

# GitHub - cdong1012/ContiUnpacker: An automatic unpacker for a Conti sample

By cdong1012

Archived: 2026-04-05 18:53:55 UTC

An automatic unpacker for a Conti sample

## Context

- This was inspired by James Bennett's blog [post](#) on how to programmatically unpack malware.
- This unpacker unpacks this specific Conti ransomware I found on [MalwareBazaar](#).

## Requirement

- Python 3
- Speakeasy

## How it works

- The unpacker uses the [Speakeasy](#) Emulation Framework to run and unpack the sample.
- When I manually unpacked this, I noticed that the sample called **VirtualAlloc** to allocate memory, wrote the unpacked PE file to it, and called **VirtualProtect** on the **.text** region before executing it.
- From this, I halted the simulation at the first **VirtualProtect** call, dumped the PE file out, and mapped it accordingly to fix the IAT.

## Usage

### Running with Command Prompt

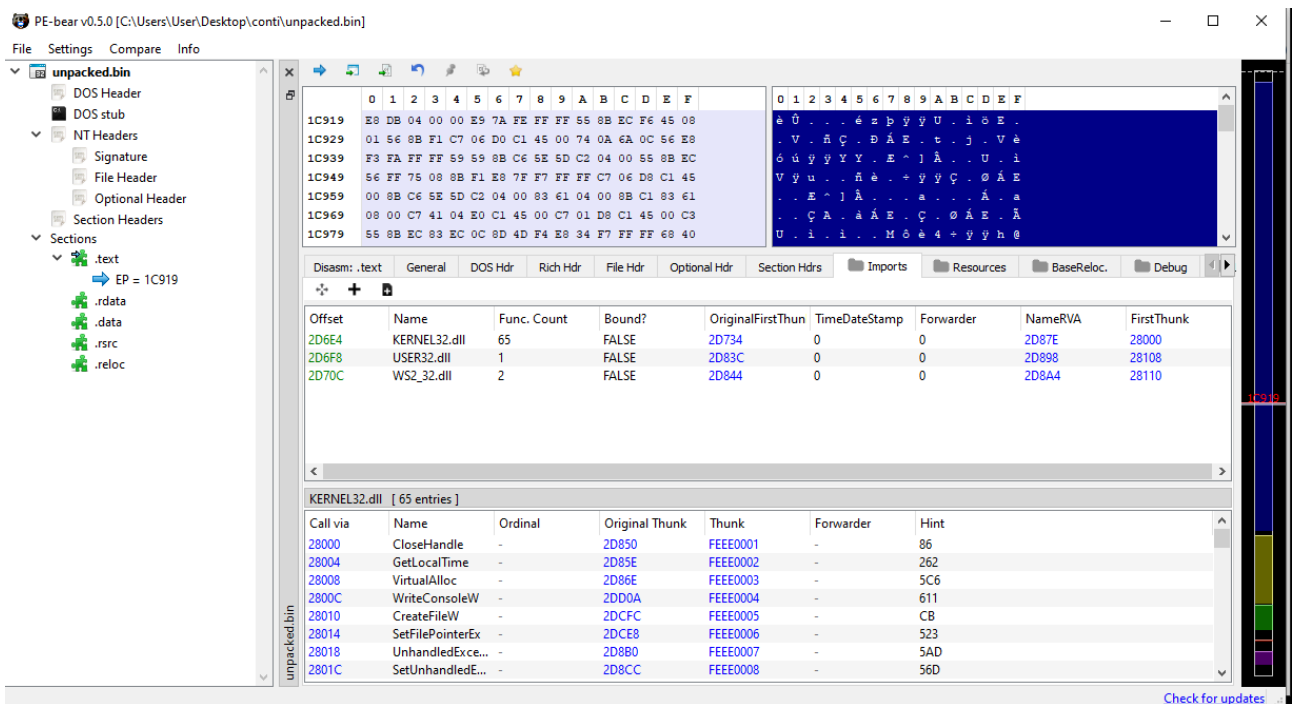
```
python ContiUnpacker.py -f conti.dll -o <output_file>
```

## Image

```

0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetCPIInfo")' -> 0xfefee0030
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetCommandLineA")' -> 0xfefee0031
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetCommandLineW")' -> 0xfefee0032
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "MultiByteToWideChar")' -> 0xfefee0033
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "WideCharToMultiByte")' -> 0xfefee0034
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetEnvironmentStringsW")' -> 0xfefee0035
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "FreeEnvironmentStringsW")' -> 0xfefee0036
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetFileType")' -> 0xfefee0037
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "LCMapStringW")' -> 0xfefee0038
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetProcessHeap")' -> 0xfefee0039
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "SetStdHandle")' -> 0xfefee003a
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetStringTypeW")' -> 0xfefee003b
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "HeapSize")' -> 0xfefee003c
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "HeapReAlloc")' -> 0xfefee003d
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "FlushFileBuffers")' -> 0xfefee003e
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetConsoleCP")' -> 0xfefee003f
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "GetConsoleMode")' -> 0xfefee0040
0x10001881: 'KERNEL32.GetProcAddress(0x77000000, "DecodePointer")' -> 0xfefee0041
0x100021ff: 'KERNEL32.IsBadReadPtr(0x4616f8, 0x14)' -> 0x0
0x1000184e: 'KERNEL32.LoadLibraryA("USER32.dll")' -> 0x77d10000
0x10001920: 'KERNEL32.VirtualQuery(0x50000, 0x1211ecc, 0x1c)' -> 0x1c
0x10001881: 'KERNEL32.GetProcAddress(0x77d10000, "wsprintfW")' -> 0xfefee0042
0x100021ff: 'KERNEL32.IsBadReadPtr(0x46170c, 0x14)' -> 0x0
0x1000184e: 'KERNEL32.LoadLibraryA("WS2_32.dll")' -> 0x78c00000
0x10001920: 'KERNEL32.VirtualQuery(0x50000, 0x1211ecc, 0x1c)' -> 0x1c
0x10001881: 'KERNEL32.GetProcAddress(0x78c00000, 0x6f)' -> 0xfefee0043
0x10001881: 'KERNEL32.GetProcAddress(0x78c00000, 0x9)' -> 0xfefee0044
0x100021ff: 'KERNEL32.IsBadReadPtr(0x461720, 0x14)' -> 0x0
[*] VirtualProtect CALLED!
[*] Dump Address: 0x434000
[*] Dump Address: 0x34000
[*] Found valid PE file
0x10001e57: 'KERNEL32.VirtualProtect(0x435000, 0x32200, 0x40, 0x1211ed8)' -> None

```



## Note

Please don't actually run this malware I included unless you know what you're doing. I'm not responsible if you end up encrypting your machine!

Also, I noticed that the function calls are a bit different on Speakeasy emulator compared to when running on x64dbg. During the VirtualProtect call, everything should technically be written into the allocated memory already, but that's not the case...

Apparently, only parts of the **.rdata** section is written, so the dumped executable won't be able to run.

I can't figure out why this is happening because Speakeasy is pretty weird, so this unpacker does not work 100%.

However, I'll still keep it here in case anyone wants to refer to this when writing their own unpacker using Speakeasy!

## **Acknowledgement**

James T. Bennett - <https://www.fireeye.com/blog/threat-research/2020/12/using-speakeasy-emulation-framework-programmatically-to-unpack-malware.html>

FireEye's Speakeasy Emulation Framework - <https://github.com/fireeye/speakeasy>.

---

Source: <https://github.com/cdong1012/ContiUnpacker>