

# GuLoader AntiVM Techniques - VinCSS Blog

By Yến Hứa

Published: 2023-05-27 · Archived: 2026-04-10 02:37:25 UTC

## Table of Contents

- [1. Anti-VM](#)
  - [1.1. Kết hợp ZwQueryVirtualMemory với các hash đã tính toán trước](#)
  - [1.2. Kiểm tra sự tồn tại của QEMU Guest Agent](#)
  - [1.3. Sử dụng CPUID](#)
- [2. Bonus](#)

Thời gian gần đây, Twitter của các chuyên gia nước ngoài liên tục xuất hiện hash tag [#GuLoader](#), đây là một downloader phổ biến đang được các nhóm tin tặc sử dụng rộng rãi nhằm tải về mã độc chính lợi dụng các dịch vụ cloud của Google Drive và Microsoft OneDrive. GuLoader rất nhỏ, nhẹ, được viết bằng VB6 và thường được nén trong file .rar. Khi người dùng vô tình thực thi loader này, nó sẽ tải xuống Trojans (RAT) hoặc các dòng mã độc đánh cắp thông tin như Agent Tesla, FormBook, NanoCore RAT, Netwire RAT, Remcos RAT, ...

Ở Việt Nam cũng không ngoại lệ, chúng tôi đã tiếp cận và có [bài phân tích](#) chi tiết về downloader này. Mới đây, khách hàng của chúng tôi tiếp tục nhận được email có file đính kèm lạ, qua kiểm tra, phân tích và so sánh, chúng tôi nhận thấy đây chính là một biến thể của GuLoader. Nhiệm vụ của nó là tải về NanoCore RAT để thực thi trên máy người dùng. Trong bài viết này, chúng tôi sẽ không đi vào phân tích chi tiết mà chỉ tập trung vào các kỹ thuật Anti-VM được sử dụng trong shellcode.

## 1. Anti-VM

### 1.1. Kết hợp ZwQueryVirtualMemory với các hash đã tính toán trước

Thông thường, để phân tích mã độc, người phân tích sẽ thực thi mã độc trên một môi trường riêng biệt nhằm thu thập các thông tin tương tác với hệ thống trong quá trình thực thi. Tuy nhiên, với các biến thể mới của Guloader, khi thực thi sẽ nhận được thông báo sau:

Debug loader sẽ tới được shellcode. Shellcode thực hiện push một loạt các hash đã tính toán trước lên stack:

```
push 0xB314751D
```

```
push 0xA7C53F01
```

```
push 0x7F21185B
```

```
push 0x3E17ADE6
```

```
push 0xF21FD920  
push 0x27AA3188  
push 0xDFCB8F12  
push 0x2D9CC76C
```

Tiến hành resolve hàm API **ZwQueryVirtualMemory** ứng với hash đã tính toán trước là **0x8802EDAC**.

*Hình 2. Lấy địa chỉ hàm API ZwQueryVirtualMemory*

Tiếp theo sử dụng vòng lặp để quét toàn bộ vùng nhớ từ **0x00010000** tới **0x7FFF000**, gọi hàm **ZwQueryVirtualMemory** kiểm tra access protection của các vùng nhớ này. Nếu vùng nhớ thỏa mãn điều kiện, sẽ quét vùng nhớ đó, gặp chuỗi sẽ gọi hàm tính toán hash cho chuỗi đó và so sánh với các hash đã thiết lập trên Stack. Khi trùng hash, loader lấy địa chỉ base của **msvbvm60.dll** để tìm hàm **MessageBoxA**, giải mã chuỗi “*This program cannot be run under virtual environment or debugging software !*” và hiển thị thông báo như Hình 1.

*Hình 3. Tính toán hash của các chuỗi trên vùng nhớ*

Hàm tính toán hash mà loader sử dụng chính là thuật toán [djb2](#):

*Hình 4. Hàm tính toán hash*

Với máy sử dụng để debug loader, chúng tôi có được chuỗi “*vmtoolsdControlWndClass*” trùng với một hash đã được loader tính toán trước là **0xB314751D**. Các hash còn lại theo phỏng đoán của chúng tôi có thể liên quan đến VirtualBox hoặc môi trường sandbox khác.

*Hình 5. Phát hiện môi trường ảo hóa sử dụng VMware*

## 1.2. Kiểm tra sự tồn tại của QEMU Guest Agent

Bên cạnh kỹ thuật nói trên, loader cũng thực hiện kiểm tra xem môi trường phân tích có sử dụng **Qemu guest agent (qga)** hay không thông qua hàm **CreateFileA**.

*Hình 6. Lấy địa chỉ hàm API CreateFileA*

Sử dụng hàm này để kiểm tra sự tồn tại của “*C:ProgramDataqemu-gaqa.state*” trên máy:

Hình 7. Gọi hàm `CreateFileA` để kiểm tra sự tồn tại của file

Nếu tồn tại file trên hệ thống sẽ hiển thị thông báo như Hình 1:

Hình 8. Hiển thị thông báo nếu có `qga.state`

### 1.3. Sử dụng **CPUID**

Bên cạnh hai kỹ thuật trên, loader còn sử dụng thêm lệnh **CPUID** để kiểm tra xem chương trình có đang thực thi trong môi trường ảo hóa hay không?

Hình 9. Sử dụng **CPUID** để kiểm tra

Lệnh **CPUID** được thực thi với giá trị của `EAX = 1`. Bit thứ 31 của `ECX` nếu trên máy vật lý sẽ bằng 0. Trên máy ảo, nó sẽ bằng 1.

## 2. Bonus

Loader này sử dụng kỹ thuật code injection phức tạp nhằm làm khó người phân tích.

- Tạo một tiến trình con là **RegAsm.exe** ở trạng thái `suspended`.
- Map **msvbvm60.dll** vào tiến trình vừa tạo.
- Thực hiện inject shellcode vào **RegAsm.exe**.
- Thiết lập context của tiến trình nhằm chuyển hướng thực thi tới đoạn code đã inject và gọi hàm **ZwResumeThread** để thực thi.

Shellcode thực thi bên trong tiến trình **RegAsm.exe** sẽ thực hiện tải về, giải mã và thực thi payload. Payload mới được lưu trên Google Drive:

Hình 10. Loader thực hiện tải payload từ Google Drive

Giải mã payload và map vào memory để thực thi. Payload có kích thước **0x32A00**:

Hình 11. Payload sau khi tải về được giải mã trên memory

Payload thu được chính là **NanoCore RAT**:

Hình 12. Payload thu được là NanoCore RAT

**Tran Trung Kien (aka m4n0w4r)**  
**Dang Dinh Phuong**  
**R&D Center – VinCSS (a member of Vingroup)**

---

Source: <https://blog.vincss.net/vi/re014-guloader-antivm-techniques/>