

Forum:[Hack Forums](#)

Thread: [Unique Method to RE Scanners ~ Virus Total & Hybrid Analysis ~ C++ POC](#)

Hackers - April 4, 2024, 3:54 am

"Reverse Engineering" Virus Total
& Hybrid Analysis
(C++ POC)

Today we will be seeing if we can identify any indicators on Virus Total or Hybrid Analysis that could help us identify when our software is being analyzed on their platform.

I started this POC without knowing if I would be successful, but I quickly figured out how to get this POC started.

The goal? To see what modules are being loaded into a program on a runtime scanning service to identify how they analyze software. Let me show you what I mean...

Note: I used .bat to increase detects and make sure each file shows up in the written/dropped files list - I did use .txt in some tests, but this may not always show up in scan details, so stick to .bat or even .exe if you want. To make sure each file is detected better, I threw in the [EICAR test string](#) to hopefully trigger more AV actions. The EICAR string didn't trigger all scans like I hoped, but it did increase detects *slightly*.

When a program runs on your computer, it may have its dependencies already included (static build), or it may dynamically link/include them when it starts up in runtime. Either way, C/C++ programs (or C# and others) will load in "modules" so that it has all of the functionality it needs. When you run your software on your own computer, you can easily see what modules are loaded by using the enumerated module list from the program. If you setup a loop to check the loaded module list, you can easily print out the names to understand what is running.

For example, if I run my software on my protected Desktop folder, I can see that AVG (an AV software I have) automatically adds 'aswhook.dll' in runtime.

However, if I run the same exact program in my whitelisted folder (a personal C++ folder that is not scanned by WD or AVG), I do not have this module loaded.

This is because AVG doesn't scan the program when it's in my whitelisted directories, so they do not load their hook into my program. This is an example of what we will be trying to find out: What modules load on Virus Total or Hybrid Analysis that don't load locally? I set out to answer the question, Can we identify any unique modules that only load when running in a VT or HA virtual

machine / sandbox?

To detect loaded modules on any computer, VM, etc, we must set up a loop that can check the loaded modules:

```
Code while (true) {
    if (EnumProcessModules(hProcess, hMods, sizeof(hMods), &cbNeeded)) {
        for (unsigned int i = 0; i < (cbNeeded / sizeof(HMODULE)); i++) {
            char szModName[MAX_PATH];
            // Get the full path to the module's file
            if (GetModuleFileNameExA(hProcess, hMods[i], szModName, sizeof(szModName) /
sizeof(char))) {
                std::string moduleName(szModName);
                // Check if the module is already tracked
                if (!isModuleTracked(trackedModules, moduleName)) {
                    // Not tracked, so add it to the list and create a log event for it
                    trackedModules.push_back(moduleName);
                    logEvent("Module loaded: " + moduleName, getBasename(moduleName));
                    createModuleFile(moduleName, hMods[i]);
                }
            }
        }
    }
    else {
        std::cerr << "Failed to enumerate process modules." << std::endl;
        break; // Exit the loop if module enumeration fails
    }
    Sleep(1000); // Sleep for 1 second to reduce CPU load
}
```

This is just some basic code using default Windows functionality. It calls the createModuleFile function which is important for tracking on all scanners. Many scanning tools will not report what modules were loaded by their programs, so to track all I came up with an idea: save the module names & info to txt files so they show up in scans as 'written files' or 'dropped files' or similar. This will let us use the file list as our visual list of loaded modules, so we can see what was used on scans!

I then modified this to include more info, such as the address calling it, and the filepath. The idea is to add more info so that we can start to identify as much as possible about each module.

We can get something like this - the .txt and info in the [] brackets are from me, trying to get more info about each. This can definitely be improved further.

```
Code "KERNELBASE.dll --- 7ffe5f00000.txt" Location: [C:\KERNELBASE.dll --- 7ffe5f00000.txt]
"gdi32full.dll --- 7ffe64d0000.txt" Location: [C:\gdi32full.dll --- 7ffe64d0000.txt]
"msvc_p_win.dll --- 7ffe5d30000.txt" Location: [C:\msvc_p_win.dll --- 7ffe5d30000.txt]
"ucrtbase.dll --- 7ffe65f0000.txt" Location: [C:\ucrtbase.dll --- 7ffe65f0000.txt]
"OLEAUT32.dll --- 7ffe7f10000.txt" Location: [C:\OLEAUT32.dll --- 7ffe7f10000.txt]
"KERNEL32.dll --- 7ffe7bc0000.txt" Location: [C:\KERNEL32.dll --- 7ffe7bc0000.txt]
"ADVAPI32.dll --- 7ffe8550000.txt" Location: [C:\ADVAPI32.dll --- 7ffe8550000.txt]
"sechost.dll --- 1ed4f9c0000.txt" Location: [C:\sechost.dll --- 1ed4f9c0000.txt]
"combase.dll --- 7ffe6760000.txt" Location: [C:\combase.dll --- 7ffe6760000.txt]
```

"apphelp.dll --- 7ffe31c0000.txt" Location: [C:\apphelp.dll --- 7ffe31c0000.txt]
"win32u.dll --- 7ffe62b0000.txt" Location: [C:\win32u.dll --- 7ffe62b0000.txt]
"FLTLIB.dll --- 7ffc4bc40000.txt" Location: [C:\FLTLIB.dll --- 7ffc4bc40000.txt]
"RPCRT4.dll --- 7ffe77f0000.txt" Location: [C:\RPCRT4.dll --- 7ffe77f0000.txt]
"WS2_32.dll --- 7ffe79a0000.txt" Location: [C:\WS2_32.dll --- 7ffe79a0000.txt]
"msvcrt.dll --- 7ffe7a60000.txt" Location: [C:\msvcrt.dll --- 7ffe7a60000.txt]
"USER32.dll --- 7ffe8600000.txt" Location: [C:\USER32.dll --- 7ffe8600000.txt]
"GDI32.dll --- 7ffe7a30000.txt" Location: [C:\GDI32.dll --- 7ffe7a30000.txt]
"ntdll.dll --- 7ffe8910000.txt" Location: [C:\ntdll.dll --- 7ffe8910000.txt]
"ole32.dll --- 7ffe81b0000.txt" Location: [C:\ole32.dll --- 7ffe81b0000.txt]
"IMM32.dll --- 7ffe84f0000.txt" Location: [C:\IMM32.dll --- 7ffe84f0000.txt]

or this, as example of my list from 1 scanning platform without any other info:

Code ADVAPI32.dll
CRYPT32.dll
DNSAPI.dll
GDI32.dll
IMM32.DLL
IPHLPAPI.DLL
KERNELBASE.dll
LPK.dll
MSASN1.dll
MSCTF.dll
NSI.dll
PSAPI.DLL
RPCRT4.dll
SHELL32.dll
SHLWAPI.dll
USER32.dll
USP10.dll
WINMM.dll
WINNSI.DLL
WS2_32.dll
api-ms-win-core-synch-l1-2-0.DLL
bcrypt.dll
cryptbase.dll
kernel32.dll
msvcrt.dll
ntdll.dll
ole32.dll
sechost.dll
KERNEL32.DLL
KERNELBASE.dll
apphelp.dll
ntdll.dll

What we are looking for is modules that only load on 1 platform, to see if we can identify any software that is analyzing our program in runtime. Boom, found one!

frida-agent is suspicious because it's associated with Frida, a [free software for runtime analysis](#). We can basically assume that if frida is loaded into our program, its being debugged/analyzed/etc.

Another indicator for your program being debugged in runtime is anything related to Flt - the [filter manager for Windows](#) that lets AV use minifilters to detect system changes & actions.

Checkout my thread here which could be useful for detecting more about minifilters:
<https://hackforums.net/showthread.php?tid=6264613&highlight=minifilter>

All in all, this POC is only intended to give us the data - it's up to you how you want to use this information. I think that there could easily be some work done to detect certain settings that are used, so your program knows 100% if it's in VT, HA, or another platform's sandbox modes. This may be useful to you. Keep in mind also that it might not always be a case of "this module was loaded" and may actually be related to not just which module is loaded, but when it is loaded, how it's loaded (order, from where, etc), and other context should be taken into account besides just looking for module names. Best of luck! :)

Here is an example of what it may look like if you have systeminfo and also the timestamp in the filename. We can see that it used Windows 7 (weird), x64, with 1GB of RAM. Keep in mind that this may be 'fake' to trick programs into triggering "all features" if they think they will work on an older Win7 machine? Not sure, but it was a surprise to see such an old version in use for this.

There are many directions to go from here, but I hope this POC helps you understand how the filelist on Virus Total, Hybrid Analysis, and similar scanners can be used to track information. Because they print all written files, you can simply use the filenames as a personal log for any runtime data you want :)

Here is the full code to the POC, which has everything needed for the module detection, system info, RAM amount, architecture, etc saved to the filename.

```
Code #include <windows.h>
#include <psapi.h>
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include <ctime>
#include <VersionHelpers.h>

std::string getSystemInfo() {
    std::ostringstream oss;
    oss << "Windows_";

    if (IsWindows10OrGreater()) {
        oss << "10_or_greater";
    }
    else if (IsWindows8Point1OrGreater()) {
        oss << "8.1";
    }
    else if (IsWindows8OrGreater()) {
        oss << "8";
    }
}
```

```

    }
    else if (IsWindows7OrGreater()) {
        oss << "7";
    }
    else {
        oss << "Older_version";
    }

    SYSTEM_INFO siSysInfo;
    GetSystemInfo(&siSysInfo);
    oss << "_" << ((siSysInfo.wProcessorArchitecture == PROCESSOR_ARCHITECTURE_AMD64)
? "x64" : "x86");

    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof(statex);
    GlobalMemoryStatusEx(&statex);
    oss << "_RAM" << statex.ullTotalPhys / (1024 * 1024 * 1024) << "GB";

    return oss.str();
}

std::string getCurrentSystemTime() {
    time_t now = time(nullptr);
    struct tm tstruct;
    char buf[80];
    localtime_s(&tstruct, &now); // Secure version of localtime
    strftime(buf, sizeof(buf), "%Y%m%d_%H%M%S", &tstruct);
    return std::string(buf);
}

// Enhanced Logging with Detailed Filenames
std::string generateLogFileName(const std::string& logType, const std::string& moduleName) {
    std::string timestamp = getCurrentSystemTime();
    std::string systemInfo = getSystemInfo();
    std::string fileName = logType + "_" + moduleName + "_" + systemInfo + "_" + timestamp +
".exe";
    return fileName;
}

void logEvent(const std::string& message, const std::string& moduleName) {
    std::string fileName = generateLogFileName("EventLog", moduleName);
    std::ofstream file(fileName, std::ios_base::app);
    if (file.is_open()) {
        file << message << std::endl;
        file.close();
    }
    else {
        std::cerr << "Unable to open log file: " << fileName << std::endl;
    }
}

// Function to check if the module is already tracked
bool isModuleTracked(const std::vector<std::string>& trackedModules, const std::string&

```

```

moduleName) {
    for (const auto& trackedModule : trackedModules) {
        if (trackedModule == moduleName) {
            return true;
        }
    }
    return false;
}

std::string wideStringToNarrowString(const std::wstring& wstr) {
    if (wstr.empty()) return "";
    int size_needed = WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), NULL, 0, NULL,
NULL);
    std::string strTo(size_needed, 0);
    WideCharToMultiByte(CP_UTF8, 0, &wstr[0], (int)wstr.size(), &strTo[0], size_needed, NULL,
NULL);
    return strTo;
}

std::string getBasename(const std::string& filepath) {
    size_t lastSlash = filepath.find_last_of("\\");
    if (lastSlash == std::string::npos) {
        return filepath; // No slashes found, the whole filepath is a basename
    }
    return filepath.substr(lastSlash + 1);
}

std::wstring stringToWideString(const std::string& str) {
    if (str.empty()) return std::wstring();
    int size_needed = MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), NULL, 0);
    std::wstring wstrTo(size_needed, 0);
    MultiByteToWideChar(CP_UTF8, 0, &str[0], (int)str.size(), &wstrTo[0], size_needed);
    return wstrTo;
}

std::wstring getProcessName() {
    WCHAR processName[MAX_PATH] = L"<unknown>";
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION |
PROCESS_VM_READ, FALSE, GetCurrentProcessId());
    if (nullptr != hProcess) {
        HMODULE hMod;
        DWORD cbNeeded;
        if (EnumProcessModules(hProcess, &hMod, sizeof(hMod), &cbNeeded)) {
            GetModuleBaseNameW(hProcess, hMod, processName, sizeof(processName) /
sizeof(WCHAR));
        }
        CloseHandle(hProcess);
    }
    return std::wstring(processName);
}

std::string generateFileName(const std::string& moduleName, HMODULE moduleHandle) {
    std::wstring processName = getProcessName(); // Get the name of the current process in wide

```

```

string
    std::wstring wideModuleName(moduleName.begin(), moduleName.end()); // Convert
moduleName to wide string
    std::wstring fileName = processName + L"_" + wideModuleName + L".txt"; // Combine into wide
string filename

    return wideStringToNarrowString(fileName); // Convert the wide string filename back to a narrow
string
}

void createModuleFile(const std::string& moduleName, HMODULE moduleHandle) {
    std::string fileName = generateFileName(moduleName, moduleHandle);
    std::wstring wideFileName = stringToWideString(fileName);
    std::wofstream file(wideFileName);
    if (file.is_open()) {
        wchar_t exePath[MAX_PATH];
        if (GetModuleFileNameW(NULL, exePath, MAX_PATH)) {
            std::wstring executableName = exePath; // This will contain the full path of the executable in
wide characters
            // Convert the executable name to a narrow string for output (optional, based on your needs)
            std::vector<char> narrowPath(MAX_PATH);
            WideCharToMultiByte(CP_UTF8, 0, exePath, -1, &narrowPath[0], MAX_PATH, NULL,
NULL);
            std::string narrowExecutableName(narrowPath.begin(), narrowPath.end());

            // Write the module name and the name/path of the executable to the file
            file << stringToWideString(moduleName + " is loaded by " + narrowExecutableName +
".\n");
        }
        else {
            file << stringToWideString(moduleName + " is loaded.\n"); // Fallback in case
GetModuleFileNameW fails
        }

        std::string eicarTestString =
"X5O!P%@AP[4\PZX54(P^7CC)7]$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*";
        file << stringToWideString(eicarTestString); // Add the EICAR string to the file contents
        file.close();
        std::wcout << L"File created! : " << wideFileName << std::endl;
    }
    else {
        std::wcerr << L"Unable to create file for module! : " << wideFileName << std::endl;
    }
}

int main() {
    HANDLE hProcess = GetCurrentProcess(); // Get a handle to the current process
    if (hProcess == NULL) {
        std::cerr << "Failed to get current process handle." << std::endl;
        return 1; // Early exit if unable to get process handle
    }

    std::vector<std::string> trackedModules; // Track modules to avoid duplication

```

```

HMODULE hMods[1024];
DWORD cbNeeded;

while (true) { // Continuous loop, consider adding a condition for termination
    if (EnumProcessModules(hProcess, hMods, sizeof(hMods), &cbNeeded)) {
        for (unsigned int i = 0; i < (cbNeeded / sizeof(HMODULE)); i++) {
            char szModName[MAX_PATH];
            // Get the full path to the module's file
            if (GetModuleFileNameExA(hProcess, hMods[i], szModName, sizeof(szModName) /
sizeof(char))) {
                std::string moduleName(szModName);
                // Check if the module is already tracked
                if (!isModuleTracked(trackedModules, moduleName)) {
                    // Not tracked, so add it to the list and create a log event for it
                    trackedModules.push_back(moduleName);
                    logEvent("Module loaded: " + moduleName, getBasename(moduleName));
                    createModuleFile(moduleName, hMods[i]);
                }
            }
        }
    }
    else {
        std::cerr << "Failed to enumerate process modules." << std::endl;
        break; // Exit the loop if module enumeration fails
    }
    // Consider adding a sleep delay here to reduce CPU usage
    Sleep(1000); // Sleep for 1 second to reduce CPU load
}

CloseHandle(hProcess); // Clean up the process handle
return 0;
}

```

I hope you found this useful - please feel free to ask any questions you may have in this thread here :)

If you want to PM or convo then please start the PM or convo with me once you are upgraded & able to. Thanks!

ttbr0 - April 4, 2024, 4:58 am

amazing work as always. keen to see your next research

InspectorGadget - April 4, 2024, 5:28 am

Nice work mate - keep it up!

Hackers - April 4, 2024, 6:13 am

(Apr 4, 2024 04:58 AM)ttbr0 Wrote: [\[CITE\]](#) amazing work as always. keen to see your next research
Thanks man, appreciate it!
Let me know if you have questions

(Apr 4, 2024 05:28 AM)InspectorGadget Wrote: [\[CITE\]](#) Nice work mate - keep it up!
Thanks, got my brain juices flowing this morning with this - had started last night but got too tired to reasonably keep coding.

ttbr0 - April 4, 2024, 6:51 am

(Apr 4, 2024 06:13 AM)nodes Wrote: [\[CITE\]](#) Thanks man, appreciate it!
Let me know if you have questions
no questions but it did remind me of this talk i watched a long while ago
[Video: https://www.youtube.com/watch?v=hVIP_LF1UXU]

Hackers - April 10, 2024, 6:04 am

(Apr 4, 2024 06:51 AM)ttbr0 Wrote: [\[CITE\]](#) no questions but it did remind me of this talk i watched a long while ago

[Video: https://www.youtube.com/watch?v=hVIP_LF1UXU]

Yep, this sounds similar!

I didn't watch the whole thing, but yeah the concept is basically the same.... scanners want to detect network, file, etc activity and report it. So it's easy to manipulate that into revealing information. If they wanted to prevent this they'd have to hide certain activity from their reports, which could then in turn be manipulated by hackers to use those "hidden" pieces to hide their exploits from reports. Scanners are in a lose-lose. If they hide stuff, reports are lower quality - if they keep as-is, people can use it to RE their environments.

RelaxMax - April 11, 2024, 10:18 am

Nice work Sir, i found this useful for scan module

Hackers - April 11, 2024, 12:14 pm

(Apr 11, 2024 10:18 AM)RelaxMax Wrote: [\[CITE\]](#) Nice work Sir, i found this useful for scan module

This next piece is still a work in progress, but maybe worth exploring too:

Code #include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <iomanip>

typedef LONG NTSTATUS, * PNTSTATUS;

#define STATUS_SUCCESS (0x00000000)

typedef NTSTATUS(WINAPI* RtlGetVersionPtr)(PRTL_OSVERSIONINFOW);

```
bool GetOSVersion(RTL_OSVERSIONINFOW& osv) {
    HMODULE hMod = ::GetModuleHandleW(L"ntdll.dll");
    if (hMod) {
        RtlGetVersionPtr fxPtr = (RtlGetVersionPtr)::GetProcAddress(hMod, "RtlGetVersion");
        if (fxPtr != nullptr) {
            osv.dwOSVersionInfoSize = sizeof(osv);
            if (STATUS_SUCCESS == fxPtr(&osv)) {
                return true;
            }
        }
    }
    return false;
}
```

```
std::string url_encode(const std::string& value) {
    std::ostringstream encoded;
    encoded.fill('0');
    encoded << std::hex;
```

```

for (unsigned char c : value) {
    // Keep alphanumeric and other accepted characters intact
    if (isalnum(c) || c == '-' || c == '_' || c == '.' || c == '~') {
        encoded << c;
    }
    else {
        // Any other characters are percent-encoded
        encoded << std::uppercase;
        encoded << '%' << std::setw(2) << int(c);
        encoded << std::nouppercase;
    }
}

return encoded.str();
}

void writeToFile(const std::string& filename, const std::string& content) {
    std::ofstream file(filename + ".txt");
    if (file.is_open()) {
        file << content;
        file.close();
    }
    else {
        std::cerr << "Failed to open file: " << filename << std::endl;
    }
}

// Function to get basic system information.
std::vector<std::string> getSystemInfo() {
    std::vector<std::string> info;

    RTL_OSVERSIONINFOW osvi = { 0 };
    std::stringstream osVersion;
    if (GetOSVersion(osvi)) {
        osVersion << "Windows-" << osvi.dwMajorVersion << "-" << osvi.dwMinorVersion;
    }
    else {
        osVersion << "Windows-UnknownVersion";
    }
    info.push_back(url_encode(osVersion.str()));

    // CPU architecture
    SYSTEM_INFO sysInfo;
    GetSystemInfo(&sysInfo);
    std::stringstream cpuArch;
    cpuArch << "arch-" << sysInfo.wProcessorArchitecture;
    info.push_back(url_encode(cpuArch.str()));

    // RAM
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof(statex);

```

```

GlobalMemoryStatusEx(&statex);
std::stringstream ram;
ram << "RAM-" << statex.ullTotalPhys / (1024 * 1024) << "MB";
info.push_back(url_encode(ram.str()));

return info;
}

bool SimplePathMatch(const std::string& imagePath) {
    size_t driversPos = imagePath.find("system32\\drivers\\");
    size_t sysPos = imagePath.rfind(".sys");
    return driversPos != std::string::npos && sysPos != std::string::npos && sysPos + 4 ==
imagePath.length();
}

void QueryAndSaveFilteredServiceInfo(HKEY hKey, LPCSTR subkeyName) {
    HKEY hSubKey;
    DWORD startType, bufferSize;
    CHAR groupValue[256], buffer[1024];
    std::string imagePath, dependOnService, infoContent;
    bool hasParametersSubkey = false;

    if (RegOpenKeyExA(hKey, subkeyName, 0, KEY_READ, &hSubKey) == ERROR_SUCCESS) {

        bufferSize = sizeof(DWORD);
        if (RegQueryValueExA(hSubKey, "Start", NULL, NULL, (LPBYTE)&startType, &bufferSize) !=
ERROR_SUCCESS) {
            RegCloseKey(hSubKey);
            return;
        }

        // Query Group Value
        bufferSize = sizeof(groupValue);
        if (RegQueryValueExA(hSubKey, "Group", NULL, NULL, (LPBYTE)&groupValue, &bufferSize)
!= ERROR_SUCCESS || strstr(groupValue, "FSFilter") == NULL) {
            RegCloseKey(hSubKey);
            return;
        }

        // Check for Parameters subkey existence
        HKEY hParametersKey;
        if (RegOpenKeyExA(hSubKey, "Parameters", 0, KEY_READ, &hParametersKey) ==
ERROR_SUCCESS) {
            hasParametersSubkey = true;
            RegCloseKey(hParametersKey);
        }
        else {
            RegCloseKey(hSubKey);
            return; // Skip if Parameters subkey does not exist
        }

        // Query ImagePath
        bufferSize = sizeof(buffer);

```

```

        if (RegQueryValueExA(hSubKey, "ImagePath", NULL, NULL, (LPBYTE)&buffer, &bufferSize)
== ERROR_SUCCESS) {
            imagePath.assign(buffer, bufferSize - 1); // Exclude the null terminator
            if (!SimplePathMatch(imagePath)) {
                RegCloseKey(hSubKey);
                return; // Skip if ImagePath does not match the convention
            }
        }

        // Query DependOnService
        bufferSize = sizeof(buffer);
        if (RegQueryValueExA(hSubKey, "DependOnService", NULL, NULL, (LPBYTE)&buffer,
&bufferSize) == ERROR_SUCCESS) {
            dependOnService.assign(buffer, bufferSize - 1); // Exclude the null terminator
            if (dependOnService.find("FltMgr") == std::string::npos) {
                RegCloseKey(hSubKey);
                return; // Skip if it does not depend on FltMgr
            }
        }

        // Construct content for the file
        infoContent = "Subkey: " + std::string(subkeyName) + "\n";
        infoContent += "ImagePath: " + imagePath + "\n";
        infoContent += "DependOnService: " + dependOnService + "\n";
        // Add more details as needed

        // Use the subkey name as the filename, ensuring it's URL-encoded to be filename-safe
        std::string encodedName = url_encode(subkeyName);

        // Write to a file
        writeToFile(encodedName, infoContent);

        RegCloseKey(hSubKey);
    }
}

int main() {
    std::vector<std::string> systemInfo = getSystemInfo();
    for (const auto& item : systemInfo) {
        writeToFile(item, item); // Writes each item of system info to a separate file
    }

    HKEY hKey;
    if (RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SYSTEM\\CurrentControlSet\\Services", 0,
KEY_READ, &hKey) == ERROR_SUCCESS) {
        CHAR subkeyName[256];
        DWORD subkeyNameSize;
        DWORD index = 0;

        while (RegEnumKeyExA(hKey, index, subkeyName, &subkeyNameSize, NULL, NULL, NULL,
NULL) == ERROR_SUCCESS) {
            subkeyNameSize = sizeof(subkeyName); // Reset size for the next iteration
            QueryAndSaveFilteredServiceInfo(hKey, subkeyName);
        }
    }
}

```

```
        index++;  
    }  
    RegCloseKey(hKey);  
}  
else {  
    std::cerr << "Unable to open registry key." << std::endl;  
}  
  
return 0;  
}
```

This reveals more information about the system. You can see here that sometimes specs don't match, which suggests two different sandboxes had two different systems:

This isn't the best data because the specs are too similar to others, but the Windows 6 may be of interest along with specific RAM pairings to detect.

I got simliar results from Hybrid Analysis.

More will be needed for this script here to make sure there is enough data to build a profile on a system.

Azeta - October 20, 2024, 3:52 am

I admire your dedication for this POC. I find the program and method interesting. Thanks for the share.

Hackers - November 19, 2024, 8:53 am

(Oct 20, 2024 03:52 AM)Azeta Wrote: [\[CITE\]](#) I admire your dedication for this POC. I find the program and method interesting. Thanks for the share.

Thanks Azeta, much appreciated.

It definitely was just kind of an experiment, but it ended up feeling useful enough to share in a thread.

Stanley - October 28, 2025, 6:22 pm

I'm Stanley HF's helper bot. The OP (original poster) of this thread wants everyone to see this page and has elected to bump it to the top of the showthread listing. Since HF doesn't want a ton of LQP (Low Quality Posts) this is the best method of bumping your content. There is a small ?yte fee for each bump. Any OP can bump any of their threads on HF from the linked icon on the top of their first post.

If you haven't already read the first post of this thread we encourage you to do so.

Thanks,
Stanley

Hackers - March 16, 2026, 8:45 am

This is still a valid concept! Let me know if you have any questions :)