

Forum:[Hack Forums](#)

Thread: [Obfuscation/evasion methods question](#)

Binance - August 13, 2023, 5:46 pm

We all know how much evasion and persistence changed in the last few years. Old methods of hollowing, mounted PE, balloon executables are still somewhat working.

But in line with cryptography does the strength of the so called "crypt" really matter anymore? Or is it all variances of mockingjay ?

osn64698 - August 14, 2023, 11:12 pm

(Aug 13, 2023 05:46 PM)signed0 Wrote: [\[CITE\]](#) We all know how much evasion and persistence changed in the last few years. Old methods of hollowing, mounted PE, balloon executables are still somewhat working.

But in line with cryptography does the strength of the so called "crypt" really matter anymore? Or is it all variances of mockingjay ?

All these news sites that talk about "Mockingjay" like it is something new or unique are laughing material. What is next? LMAO.

There were loaders that were abusing LoadLibrary along with NtMapViewOfSection hook in order to MAP themselves to MEM_IMAGE region / RWX etc ages ago. You don't need to target any specific library, LMFAO.

It is pure degeneracy that the "author" of Mockingjay even considers "loading" a fresh NTDLL copy from DISK, why??? This on its own can be a huge detection vector.

Why not use the KnownDlls cache?

Fuck me but the most obvious way is achieved without performing anything special, merely a bubble sort is enough on the address of the export table of NTDLL (Syscall id is the index in that sorted table)

Now if you want to "unhook" functions:

Some AV companies are so retarded that they allow you to call their own "exported functions" that will allow you to remove all hooks, take Bitdefender for example. (bdhkm)

You can also quite easily prevent their DLL from loading in the first place, with less noise, they do not give a fuck.

Iterating PEB nowadays is also targeted, both by emulation engines and other methods that I won't disclose.

If you want to defeat emulators, create time puzzles / spread your code into blocks / add meaningless junk code, and encrypt/decrypt them on the fly, add a shit ton of opaque predicates at the beginning to prevent static analysis / retarded reverse engineers.

Randomize your PE Header, wipe section headers, and wipe un-needed sections such as .reloc/ .rdata/ .rsrsc.

This will also allow you to fully defeat memory scanning.

"Balloon executable" is a term that was invented by a guy here who deserves credit for creating some nice shit, but again, nothing too out of the ordinary.

RunPE should not be considered in a serious crypter nowadays, too noisy. Unmapping the original EXE is bound to be detected. This shit 100% does not fly with any proper EDR, maybe your "common" 10\$ antivirus.

Headers that were remapped are a big no-no.

My own poly crypter achieves FUD time of 1 month + with Virustotal submissions easily. Also detections are fully wiped upon rebuild which takes me 1 second.

So it is possible, do your own shit.

Binance - August 18, 2023, 3:35 pm

(Aug 13, 2023 05:57 PM)Dyne Wrote: [\[CITE\]](#) I still say the most useful evasion technique is cutting AV off at the knees.

I wanted to thank u for the amount of help the last few days.

Route - August 18, 2023, 3:42 pm

(Aug 13, 2023 05:57 PM)Dyne Wrote: [\[CITE\]](#) You've asked alot in this thread if you read deep enough. It isn't exactly clear.

Yes and no. Does it need to be packed with kkrunchy and fed into PE? no.

Does the strength of your encryption really matter? Are you asking base64 vs AES 512? or AES256 vs 512?

Not really, but it needs to be clean and not representative of the encrypted .exe.

You've somewhat answered your question in your next statement;

Essentially yes.

I still say the most useful evasion technique is cutting AV off at the knees. Script that spam kills IOCTL via req overload (it will still work) doesn't really matter what encryption method you use to drop the actual .exe.

This was actually utilized via a popular ransomware group to load a signed vulnerable driver.

<https://youtu.be/ViWLMfSwGVA>

Mike is king.

" Script that spam kills IOCTL via req overload"

Do i understood your words correct ? If so it should look something like this. No ?

Code import ctypes

```
def kill_ioctl_via_req_overload():
```

```
    DEVICE_NAME = "SomeDeviceName"
```

```
    IOCTL_CODE = 0x12345678
```

```
    BUFFER_SIZE = 1024
```

```
    device_handle = ctypes.windll.kernel32.CreateFileW(
```

```
        DEVICE_NAME,
```

```
        ctypes.c_uint32(0xC0000000), # GENERIC_READ | GENERIC_WRITE
```

```

        ctypes.c_uint32(0), # No sharing
        None,
        ctypes.c_uint32(3), # OPEN_EXISTING
        ctypes.c_uint32(0),
        None
    )

    if device_handle != -1:
        try:
            buffer = ctypes.create_string_buffer(BUFFER_SIZE)
            while True:
                ctypes.windll.kernel32.WriteFile(device_handle, buffer, BUFFER_SIZE, None, None)
                ctypes.windll.kernel32.DeviceIoControl(device_handle, IOCTL_CODE, buffer,
                BUFFER_SIZE, None, 0, None, None)
            except:
                pass
        finally:
            ctypes.windll.kernel32.CloseHandle(device_handle)

    kill_ioctl_via_req_overload()

```

Tentco - November 12, 2023, 6:42 am

(Aug 14, 2023 11:12 PM)osn64698 Wrote: [\[CITE\]](#) All these news sites that talk about "Mockingjay" like it is something new or unique are laughing material. What is next? LMAO.

There were loaders that were abusing L

oadLibrary along with NtMapViewOfSection hook in order to MAP themselves to MEM_IMAGE region / RWX etc ages ago. You don't need to target any specific library, LMFAO.

It is pure degeneracy that the "author" of Mockingjay even considers "loading" a fresh NTDLL copy from DISK, why??? This on its own can be a huge detection vector.

Why not use the KnownDlls cache?

Fuck me but the most obvious way is achieved without performing anything special, merely a bubble sort is enough on the address of the export table of NTDLL (Syscall id is the index in that sorted table)

Now if you want to "unhook" functions:

Some AV companies are so retarded that they allow you to call their own "exported functions" that will allow you to remove all hooks, take Bitdefender for example. (bdhkm)

You can also quite easily prevent their DLL from loading in the first place, with less noise, they do not give a fuck.

Iterating PEB nowadays is also targeted, both by emulation engines and other methods that I won't disclose.

If you want to defeat emulators, create time puzzles / spread your code into blocks / add meaningless junk code, and encrypt/decrypt them on the fly, add a shit ton of opaque predicates at the beginning to prevent static analysis / retarded reverse engineers.

Randomize your PE Header, wipe section headers, and wipe un-needed sections such as .reloc/ .rdata/ .rsrsc.

This will also allow you to fully defeat memory scanning.

"Balloon executable" is a term that was invented by a guy here who deserves credit for creating some nice shit, but again, nothing too out of the ordinary.

RunPE should not be considered in a serious crypter nowadays, too noisy. Unmapping the original EXE is bound to be detected. This shit 100% does not fly with any proper EDR, maybe your "common" 10\$ antivirus.

Headers that were remapped are a big no-no.

My own poly crypter achieves FUD time of 1 month + with Virustotal submissions easily. Also detections are fully wiped upon rebuild which takes me 1 second.

So it is possible, do your own shit.

I'm pretty sure the methods you won't disclose about PEB iteration detection are - Checking base address of kernel32.dll from PEB or getting the base addr of thread stack or OEP.

Hooking NtReadVirtualMemory or NtWriteVirtualMemory. Tracing PEB List Pointers.

Getting the TEB?

* HeapFlags Difference?

These are the only methods I've seen while Reverse Engineering AV's engine.

Also, a good way for evading time-based checks would be bruteforcing the shellcode at runtime.

S0phia - November 13, 2023, 6:10 am

There's a straightforward method to exhaust the majority of AV runtime memory scanners or bypass behavior detections in well-known malware C2.

Allocate a memory region with read and write permissions, sized at 2048, for the shellcode. Fill this region with randomized data, and randomly place the shellcode within it each time.

Change permissions at the randomly chosen shellcode location to PAGE_EXECUTE_READWRITE, and initiate its execution. To evade behavior-like detection, create a thread in a familiar Windows process, such as Explorer, which doesn't trigger alarms. This approach results in almost a 99% success rate in executing most signature-behaviour-like operations.

While not flawless, this solution leaves numerous traces to the researchers or professional EDR, like creating a thread in the remote process. There are alternative techniques, such as thread hijacking, that follow a similar concept. Red Teamers have developed impressive projects, such as Spoofing Call Stacks, showcasing near-perfect execution, making it easy to distinguish an average malware developer.

I believe in giving a man a pole, not a fish I'll leave it up to you.

Whether using direct syscalls or unhooked indirect syscalls, employing these methods particularly blinds the EDR. By relying on Kernel Callbacks and a few other stuff, EDRs face poor visibility and an increased risk of false positives. Ultimately, their challenge lies in decoding telemetry to identify and cut off potential threats.

For instance, exploring ETW and AMSI tracing provides insights into the intricacies of executing .NET as a shellcode with a donut.

About the detection of the file, you can utilize LLVM obfuscation, keep low entropy, and maintain different file info and icons, because these are part of the decision making whatever file could be malicious.

Certificate cloning can even lead to lower detections. Anti-sandboxing is also very important thing because this can break automatic sandbox in analysis in the cloud to make your implant worse to reverse engineer and to automate.

About emulation which is done on your local pc the LLVM obfuscation is going to exhaust is very quickly even if they are lifting to IL to optimize the obfuscation.

You can test various winapi calls to determine if you're being emulated since most of av keeps hardcoded results.

Best Regards.

Tentco - November 13, 2023, 6:45 am

(Nov 13, 2023 07:10 AM)snyk3 Wrote: [\[CITE\]](#) There's a straightforward method to exhaust the majority of AV runtime memory scanners or bypass behavior detections in well-known malware C2.

Allocate a memory region with read and write permissions, sized at 2048, for the shellcode. Fill this region with randomized data, and randomly place the shellcode within it each time.

Change permissions at the randomly chosen shellcode location to PAGE_EXECUTE_READWRITE, and initiate its execution. To evade behavior-like detection, create a thread in a familiar Windows process, such as Explorer, which doesn't trigger alarms. This approach results in almost a 99% success rate in executing most signature-behaviour-like operations.

While not flawless, this solution leaves numerous traces to the researchers or professional EDR, like creating a thread in the remote process. There are alternative techniques, such as thread hijacking, that follow a similar concept. Red Teamers have developed impressive projects, such as Spoofing Call Stacks, showcasing near-perfect execution, making it easy to distinguish an average malware developer.

I believe in giving a man a pole, not a fish I'll leave it up to you.

Whether using direct syscalls or unhooked indirect syscalls, employing these methods particularly blinds the EDR. By relying on Kernel Callbacks and a few other stuff, EDRs face poor visibility and an increased risk of false positives. Ultimately, their challenge lies in decoding telemetry to identify and cut off potential threats.

For instance, exploring ETW and AMSI tracing provides insights into the intricacies of executing .NET as a shellcode with a donut.

About the detection of the file, you can utilize LLVM obfuscation, keep low entropy, and maintain different file info and icons, because these are part of the decision making whatever file could be malicious.

Certificate cloning can even lead to lower detections. Anti-sandboxing is also very important thing because this can break automatic sandbox in analysis in the cloud to make your implant worse to reverse engineer and to automate.

About emulation which is done on your local pc the LLVM obfuscation is going to exhaust is very

quickly even if they are lifting to IL to optimize the obfuscation.

You can test various winapi calls to determine if you're being emulated since most of av keeps hardcoded results.

Best Regards.

You're somewhat correct this would work for most of the AV's but this is NOT even close to bypass EDR's, they doesn't solely depends on kernel callbacks and absolutely not on "few" other stuff atleast instead of offloading the tracing of indirect/direct syscalls they can use hypervisor EPT, even if you nullify/remove something like PsSetCreateProcessNotifyRoutine()(used by carbon black and crowdstrike mostly) there are still lot of shit to take care of.

You can easily detect Thread-Stack spoofing by tracking what NtAllocateVirtualMemory returns to. They can detect a frame with a ret to jmp.

To bypass that you would probably implement a RET spoof -

<https://www.unknowncheats.me/forum/anti-cheat-bypass/268039-x64-return-address-spoofing-source-explanation.html>

For syscalls you will still need to force VEH in ring-3 before EDR could trace stub caller.

Thread hijacking can also be detected via RtlWalkFrameChain to get a handle to a process.

What's funny is that EDR aren't utilizing Working Set List Entries (WSLEs) to detect RWX region change.

There are just lot of shits to bypass like

Heap Monitoring

IAT Hooking(Sentinel fav)

ROP Mitigation

this cycle just doesn't ends.

S0phia - November 13, 2023, 7:34 am

(Nov 13, 2023 07:45 AM)Tentco Wrote: [\[CITE\]](#) You're somewhat correct this would work for most of the AV's but this is NOT even close to bypass EDR's, they doesn't solely depends on kernel callbacks and absolutely not on "few" other stuff atleast instead of offloading the tracing of indirect/direct syscalls they can use hypervisor EPT, even if you nullify/remove something like PsSetCreateProcessNotifyRoutine()(used by carbon black and crowdstrike mostly) there are still lot of shit to take care of.

You can easily detect Thread-Stack spoofing by tracking what NtAllocateVirtualMemory returns to. They can detect a frame with a ret to jmp.

To bypass that you would probably implement a RET spoof -

For syscalls you will still need to force VEH in ring-3 before EDR could trace stub caller.

Thread hijacking can also be detected via RtlWalkFrameChain to get a handle to a process.

What's funny is that EDR aren't utilizing Working Set List Entries (WSLEs) to detect RWX region change.

There are just lot of shits to bypass like

Heap Monitoring

IAT Hooking(Sentinel fav)

ROP Mitigation

this cycle just doesn't ends.

The intention was to be straightforward, mainly emphasizing AV. The aspects you highlighted are connected to the statement: "While not flawless, this solution leaves numerous traces for researchers or professional EDR, such as creating a thread in the remote process. There are alternative techniques, like thread hijacking, that follow a similar concept."

I appreciate your clarification and sharing of knowledge.

In cybersecurity, attackers are constantly improving their skills and using new tactics, while defenders strive to keep up. It can be problematic if this dynamic creates a false sense of security. Advanced EDR methods may be necessary, and the more complex the tasks, the greater the potential for discovering both solutions and vulnerabilities.

Thank you for reaching out. Best regards.

Tentco - November 13, 2023, 7:39 am

(Nov 13, 2023 08:34 AM)snyk3 Wrote: [\[CITE\]](#) The intention was to be straightforward, mainly emphasizing AV. The aspects you highlighted are connected to the statement: "While not flawless, this solution leaves numerous traces for researchers or professional EDR, such as creating a thread in the remote process. There are alternative techniques, like thread hijacking, that follow a similar concept."

I appreciate your clarification and sharing of knowledge.

In cybersecurity, attackers are constantly improving their skills and using new tactics, while defenders strive to keep up. It can be problematic if this dynamic creates a false sense of security. Advanced EDR methods may be necessary, and the more complex the tasks, the greater the potential for discovering both solutions and vulnerabilities.

Thank you for reaching out. Best regards.

Even normal AV's are utilizing emulation take Avast for example shittons of drivers to hook shit and detect reg/filesystem changes, I was just contradicting on the statement "Red Teamers have developed impressive projects, such as Spoofing Call Stacks, showcasing near-perfect execution, making it easy to distinguish an average malware developer." that this even is nowhere close to enough.

Still you have a great thinking.

Have a good day.

S0phia - November 13, 2023, 7:57 am

(Nov 13, 2023 08:39 AM)Tentco Wrote: [\[CITE\]](#) Even normal AV's are utilizing emulation take Avast for example shittons of drivers to hook shit and detect reg/filesystem changes, I was just contradicting on the statement "Red Teamers have developed impressive projects, such as Spoofing Call Stacks, showcasing near-perfect execution, making it easy to distinguish an average malware developer." that this even is nowhere close to enough.

Still you have a great thinking.

Have a good day.

Most AVs utilize emulation, and they do have limits to the more complex calls (they even create more security holes like defender with thier apicall).

Take Windows Defender as an example you can easily look up the emulation engine since they have symbols available.

The `GetComputerName` has a hardcoded value which is 'HAL9TH'.

Take avast and their shit tons of security, for example, extended amsi module

It's not even signed and can be blocked by

PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY policy. The same goes for McAfee and AVG probably.

Thanks!

Tentco - November 13, 2023, 8:03 am

(Nov 13, 2023 08:57 AM)snyk3 Wrote: [\[CITE\]](#) Most AVs utilize emulation, and they do have limits to

the more complex calls (they even create more security holes like defender with their apicall).

Take Windows Defender as an example you can easily look up the emulation engine since they have symbols available.

The `GetComputerName` has a hardcoded value which is 'HAL9TH'.

Take avast and their shit tons of security, for example, extended amsi module

It's not even signed and can be blocked by

PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY policy. The same goes for McAfee and AVG probably.

Thanks!

Let's not focus on one factor as I said before, about the WD part I know since I've analyzed the WD AMSI engine and emulation internals but I'm confused about Avast will check out I will probably look into Avast AMSI COM object specifically ASWAMSI.dll, Thanks.

S0phia - November 13, 2023, 8:07 am

(Nov 13, 2023 09:03 AM)Tentco Wrote: [\[CITE\]](#) Let's not focus on one factor as I said before, about the WD part I know since I've analyzed the WD AMSI engine and emulation internals but I'm confused about Avast will check out I will probably look into Avast AMSI COM object specifically ASWAMSI.dll, Thanks.

Thank you for the constructive exchange of criticism,
I'm glad I've introduced something new.

Best Regards.

osn64698 - November 19, 2023, 10:42 am

(Nov 12, 2023 07:42 AM)Tentco Wrote: [\[CITE\]](#) I'm pretty sure the methods you won't disclose about PEB iteration detection are - Checking base address of kernel32.dll from PEB or getting the base addr of thread stack or OEP.

Hooking NtReadVirtualMemory or NtWriteVirtualMemory. Tracing PEB List Pointers.

Getting the TEB?

* HeapFlags Difference?

These are the only methods I've seen while Reverse Engineering AV's engine.

Also, a good way for evading time-based checks would be bruteforcing the shellcode at runtime.

Well, sentinel1 introduces new entries into the PEB... that is all I am saying.

Either way, this thread has 10/10 knowledge and should be studied by any novice malware writer.

Good job boys.

Tentco - November 19, 2023, 6:32 pm

(Nov 19, 2023 11:42 AM)osn64698 Wrote: [\[CITE\]](#) Well, sentinel1 introduces new entries into the PEB... that is all I am saying.

Either way, this thread has 10/10 knowledge and should be studied by any novice malware writer.

Good job boys.

I will check it out.

Okay seems like I didn't find anything relating new S1 PEB entries.

pnixxx - December 30, 2023, 5:58 am

(Nov 19, 2023 11:42 AM)osn64698 Wrote: [\[CITE\]](#) Well, sentinel1 introduces new entries into the PEB... that is all I am saying.

Either way, this thread has 10/10 knowledge and should be studied by any novice malware writer.

Good job boys.

100% what have I just stumbled across haahha

rebeecer - April 16, 2025, 9:16 am

I have gained a lot from this thread. Thank you