

Forum:[Hack Forums](#)

Thread: [CPU Monitoring Tool \[C++ Source Code\] - Scan CPU % Every X Seconds](#)

Hackers - May 8, 2024, 7:48 pm

CPU Monitoring Tool C++ Source Code

When it comes to keeping your computer safe from malware or other malicious software, the first thing is to detect that it's there in the first place. There are many ways to do this, but one thing you can do is look for abnormal CPU usage. Of course, having hundreds of Chrome tabs, or opening a large Adobe file will eat through resources (RAM, CPU, GPU, etc). However, when your computer is truly idle, not updating, not refreshing in the background, your resource usage should be minimal.

For that reason, I asked GPT-4 to create a program using C++ that can find the current CPU % usage, via WMI query lookups (WQL).

Checking CPU % patterns can help identify if a software is running processes "secretly" on your computer.

The full source code can be seen below, but first I will show you some highlights to give you a better idea of how this program works.

Code hres = CoInitializeEx(0, COINIT_MULTITHREADED);
Here is a snippet about this function:

<https://pubs.opengroup.org/onlinepubs/009899899/CHP05GDC.HTM#:~:text=Library%20Initialization%20/%20Uninitialization>

Quote: To use basic COM services, all COM threads of execution in clients and out-of-process servers must call either the CoInitialize() or the CoInitializeEx() function before calling any other COM function except memory allocation calls. CoInitializeEx() replaces the other function, adding a parameter that allows you to specify the threading model of the thread--either apartment-threaded or free-threaded. A call to CoInitialize() simply sets the threading model to apartment-threaded. For information on threading in clients and servers, refer to Chapter 6.

To use COM, you set the security levels before usage.

<https://learn.microsoft.com/en-us/windows/win32/api/combaseapi/nf-combaseapi-coinitializesecurity>

Here are the security levels for the code I'm sharing:

```
Code hres = CoInitializeSecurity(  
    NULL,
```

```

-1,
NULL,
NULL,
RPC_C_AUTHN_LEVEL_DEFAULT,
RPC_C_IMP_LEVEL_IMPERSONATE,
NULL,
EOAC_NONE,
NULL
);

```

According to the docs, once we've established our COM variables, we can then continue on to connect to a WMI namespace. Again, still learning about this myself, but from what I understand, it's a "bridge" or a "proxy" to the core WMI database on your machine.

<https://learn.microsoft.com/en-us/windows/win32/wmisdk/creating-a-connection-to-a-wmi-namespace>

```

Code hres = CoCreateInstance(
    CLSID_WbemLocator,
    0,
    CLSCTX_INPROC_SERVER,
    IID_IWbemLocator, (LPVOID*)&pLoc);

```

// snipped code

```

hres = pLoc->ConnectServer(
    _bstr_t(L"ROOT\\CIMV2"),
    NULL,
    NULL,
    0,
    NULL,
    0,
    0,
    &pSvc
);

```

Finally, now that we have prepared our WMI connection, we can start querying!

Lets get the CPU load percentage:

```

Code HRESULT hres = pSvc->ExecQuery(
    bstr_t("WQL"),
    bstr_t("SELECT LoadPercentage FROM Win32_Processor"),
    WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
    NULL,
    &pEnumerator
);

```

If you want to try out other WQL queries, here's more info:

<https://learn.microsoft.com/en-us/windows/win32/api/wbemcli/nf-wbemcli-iwbemservices-execquery>

```

Code HRESULT ExecQuery(
    [in] const BSTR      strQueryLanguage,
    [in] const BSTR      strQuery,
    [in] long            IFlags,
    [in] IWbemContext    *pCtx,
    [out] IEnumWbemClassObject **ppEnum
);

```

[/align]

Source Code

Spoiler ([Click to View](#))

```
Code #include <iostream>
```

```
#include <comdef.h>
```

```
#include <Wbemidl.h>
```

```
#include <chrono>
```

```
#include <thread>
```

```
#pragma comment(lib, "wbemuuid.lib")
```

```
// Utility function to initialize COM and set up WMI connection
```

```
IWbemServices* InitializeWmi() {
```

```
    HRESULT hres;
```

```
    // Initialize COM
```

```
    hres = CoInitializeEx(0, COINIT_MULTITHREADED);
```

```
    if (FAILED(hres)) {
```

```
        std::cerr << "Failed to initialize COM library. Error code = " << hres << std::endl;
```

```
        return nullptr;
```

```
    }
```

```
    // Set security levels
```

```
    hres = CoInitializeSecurity(
```

```
        NULL,
```

```
        -1,
```

```
        NULL,
```

```
        NULL,
```

```
        RPC_C_AUTHN_LEVEL_DEFAULT,
```

```
        RPC_C_IMP_LEVEL_IMPERSONATE,
```

```
        NULL,
```

```
        EOAC_NONE,
```

```
        NULL
```

```
    );
```

```
    if (FAILED(hres)) {
```

```
        CoUninitialize();
```

```
        std::cerr << "Failed to initialize security. Error code = " << hres << std::endl;
```

```
        return nullptr;
```

```
    }
```

```
IWbemLocator* pLoc = NULL;
```

```
// Create a WMI locator
```

```
hres = CoCreateInstance(
```

```
    CLSID_WbemLocator,
```

```
    0,
```

```
    CLSCTX_INPROC_SERVER,
```

```
    IID_IWbemLocator, (LPVOID*)&pLoc);
```

```

if (FAILED(hres)) {
    CoUninitialize();
    std::cerr << "Failed to create IWbemLocator object. Error code = " << hres << std::endl;
    return nullptr;
}

IWbemServices* pSvc = NULL;

// Connect to WMI namespace
hres = pLoc->ConnectServer(
    _bstr_t(L"ROOT\\CIMV2"),
    NULL,
    NULL,
    0,
    NULL,
    0,
    0,
    &pSvc
);

if (FAILED(hres)) {
    pLoc->Release();
    CoUninitialize();
    std::cerr << "Could not connect. Error code = " << hres << std::endl;
    return nullptr;
}

pLoc->Release();
return pSvc;
}

// Function to get and print CPU usage using WMI
void MonitorCpuUsage(IWbemServices* pSvc) {
    IEnumWbemClassObject* pEnumerator = NULL;
    HRESULT hres = pSvc->ExecQuery(
        bstr_t("WQL"),
        bstr_t("SELECT LoadPercentage FROM Win32_Processor"),
        WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
        NULL,
        &pEnumerator
    );

    if (FAILED(hres)) {
        std::cerr << "Query for processor load failed. Error code = " << hres << std::endl;
        return;
    }

    IWbemClassObject* pclsObj = NULL;
    ULONG uReturn = 0;

    while (pEnumerator) {
        HRESULT hr = pEnumerator->Next(WBEM_INFINITE, 1, &pclsObj, &uReturn);
        if (0 == uReturn) {

```

```

        break;
    }

    VARIANT vtProp;
    hr = pclsObj->Get(L"LoadPercentage", 0, &vtProp, 0, 0);
    if (SUCCEEDED(hr) && vtProp.vt == VT_I4) {
        std::cout << "CPU Load: " << vtProp.intVal << "%" << std::endl;
    }
    VariantClear(&vtProp);
    pclsObj->Release();
}

pEnumerator->Release();
}

int main() {
    IWbemServices* pSvc = InitializeWmi();
    if (!pSvc) {
        return 1; // Failed to initialize WMI
    }

    // Continuously monitor CPU usage every 3 seconds
    while (true) {
        MonitorCpuUsage(pSvc);
        std::this_thread::sleep_for(std::chrono::seconds(3));
    }

    pSvc->Release();
    CoUninitialize();
    return 0;
}

```

I am new to WQL, so excuse me if there are mistakes here. I am still learning from this code myself, and this source code is originally AI-generated, with a good few new functions that I've never used before. However, I can confirm it works on Windows 10 & 11 and it was safe to use on my personal computer.

Stanley - March 16, 2026, 8:34 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

Null State - March 17, 2026, 12:42 am

(May 8, 2024 07:48 PM)Hackers Wrote: [\[CITE\]](#)

CPU Monitoring Tool

C++ Source Code

When it comes to keeping your computer safe from malware or other malicious software, the first thing is to detect that it's there in the first place. There are many ways to do this, but one thing you can do is look for abnormal CPU usage. Of course, having hundreds of Chrome tabs, or opening a large Adobe file will eat through resources (RAM, CPU, GPU, etc). However, when your computer is truly idle, not updating, not refreshing in the background, your resource usage should be minimal.

For that reason, I asked GPT-4 to create a program using C++ that can find the current CPU % usage, via WMI query lookups (WQL).

Checking CPU % patterns can help identify if a software is running processes "secretly" on your computer.

The full source code can be seen below, but first I will show you some highlights to give you a better idea of how this program works.

Code hres = CoInitializeEx(0, COINIT_MULTITHREADED);
Here is a snippet about this function:

<https://pubs.opengroup.org/onlinepubs/009899899/CHP05GDC.HTM#:~:text=Library%20Initialization%20/%20Uninitialization>

To use COM, you set the security levels before usage.

<https://learn.microsoft.com/en-us/windows/win32/api/combaseapi/nf-combaseapi-coinitializesecurity>

Here are the security levels for the code I'm sharing:

```
Code hres = CoInitializeSecurity(  
    NULL,  
    -1,  
    NULL,  
    NULL,  
    RPC_C_AUTHN_LEVEL_DEFAULT,  
    RPC_C_IMP_LEVEL_IMPERSONATE,  
    NULL,  
    EOAC_NONE,  
    NULL  
);
```

According to the docs, once we've established our COM variables, we can then continue on to connect to a WMI namespace. Again, still learning about this myself, but from what I understand, it's a "bridge" or a "proxy" to the core WMI database on your machine.

<https://learn.microsoft.com/en-us/windows/win32/wmisdk/creating-a-connection-to-a-wmi-namespace>

```
Code hres = CoCreateInstance(
    CLSID_WbemLocator,
    0,
    CLSCTX_INPROC_SERVER,
    IID_IWbemLocator, (LPVOID*)&pLoc);
```

// snipped code

```
hres = pLoc->ConnectServer(
    _bstr_t(L"ROOT\\CIMV2"),
    NULL,
    NULL,
    0,
    NULL,
    0,
    0,
    &pSvc
);
```

Finally, now that we have prepared our WMI connection, we can start querying!

Lets get the CPU load percentage:

```
Code HRESULT hres = pSvc->ExecQuery(
    bstr_t("WQL"),
    bstr_t("SELECT LoadPercentage FROM Win32_Processor"),
    WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
    NULL,
    &pEnumerator
);
```

If you want to try out other WQL queries, here's more info:

<https://learn.microsoft.com/en-us/windows/win32/api/wbemcli/nf-wbemcli-iwbemservices-execquery>

```
Code HRESULT ExecQuery(
    [in] const BSTR      strQueryLanguage,
    [in] const BSTR      strQuery,
    [in] long            IFlags,
    [in] IWbemContext    *pCtx,
    [out] IEnumWbemClassObject **ppEnum
);
```

[/align]

Source Code

Spoiler ([Click to View](#))

```
Code #include <iostream>
#include <comdef.h>
#include <Wbemidl.h>
#include <chrono>
#include <thread>
```

```
#pragma comment(lib, "wbemuuid.lib")
```

```
// Utility function to initialize COM and set up WMI connection
IWbemServices* InitializeWmi() {
```

```
HRESULT hres;
```

```
// Initialize COM
```

```
hres = CoInitializeEx(0, COINIT_MULTITHREADED);
```

```
if (FAILED(hres)) {
```

```
    std::cerr << "Failed to initialize COM library. Error code = " << hres << std::endl;
```

```
    return nullptr;
```

```
}
```

```
// Set security levels
```

```
hres = CoInitializeSecurity(
```

```
    NULL,
```

```
    -1,
```

```
    NULL,
```

```
    NULL,
```

```
    RPC_C_AUTHN_LEVEL_DEFAULT,
```

```
    RPC_C_IMP_LEVEL_IMPERSONATE,
```

```
    NULL,
```

```
    EOAC_NONE,
```

```
    NULL
```

```
);
```

```
if (FAILED(hres)) {
```

```
    CoUninitialize();
```

```
    std::cerr << "Failed to initialize security. Error code = " << hres << std::endl;
```

```
    return nullptr;
```

```
}
```

```
IWbemLocator* pLoc = NULL;
```

```
// Create a WMI locator
```

```
hres = CoCreateInstance(
```

```
    CLSID_WbemLocator,
```

```
    0,
```

```
    CLSCTX_INPROC_SERVER,
```

```
    IID_IWbemLocator, (LPVOID*)&pLoc);
```

```
if (FAILED(hres)) {
```

```
    CoUninitialize();
```

```
    std::cerr << "Failed to create IWbemLocator object. Error code = " << hres << std::endl;
```

```
    return nullptr;
```

```
}
```

```
IWbemServices* pSvc = NULL;
```

```
// Connect to WMI namespace
```

```
hres = pLoc->ConnectServer(
```

```
    _bstr_t(L"ROOT\\CIMV2"),
```

```
    NULL,
```

```
    NULL,
```

```
    0,
```

```
    NULL,
```

```
    0,
```

```

    0,
    &pSvc
);

if (FAILED(hres)) {
    pLoc->Release();
    CoUninitialize();
    std::cerr << "Could not connect. Error code = " << hres << std::endl;
    return nullptr;
}

pLoc->Release();
return pSvc;
}

// Function to get and print CPU usage using WMI
void MonitorCpuUsage(IWbemServices* pSvc) {
    IEnumWbemClassObject* pEnumerator = NULL;
    HRESULT hres = pSvc->ExecQuery(
        bstr_t("WQL"),
        bstr_t("SELECT LoadPercentage FROM Win32_Processor"),
        WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
        NULL,
        &pEnumerator
    );

    if (FAILED(hres)) {
        std::cerr << "Query for processor load failed. Error code = " << hres << std::endl;
        return;
    }

    IWbemClassObject* pclsObj = NULL;
    ULONG uReturn = 0;

    while (pEnumerator) {
        HRESULT hr = pEnumerator->Next(WBEM_INFINITE, 1, &pclsObj, &uReturn);
        if (0 == uReturn) {
            break;
        }

        VARIANT vtProp;
        hr = pclsObj->Get(L"LoadPercentage", 0, &vtProp, 0, 0);
        if (SUCCEEDED(hr) && vtProp.vt == VT_I4) {
            std::cout << "CPU Load: " << vtProp.intVal << "%" << std::endl;
        }
        VariantClear(&vtProp);
        pclsObj->Release();
    }

    pEnumerator->Release();
}

int main() {

```

```

IWbemServices* pSvc = InitializeWmi();
if (!pSvc) {
    return 1; // Failed to initialize WMI
}

// Continuously monitor CPU usage every 3 seconds
while (true) {
    MonitorCpuUsage(pSvc);
    std::this_thread::sleep_for(std::chrono::seconds(3));
}

pSvc->Release();
CoUninitialize();
return 0;
}

```

I am new to WQL, so excuse me if there are mistakes here. I am still learning from this code myself, and this source code is originally AI-generated, with a good few new functions that I've never used before. However, I can confirm it works on Windows 10 & 11 and it was safe to use on my personal computer.

Do you have a repo anywhere for this?

zyleak - April 2, 2026, 1:56 pm

hey bro u didn't need to block me and give me a -12, im new here and i thought this was a hacker forum, where you are allowed to ask anything.

UPGRADING - May 9, 2026, 6:40 am

(May 8, 2024 07:48 PM)Hackers Wrote: [\[CITE\]](#)

CPU Monitoring Tool
C++ Source Code

When it comes to keeping your computer safe from malware or other malicious software, the first thing is to detect that its there in the first place. There are many ways to do this, but one thing you can do is look for abnormal CPU usage. Of course, having hundreds of Chrome tabs, or opening a large Adobe file will eat through resources (RAM, CPU, GPU, etc). However, when your computer is truly idle, not updating, not refreshing in the background, your resource usage should be minimal.

For that reason, I asked GPT-4 to create a program using C++ that can find the current CPU % usage, via WMI query lookups (WQL).

Checking CPU % patterns can help identify if a software is running processes "secretly" on your computer.

The full source code can be seen below, but first I will show you some highlights to give you a better idea of how this program works.

```
Code hres = CoInitializeEx(0, COINIT_MULTITHREADED);
```

Here is a snippet about this function:

<https://pubs.opengroup.org/onlinepubs/009899899/CHP05GDC.HTM#:~:text=Library%20Initialization%20%20Uninitialization>

To use COM, you set the security levels before usage.

<https://learn.microsoft.com/en-us/windows/win32/api/combaseapi/nf-combaseapi-coinitializesecurity>

Here are the security levels for the code I'm sharing:

```
Code hres = CoInitializeSecurity(  
    NULL,  
    -1,  
    NULL,  
    NULL,  
    RPC_C_AUTHN_LEVEL_DEFAULT,  
    RPC_C_IMP_LEVEL_IMPERSONATE,  
    NULL,  
    EOAC_NONE,  
    NULL  
);
```

According to the docs, once we've established our COM variables, we can then continue on to connect to a WMI namespace. Again, still learning about this myself, but from what I understand, it's a "bridge" or a "proxy" to the core WMI database on your machine.

<https://learn.microsoft.com/en-us/windows/win32/wmisdk/creating-a-connection-to-a-wmi-namespace>

```
Code hres = CoCreateInstance(  
    CLSID_WbemLocator,  
    0,  
    CLSCTX_INPROC_SERVER,  
    IID_IWbemLocator, (LPVOID*)&pLoc);
```

// snipped code

```
hres = pLoc->ConnectServer(  
    _bstr_t(L"ROOT\\CIMV2"),  
    NULL,  
    NULL,  
    0,  
    NULL,  
    0,  
    0,  
    &pSvc  
);
```

Finally, now that we have prepared our WMI connection, we can start querying!

Lets get the CPU load percentage:

```
Code HRESULT hres = pSvc->ExecQuery(  
    bstr_t("WQL"),  
    bstr_t("SELECT LoadPercentage FROM Win32_Processor"),
```

```

    WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
    NULL,
    &pEnumerator
);

```

If you want to try out other WQL queries, here's more info:

<https://learn.microsoft.com/en-us/windows/win32/api/wbemcli/nf-wbemcli-iwbemservices-execquery>

```

Code HRESULT ExecQuery(
    [in] const BSTR      strQueryLanguage,
    [in] const BSTR      strQuery,
    [in] long            IFlags,
    [in] IWbemContext    *pCtx,
    [out] IEnumWbemClassObject **ppEnum
);

```

[/align]

Source Code

Spoiler ([Click to View](#))

```

Code #include <iostream>
#include <comdef.h>
#include <Wbemidl.h>
#include <chrono>
#include <thread>

```

```

#pragma comment(lib, "wbemuuid.lib")

```

```

// Utility function to initialize COM and set up WMI connection

```

```

IWbemServices* InitializeWmi() {
    HRESULT hres;

    // Initialize COM
    hres = CoInitializeEx(0, COINIT_MULTITHREADED);
    if (FAILED(hres)) {
        std::cerr << "Failed to initialize COM library. Error code = " << hres << std::endl;
        return nullptr;
    }
}

```

```

// Set security levels

```

```

hres = CoInitializeSecurity(
    NULL,
    -1,
    NULL,
    NULL,
    RPC_C_AUTHN_LEVEL_DEFAULT,
    RPC_C_IMP_LEVEL_IMPERSONATE,
    NULL,
    EOAC_NONE,
    NULL
);

```

```

if (FAILED(hres)) {

```

```

    CoUninitialize();
    std::cerr << "Failed to initialize security. Error code = " << hres << std::endl;
    return nullptr;
}

IWbemLocator* pLoc = NULL;

// Create a WMI locator
hres = CoCreateInstance(
    CLSID_WbemLocator,
    0,
    CLSCTX_INPROC_SERVER,
    IID_IWbemLocator, (LPVOID*)&pLoc);

if (FAILED(hres)) {
    CoUninitialize();
    std::cerr << "Failed to create IWbemLocator object. Error code = " << hres << std::endl;
    return nullptr;
}

IWbemServices* pSvc = NULL;

// Connect to WMI namespace
hres = pLoc->ConnectServer(
    _bstr_t(L"ROOT\\CIMV2"),
    NULL,
    NULL,
    0,
    NULL,
    0,
    0,
    &pSvc
);

if (FAILED(hres)) {
    pLoc->Release();
    CoUninitialize();
    std::cerr << "Could not connect. Error code = " << hres << std::endl;
    return nullptr;
}

pLoc->Release();
return pSvc;
}

// Function to get and print CPU usage using WMI
void MonitorCpuUsage(IWbemServices* pSvc) {
    IEnumWbemClassObject* pEnumerator = NULL;
    HRESULT hres = pSvc->ExecQuery(
        bstr_t("WQL"),
        bstr_t("SELECT LoadPercentage FROM Win32_Processor"),
        WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY,
        NULL,

```

```

        &pEnumerator
    );

    if (FAILED(hres)) {
        std::cerr << "Query for processor load failed. Error code = " << hres << std::endl;
        return;
    }

    IWbemClassObject* pclsObj = NULL;
    ULONG uReturn = 0;

    while (pEnumerator) {
        HRESULT hr = pEnumerator->Next(WBEM_INFINITE, 1, &pclsObj, &uReturn);
        if (0 == uReturn) {
            break;
        }

        VARIANT vtProp;
        hr = pclsObj->Get(L"LoadPercentage", 0, &vtProp, 0, 0);
        if (SUCCEEDED(hr) && vtProp.vt == VT_I4) {
            std::cout << "CPU Load: " << vtProp.intVal << "%" << std::endl;
        }
        VariantClear(&vtProp);
        pclsObj->Release();
    }

    pEnumerator->Release();
}

int main() {
    IWbemServices* pSvc = InitializeWmi();
    if (!pSvc) {
        return 1; // Failed to initialize WMI
    }

    // Continuously monitor CPU usage every 3 seconds
    while (true) {
        MonitorCpuUsage(pSvc);
        std::this_thread::sleep_for(std::chrono::seconds(3));
    }

    pSvc->Release();
    CoUninitialize();
    return 0;
}

```

I am new to WQL, so excuse me if there are mistakes here. I am still learning from this code myself, and this source code is originally AI-generated, with a good few new functions that I've never used before. However, I can confirm it works on Windows 10 & 11 and it was safe to use on my personal computer.

how is it different from <https://www.alcpu.com/CoreTemp/>

