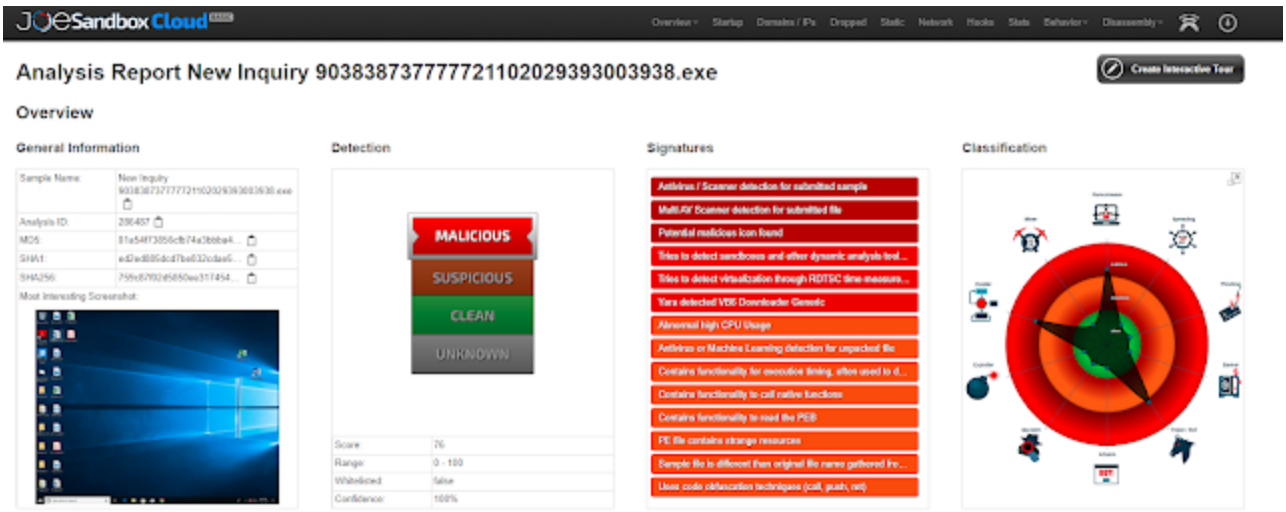
# GuLoader's VM-Exit Instruction Hammering explained

joesecurity.org/blog/3535317197858305930

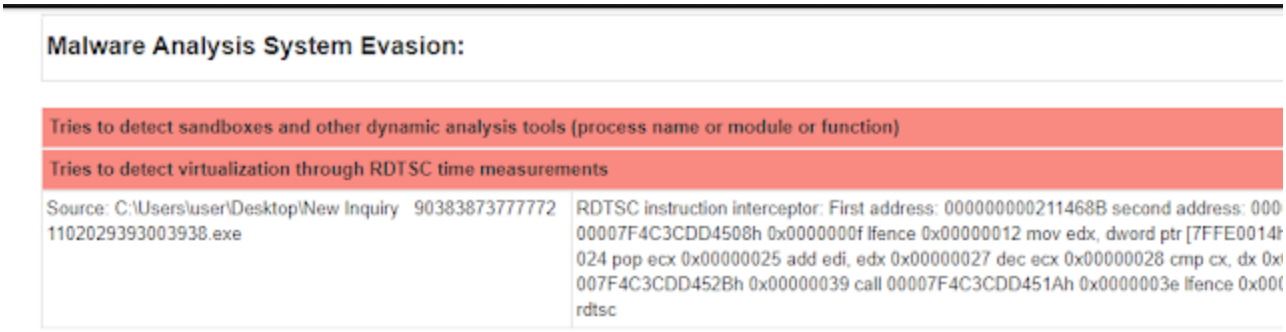

In Joe Sandbox Cloud Basic, our community version of Joe Sandbox, we often get very interesting and recent malware samples. On the September 16th, 2020 we came across a new GuLoader variant (MD5: 01a54f73856cfb74a3bbba47bcec227b). GuLoader is a malware loader well known for its anti-evasion techniques.

## Slow VM Exits

The initial analysis on a virtual machine showed the following results:



As we can see in the Signature section, there are some RDTSC based evasion checks executed:



**Malware Analysis System Evasion:**

| |
|---|
| Tries to detect sandboxes and other dynamic analysis tools (process name or module or function) |
| Tries to detect virtualization through RDTSC time measurements |

| | |
|---|---|
| Source: C:\Users\user\Desktop\New Inquiry 90383873777772 1102029393003938.exe | RDTSC instruction interceptor: First address: 000000000211468B second address: 000 00007F4C3CDD4508h 0x0000000f lfence 0x00000012 mov edx, dword ptr [7FFE0014h 024 pop ecx 0x00000025 add edi, edx 0x00000027 dec ecx 0x00000028 cmp cx, dx 0x 007F4C3CDD452Bh 0x00000039 call 00007F4C3CDD451Ah 0x0000003e lfence 0x000 rdtsc |

Among many other anti-evasion checks, GuLoader uses the following code to detect that it is running in a virtual machine:

```
#include <stdio.h>
#include <windows.h>

__declspec(naked) int fun() {
  __asm {
  _loop:
      lfence
      rdtsc    // (1)
      lfence
      shl edx, 0x20
      or edx, eax
      mov esi, edx
      pushad
      mov eax, 0x1
      cpuid
      bt ecx, 0x1F
      jb _here
  _here:
      popad
      lfence
      rdtsc    // (2)
      lfence
      shl edx, 0x20
      or edx, eax
      sub edx, esi
      cmp edx, 0x0
      jle _loop
      mov eax, edx
      retn
  };
}

int main() {
  for (int r = 5; r; r--) {
    int sum = 0;
    for (int i = 100000; i; i--) {
      sum += fun();
    }

    int expected = 110000000;
    printf("expected between: [0, %d)\n            got :  %d\n         result : %s\n\n",
                    expected, sum, (sum < 0 || sum >= expected) ? "failed" : "passed");
    Sleep(500);
  }

  Sleep(5000);
  return 0;
}
```

The code has two main purposes. First, it measures how long the execution of the CPUID instructions takes. On real hardware, CPUID is directly executed by the CPU. Inside a virtual machine, the CPUID instruction forces a VM exit - execution is transferred from the guest VM to the host. The hypervisor handles the instructions and switches back. This transition is much slower compared to direct CPU execution. The same is true for other instructions like RDTSC. This difference is measured and used to decide if the loader is going to execute the payload or not.

## Instruction Hammering

Secondly, the measurements are not executed once but executed thousands of times. The result is an overall delay which often exceeds the execution time on a sandboxed analyzer. As a result, the payload execution is never reached. This method of executing massive amounts of delay instructions to prevent the execution - also known as **Instruction Hammering** - is very similar to API hammering, a technique we saw in TrickBot and many other malware samples.

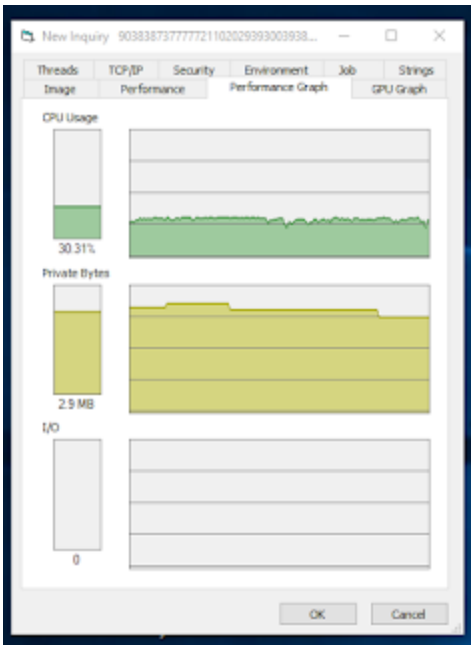
Instruction Hammering is extremely powerful since it is hard to detect and challenging to bypass, as it exploits the architecture of virtualization. The GuLoader creators seem to have noticed that, and in the new version they have even increased the number of delay instructions being executed:

Disassembly:

```
0:  0f 31                   rdtsc
2:  b8 01 00 00 00          mov    eax,0x1
7:  0f a2                   cpuid
9:  61                      popa
a:  e8 03 00 00 00          call   0x12
f:  0f ae e8                lfence
12: 8b 15 14 00 fe 7f       mov    edx,DWORD PTR ds:0x7ffe0014
18: 0f ae e8                lfence
1b: c3                      ret
1c: 29 f2                   sub    edx,esi
1e: c3                      ret
1f: 66 39 c8                cmp    ax,cx
22: 85 c2                   test   edx,eax
24: 59                      pop    ecx
25: 01 d7                   add    edi,edx
27: 49                      dec    ecx
28: 66 39 d1                cmp    cx,dx
2b: 83 f9 00                cmp    ecx,0x0
2e: 75 e6                   jne    0x16
30: 51                      push   ecx
31: 66 39 c0                cmp    ax,ax
34: e8 26 00 00 00          call   0x5f
39: e8 15 00 00 00          call   0x53
3e: 0f ae e8                lfence
41: 8b 15 14 00 fe 7f       mov    edx,DWORD PTR ds:0x7ffe0014
47: 0f ae e8                lfence
4a: c3                      ret
4b: 89 d6                   mov    esi,edx
4d: 60                      pusha
4e: 0f 31                   rdtsc
```

This code executes RDTSC and CPUID 11 million times. In addition, *UserSharedData.SystemTime* is being used for time measurements.

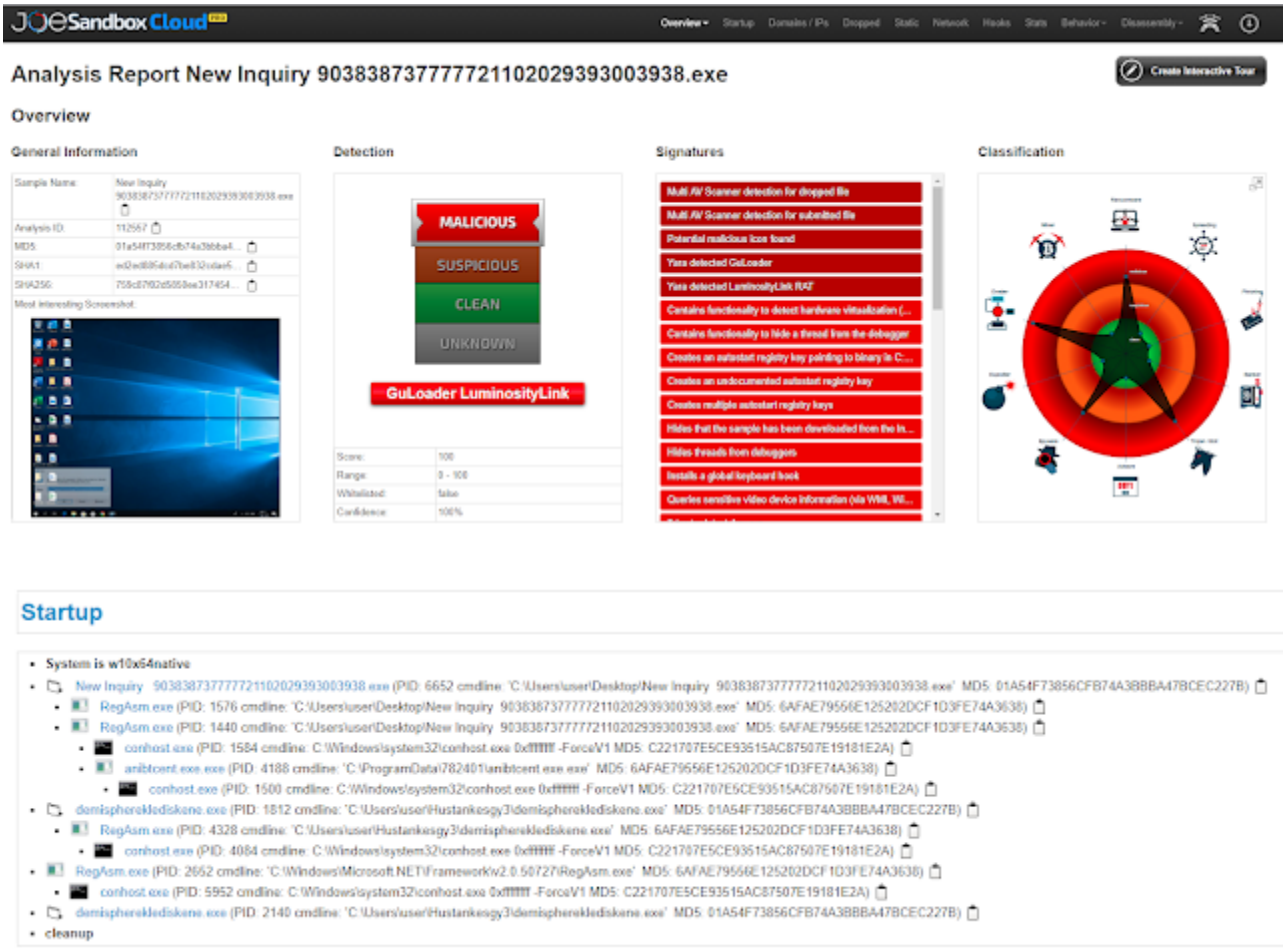
On a Windows 10 x64 system running on VirtualBox the delay loop takes several minutes to finish:




On real hardware, the loop is executed in under one second!

## Bare Metal Analysis to the Rescue

Joe Sandbox is one of a few vendors offering analysis on bare metal. In this setup, the malware sample is run on a real physical machine. Physical machines are much closer to the real target of the malware. As a result, VM-based evasions don't work and the sandbox can catch and record the real payload. If we analyze GuLoader on bare metal the delay loop is passed in under a second and we can see that the LuminosityLink RAT is dropped:




The full analysis report of the GuLoader variant is available here.

Interested in Joe Sandbox? Register for free at Joe Sandbox Cloud Basic or contact us for an in-depth technical demo!