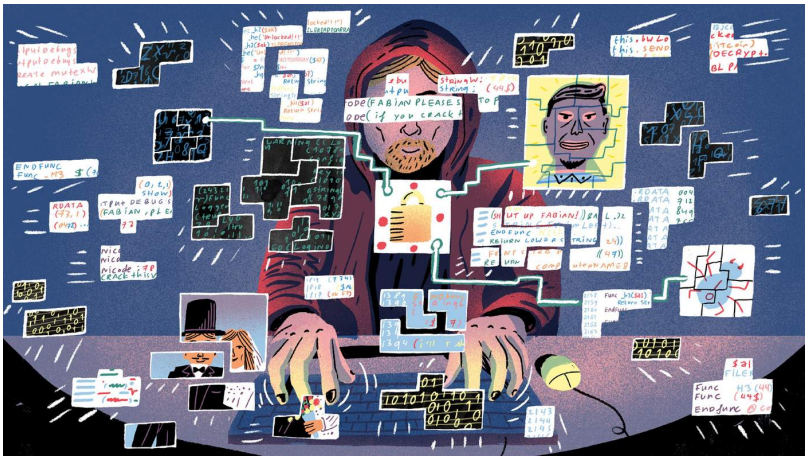
Defeat the Castle - Bypass AV & Advanced XDR solutions.



Generated By 0xsp.com

Introduction

This research paper will demonstrate the unique process hollowing technique used to bypass and divert detection analysis. the following research has been introduced first on CrestCon Asia 2021, and you may watch it on Youtube(https://www.youtube.com/watch?v=H7EMBz7GLMk)

With an advanced newer security defense solution (e.g., EDR, XDR, NGAV), it becomes much harder for offensive security experts to deploy well-known malicious scripts needed for any red teaming simulation engagement such as cobalt strike beacon, Mimikatz, or other tools. Alternatively, a red teamer goes to another way to deploy its scripts through PowerShell after patching AMSI, such as loading a modified version of Mimikatz or other scripts. That's good enough, but if we compare the latest version of Mimikatz with the PowerShell version, we can observe that the newer added techniques and functions are not available yet in the PowerShell available version.

Root Cause

- *Obfuscation of scripts or binaries can still be detected and prevented by 90 % of AV.*
- *The majority of AV/XDR can detect and prevent malicious memory executions.*
- *Tampering and behavioral analysis*
- *indicators of compromise*

Even though you attempt to perform code obfuscation or recompile the sources code with your modified, a chance to bypass the detection is prior low. Advanced solutions will detect it by behavior analysis (HIPS) multi-layer security protection.

So, deploying a unique technique and weaponization to your arsenal is one key factor in achieving complete bypass and reducing the chance of detection from 90 % to 30 % or even less.

Dynamic forking

a technique known as "Process hollowing RUNPE "allows the execution of an executable image within another process's address space. the method works by creating a host process in a suspended state and unmapping the original executable image, followed by a memory allocation. Therefore, it becomes possible to write the replacement executable into the allocated memory space and set the

EAX/RAX of the process's primary thread to the entry point of the replacement executable.

```
NtUnmapViewOfSection
NtReadVirtualMemory
NtWriteVirtualMemory
NtGetContextThread
NtSetContextThread
NtResumeThread
NtWaitForSingleObject
NtClose
NtTerminateProcess
```

Mortar Evasion Technique

Although Dynamic forking is an effective technique with newer detection and prevention mechanisms, it becomes easy to detect and prevent when attempting to load the malicious executable image. Nowadays, any Basic AV solution will take immediate action by isolating the malicious code section and successfully quarantining the malicious code.

Starting from that root cause, I developed a technique that can encrypt and decrypt the executable image inside the memory stream and seek position accordingly.

The mortar technique is an encryption and decryption mechanism for malicious binary using the fly blowfish encryption/ decryption stream. Blowfish is a cipher based on blocks (64 bits) that also uses symmetric keys (both encryption and decryption). In addition, blowfish is based on a Feistel Network(A Feistel iterates a specific function a certain number of times, and each cycle is called around).

Debugging our execution

Mortar encryptor will take the selected binary into base64 format, forward the output into Blowfish on-fly encryption stream, and finalize the process by writing a completed encrypted version of malicious binary.

While the decryption stream process concludes that all data that is read from the source stream is decrypted before it is placed in the output buffer that the AV advanced memory scanner can still

detect. to bypass that, I put some encoding into the decrypted bytes and then moved the decoded data into another memory string stream. For further explanation, attached is the Mortal Loader into the x64gdb debugger. I have captured the exciting stack call addresses from the Coding IDE environment that we may need to toggle a breakpoint for disassembling the ASM instructions.

I made a file containing the following payload tag string "lawlaw" to demonstrate straightforward proof of concept. (the PoC is not valid EXE). After toggling a breakpoint into the first procedure when loading the encrypted file into the string memory, the following lea instruction has been used to load the effective address of encrypted input data in the Assigned address in ESP/RSP.

After that, the loader will start decrypting the encrypted data inside the memory stream with the valid decryption key. As the figure below, we can observe the assembly instruction of the blowfish on fly decryption. Later on, since this output format is base64, it becomes challenging for AV to detect and isolate the malicious image. The technique will copy the malicious payload into another allocated memory address to make it harder.

Test Cases

Below is a detailed table of test cases done so far with different security products

Starting with ESET, internet security was effortless and didn't take a while to run the PE inside the memory without detection; I am not here saying that ESET is good or bad; I am just indicating that ESET becomes blind while dealing with malware execution in memory blocks.

Moving our test into Kaspersky was challenging, but at the end of testing, I was able to bypass and lunch a complete act of compromise, including loading Mimikatz,cobalt strike, MSF

Standing by the previous results, I conclude that All AV solutions are the same, and by using this technique, it is possible to achieve a bypass either by EXE or DLL loader.

Cortex has detected and responded perfectly while performing the attack on an active red teaming engagement. I still believe Cortex is one of the leading XDR solutions to detect and prevent advanced attacks.

Bypass Cortex XDR

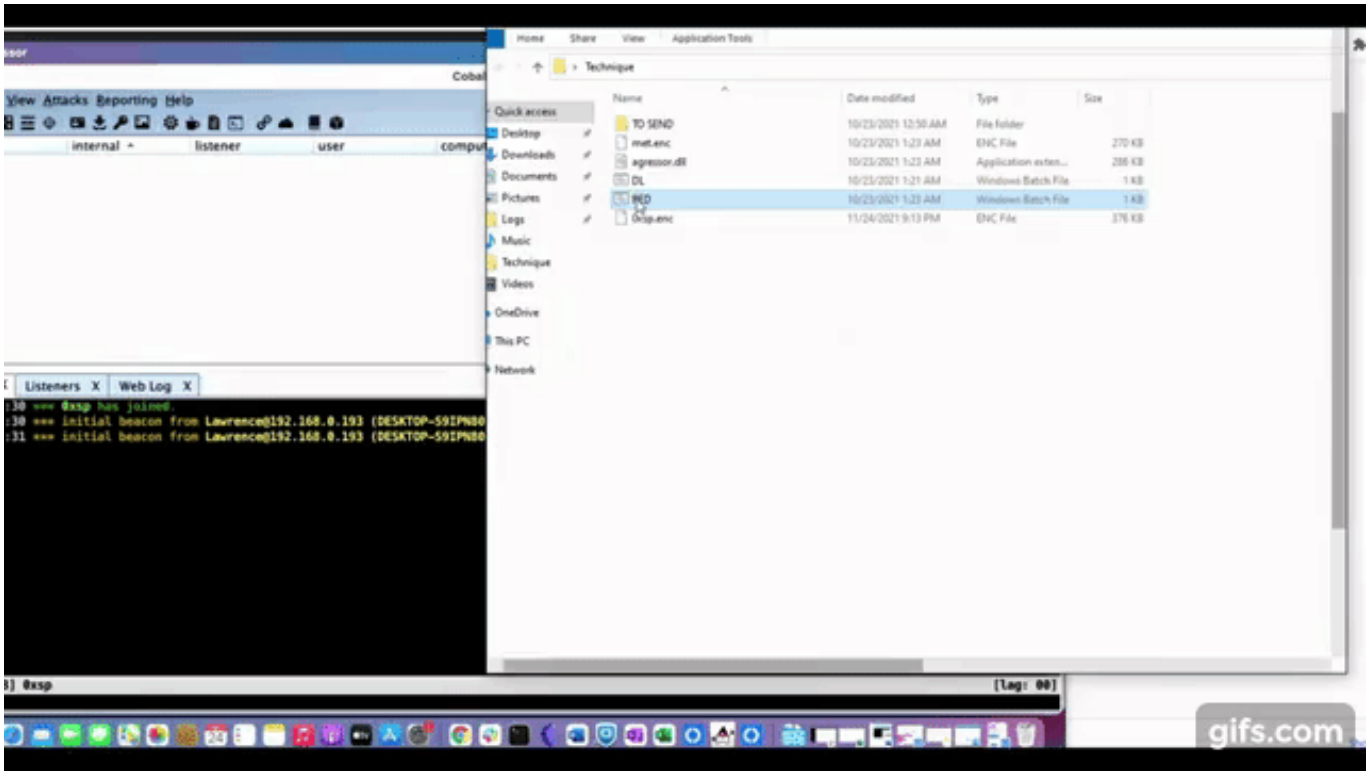
we can notice from the previous figure that Cortex detects any malicious child process execution that includes process spoofing or process hollowing with advanced anti tempering techniques and prevents any possible modification into the newly created process's memory.

After debugging the source code, I discovered an interesting trick on the following Windows API (ReadProcessMemory). So what I have done is by extending the previous value of Thread Context lpbaseAddress from RDX+ $10 into RDX+$100 and decrease the number of bytes to be read from the process from 8 Bytes into 2 bytes. After compiling the source code again and running the attack vector, Cortex XDR prevented it, and it is still not possible to probably get the Cobalt strike beacon working. After further execution of the same payload looks like XDR has prevented the execution again, to craft a bypass, I have to do analysis for recorded events and understand the logic of the prevention module.

So at this point, instead of performing process hollowing to CMD.exe process, I have performed into the primary process of XDR and recompiled the whole loader as DLL. after doing that, XDR prevented the attack again, so the final thing I do is by scripting a bat file that can perform the attack as one-click execution.

```
@echo off
cmd.exe /c loader.dll,stealth
```

in summary, when we click on the bat file, it will execute rundll32.exe to run the loader and load the encrypted binary and execute it inside the memory



Repository

since the project is open-source, all contributions or shown of appreciation are welcomed by opening an issue or modification of the code

https://github.com/0xsp-SRD/mortar

Resources

- Black Hat Asia 17
- Malewarebytes blogpost