1],
            [0,0,0,0,1,0,0,0,0,0,1,0,0,1],
            [0,0,0,0,1,1,0,0,0,1,1,0,1,1],
            [0,0,0,0,1,1,1,1,1,1,1,1,1,0]],
      '.'=>[[0,0,0,0,0,0,0,0,0,0,1,0,0,0]]

}

def bw(a,y,x)
      i=y*WIDTH+x
      (a[15+i*3].ord>112 && a[15+i*3+1].ord>124 && a[15+i*3+2].ord>124) ? 1
: 0
end

a = File.open("screen.ppm","rb").read

c=0
b=[]

62.upto WIDTH do |d|
      f = 0
      t=[]
      55.upto 68 do |i|
            t << bw(a,i,d)
      end
      unless t.include?1
            c += 1
            hash.each do |k,v|
                  print k if v == b
            end
            b = []
      else
            b << t
            c=0
      end
      break if c == 7
end

puts
```

Parsing the string

Now that we can easily retrieve the text from the image, there is little more to do to finally complete our script. Basically, we need to convert the text to a time. Being this a Proof of Concept, we are going to assume that the tests we are running will cover a day (12:00 AM to 11:59 PM). There is a number of different ways for us to store the fact that the target was either online or not at any given moment.

The first one that comes to mind is using an array of booleans `x` containing values for each minute of the day (24*60 = 1440 values), every single element will be defaulted to `false` . Then a screenshot is retrieved every 60 seconds, the content of the image's text is read and converted to an index `i`, representing the last time the target was seen online: the value `x[i]` will then be set to `true`. By the end, `x` will contain all the information, minute per minute.

There are mainly 2 different situations that can be found when reading the text: either the target is online, or not.

- The target is online when the text read is either “online” or “typing...”. In those cases, the index returned is the one associated with the current time.

- The target is not online when the text starts with the “last seen” string. In this case, the target was either last seen today or earlier. I will only cover the “last seen today” scenario, since the Proof of Concept is written so that it only analyzes the current day. Using some basic regex (`/^lastseentodayat([0-9]{1,2}):([0-9]{2})(A|P)M$/`) the string can be parsed, and the time retrieved. It can then be converted to the “1440 format” using some basic operations.

The following is the final script: it stores the activity in the `x` array, for further analysis.

```
$hash = { … }

def bw(a,y,x)
      i=y*WIDTH+x
      (a[15+i*3].ord>112 && a[15+i*3+1].ord>124 && a[15+i*3+2].ord>124) ? 1
: 0
end

def get_string
      `adb shell screencap -p | sed 's/\r$//' > screen.png`
      `convert screen.png screen.ppm`
      a = File.open("screen.ppm","rb").read
      c=0
      b=[]
      s = ""
      62.upto WIDTH do |d|
            f = 0
            t=[]
            55.upto 68 do |i|
                  t << bw(a,i,d)
            end
            unless t.include?1
```

```
                c += 1
                $hash.each do |k,v|
                     s << k if v == b
                end
                b = []
           else
                b << t
                c=0
           end
           break if c == 7
      end
      File.delete "screen.ppm"
      File.delete "screen.png"
      s
end

def last(str, time)
      if str == "online" || str == "typing..."
           return time
      end
      if m = str.match(/^lastseentodayat([0-9]{1,2}):([0-9]{2})(A|P)M$/)
           t = m[1].to_i*60+m[2].to_i
           t -= 12*60 if m[3]=="A"&&m[1].to_i==12
           t += 12*60 if m[3]=="P"&&m[1].to_i!=12
           return t
      end
      -1
end
m = t.min-1
d = t.day

while d == t.day
      t = Time.new
      unless m == t.min
           m = t.min
           now = t.hour*60+t.min
           # retrieve string
           str = get_string
           now = last(str,now)
           x[now] = true if now != -1
           puts "online @ #{now}"
      else
           sleep 1
      end
end

x = Array.new(false)
t = Time.new
m = t.min-1
d = t.day

while d == t.day
      t = Time.new
```

```
     unless m == t.min
          m = t.min
          now = t.hour*60+t.min
          str = get_string
          now = last(str,now)
          x[now] = true if now != -1
     else
          sleep 1
     end
end
```

You may have noticed that the Ruby code I am using is rather inelegant: I apologize about that, but I only have studied Ruby a few weeks, and this is the first actual script I have written using this scripting language.

Flavio Giobergia

flavio.giobergia@gmail.com

@FlavioGiobergia