

AMBER: Reflective PE Packer

Ege Balci

INVICTUS EUROPE Security Consulting and Intelligence Services LLC
ege.balci@invictuseurope.com

Abstract

Spreading malicious code is a complex problem for malware authors. Because of the recent advancements on malware detection technologies both malware authors and penetration testers having hard time with bypassing security measures and products such as anti-viruses and OS level mitigations. The generally known approaches for malware to bypass security measures has been using obfuscation, encryption and anti-analysis tricks inside malware executable file, but these approaches are losing effect on up to date detection technologies. Machine learning and cloud based analysis mechanisms are very effective for detecting new and novel malware in the form of executable files, using more obfuscation and anti-detection methods makes these systems smarter and harder to bypass. Because of the rapidly increasing datasets of automated malware analysis technologies it is getting very easy to detect obfuscation, encryption or similar anti detection techniques that is used maliciously. The main purpose of this paper is developing a new packing methodology for PE files that can alter the way of delivering the malware to the systems. Instead of trying to find new anti-detection techniques that feed the machine learning datasets, delivering the payload to the systems via fileless code injections directly bypasses most of the security mechanisms. With this new packing method it is possible to convert compiled PE files into multi stage infection payloads that can be used with common software vulnerabilities such as buffer overflows.

1. Introduction

Because of the increasing security standards inside operating systems and rapid improvements on malware detection technologies today's malware authors takes advantage of the transparency offered by in-memory execution methods. In-memory execution or fileless execution of a PE file can be defined as executing a compiled PE file inside the memory with manually

performing the operations that OS loader supposed to do when executing the PE file normally. In-memory execution of a malware facilitates the obfuscation and anti-emulation techniques. Additionally the malware that is using such methods leaves less footprints on the system since it does not have to possess a file inside the hard drive. Combining in-memory execution methods and multi stage infection models allows malware to infect systems with very small sized loader programs; only purpose of a loader is loading and executing the actual malware code via connecting to a remote system. Using small loader codes are hard to detect by security products because of the purpose and the code fragments of loaders are very common among legitimate applications. Malware that are using this approach can still be detected with scanning the memory and inspecting the behaviors of processes but in terms of security products these operation are harder to implement and costly because of the higher resource usage (Ramilli, 2010 [1]). Current rising trend on malware detection technologies is to use the machine learning mechanisms to automate the detection of malwares with feeding very big datasets into the system, as in all machine learning applications this mechanism gets smarter and more accurate in time with absorbing more samples of malware. These mechanisms can feed large numbers of systems that human malware analysts can't scale. Malware Detection Using Machine Learning [2] paper by Gavriluț Dragoș et al from BitDefender Romania Labs widely explains the inner workings of machine learning usage on malware detection. According to the Automatic Analysis of Malware Behavior using Machine Learning [3] paper by Konrad Rieck et al, with enough data and time false positive results will get close to zero percent and deterministic detection of malware will be significantly effective on new and novel malware samples. These claims are the main encouragement of developing this packing method.

2. Related Works

Reflective DLL Injection [4] is a great library injection technique developed by Stephen Fewer and it

is the main inspiration point for developing this new packer. This technique allows in-memory execution of a specially crafted DLL that is written with reflective programming approach. Because of the adopted reflective programming approach this technique allows multi stage payload deployment. Besides the many advantages of this technique it has few limitations. First limitation is the required file format, this technique expects the malware to be developed or recompiled as a DLL file, and unfortunately in most cases converting an already compiled EXE file to DLL is not possible or requires extensive work on the binary. Second limitation is the need for relocation data. Reflective DLL injection technique requires the relocation data for adjusting the base address of the DLL inside the memory. Also this method has been around for a long time, this means up to date security products can easily detect the usage of Reflective DLL injection. Amber will provide solutions for each of these limitations.

Process Hollowing [5] is another commonly known in-memory malware execution method that is using the documented Windows API functions for creating a new process and mapping an EXE file inside it. This method is popular among crypters and packers that are designed to decrease the detection rate of malwares. But this method also has several drawbacks. Because of the Address Space Layout Randomization (ASLR) security measure inside the up-to-date Windows operating systems, the address of memory region when creating a new process is randomized, because of this process hollowing also needs to implement image base relocation on up-to-date Windows systems, As mentioned earlier base relocation requires relocation data inside PE files. Another drawback is because of the usage of specific file mapping and process creation API functions in specific order this method is easy to identify by security products.

Hyperion [6] is a crypter for PE files, developed and presented by Christian Amman in 2012. It explains the theoretic aspects of runtime crypters and how to implement it. The PE parsing approach in assembly and the design perspective used while developing Hyperion helped us for our POC packer.

3. Amber Workflow

The fundamental principle of executing a compiled binary inside the OS memory is possible with imitating the PE loader of the OS. On Windows, PE loader does many important things, between them mapping a file to memory and resolving the addresses of imported

functions are the most important stages for executing a EXE file. Current methods for executing EXE files in memory uses specific windows API functions for mimicking the windows PE loader. Common approach is to create a new suspended process with calling CreateProcess windows API function and mapping the entire EXE image inside it with the help of NtMapViewOfSection, MapViewOfFile and CreateFileMapping functions. Usage of such functions indicates suspicious behavior and increases the detection possibility of the malware. One of the key aspects while developing our packer is using less API functions as possible, in order to avoid the usage of suspicious file mapping API functions our packer uses premapped PE images moreover execution of the malware occurs inside of the target process itself without using the CreateProcess windows API function. The malware executed inside the target process is run with the same process privileges because of the shared _TEB block which is containing the privilege information and configuration of a process. Amber has 2 types of stub, one of them is designed for EXE files that are supporting the ASLR and the other one is for EXE files that are stripped or doesn't have any relocation data inside. The ASLR supported stub uses total of 4 windows API calls and other stub only uses 3 that are very commonly used by majority of legitimate applications.

ASLR Supported Stub:

- VirtualAlloc
- CreateThread
- LoadLibraryA
- GetProcAddress

Non-ASLR Stub:

- VirtualProtect
- LoadLibraryA
- GetProcAddress

In order to call these API's on runtime Amber uses a publicly known EAT parsing technique that is used by Stephen Fewer's Reflective DLL injection [4] method, technique simply locates the InMemoryOrderModuleList structure with navigating through Process Environment Block (PEB) inside memory. After locating the structure it is possible to reach export tables of all loaded DLLs with reading each _LDR_DATA_TABLE_ENTRY structure pointed by the InMemoryOrderModuleList. After reaching the export table of a loaded DLL it compares the previously calculated ROR (rotate right) 13 hash of each exported function name until a match occurs. Amber's packing method also provides several alternative windows API usage methods, one of them

is using fixed API addresses, this is the best option if the user is familiar on the remote process that will host the Amber payload. Using fixed API addresses will directly bypass the latest OS level exploit mitigations that are inspecting export address table calls also removing API address finding code will reduce the overall payload size. Another alternative techniques can be used for locating the addresses of required functions such as IAT parsing technique used by Josh Pitts in “Teaching Old Shellcode New Tricks”[7] presentation. Current POC packer versions only supports Fixed API addresses and EAT parsing techniques but IAT parsing will be added on next versions.

4. Generating the Payload

For generating the actual Amber payload first packer creates a memory mapping image of the malware , generated memory mapping file contains all sections, optional PE header and null byte padding for unallocated memory space between sections.

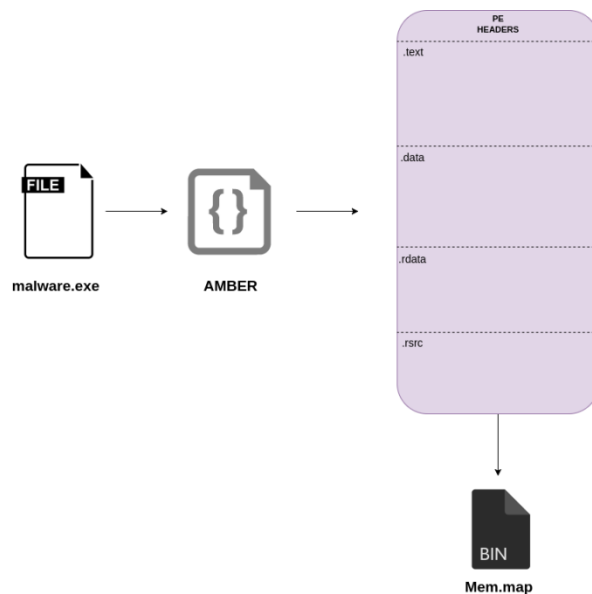


Figure 1, extracting the file mapping

After obtaining the mapping of the malware as shown inside figure 1, packer checks the ASLR compatibility of the supplied EXE, if the EXE is ASLR compatible packer adds the related Amber stub if not it uses the stub for EXE files that has fixed image base. From this point Amber payload is complete. Below image describes the Amber payload inside the target process,

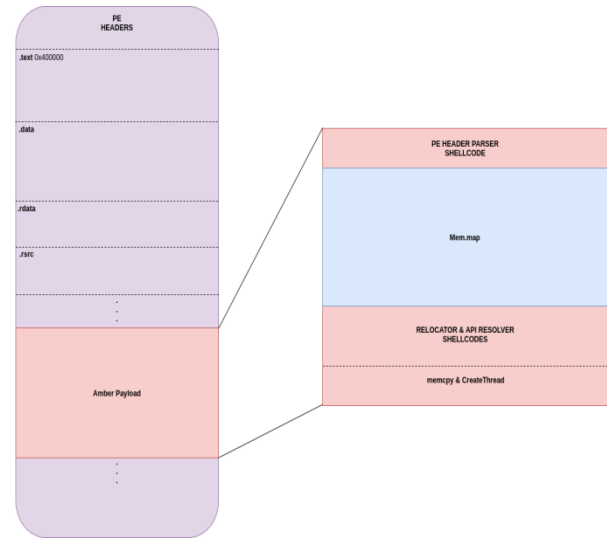


Figure 2, Amber payload inside the target process

5. ASLR Stub Execution

Execution of ASLR supported stub consists of 5 phases,

1. Base Allocation
2. Resolving API Functions
3. Base Relocation
4. Placement Of File Mapping
5. Execution

At the base allocation phase stub allocates a read/write/execute privileged memory space at the size of mapped image of malware with calling the VirtualAlloc windows API function,

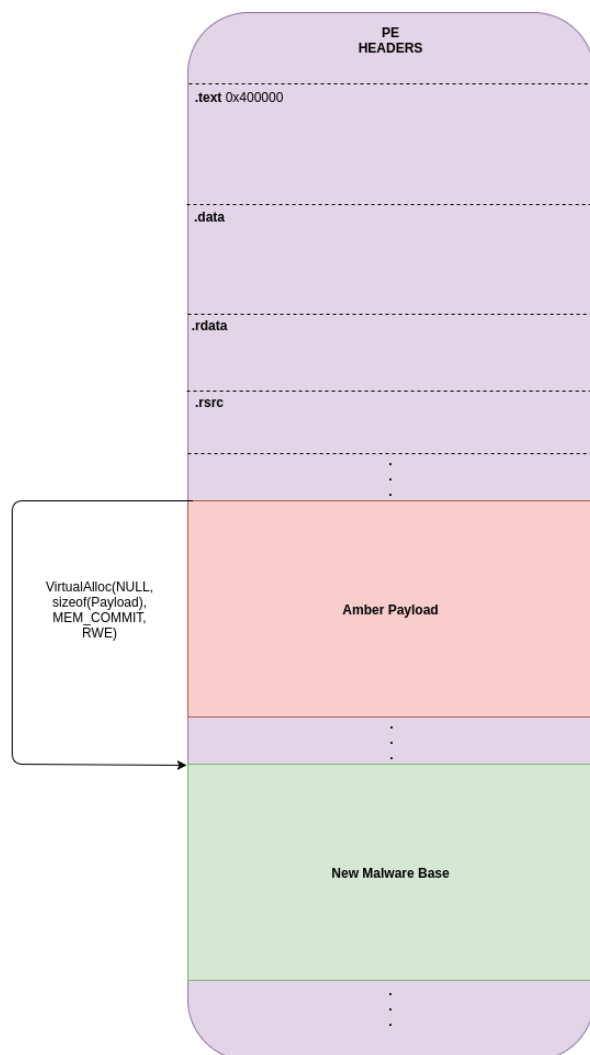


Figure 3, VirtualAlloc call by Amber stub

This memory space will be the new base of malware after the relocation process. In the second phase Amber stub will resolve the addresses of functions that is imported by the malware and write the addresses to the import address table of the mapped image of malware.

Address resolution phase is very similar to the approach used by the PE loader of Windows, Amber stub will parse the import table entries of the mapped malware image and load each DLL used by the malware with calling the LoadLibraryA windows API function, each `_IMAGE_IMPORT_DESCRIPTOR` entry inside import table contains pointer to the names of loaded DLL's as string, stub will take advantage of existing strings and pass them as parameters to the LoadLibraryA function,

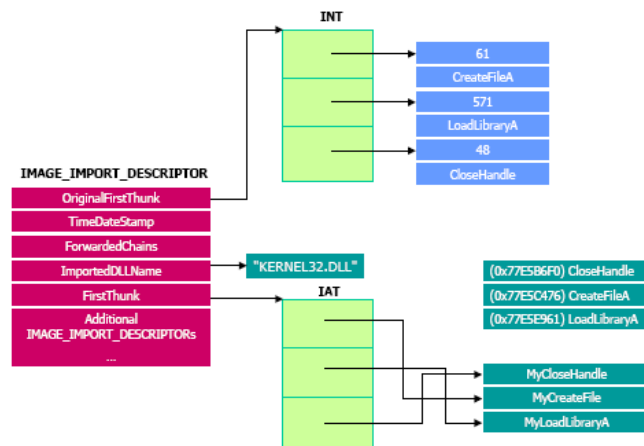


Figure 4, `_IMAGE_IMPORT_DESCRIPTOR`

after loading the required DLL Amber stub saves the DLL handle and starts finding the addresses of imported functions from the loaded DLL with the help of GetProcAddress windows API function, `_IMAGE_IMPORT_DESCRIPTOR` structure also contains a pointer to a structure called import names table, this structure contains the names of the imported functions in the same order with import address table (IAT), before calling the GetProcAddress function Amber stub passes the saved handle of the previously loaded DLL and the name of the imported function from import name table structure. Each returned function address is written to the malwares import address table (IAT) with 4 padding byte between them. This process continuous until the end of the import table, after loading all required DLL's and resolving all the imported function addresses second phase is complete.

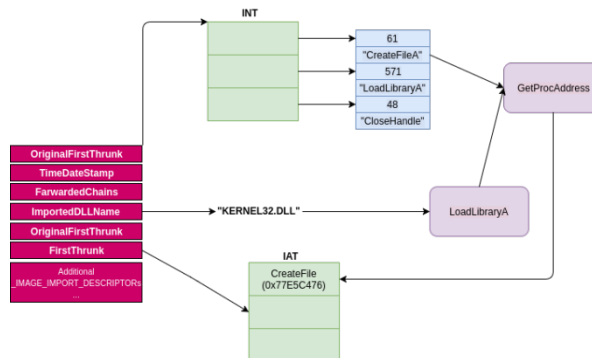


Figure 5, Resolving API addresses with EAT parsing

At the third phase Amber stub will start the relocation process with adjusting the addresses according to the address returned by the VirtualAlloc call, this is almost the same approach used by the PE loader of the

windows itself, stub first calculates the delta value with the address returned by the VirtualAlloc call and the preferred base address of the malware, delta value is added to the every entry inside the relocation table. In fourth phase Amber stub will place the file mapping to the previously allocated space, moving the mapped image is done with a simple assembly loop that does byte by byte move operation.

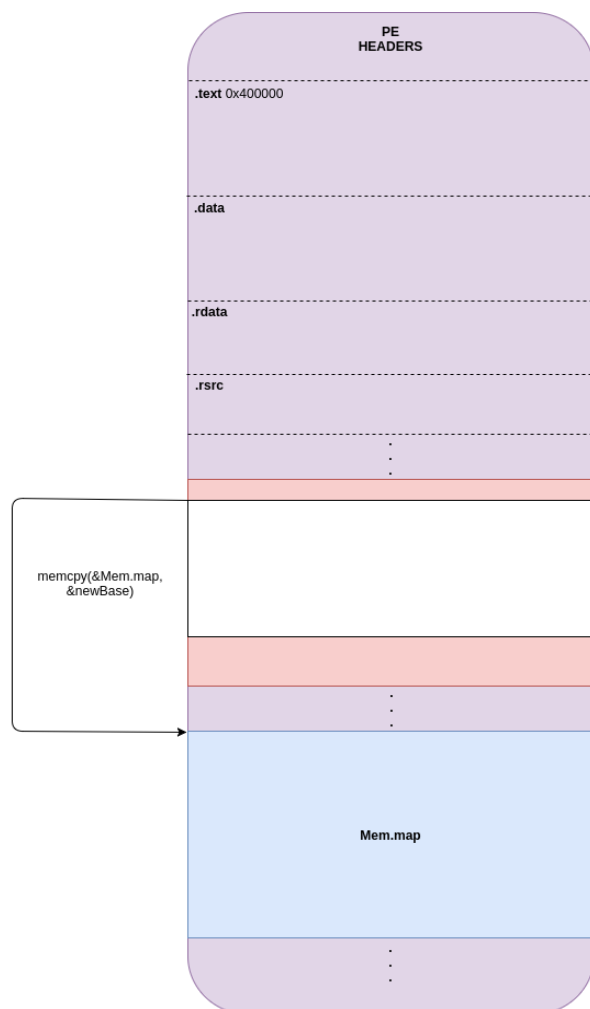


Figure 6, Replacement of ASLR supported file mapping.

At the final phase Amber stub will create a new thread starting from the entry point of the malware with calling the CreateThread API function. The reason of creating a new thread is to create a new growable stack for the malware and additionally executing the malware inside a new thread will allow the target process to continue from its previous state. After creating the malware thread stub will restore the execution with returning to the first caller or stub will jump inside a infinite loop that will stall the current thread while the malware thread successfully runs.

6. Non-ASLR Stub Execution

Execution of Non-ASLR supported stub consists of 4 phases,

1. Base Allocation
2. Resolving API functions
3. Placement Of File Mapping
4. Execution

If the malware is stripped or has no relocation data inside there is no other way than placing it to its preferred base address. In such condition stub tries to change the memory access privileges of the target process with calling VirtualProtect windows API function starting from image base of the malware through the size of the mapped image. If this condition occurs preferred base address and target process sections may overlap and target process will not be able to continue after the execution of Amber payload.



Figure 7, VirtualProtect call by Amber stub.

Fixed Amber stub may not be able to change the access privileges of the specified memory region, this may have multiple reasons such as specified memory range is not inside the current process page boundaries (reason is most probably ASLR) or the specified

address is overlapping with the stack guard regions inside memory. This is the main limitation for Amber payloads, if the supplied malware don't have ASLR support (has no relocation data inside) and stub can't change the memory access privileges of the target process payload execution is not possible. In some situations stub successfully changes the memory region privileges but process crashes immediately, this is caused by the multiple threads running inside the overwritten sections. If the target process owns multiple threads at the time of fixed stub execution it may crash because of the changing memory privileges or overwriting to a running section. However these limitations doesn't matter if it's not using the multi stage infection payload with fixed stub, current POC packer can adjust the image base of generated EXE file and the location of Amber payload accordingly. If the allocation attempt ends up successful first phase is complete. Second phase is identical with the approach used by the ASLR supported stub. After finishing the resolution of the API addresses, same assembly loop used for placing the completed file mapping to the previously amended memory region.

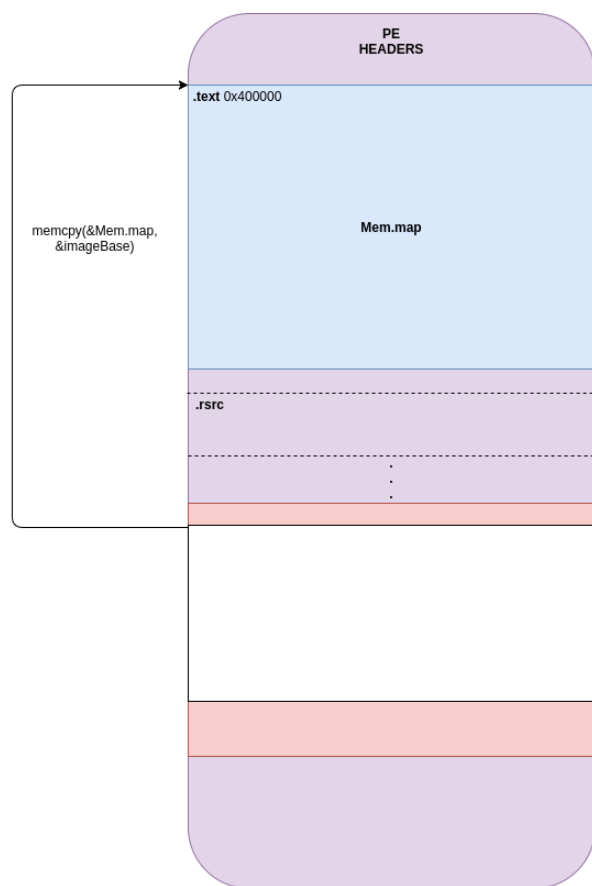


Figure 8, Replacement of the Non-ASLR memory mapping.

At the final phase stub jumps to the entry point of the malware and starts the execution without creating a new thread. Unfortunately, usage of Non-ASLR Amber stub does not allow the target process to continue with its previous state.

7. Multi Stage Applications

Security measures that will be taken by operating systems in the near future will shrink the attack surface even more for malwares. Microsoft has announced Windows 10 S on May 2 2017[8], this operating system is basically a configured version of Windows 10 for more security, one of the main precautions taken by this new operating system is doesn't allow to install applications other than those from Windows Store. This kind of white listing approach adopted by the operating systems will have a huge impact on malwares that is infecting systems via executable files. In such scenario usage of multi stage in-memory execution payloads becomes one of the most effective attack vectors. Because of the position independent nature of the Amber stubs it allows multi stage attack models, current POC packer is able to generate a stage payload from a complex compiled PE file that can be loaded and executed directly from memory like a regular shellcode injection attack. In such overly restrictive systems multi stage compatibility of Amber allows exploitation of common memory based software vulnerabilities such as stack and heap based buffer overflows.

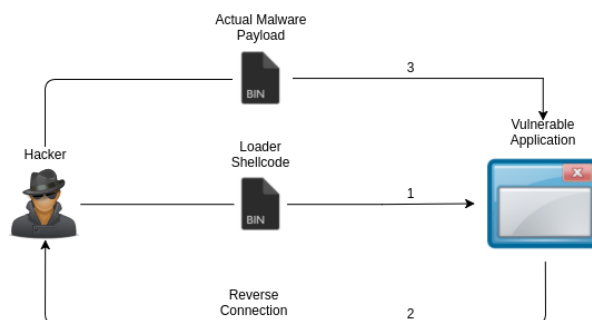


Figure 9, Multi stage payload attack model.

However due to the limitations of the fixed Amber stub it is suggested to use ASLR supported EXE files while performing multi stage infection attacks. Stage payloads generated by the POC packer are compatible with the small loader shellcodes and payloads generated from Metasploit Framework [9], this also means Amber payloads can be used with all the exploits inside the Metasploit Framework [9] that is using the multi stage meterpreter shellcodes.

8. Future Work

This paper introduces a new generation malware packing methodology for PE files but does not support .NET executables, future work may include the support for 64 bit PE files and .NET executables. Also in terms of stealthiness of this method there can be more advancement. Allocation of memory regions for entire mapped image done with read/write/execute privileges, after placing the mapped image changing the memory region privileges according to the mapped image sections may decrease the detection rate. Also wiping the PE header after the address resolution phase can make detection harder for memory scanners. The developments of Amber POC packer will continue as a open source project.

9. References

- [1] Ramilli, Marco, and Matt Bishop. "Multi-stage delivery of malware." Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. IEEE, 2010.
- [2] Gavriluț, Dragoș, et al. "Malware detection using machine learning." Computer Science and Information Technology, 2009. IMCSIT'09. International Multiconference on. IEEE, 2009.
- [3] Rieck, Konrad, et al. "Automatic analysis of malware behavior using machine learning." Journal of Computer Security 19.4 (2011): 639-668.
- [4] Fewer, Stephen. "Reflective DLL injection." Harmony Security, Version 1 (2008).
- [5] Leitch, John. "Process hollowing." (2013).
- [6] Ammann, Christian. "Hyperion: Implementation of a PE-Crypter." (2012).
- [7] Pitts, Josh. "Teaching Old Shellcode New Tricks" https://recon.cx/2017/brussels/resources/slides/RECON-BRX-2017_Teaching_Old_Shellcode_New_Tricks.pdf (2017)
- [8] <https://news.microsoft.com/europe/2017/05/02/microsoft-empowers-students-and-teachers-with-windows-10-s-affordable-pcs-new-surface-laptop-and-more/>
- [9] Rapid7 Inc, Metasploit Framework <https://www.metasploit.com>
- [10] Desimone, Joe. "Hunting In Memory" <https://www.endgame.com/blog/technical-blog/hunting-memory> (2017)
- [11] Lyda, Robert, and James Hamrock. "Using entropy analysis to find encrypted and packed malware." IEEE Security & Privacy 5.2 (2007).
- [12] Nasi, Emeric. "PE Injection Explained Advanced memory code injection technique" Creative Commons Attribution-NonCommercial-NoDerivs 3.0 License (2014)
- [13] Pietrek, Matt. "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format" <https://msdn.microsoft.com/en-us/library/ms809762.aspx> (1994)