AnalyzingMARCH

November 25, 2021

1 MARCH Tests

1.1 March D-nv-SRAM

In this example, we are investigating a 128kx8 SRAM that was exposed to radiation in March C & D tests. Hence, the content inside the memory was continuosly written and red. At any rate, from the point of view of statistical analysisis, this test is equivalent to a pseudostatic one, with two different patterns and with addresses in switching order.

1.2 Loading packages

The very first thing we must do is to load the packages required to load files (*DelimitedFiles*) as well as the LELAPE module. I suppose you have installed both. Load is done with:

[17]:
```
push!(LOAD_PATH,"/home/francisco/Escritorio/LELAPE-main/LELAPE/src")
using DelimitedFiles, LELAPE
```

1.3 Defining variables

Previous paragraph allows us to define several variables for checking the tests:

- Word width : 8 bits
- Memory size in words: 2M is just 2^21.
- In SRAMs, it seems more likely to succeed the XOR operation.
- Tests were pseudostatic. Therefore, it is intelligent to keep information about the different cycles.

Ok, let us use this information to set these variables:

[18]:
```
LA = 2^17 # Memory size in words
WordWidth = 8 # Selfexplaining.
Operation = "XOR" # Only "XOR" or "POS" are allowed.
KeepCycles = true # This is a Bool variable and only true false are accepted.
```

[18]:
```
true
```

1.4 Loading data

Results are stored in three different files following the required format: * CSV files * Every row is formed as WORD ADDRESS, READ VALUE, PATTERN, CYCLE. Besides, the first row contains

column heading (must be skipped), separators are commas and EOL character is the standard.

We will use the *readdlm* function provided by the *DelimitedFiles* package to load the first CSV file and to store everything in the new variable, DATA. Finally, it is important to indicate that DATA must be an array of UInt32 numbers.

[20]:
```
DATA1 = readdlm("MarchD - nv-SRAM.csv", ',', UInt32, '\n', skipstart=1)
```

[20]:
```
970×4 Matrix{UInt32}:
 0x00006eb2  0x000000ef  0x000000ff  0x00000001
 0x00007611  0x000000fb  0x000000ff  0x00000001
 0x00007af3  0x000000fd  0x000000ff  0x00000001
 0x00007f70  0x000000bf  0x000000ff  0x00000001
 0x00008754  0x0000007f  0x000000ff  0x00000001
 0x00009688  0x000000fd  0x000000ff  0x00000001
 0x000099f7  0x000000fd  0x000000ff  0x00000001
 0x0000a2c8  0x000000fe  0x000000ff  0x00000001
 0x0000a8aa  0x000000bf  0x000000ff  0x00000001
 0x0000acfb  0x000000df  0x000000ff  0x00000001
 0x0000af30  0x000000df  0x000000ff  0x00000001
 0x0000b00b  0x000000ef  0x000000ff  0x00000001
 0x0000b0a3  0x000000ef  0x000000ff  0x00000001

 0x0001ee5e  0x00000010  0x00000000  0x00000006
 0x0001ee9c  0x00000004  0x00000000  0x00000006
 0x0001f002  0x00000040  0x00000000  0x00000006
 0x0001f016  0x00000001  0x00000000  0x00000006
 0x0001f0f8  0x00000004  0x00000000  0x00000006
 0x0001f2b4  0x00000002  0x00000000  0x00000006
 0x0001f5b2  0x00000001  0x00000000  0x00000006
 0x0001f5d1  0x00000040  0x00000000  0x00000006
 0x0001f79e  0x00000080  0x00000000  0x00000006
 0x0001fb70  0x00000040  0x00000000  0x00000006
 0x0001fc27  0x00000008  0x00000000  0x00000006
 0x0001fdaf  0x00000010  0x00000000  0x00000006
```

1.5 Looking for MBUs

This analyisis is quite simple. We will call the *CheckMBUs* functionthat returns the MBUs present in DATA.Input arguments are the second and third columns, and the wordwidth.

This function returns two vectors. The first one indicates in position k the number of bitflips observed in the *kth* word. The second one is a vector of vectors and contains more detailed information: not only the number of bitflips per word but the position of the flipped bit (0 = LSB, WordWidth-1 = MSB).

[21]:
```
MBUSize, MBU_bit_pos = CheckMBUs(DATA1[:,2], DATA1[:,3], WordWidth)
```

[21]:
```
([1, 1, 1, 1, 1, 1, 1, 1, 1, 1  …  1, 1, 1, 1, 1, 1, 1, 1, 1, 1], Any[[4], [2],
[1], [6], [7], [1], [1], [0], [6], [5]  …  [6], [0], [2], [1], [0], [6], [7],
[6], [3], [4]])
```

The following loop will show how many MBUs per number of flipped bits were observed:

[22]:
```
for size = 1: WordWidth
    println("$size-bit MBUs: ", length(findall(MBUSize.==size)))
end
```

```
1-bit MBUs: 970
2-bit MBUs: 0
3-bit MBUs: 0
4-bit MBUs: 0
5-bit MBUs: 0
6-bit MBUs: 0
7-bit MBUs: 0
8-bit MBUs: 0
```

1.6 Looking for MCUs

As modern memories are interleaved, it is not worth investigating MBUs but MCUs. Now, the system will combine addresses in all the possible pairs and operate them to create a DV set. If there were no MCUs, their characteristics are known.

In particular, we can state that if the expected number of elements repeated k times in this set is lower than a very low positive number, it is impossible to observe this number of repetitions unless the Only SBU assumption fails. We will define this threshold as 0.001 (default, 0.05).

Although without a solid theoretical background, it seems that using pseudoaddress instead of word address provides better results.

Some experiments seem to show that if an element with very few number of 1s in binary format is too often repeated, it is indicative of the presence of MCUs. This is the Trace Rule and, in our analyisis, we want to keep all those too often repeated elements such that contain 2 ones or less in binary format.

Finally, perhaps we know that MCUs will not very large. For example, we may guess that MCUs with more than 20 bitflips are totally rejected. Therefore, to help the software and to avoid running out of memory, we will say the program *"Don't be silly and do not expect events larger than 20!!"* If somehow this idea was wrong, we can change this value again and repeat the calculations.

[23]:
```
 = 0.001   # If the expected number of elements repeated k times is lower than␣
↪ ,
            # we can afirm that this is virtually impossible.
UsePseudoAddress = true
TraceRuleLength = 2
LargestMCUSize = 20
```

[23]: 20

Time to test!!! We will call the function. Deppending on the set size or even if this is your first test, it will take you more or less time (Don't get up from your chair, though!!!!)

The following instruction will look for: 1. Values that pass the self-consistency test (C1_SCY) 2. Values found after inspecting MCUs derived from self-consistency-test (C1_MCU). 3. Values with less than or equal to *TraceRuleLength* 1s in binary format that appear too often in the DV set (C1_TRC). 4. Values that, after combining in pairs the union of all the previous three sets and applying the operation and that appear too many times within the DV set (C1_SHF).

The first column of each matrix are the possible values and the second one the times it appeared.

[24]:
```
C1_SCY, C1_MCU, C1_TRC, C1_SHF = DetectAnomalies_FullCheck(DATA1, WordWidth,␣
↪LA, Operation, TraceRuleLength, UsePseudoAddress, KeepCycles,  ,␣
↪LargestMCUSize)
```

[24]:
```
(Matrix{UInt32}(undef, 0, 2), Matrix{UInt32}(undef, 0, 2), Matrix{UInt32}(undef,
0, 2), Matrix{UInt32}(undef, 0, 2))
```

Perhaps these matrices are hard to read since, for efficiency, they were returned in UInt32 format, even the number of occurrences!!! Execute the following instrucction for a better comprehension.

[25]:
```
println("Elements appearing more than expected and passing the Self-Consistency␣
↪test:\n")
for index in 1:length(C1_SCY[:, 1])
    println("Value: 0x", string(C1_SCY[index, 1], base=16, pad = 6), " --> ",␣
↪Int(C1_SCY[index, 2]),".")
end

UsePseudoAddress ? L = LA*WordWidth : L = LA

print("\nOnly up to ", MaxExpectedRepetitions(NPairs(DATA1, UsePseudoAddress,␣
↪WordWidth, KeepCycles), L, Operation,  )-1, " repetitions are explained by␣
↪randomness.")
```

```
Elements appearing more than expected and passing the Self-Consistency test:


Only up to 6 repetitions are explained by randomness.
```

In this example, it is not worth to check the other sets since they did not yield any positive result. If you had had success, you would only have to do the following:

[26]:
```
C1_All = [C1_SCY; C1_MCU; C1_TRC; C1_SHF]
```

[26]:
```
0×2 Matrix{UInt32}
```

1.7 Grouping bitflips

Now, we have discovered those values relating pairs of pseudoaddresses. Now, let us go to group events in DATA.

The first step consists in labeling all the pseudoaddresses and grouping their assigned indexes to a matrix containing information for the possible MCUs. It is an intermediate step and is done with the instruction *MCU_Indexes* with the required and already defined parameters.

[27]:
```
Labeled_addresses = MCU_Indexes(DATA1, Operation, C1_All[:, 1],␣
↪UsePseudoAddress, WordWidth)
```

[27]:
```
0×2 Matrix{Int64}
```

Using this information, we can group the addresses.

[28]:
```
Events = Classify_Addresses_in_MCU(DATA1, Labeled_addresses, UsePseudoAddress,␣
↪WordWidth)
```

[28]:
```
2-element Vector{Any}:
 0×2 Matrix{UInt32}
 UInt32[0x00037594, 0x0003b08a, 0x0003d799, 0x0003fb86, 0x00043aa7, 0x0004b441,
0x0004cfb9, 0x00051640, 0x00054556, 0x000567dd  …  0x000f8016, 0x000f80b0,
0x000f87c2, 0x000f95a1, 0x000fad90, 0x000fae8e, 0x000fbcf7, 0x000fdb86,
0x000fe13b, 0x000fed7c]
```

Difficult to read, isn't it? The following instruction makes the content more readable:

[29]:
```
for k = 1:length(Events)
    NMCUs = length(Events[k][:, 1])
    println("Pseudoaddresses involved in $(length(Events)-k+1)-bit MCUs ($NMCUs␣
↪events):")
    for row = 1:NMCUs
        for bit = 1:length(Events)-k+1
            print("0x", string(Events[k][row, bit], base=16, pad = 6), )

            bit != length(Events)-k+1 ? print(", ") : print("\n")

        end
    end
    println()
end
```

```
Pseudoaddresses involved in 2-bit MCUs (0 events):

Pseudoaddresses involved in 1-bit MCUs (970 events):
0x037594
0x03b08a
0x03d799
0x03fb86
0x043aa7
0x04b441
```

0x04cfb9
0x051640
0x054556
0x0567dd
0x057985
0x05805c
0x05851c
0x0585ab
0x061863
0x06dc94
0x06e430
0x078cad
0x07901f
0x079efd
0x07a841
0x07ad0f
0x07afb9
0x07baae
0x07cc7c
0x07de28
0x081cfe
0x082e19
0x085cf1
0x0861d2
0x08d691
0x091634
0x091f30
0x092849
0x09a1f4
0x09c184
0x09cea6
0x09dca9
0x0a1f62
0x0a6a38
0x0a88f4
0x0a9c7a
0x0acf66
0x0acf76
0x0ae0bd
0x0ae23c
0x0af1d0
0x0af9a9
0x0b0f33
0x0b48e6
0x0b48ef
0x0b8337
0x0b8f48
0x0bb145

0x0bb55d
0x0bb871
0x0bc52d
0x0bc581
0x0bea19
0x0beef8
0x0c7511
0x0cb461
0x0ce2f4
0x0d0d30
0x0d1b85
0x0d2aad
0x0d5256
0x0d68a2
0x0d81b2
0x0d8fe0
0x0d9445
0x0dbc31
0x0dc7b6
0x0e2156
0x0e3039
0x0e315d
0x0e4792
0x0e781f
0x0e7db8
0x0e9082
0x0e93fb
0x0e98fc
0x0ee959
0x0ef1a4
0x0f09e7
0x0f1490
0x0f15d5
0x0f1e62
0x0f1e96
0x0f2586
0x0f2b47
0x0f4136
0x0f4e28
0x0f7f66
0x0fb417
0x0fb76c
0x0fbb49
0x0fbc42
0x0fd2c5
0x0ff4f6
0x00317a
0x003bfe

0x00490d
0x004929
0x005bc9
0x00619d
0x0092cf
0x00978d
0x00a17f
0x00bf70
0x00c581
0x00d0a9
0x016a5b
0x01b166
0x01dc3a
0x0237de
0x026272
0x02798b
0x029d72
0x02d94b
0x02df65
0x02ea01
0x03004e
0x0325e4
0x035a8d
0x037412
0x038a48
0x0393e6
0x039685
0x03afea
0x03b272
0x03f7d6
0x040a55
0x043748
0x045280
0x049339
0x04acd6
0x04ad2d
0x04cfb7
0x04e3b2
0x04ed6b
0x04fb11
0x053979
0x059345
0x0599db
0x05aec4
0x05b4ad
0x05cf1e
0x05d841
0x05eebc

0x0635ba
0x063d43
0x064672
0x065277
0x06a30f
0x06d0d2
0x06ddc9
0x06f1ae
0x0713c4
0x0716c5
0x073d40
0x073f71
0x0756eb
0x077562
0x0796f9
0x07a10f
0x07b271
0x07dbec
0x0804fc
0x0811cc
0x0827de
0x084e8f
0x084f94
0x087607
0x08a7f4
0x08acee
0x08b2ee
0x08b5b1
0x08b95a
0x08dab0
0x08f90f
0x0902d6
0x093e77
0x094a37
0x098181
0x09a037
0x09a99a
0x09d688
0x0a1abb
0x0a7bc9
0x0a8bd1
0x0a8ce3
0x0a9d19
0x0a9f55
0x0aa224
0x0abd5c
0x0ad348
0x0ae42e

0x0b230f
0x0b297c
0x0b2d80
0x0b3cb9
0x0b3ce1
0x0b42b2
0x0b49f2
0x0b5fea
0x0b615a
0x0b6803
0x0b82b2
0x0b9e9d
0x0ba856
0x0ba984
0x0bd048
0x0bd172
0x0c2895
0x0c34b2
0x0c483e
0x0c484c
0x0cb400
0x0cb48b
0x0cd96c
0x0cdbce
0x0cdf36
0x0cf0f9
0x0d20a0
0x0d2b8e
0x0d6956
0x0d7a85
0x0d8fe5
0x0d981c
0x0da1a3
0x0dacc7
0x0dae1e
0x0dbdee
0x0e3740
0x0e3959
0x0e3d3d
0x0e3d49
0x0e764d
0x0e85f1
0x0f0656
0x0f1065
0x0f3b2f
0x0f4dd8
0x0f4fe6
0x0f5022

0x0f8c4a
0x0fc871
0x0fd470
0x0fe338
0x0029ca
0x002e13
0x003d34
0x004b6e
0x006a1f
0x007e8a
0x009f4a
0x00b02a
0x00bf7b
0x00e457
0x00f479
0x00fadc
0x01084b
0x011169
0x012a82
0x014755
0x015347
0x0156c0
0x016e04
0x017176
0x018345
0x01a2ad
0x01a75e
0x01b696
0x01b9f4
0x01bbf2
0x01e789
0x01ebec
0x01f015
0x01f534
0x021115
0x02124b
0x02228c
0x022463
0x02397c
0x02519f
0x026259
0x02a5e9
0x02a6d0
0x02bb43
0x02d0cf
0x02e740
0x0303e1
0x031802

0x031aa8
0x0325b3
0x03300e
0x037244
0x0396db
0x03a5be
0x03b4a9
0x03bf29
0x03e326
0x0446ab
0x0449d9
0x044bbe
0x044dd7
0x044fe5
0x045dfd
0x048737
0x048846
0x048c84
0x048ec9
0x048f93
0x049062
0x049567
0x04a356
0x04b8d3
0x04c9c1
0x04e009
0x04e8ce
0x051dec
0x052631
0x054a86
0x0556f4
0x055725
0x0573e6
0x057aab
0x0587b7
0x05c522
0x05dcc6
0x061e60
0x061e68
0x063e50
0x064b8e
0x0653df
0x065e82
0x068b8e
0x06a1ab
0x06b458
0x06d223
0x06dbb3

0x06ed35
0x06fbb7
0x071605
0x073b10
0x075fc7
0x07663f
0x079434
0x07a527
0x07e6b5
0x07f5e2
0x081e79
0x086a84
0x087ea2
0x087f03
0x088ad8
0x089f97
0x08b68b
0x08b8ab
0x08de57
0x08e3ef
0x08f055
0x090435
0x092439
0x09264d
0x092732
0x093055
0x095266
0x09671c
0x0967e8
0x0996b3
0x09c0e3
0x09e253
0x0a23b2
0x0a5f2a
0x0a6586
0x0a9655
0x0a9c60
0x0ac187
0x0acd30
0x0ad25f
0x0ad95c
0x0afcfc
0x0b2aa5
0x0b428c
0x0b5719
0x0b6692
0x0b7de1
0x0b93bc

0x0ba223
0x0bb791
0x0bbdd6
0x0bd216
0x0c025d
0x0c3c99
0x0c3fa3
0x0c4474
0x0c55be
0x0c6eff
0x0c7379
0x0c87dd
0x0ca446
0x0cc513
0x0cf9c5
0x0d1d00
0x0d1d10
0x0d25c6
0x0d4f38
0x0d5b08
0x0d716a
0x0d92ef
0x0da732
0x0dab77
0x0db0e6
0x0dcd5b
0x0ddb59
0x0de211
0x0e66b9
0x0e8333
0x0e9735
0x0ea800
0x0eb4f9
0x0ebaa6
0x0ec2fa
0x0ece98
0x0ee906
0x0efa92
0x0f0414
0x0f11e2
0x0f23c9
0x0f2aec
0x0f4326
0x0f5808
0x0f5cae
0x0f627c
0x0fc5b0
0x0e591f

0x0e5668
0x0dae1c
0x0d9fc3
0x0d9af8
0x0d8ea1
0x0d6465
0x0d193f
0x0cc57d
0x0c9342
0x0c6cfb
0x0c668f
0x0c3c9b
0x0c3b3f
0x0c36d5
0x0c05c1
0x0b7403
0x0b5af1
0x0b295a
0x0b2942
0x0b079b
0x0b059e
0x0b0570
0x0afa69
0x0ad569
0x0ab1e7
0x0a995e
0x0a5863
0x0a52a9
0x0a3d55
0x0a2da1
0x096735
0x096303
0x096267
0x093df6
0x093583
0x091ffb
0x0914e6
0x091354
0x0906b4
0x08fcc8
0x08f3fc
0x08df46
0x08dd3c
0x08af57
0x088fcd
0x087cc4
0x087357
0x0871a9

0x086cd5
0x0858c9
0x084c0f
0x084af0
0x0839c8
0x082fb6
0x07f53f
0x07e08f
0x07d90d
0x07bdeb
0x074a3f
0x0747c7
0x073ac2
0x072ec0
0x070ab0
0x06f582
0x06ec2c
0x06a1be
0x0695c4
0x065a7f
0x060cef
0x05f8ae
0x05e785
0x05d753
0x05c939
0x05b0ef
0x05a5df
0x05a151
0x0599ae
0x058581
0x056f23
0x05611d
0x054f0e
0x054c47
0x053b55
0x052ea4
0x05160c
0x050dcd
0x04e7d2
0x04e030
0x04dac7
0x04cc04
0x04c5c7
0x048bd1
0x046212
0x0461b2
0x045e07
0x04404c

0x04387b
0x0407cc
0x03fd3e
0x03aeb9
0x039ade
0x03988a
0x037aa7
0x0378d4
0x037811
0x0354b1
0x0338d1
0x032c59
0x031e2a
0x031420
0x0300b7
0x02e96e
0x02cd8b
0x02c220
0x02b9cc
0x02b30e
0x02aa30
0x02a109
0x028f18
0x027e7e
0x027e0c
0x027265
0x026e07
0x024602
0x0226bf
0x01eafc
0x01e31e
0x01e2be
0x01da3c
0x01d6ef
0x01d1f4
0x01ccdb
0x01b741
0x01b200
0x01a220
0x0199e9
0x01951a
0x0191dd
0x0190a3
0x0181b2
0x0180f8
0x017b7c
0x016e0f
0x01671d

0x0161cd
0x015c51
0x015662
0x0150ba
0x0142a6
0x01403d
0x012b57
0x0124f0
0x00b9f2
0x00a74f
0x00a675
0x009ca4
0x009c0c
0x008ffa
0x0050ea
0x003b58
0x003090
0x002c3b
0x002ad6
0x0ff102
0x0fee06
0x0fe6bb
0x0fe698
0x0fc582
0x0fbfaa
0x0f9088
0x0f54c7
0x0f3221
0x0f2074
0x0f1429
0x0ef90f
0x0ecaaa
0x0ea093
0x0e73e1
0x0e639e
0x0e5462
0x0e3754
0x0e310a
0x0e1583
0x0e0c49
0x0e0876
0x0df981
0x0dcdba
0x0d6c41
0x0d63bc
0x0d529e
0x0d3a61
0x0d29e6

0x0d1fa6
0x0d0ecd
0x0cc5f6
0x0cb23a
0x0cadb4
0x0ca9d7
0x0ca417
0x0c8781
0x0c52f8
0x0c4df2
0x0c4862
0x0c4568
0x0c44aa
0x0c0f53
0x0bf590
0x0bdf73
0x0bd88f
0x0b7e7d
0x0b66db
0x0b3cb8
0x0b25fe
0x0b0816
0x0b04b3
0x0aec9a
0x0ae19a
0x0ada58
0x0a9a75
0x0a88d8
0x0a83fc
0x0a7ffd
0x0a6c2d
0x0a4f5b
0x0a22e9
0x0a1800
0x09e8c7
0x09ca90
0x099d1c
0x098456
0x09837a
0x0965e0
0x091f66
0x0902c5
0x08d790
0x08b038
0x08932c
0x089207
0x088909
0x0882f0

0x088239
0x087d77
0x085a63
0x0858a2
0x0842f0
0x08307b
0x082b8f
0x082a87
0x081a13
0x0812b9
0x080df2
0x07ff94
0x07d44a
0x0794d0
0x07925f
0x078806
0x077e50
0x077770
0x0772b9
0x075438
0x073e95
0x07349e
0x0723e7
0x0713a2
0x06b3f1
0x06a263
0x068587
0x0663d0
0x065ee3
0x0654ec
0x06547d
0x063eb3
0x05ffaa
0x05fe93
0x05b4db
0x05aab4
0x059f3b
0x0599b1
0x055c93
0x04efa5
0x04ec48
0x04d4bf
0x04ad91
0x049b91
0x0499d3
0x049445
0x048517
0x0482cc

0x046223
0x044c81
0x04425e
0x0441f4
0x042ed4
0x04131b
0x04002a
0x03e4e9
0x03df5a
0x03d923
0x03d624
0x039dba
0x0395ab
0x039236
0x0383de
0x0370a4
0x036792
0x036025
0x02e31a
0x02cf9d
0x02b126
0x029cc5
0x027773
0x027272
0x024f26
0x0247a8
0x0207ab
0x02016b
0x01d573
0x01c99d
0x01c8e5
0x01bf6d
0x01bd02
0x01a8c8
0x0194d9
0x0191fd
0x01803f
0x017d02
0x015121
0x010bf4
0x010366
0x00f357
0x00dc88
0x00ca06
0x00c171
0x00b952
0x00aae2
0x009db8

0x006868
0x0066af
0x006638
0x004719
0x0043c2
0x00392c
0x0037ca
0x002609
0x0025d0
0x002038
0x000bbc
0x000502
0x0004a9
0x00638a
0x00a43f
0x014431
0x02154a
0x02736c
0x02935e
0x02db59
0x038c67
0x03aa75
0x03b313
0x03c622
0x04548a
0x0474f8
0x047bee
0x048077
0x04948a
0x049dd8
0x04ad80
0x04d916
0x051abb
0x055f01
0x056071
0x05752a
0x057866
0x0594ac
0x05de56
0x05e901
0x05ec5e
0x05fd05
0x062b6c
0x065280
0x065daa
0x0669ae
0x066c6e
0x06a094

0x06ab8c
0x06b79b
0x06da0c
0x06db44
0x071f48
0x073583
0x073b17
0x074cee
0x076065
0x077693
0x077b1a
0x079092
0x07dc84
0x07dfeb
0x081a08
0x081c48
0x0843a9
0x084425
0x0874ca
0x089dbb
0x08a820
0x08b252
0x08c0fe
0x0903da
0x091cb3
0x092361
0x092996
0x093686
0x094f71
0x095109
0x095577
0x096135
0x09644e
0x0980fd
0x098e9e
0x099b53
0x09bfb6
0x09c773
0x09c7b2
0x09e87f
0x09fe15
0x0a01de
0x0a22f4
0x0a3006
0x0a690a
0x0a69d1
0x0a7588
0x0a7c4e

0x0a84a7
0x0a951f
0x0aaec7
0x0ab292
0x0ad091
0x0adcc4
0x0ae65f
0x0ae903
0x0ae9b9
0x0b1225
0x0b48d5
0x0bbff3
0x0bc11f
0x0bd369
0x0bdca4
0x0bf0e6
0x0c1626
0x0c24dc
0x0c314d
0x0c416f
0x0c4860
0x0c5586
0x0c5880
0x0c5f9c
0x0c6173
0x0c638d
0x0c75bd
0x0c8473
0x0ca5cb
0x0cae8b
0x0cbb20
0x0ccc38
0x0ce426
0x0ce6e0
0x0cf52b
0x0d0f5d
0x0d10b6
0x0d14aa
0x0d27a6
0x0d3099
0x0d337a
0x0d3e59
0x0d4255
0x0d4550
0x0d795b
0x0d8b47
0x0d8cf7
0x0d8dc5

0x0d9211
0x0dcf80
0x0de332
0x0de338
0x0de3e0
0x0dfddc
0x0e0cc0
0x0e0d7a
0x0e1416
0x0e169d
0x0e1db2
0x0e3131
0x0e3959
0x0e4022
0x0e4890
0x0e4902
0x0e4b5f
0x0e4d90
0x0e54ad
0x0e5a68
0x0e61f8
0x0e65b6
0x0e70f6
0x0e7ef8
0x0ea3e2
0x0eaf46
0x0ecffa
0x0ee018
0x0eef67
0x0ef338
0x0f0f1f
0x0f117a
0x0f13e2
0x0f2a14
0x0f347e
0x0f3a22
0x0f46dc
0x0f573c
0x0f6249
0x0f6cee
0x0f72f4
0x0f74e2
0x0f8016
0x0f80b0
0x0f87c2
0x0f95a1
0x0fad90
0x0fae8e

```
0x0fbcf7
0x0fdb86
0x0fe13b
0x0fed7c
```

1.8 Analysis completed!