Understanding BumbleBee: The malicious behavior

vmray.com/cyber-security-blog/understanding-bumblebee-the-malicious-behavior/

Understanding BumbleBee:

The malicious behavior of BumbleBee

In this article, we're exploring the malicious behavior of BumbleBee, and diving into its evasion techniques.

BumbleBee Blog Series – 2

18 August 2023

DOWNLOAD THE E-BOOK

Table of Contents

Introduction

Having meticulously dissected the intricate delivery methods employed by BumbleBee in our previous blog post, we embarked on a journey through the multifaceted and complex delivery chains that enable its stealthy penetration. From the covert utilization of seemingly innocuous files to ingenious tactics that evade detection, we navigated through the diverse arsenal BumbleBee wields to gain access.

Now, our exploration of this malicious loader takes us deeper into its heart — its behavior within the targeted environment. As we journey through its maneuvers, we seek to unravel the layers of complexity that lie beneath its surface. With a focus on unveiling the true essence of BumbleBee's actions post-infiltration, we uncover the evolution and adaptation that contribute to its resilience.

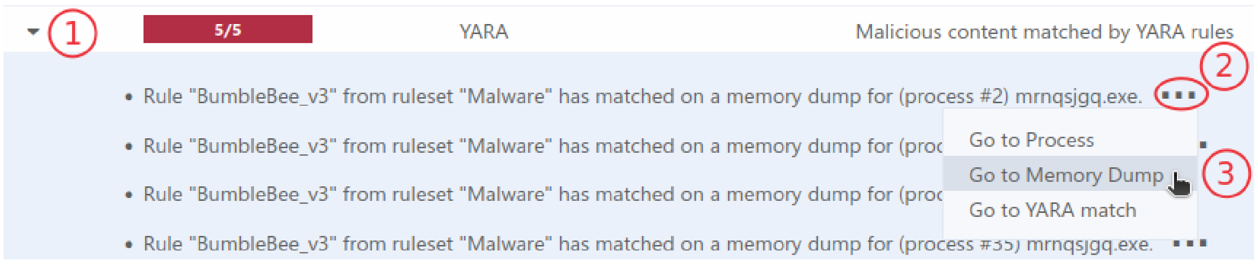
Unpacking Executables

Malicious applications need to stay undetected as long as possible so victims can be infected without being caught by malware researchers, antivirus scanners or dynamic behavior-based analysis engines. To protect against the first two, malware is often packed, encrypted or obfuscated. For one, signature-based antivirus engines might fail to trigger even on known malware families when the sample is manipulated to a certain degree. Obfuscating the sample also increases the workload of threat researchers analyzing the malware manually, thus this either slows them down or potentially prevents researchers from dissecting the malware altogether.

This is why one major advantage of dynamic behavior-based analysis is that the malware is executed in a simulated environment which allows us to dynamically observe it's behavior at runtime. As packed and encrypted samples will at some point execute the original, unprotected executable, our system is able to automatically save a memory dump of the unpacked binary. This is such a powerful feature that threat researchers rarely have to manually unpack a malware to analyze it's behavior. Instead, they can let VMRay Platform perform the unpacking.

To demonstrate this feature, we let VMRay Platform analyze a BumbleBee sample which is often protected by such measures. Once the dynamic analysis is done, we try to identify the memory dump containing the unpacked binary.

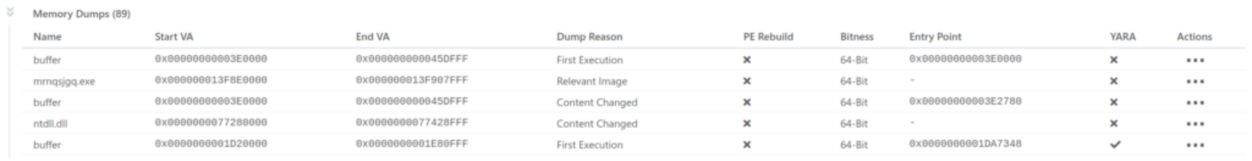
As we support multiple trigger reasons to dump a memory region (for example, when the memory content changes, when the first network connection is established, when the application is terminated, etc.) there could be multiple memory dumps to select from, many of which are not fully unpacked yet. The easiest method to find a good memory dump is to select one with a YARA match, which implies that there must be enough strings and code snippets to classify it as belonging to a certain malware family (see Figure 1 and Figure 2)



5/5 YARA Malicious content matched by YARA rules

- Rule "BumbleBee_v3" from ruleset "Malware" has matched on a memory dump for (process #2) mrnqsjgq.exe.
- Rule "BumbleBee_v3" from ruleset "Malware" has matched on a memory dump for (proc
- Rule "BumbleBee_v3" from ruleset "Malware" has matched on a memory dump for (proc
- Rule "BumbleBee_v3" from ruleset "Malware" has matched on a memory dump for (process #35) mrnqsjgq.exe.

Figure 1: Following a YARA match is a good way to find a memory dump for the unpacked binary.

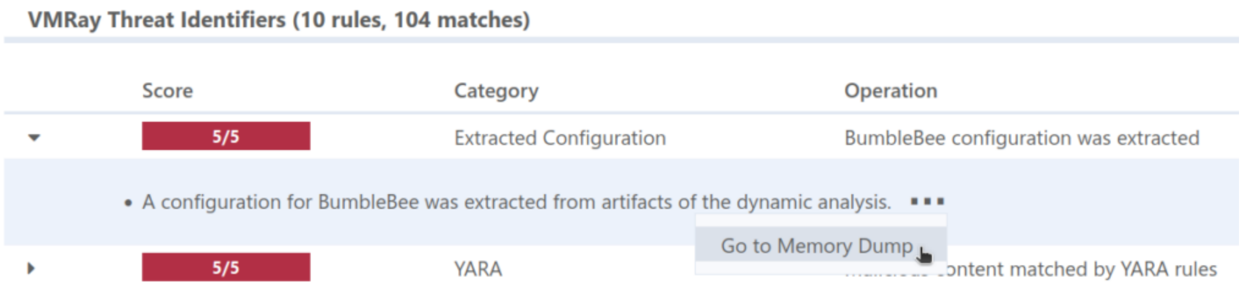


Memory Dumps (89)

Name	Start VA	End VA	Dump Reason	PE Rebuild	Bitness	Entry Point	YARA	Actions
buffer	0x00000000003E0000	0x000000000045DFFF	First Execution	✗	64-Bit	0x00000000003E0000	✗	•••
mrnqsjgq.exe	0x000000013F8E0000	0x000000013F907FFF	Relevant Image	✗	64-Bit	-	✗	•••
buffer	0x00000000003E0000	0x000000000045DFFF	Content Changed	✗	64-Bit	0x00000000003E2780	✗	•••
ntdll.dll	0x0000000077280000	0x0000000077428FFF	Content Changed	✗	64-Bit	-	✗	•••
buffer	0x0000000001D20000	0x0000000001E80FFF	First Execution	✗	64-Bit	0x0000000001DA7348	✓	•••

Figure 2: Partial list of memory dumps for BumbleBee. Note that the last memory dump in the screenshot reports a YARA match.

An even better approach is to find the memory dump for which the config extractor was triggered (see Figure 3). A successful config extraction is a stronger indicator for an unpacked binary.



VMRay Threat Identifiers (10 rules, 104 matches)

Score	Category	Operation
5/5	Extracted Configuration	BumbleBee configuration was extracted
5/5	YARA	…ontent matched by YARA rules

- A configuration for BumbleBee was extracted from artifacts of the dynamic analysis.

Figure 3: Another approach to download the unpacked binary is to find the memory dump that was used to extract the configuration.

This memory dump can now be analyzed by popular tools such as IDA Pro. It might be necessary to run it through pe_unmapper first. To benefit from the information extracted during the analysis, we also have an IDA plugin that enriches the disassembly and decompilation with runtime information, such as function parameters observed during execution (see here for more information).

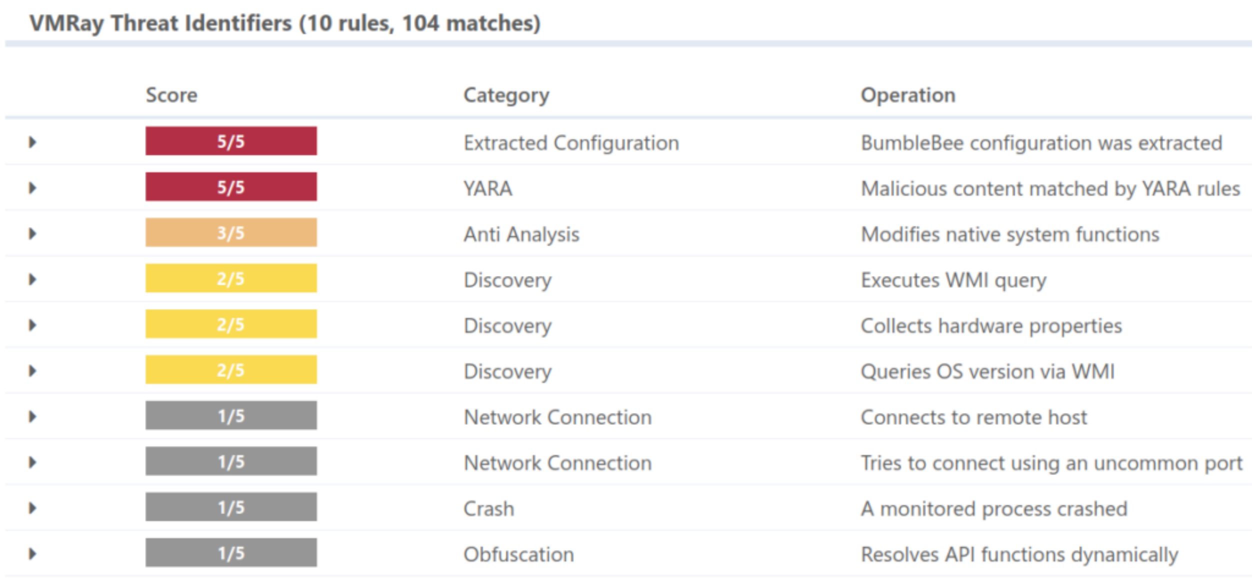
While this approach of unpacking malware has some limitations, e.g., packers employing virtual machine-based protection such as VMProtect may never “unpack” the original executable as some of the code was replaced by virtual operations by the protector, and thus this will prevent a memory dump containing the original instructions, VMRay Platform will still be able to dynamically observe the malware unhindered, as the original behavior will remain the same.

Malicious behavior

The main goal of a malware loader is to infect the system with additional malware, but most loaders also try to ensure that the malicious executables are not downloaded or executed while being observed either by threat researchers during manual analysis or by a behavior-based dynamic analysis system (see Figure 4).

For this purpose, BumbleBee includes more than 50 evasion techniques, ranging from tracking mouse movements to checking for artifacts of common virtualization tools such as VirtualBox. Our analysis shows that these techniques were mostly copied from Al-Khaser, an open source collection of known evasion techniques.

We provide a list of all functions imported from Al-Khaser we have identified so far in the Appendix.



VMRay Threat Identifiers (10 rules, 104 matches)

Score	Category	Operation
5/5	Extracted Configuration	BumbleBee configuration was extracted
5/5	YARA	Malicious content matched by YARA rules
3/5	Anti Analysis	Modifies native system functions
2/5	Discovery	Executes WMI query
2/5	Discovery	Collects hardware properties
2/5	Discovery	Queries OS version via WMI
1/5	Network Connection	Connects to remote host
1/5	Network Connection	Tries to connect using an uncommon port
1/5	Crash	A monitored process crashed
1/5	Obfuscation	Resolves API functions dynamically

Figure 4: VMRay's VTI's triggered for a BumbleBee sample.

Network

To download additional malware, BumbleBee makes contact with a remote C2 server. The oldest BumbleBee sample we could find uses HTTP as the network protocol while more recent versions have adopted WebSockets. Once the connection is established, the infected system waits for one of seven possible commands.

Command name: shi
Function: Shellcode injection

Command name: dij
Function: DLL injection

Command name: dex
Function: Download & Execute

Command name: sdl
Function: Uninstall

Command name: ins
Function: Persistence

Command name: gdt
Function: Execute shell commands

Command name: plg
Function: Load plugins

We have been able to observe that the last two commands were added somewhere around April and August 2022. Interestingly, the “plg” instruction is just a reference to the “dij” command, a hint that this functionality might be updated in the future.

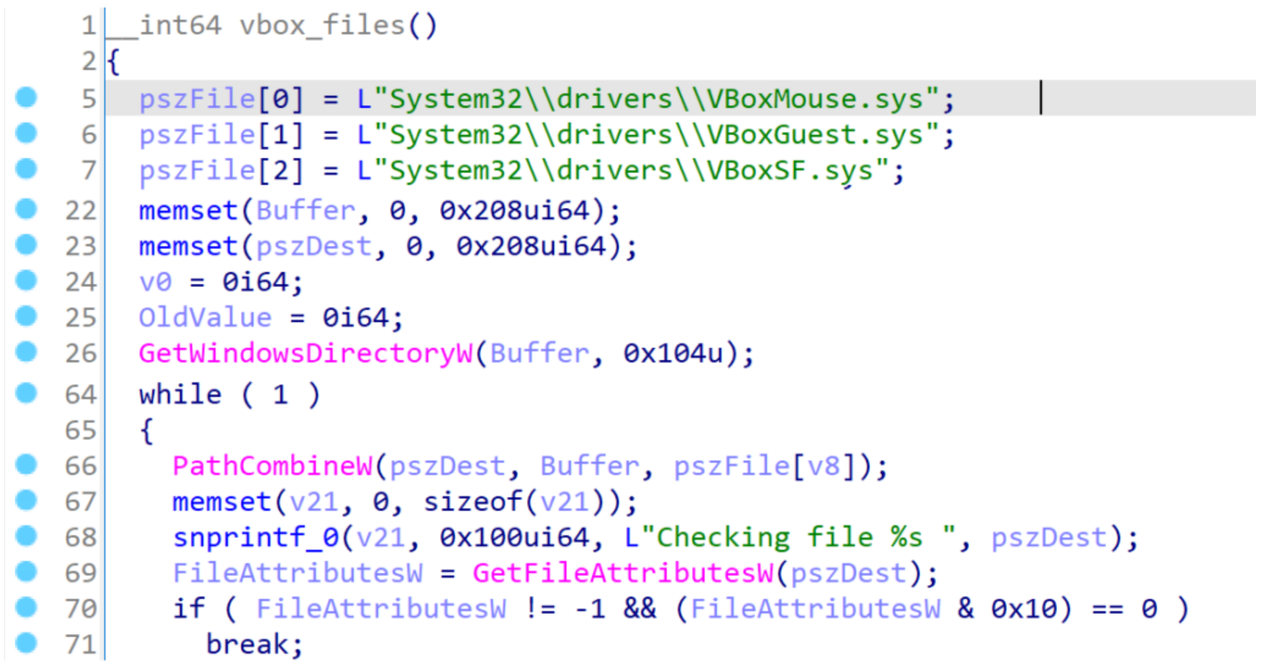
Evasion techniques

In it’s most recent version, BumbleBee employs a total of over 50 evasion techniques, a majority of which are imported from the Al-Khaser open-source project.

They can be summarized in the following categories:

Detecting virtualization software and related tools

Most evasion techniques we have discovered are related to virtualization tools often used either during manual analysis by threat researchers (VirtualBox, VMWare, Xen) or potentially used during dynamic, behavior-based analysis, such as QEMU and KVM. One of the techniques, for example, checks if certain files such as “System32\drivers\VBoxMouse.sys” exist on the system to detect the presence of VirtualBox based on its drivers (see Figure 13).



```
__int64 vbox_files()
{
  pszFile[0] = L"System32\\drivers\\VBoxMouse.sys";
  pszFile[1] = L"System32\\drivers\\VBoxGuest.sys";
  pszFile[2] = L"System32\\drivers\\VBoxSF.sys";
  memset(Buffer, 0, 0x208ui64);
  memset(pszDest, 0, 0x208ui64);
  v0 = 0i64;
  OldValue = 0i64;
  GetWindowsDirectoryW(Buffer, 0x104u);
  while ( 1 )
  {
    PathCombineW(pszDest, Buffer, pszFile[v8]);
    memset(v21, 0, sizeof(v21));
    snprintf_0(v21, 0x100ui64, L"Checking file %s ", pszDest);
    FileAttributesW = GetFileAttributesW(pszDest);
    if ( FileAttributesW != -1 && (FileAttributesW & 0x10) == 0 )
      break;
```

Figure 5: Function from Al-Khaser used by BumbleBee that demonstrates one of the detection methods for VirtualBox (cropped for readability).

There are also functions related to detecting Wine, a compatibility layer on Linux to execute Windows applications.

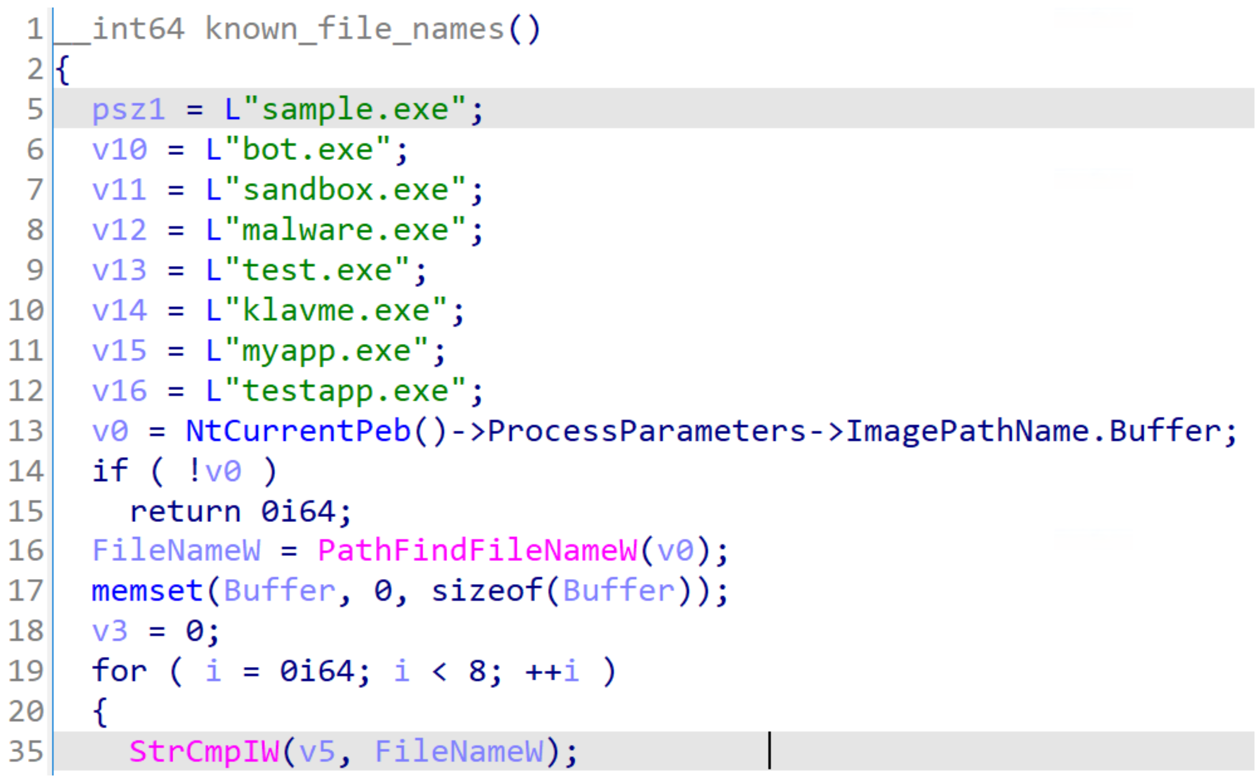
Detecting reverse engineering tools

Particularly in older BumbleBee versions, we found checks for OllyDbg, a popular tool used by threat researchers to analyze malware and other Windows executables.

This check was removed in more recent versions, but newer samples still check for Process Explorer, another useful utility sometimes used by threat researchers to explore processes running on the system.

Detecting sandboxing artifacts

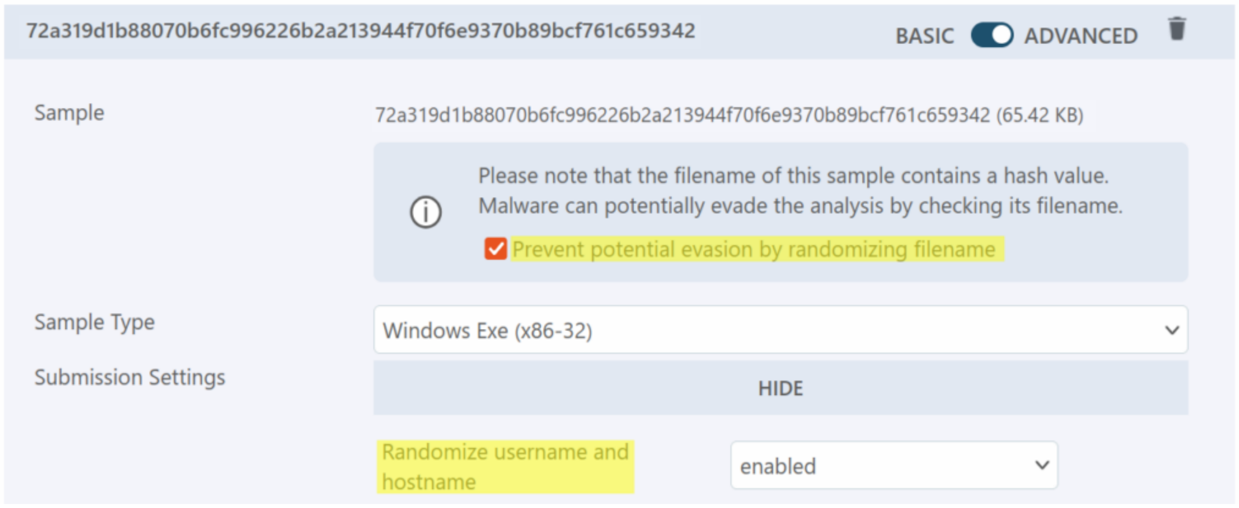
To detect some sandboxing solutions, BumbleBee checks if the currently logged-in username or it’s own filename is a name primarily used by sandbox technologies, for example “sample.exe” (see Figure 6).



```
__int64 known_file_names()
{
  psz1 = L"sample.exe";
  v10 = L"bot.exe";
  v11 = L"sandbox.exe";
  v12 = L"malware.exe";
  v13 = L"test.exe";
  v14 = L"klavme.exe";
  v15 = L"myapp.exe";
  v16 = L"testapp.exe";
  v0 = NtCurrentPeb()->ProcessParameters->ImagePathName.Buffer;
  if ( !v0 )
    return 0i64;
  FileNameW = PathFindFileNameW(v0);
  memset(Buffer, 0, sizeof(Buffer));
  v3 = 0;
  for ( i = 0i64; i < 8; ++i )
  {
    StrCmpIW(v5, FileNameW);
```

Figure 6: Al-Khaser function used by BumbleBee to check if the filename matches known filenames used by sandboxing solutions (cropped for readability).

This is one of the reasons VMRay Platform can randomize the username, the hostname and the filename (see Figure7). Additionally, one common evasion technique is checking if the filename is the hash of the file itself, which is often the case in dynamic analysis engines. This can also be prevented by randomizing the filename.



72a319d1b88070b6fc996226b2a213944f70f6e9370b89bcf761c659342 BASIC ADVANCED

Sample: 72a319d1b88070b6fc996226b2a213944f70f6e9370b89bcf761c659342 (65.42 KB)

Please note that the filename of this sample contains a hash value. Malware can potentially evade the analysis by checking its filename.

[✓] Prevent potential evasion by randomizing filename

Sample Type: Windows Exe (x86-32)

Submission Settings: HIDE

Randomize username and hostname: enabled

Figure 7: VMRay Platform allows the randomization of the username, hostname and the sample name to prevent some evasion techniques.

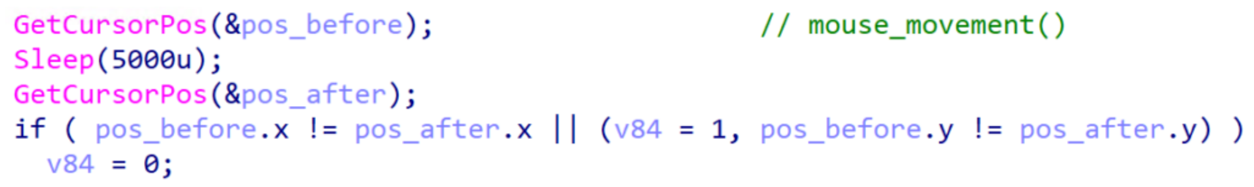
Detection via hardware configuration

In addition to searching for evidence that the sample is running inside of a virtualized or simulated environment, BumbleBee also checks the hardware of the system for specific configuration artifacts mostly associated with sandboxing.

One method checks if the RAM and the disk size are above a certain threshold, another checks if information about the CPU fan can be retrieved.

Detection via user behavior

Not all evasion techniques rely on detecting the environment itself. Some smart techniques revolve around detecting the presence of a user, which is usually not the case for automated sandbox solutions. In particular, BumbleBee checks if the mouse pointer moves (see Figure 8).



```
GetCursorPos(&pos_before);                         // mouse_movement()
Sleep(5000u);
GetCursorPos(&pos_after);
if ( pos_before.x != pos_after.x || (v84 = 1, pos_before.y != pos_after.y) )
  v84 = 0;
```

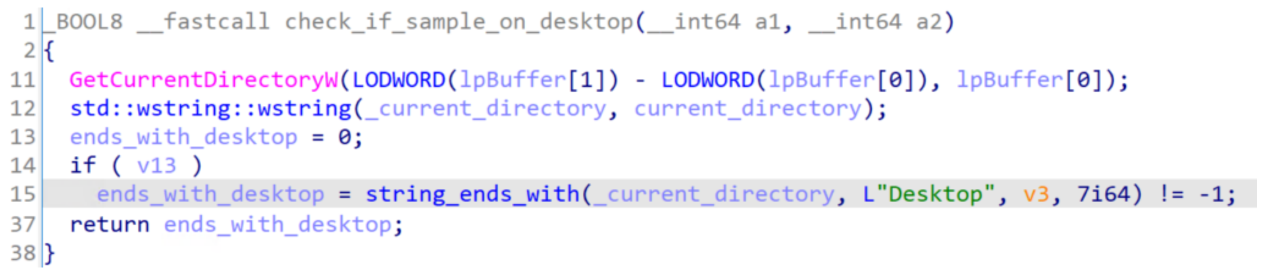
Figure 8: Al-Khaser function used by BumbleBee to detect mouse movement.

VMRay employs a variety of techniques to simulate user behavior to avoid these evasion techniques from detecting the monitored environment. For example, the mouse is moved regularly and we even simulate interactions with the user interface. Additionally, our customers can also manually interact with the virtual environment during the analysis as if they were sitting in front of it – right from the web browser.

Custom evasion techniques

We have discovered one evasion technique seemingly not covered by Al-Khaser: BumbleBee checks if it was started from the desktop (see Figure 9).

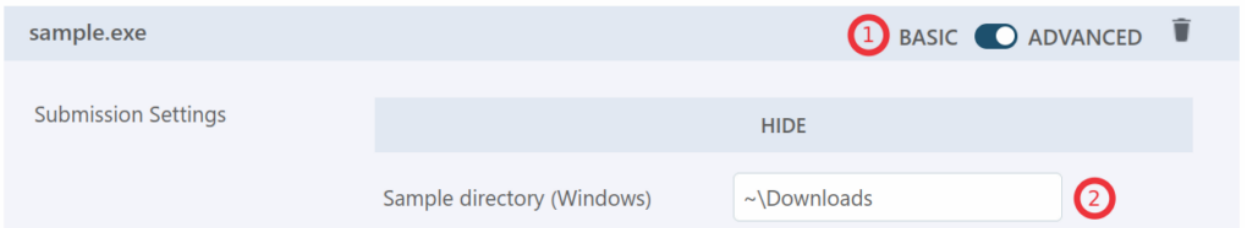
The reasoning here is that BumbleBee's delivery chain usually results in the sample being located in, for example, the temp directory or on a CD-ROM drive mounted from an ISO file, and not the desktop.



```
_BOOL8 __fastcall check_if_sample_on_desktop(__int64 a1, __int64 a2)
{
  GetCurrentDirectoryW(LODWORD(lpBuffer[1]) - LODWORD(lpBuffer[0]), lpBuffer[0]);
  std::wstring::wstring(_current_directory, current_directory);
  ends_with_desktop = 0;
  if ( v13 )
    ends_with_desktop = string_ends_with(_current_directory, L"Desktop", v3, 7i64) != -1;
  return ends_with_desktop;
}
```

Figure 9: BumbleBee checks if the sample was started from the Desktop, in which case it aborts the execution.

VMRay Platform allows customers to set the sample directory while uploading submissions (see Figure 10).


sample.exe

1 BASIC ADVANCED

Submission Settings

HIDE

Sample directory (Windows) ~\Downloads 2

Figure 10: VMRay Platform allows users to set the directory from which the sample is started from by opening the advanced settings.

Note that having multiple evasion techniques can actually backfire for the malware, as performing the evasion techniques themselves can be detected as malicious behavior. For example, our VMRay Platform is able to observe the checks for virtualized environments, which contributes to the final malicious verdict.

The evolution of BumbleBee

We have tracked BumbleBee since it first appeared in the wild and were able to create a rough timeline of when particular important changes were introduced.

In summary, the network communication was changed from HTTP to WebSockets, and two new C2 operations were introduced (see Figure 11) alongside nearly 20 additional evasion techniques. Notably, we have identified samples where all evasion techniques were removed for a period of time around November 2022, but they were reintroduced later on (see Figure 12).

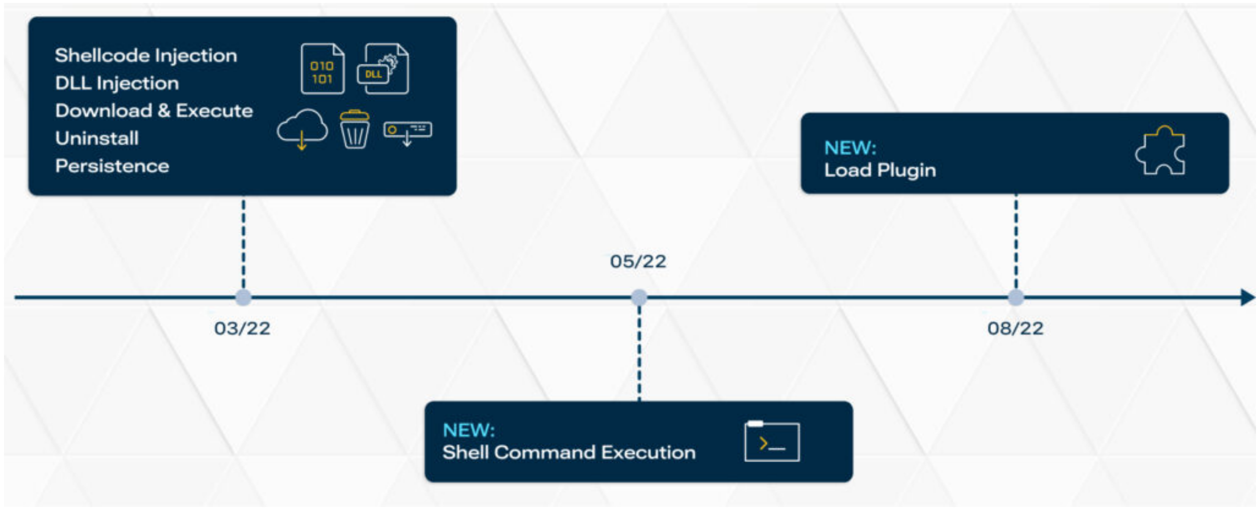


Figure 11: Rough timeline for the changes in the C2 functionality.

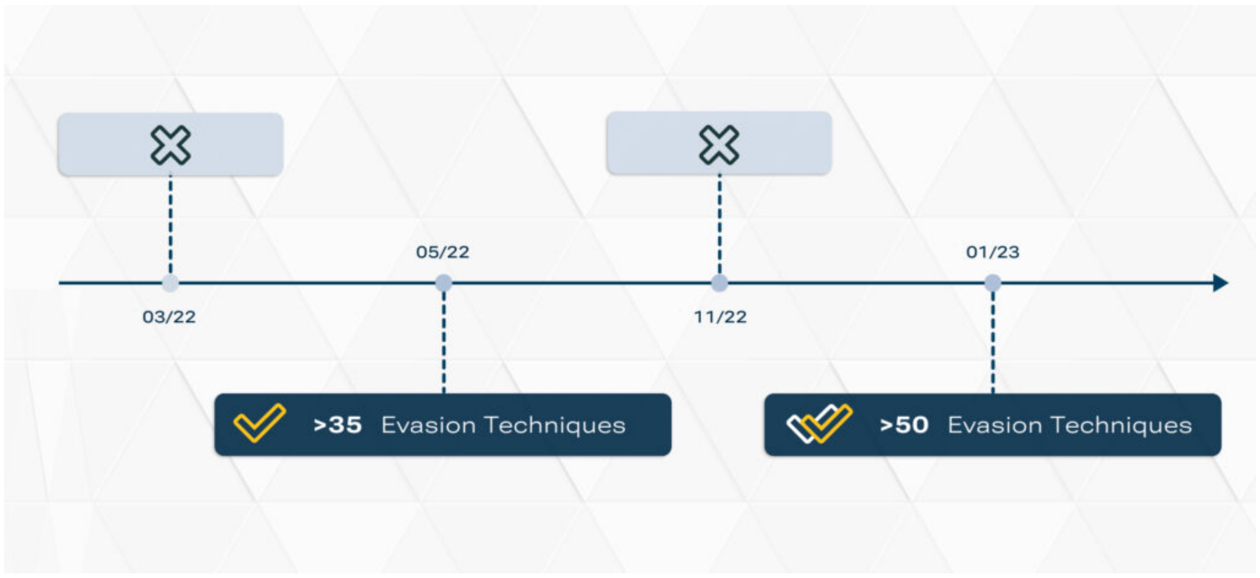


Figure 12: Rough timeline of changes for the evasion techniques

Conclusion

In the world of cybersecurity, staying informed is our strongest shield. Our exploration of BumbleBee's delivery methods and malicious behavior has unveiled a landscape of complexity, where seemingly innocuous file formats are manipulated for malicious intent. With over 50 evasion techniques in its arsenal, BumbleBee showcases the evolving art of deception employed by threat actors.

In our upcoming article, we'll delve deeper into BumbleBee's behavior, exploring its configurations and the clusters emerging from our analysis. Stay tuned for the insights that fortify our understanding of this intricate menace.

Appendix

Evasion Techniques imported from Al-Khaser

vbox_files
vbox_dir
vbox_mac_wmi
vbox_eventlogfile_wmi
vbox_firmware_ACPI
vbox_pnpentity_pcideviceid_wmi
vbox_pnpentity_pcideviceid_wmi_2
vbox_bus_wmi
vbox_baseboard_wmi
vbox_pnpentity_vboxname_wmi
vmware_files
vmware_firmware_ACPI

qemu_dir
qemu_firmware_ACPI
kvm_files
known_file_names
known_usernames
disk_size_wmi
model_computer_system_wmi
registry_services_disk_enum
get_services
check_mac_addr
GetProcessIdFromName
get_system_firmware
wine_reg_key
wine_exports
vbox_window_class
vbox_reg_key_value
vbox_reg_keys
vbox_devices
vbox_network_share
vbox_processes
vbox_window_class
vbox_devices
vbox_firmware_SMBIOS
vmware_reg_keys
vmware_mac
vmware_devices
vmware_firmware_SMBIOS
virtual_pc_process
virtual_pc_reg_keys
qemu_reg_key_value
qemu_processes
qemu_firmware_SMBIOS
kvm_reg_keys
kvm_dir
parallels_process
mouse_movement
cpu_fan_wmi
memory_space
xen_check_mac

Emre Güler
Threat Researcher

BumbleBee Series – 1:
The delivery chains

BumbleBee Series – 3:
The malware configuration and clusters