



Cerbero Journal

"Vita sine proposito
vaga est." – Lucius
Annaeus Seneca

ISSUE NR. 1

CERBERO LABS

JUNE 20, 2022

Welcome to the first issue of Cerbero Journal!

We do understand that our customers cannot follow every blog and social media post we publish, so we decided to create a journal to offer a general overview of our latest endeavors as a company.

Plus a journal offers the added nostalgia bonus if, like us, you are old enough to remember the golden era of e-zines like 'NT Insider'.

Among many other things, in this issue we discuss the introduction of [Cerbero Store](#), which we consider to be a game-changer, and we write about a selection of some of the packages we recently released on it.

FULL METAL ENGINE



Since March of this year Cerbero Engine comes in two different editions: Classic and Metal. The Metal edition is designed to be run in cloud and server environments which usually lack a graphical stack.

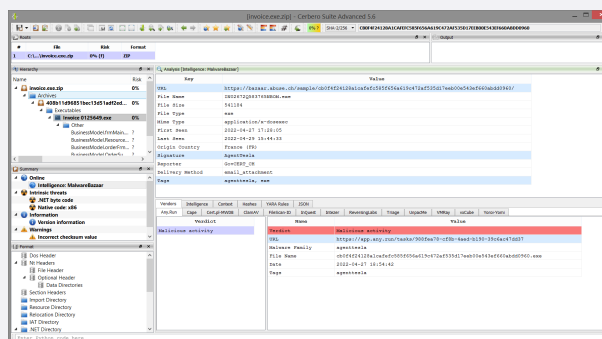
Plugins which import graphical functions are compatible with the Metal edition: all UI functions are available, although they are provided only as stubs. A few graphical methods like `msgBox` fall back to console I/O.

Providing two editions of Cerbero Engine allows us to offer the perfect fit for organizations which need a powerful and flexible back-end for their online services.

If you're not yet familiar with Cerbero Engine, you can read the [engine intermezzo](#) or visit our [web-page](#).

MALWAREBAZAAR INTELLIGENCE

We recently released the 'MalwareBazaar Intelligence' package that enables immediate access to intelligence from [MalwareBazaar](#) directly from the file report.



[continued on page 3]

CROSS-PLATFORM MICROSOFT AUTHENTICODE

In April we released a package for commercial licenses of Cerbero Suite Advanced and Cerbero Engine that enables verification of Microsoft Authenticode signatures on platforms other than Windows such as Linux and macOS.

Our Authenticode support includes full-chain certificate and time-stamp verification.

[continued on page 4]

UPX UNPACKER

We recently released the 'UPX Unpacker' package available for all of our products.

By installing the package, binaries compressed with UPX are automatically identified and unpacked as child objects.

[continued on page 5]

HAVING FUN DOING CTFs

While our 'String Decrypter' package isn't just for CTFs, it can be very useful in that context.

How often do we find ourselves in the situation of having to brute-force the decryption of either a string or a byte-array?

[continued on page 10]

CORE SDK DOCUMENTATION

For the past year we've been expanding our SDK documentation and we have now completely documented our core modules.

[continued on page 3]

CERBERO STORE

One of the major features introduced for this series of Cerbero Suite and Cerbero Engine is Cerbero Store: a simple way to install and update packages.

Chief among the reasons we had to create Cerbero Store was the necessity to release faster updates. It didn't make sense to update the whole application just to update a specific part.

Additionally, our software runs on multiple platforms. Which means that each update requires us to create multiple software packages. This problem is solved by Cerbero Store, since all platforms share the same package code.

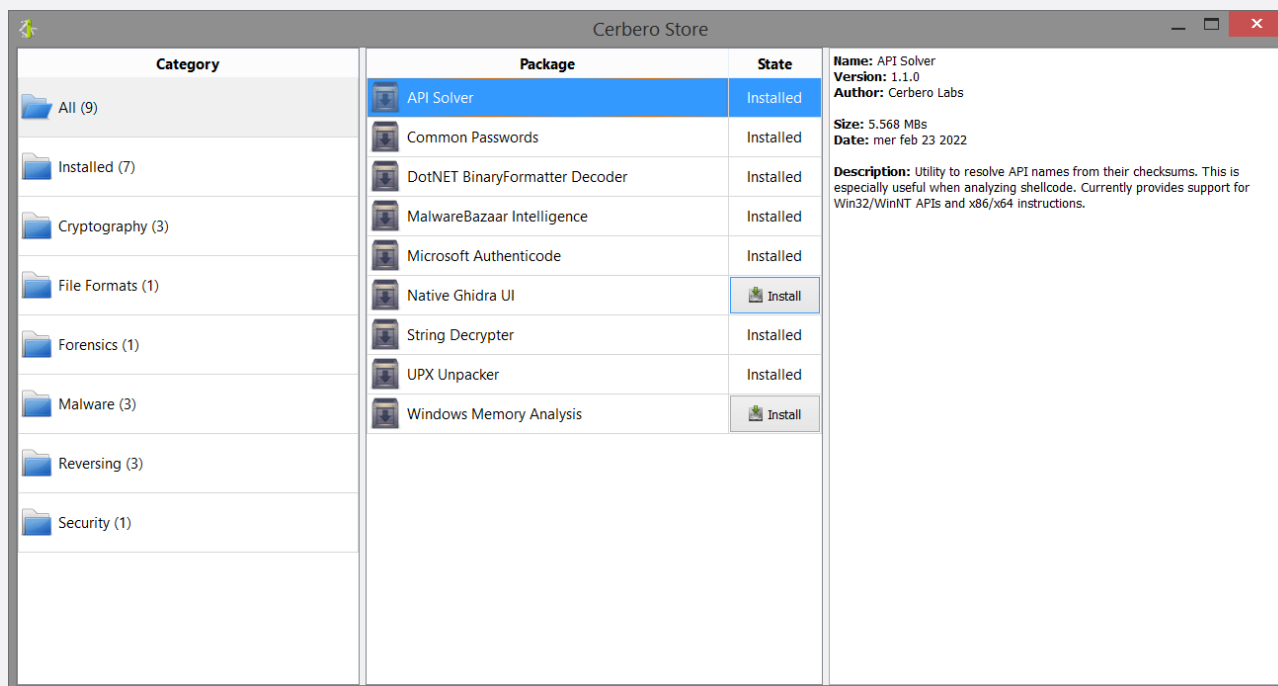
Another advantage of Cerbero Store is that some components which are used by a minority of users can now be decoupled from the main application. In fact, we moved our Windows memory analysis functionality to a package on Cerbero Store. That made all of our main software packages sensibly lighter.

Yet another component we have moved to a package on Cerbero Store is our native UI for Ghidra. The reason for this is that Ghidra sometimes changes its API between releases and breaks our plugin code. Thus, it happened in the past that we had to update our whole application just to update the plugin for Ghidra. This issue has now been solved by having the plugin for Ghidra as a separate package.

INDEX

CERBERO STORE	2
MALWAREBAZAAR INTELLIGENCE	3
CORE SDK DOCUMENTATION	3
CROSS-PLATFORM AUTHENTICODE	4
BLITZ MALWARE ANALYSIS	4
UPX UNPACKER	5
PARALLEL DECOMPILED	5
EXCEL MALWARE STEP BY STEP	6
ENGINE INTERMEZZO	7
CERTIFICATES SUPPORT	8
UNCOMMON FORMATS	8
INTERNAL PROJECT FILES	9
HAVING FUN DOING CTFs	10
CHALLENGE: CTF-LIKE MALWARE	10
API SOLVER	11
TIPS & TRICKS	12

In the last months we have started filling Cerbero Store with packages for all kind of purposes. Among others we have released the '[MalwareBazaar Intelligence](#)' package, the '[Microsoft Authenticode](#)' package, the '[String Decrypter](#)' package, the '[UPX Unpacker](#)' package, the '[DotNET BinaryFormatter Decoder](#)' package and the '[API Solver](#)' package.

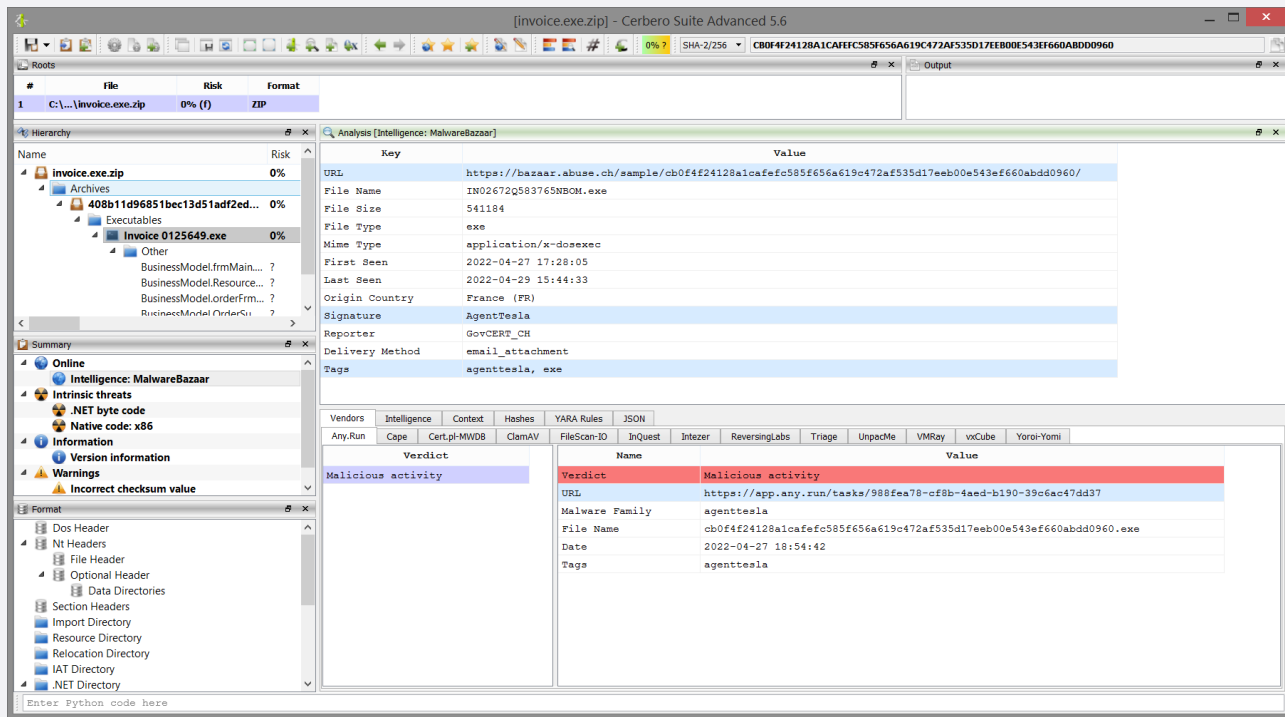


COMMERCIAL-ONLY PACKAGES

Personal license holders of Cerbero Suite have access to many packages on Cerbero Store. However, we reserve some packages such as the '[MalwareBazaar Intelligence](#)' package to commercial licenses. We try to limit the number of packages reserved to commercial licenses to those which we think fulfill a commercial activity. Additionally, some packages may be available to Cerbero Suite Advanced and not to Cerbero Suite Standard, in case they rely on features not available to the latter.

MALWAREBAZAAR INTELLIGENCE

...continued from page 1.



The MalwareBazaar intelligence report for a malicious sample analyzed in Cerbero Suite.

The 'MalwareBazaar Intelligence' package enables immediate access to intelligence from [MalwareBazaar](#) directly from the file report. The package graphically visualizes all the various sections of the MalwareBazaar report, including all external scan providers.

The interface uses different colors to highlight the maliciousness of a file and all the other highlighted items are hyper-links which bring you to the relevant MalwareBazaar

web-page.

Although the package is already incredibly useful as it is, we're not nearly finished with it!

There are many features we wanted to squeeze in, but we're unable to ahead of the release of Cerbero Suite 5.6.

In fact, the most interesting features have yet to come. So expect an update soon!

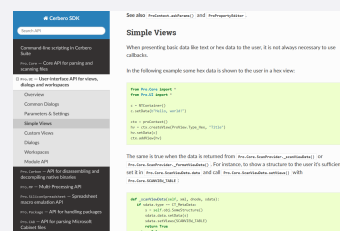
CORE SDK DOCUMENTATION

...continued from page 1.

Slowly but surely, in the last year we've been expanding our SDK documentation one module at a time and while we have just started documenting our built-in file format modules, we've already documented all of our core modules.

Namely, our core modules are:

- **Pro.Core** – Core API for parsing and scanning files
- **Pro.UI** – User-interface API for views, dialogs and workspaces
- **Pro.Carbon** – API for disassembling and decompiling native binaries
- **Pro.MP** – Multi-Processing API
- **Pro.SiliconSpreadsheet** – Spreadsheet macro emulation API
- **Pro.Package** – API for handling packages



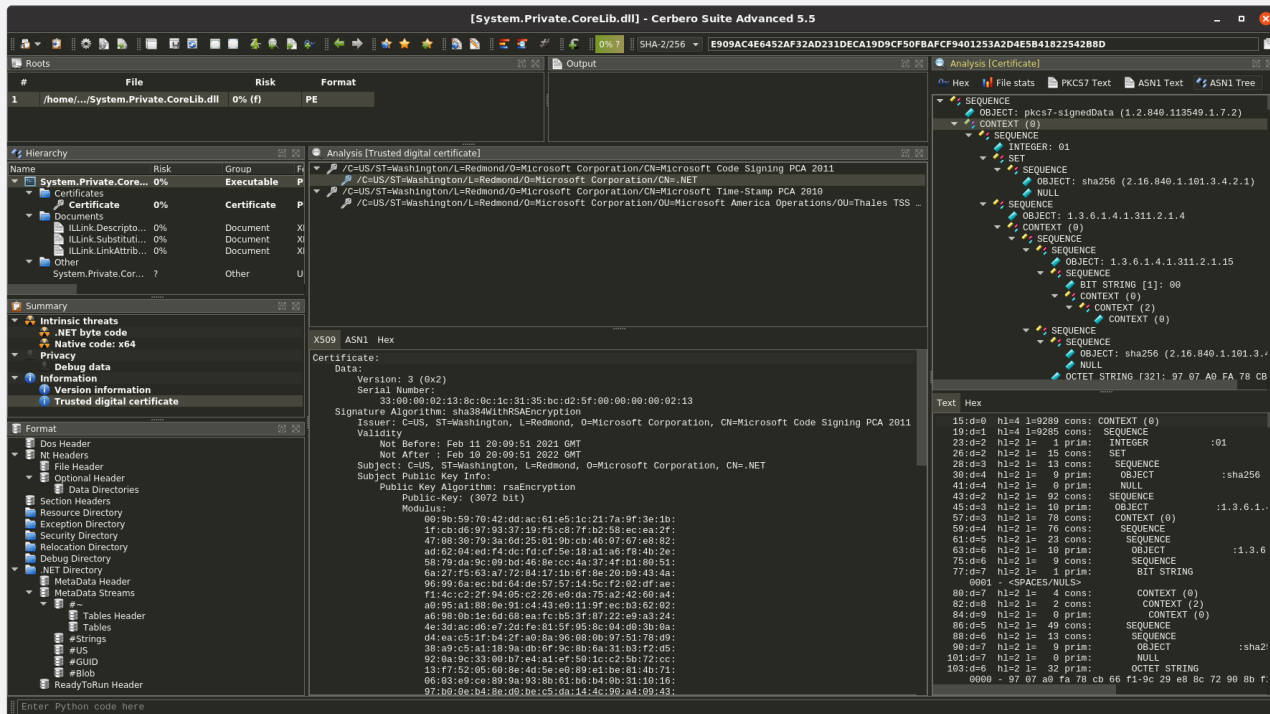
The documentation contains not only detailed explanations and code examples, but also descriptions of every class, function and constant present in each module.

With the already documented core modules, it is already possible to create every type of extension for Cerbero Suite and Cerbero Engine, including scan providers and disassembly loaders for custom file formats, scan hooks, actions, logic and key providers, graphical tools and much more.

Visit our [SDK documentation web-page](#) for more information.

CROSS-PLATFORM MICROSOFT AUTHENTICODE

... continued from page 1.



Verification of a DLL signed with Microsoft Authenticode on Linux.

In conjunction with our recently extended support for [certificate file formats](#), we now provide complete support for inspecting signed Portable Executable binaries on non-Windows systems.

The only required step to verify Authenticode signatures on systems such as Linux or macOS is to install our 'Microsoft Authenticode' package from Cerbero Store.

Cerbero Suite has been using its own implementation of Microsoft Authenticode for performance reasons since the [very beginning](#), back in 2012. However, thanks to the recently

introduced [Cerbero Store](#) we can now offer this feature on systems other than Windows.

We have also exposed Authenticode validation to our Python SDK.

```
from Pro.PE import *
print(PE_VerifyAuthenticode(obj))
```

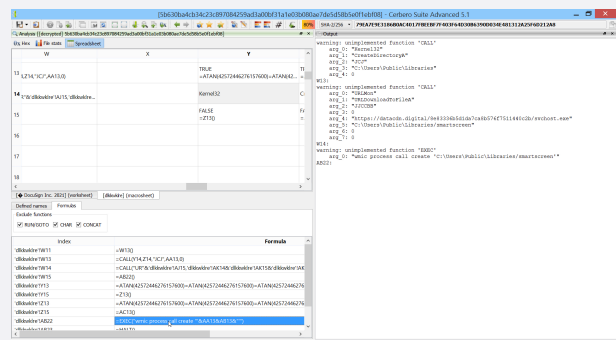
Alternatively, scan hooking extensions can check the generated report for the validation scan entries.

BLITZ MALWARE ANALYSIS

Do you get easily bored and distracted by trying to follow long malware analysis videos? Then perhaps we have a solution for you!

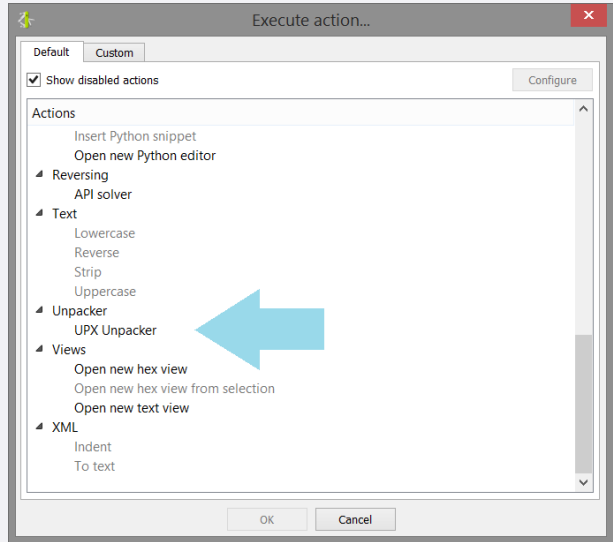
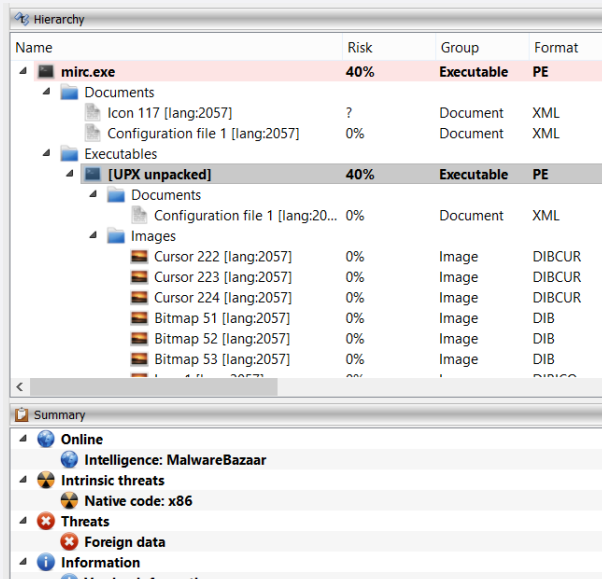
In a not-to-be-taken-too-seriously effort to showcase the manual analysis capabilities of Cerbero Suite, we have created a series of videos where we analyze malware samples in 3 minutes or less.

In our quest to reach virtuoso level, we analyzed an encrypted and obfuscated Excel malware sample in 19 seconds. You can watch the video on [YouTube](#)!



UPX UNPACKER

...continued from page 1.



Additionally, the unpacker can be invoked from Python.

But what is UPX?

From the [UPX web-site](#): "UPX is a free, portable, extendable, high-performance executable packer for several executable formats."

PE, ELF and Mach-O binaries are all supported.

If for some reason a binary is not automatically unpacked, the unpacker can be invoked manually as an action.

```
from Pkg.UPXUnpacker.Unpack import unpack

ret, output = unpack(file_name)
# print the unpacker output
print(output)
```

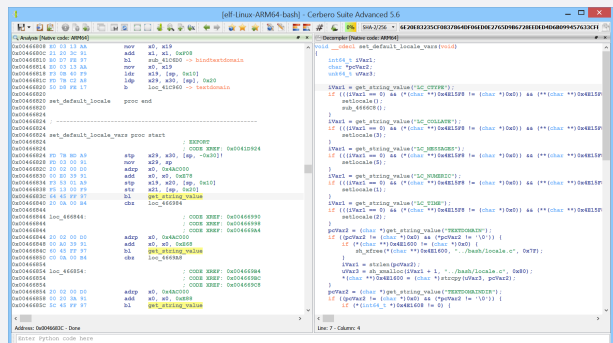
The 'UPX Unpacker' package is open-source to help our users write similar plugins. An important functionality we introduced to support this kind of extension is [the support for internal files](#).

PARALLEL DECOMPILING

Since version 5.2 of Cerbero Suite we used our [multi-processing technology](#) to parallelize the Sleigh decompiler by running it in a different process. This guarantees complete stability in case Sleigh encounters an issue and it makes every decompiling operation safe to cancel.

By parallelizing the decompiler we were also able to initialize it ahead of time, during the loading of the disassembly. Therefore, when the decompiler is invoked for the first time there is no initial delay, making it extra-snappy.

If you haven't done so already, give it a try: we're sure you'll love it!



DECOMPILED PLATFORMS

The decompiler supports the same platforms as our Carbon disassembler: x86, x64, ARM32 and ARM64.

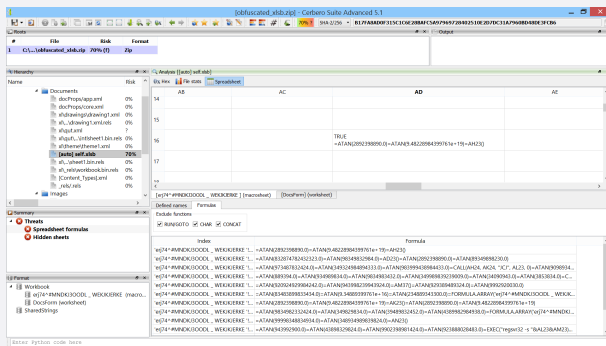
If you're wondering what Carbon is: it's an ultra-fast disassembler. Carbon has been specifically designed for malware triage and for handling vast amount of data such as memory images and crash dumps.

EXCEL MALWARE STEP BY STEP

Sample SHA-256: B17FA8AD0F315C1C6E28BAFC5A97969728402510E2D7DC31A7960BD48DE3FCB6

What follows is a step-by-step analysis of a malicious obfuscated Microsoft Office XLSB document. We deobfuscate the macros and use our proprietary **Silicon Excel Emulator** to understand the behavior of the code.

1. By previewing the spreadsheet in Cerbero Suite, we can see that the macros are obfuscated.

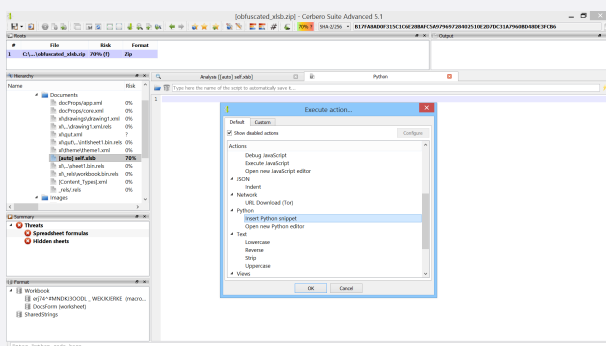


An obfuscated formula looks as follows:

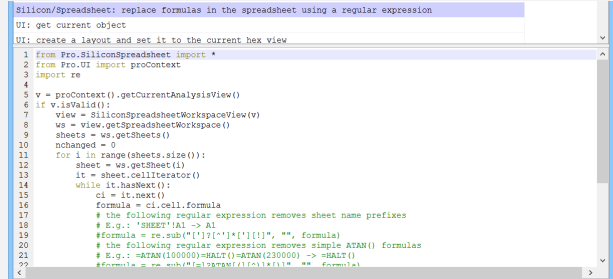
```
=ATAN(83483899833434.0)=ATAN
→ (9.34889399761e+16)=ATAN
→ (234889343300.0)=FORMULA.ARRAY('
→ erj74^#MNDKJ3OODL _ WEKJKJERKE '!
→ AT24&'erj74^#MNDKJ3OODL
→ WEKJKJERKE '!AT27&'erj74^#
→ MNDKJ3OODL _ WEKJKJERKE '!AT29&'
→ erj74^#MNDKJ3OODL
etc.
```

The malware uses the 'ATAN' macro and a very long sheet name for obfuscation.

2. We open a new Python editor and execute the action 'Insert Python snippet' (Ctrl+R).

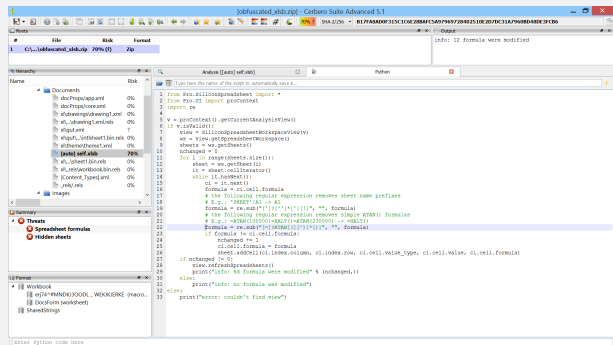


3. We insert the Silicon/Spreadsheet snippet to replace formulas.



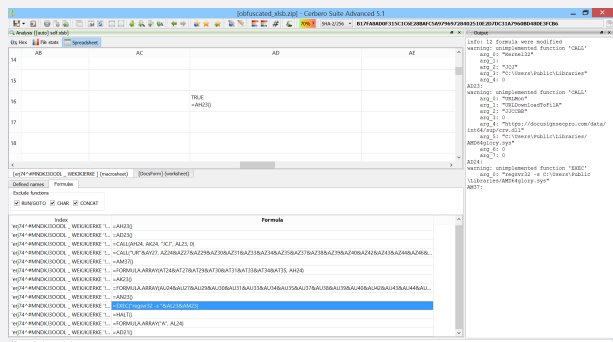
4. We uncomment both example regular expressions, as they were written based on this sample. One regex removes the 'ATAN' macro and the other removes the sheet name from cell names. Since there's only one spreadsheet, no extra logic is needed.

We then execute the script (Ctrl+E).



5. The script modifies 12 formulas. At this point we can easily identify 'CALL' and 'EXEC' macros and use the Silicon Excel Emulator to emulate them (Ctrl+E).

Just by emulating these macros, we can see that the malware creates a directory, downloads a file into it and executes it.



Done.

ENGINE INTERMEZZO



In case you're not yet familiar with Cerbero Engine, here is a quick introduction. You can read more on our [web-page](#).

WHAT IS CERBERO ENGINE?

Cerbero Engine is our solution for enterprise projects such as cloud or in-house services. It offers the same SDK as Cerbero Suite Advanced and has already been used to analyze billions of files.

WHAT CAN IT DO?

Our SDK is extensive. It features support for dozens of file formats, scanning, disassembly, decompiling, emulation, signature matching, file carving, decompression, decryption and much more.

We make sure Cerbero Engine keeps up with the latest threats and challenges presented by file formats which are difficult to analyze. We offer state-of-the-art support for various file types such as Adobe PDF and Microsoft Office.

HOW SECURE IS IT?

Cerbero Engine has been designed taking into account any type of security issue when analyzing malicious files: buffer overflows, integer overflows, infinite loops, infinite recursion, decompression bombs, denial-of-service etc.

WHAT PLATFORMS DOES IT SUPPORT?

Just like Cerbero Suite, Cerbero Engine is cross-platform. Currently we offer it for both Windows (x86, x64) and Linux (x64). It is also compatible with older version of Windows and Linux.

CAN IT BE EMBEDDED?

Cerbero Engine is deployed as an embeddable module: a Dynamic-Link Library (DLL) on Windows and a Shared Library on Linux. The engine can be loaded from both C/C++ and Python 3.

Loading the engine from Python is extremely simple.

```
from ProEngine import *

# initialize the engine
proEngineInit()

# from here on the SDK can be accessed
from Pro.Core import *
# ...

# finalize the engine before exiting
proEngineFinal()
```

Loading the engine from C/C++ is also very simple: it only requires including the 'ProEngine' header and specifying the location of the engine on disk.

```
#define PRO_ENGINE_INIT
#include "ProEngine.h"

int main()
{
    // initialize the engine
    if (!proEngineInit("/path/to/the/
    ↪ engine", ProEngine_InitPython))
        return -1;

    // from here on the SDK can be
    ↪ accessed

    // finalize the engine before exiting
    proEngineFinal();
    return 0;
}
```

IS IT FAST?

While our SDK is in Python, our engine is written in C++ and is both multi-thread and multi-process. This design decision guarantees maximum speed, while also giving our customers the capability to write cross-platform code that is compatible across all of our products.

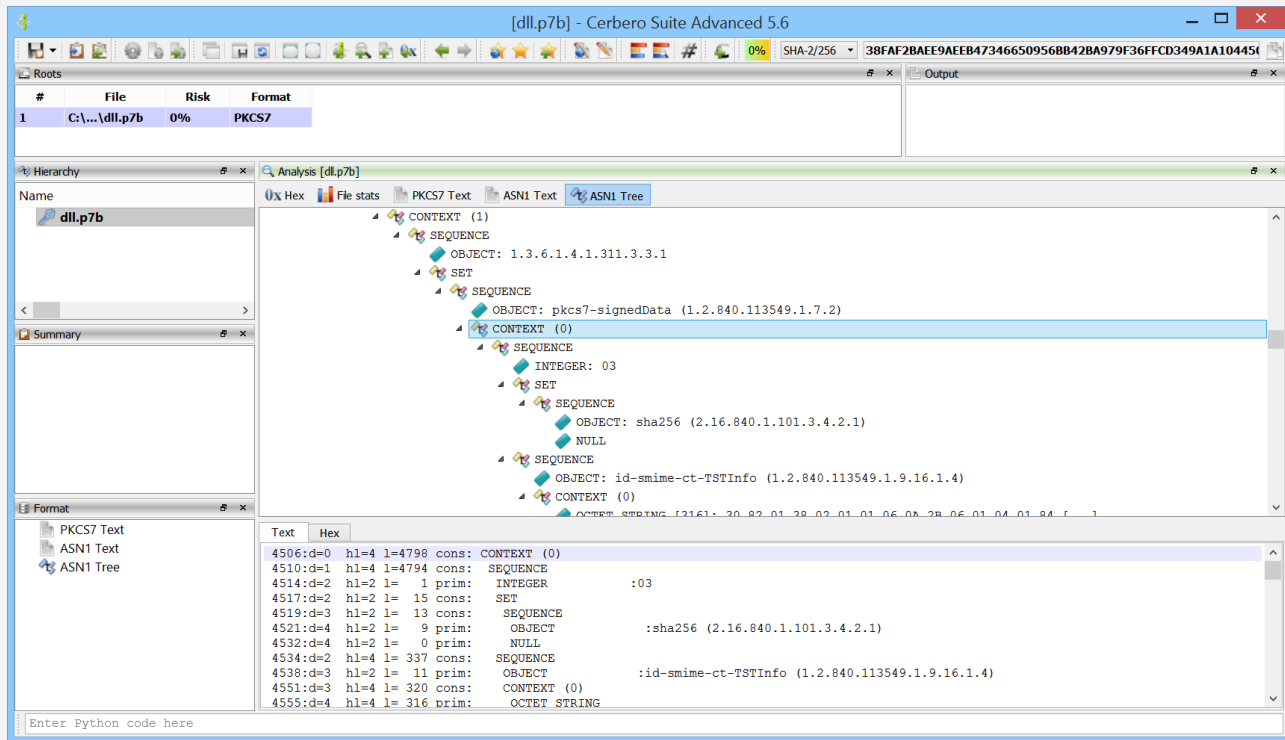
Since the SDK is in Python, our customers don't need to worry about rebuilding their project when the engine is updated. Moreover, we take great care not to introduce breaking changes to the SDK: we don't want our customers to worry that an update could cause their code to stop working!

HOW DO YOU LICENSE IT?

We license Cerbero Engine on a per-case basis. The licensing depends upon the scope of the project. If you are interested in a quotation, please [contact us](#).

Purchasing a license of Cerbero Engine comes with discounted lab licenses of Cerbero Suite. By using Cerbero Suite, your engineers can interactively debug parsing issues, analyze edge cases, use our Python editor for development and create graphical applications that work in conjunction with the engine.

CERTIFICATES SUPPORT



The ASN.1 tree.

Since version 5.5 of Cerbero Suite and version 2.5 of Cerbero Engine we fully support certificate formats. While Cerbero Suite would already let you inspect certificates inside binaries, now it can load them directly from disk and also lets you inspect each individual ASN.1 object.

Both DER and PEM encodings for certificates are supported and you can inspect all types of certificates, including X509, PKCS7 and PKCS12.

We have also exposed the code to our Python SDK in order to make the programmatic parsing of certificates a simple task.

For example, the following code enumerates all the objects in an ASN.1 DER file.

```
from Pro.Core import *
from Pro.Certificates import *

def enumerateObjects(fname):
```

```
c = createContainerFromFile(fname)
if c.isNull():
    return
obj = DERObject()
if not obj.Load(c):
    return
class Visitor(DERObjectVisitor):
    def Visit(self, obj, oi):
        print(obj)
        ↪ GetObjectDescription(oi
        ↪ ))
        print("    offset:", hex(oi.
        ↪ offset), "size:", hex(
        ↪ oi.content_size))
        return 0
v = Visitor()
obj.VisitObjects(v)
```

You can check out the [SDK documentation](#) for the Pro.Certificates module for more code examples.

UNCOMMON FORMATS

While the support for common file formats such as certificates is available to personal licenses, the support for some uncommon file and data formats is sometimes restricted to commercial licenses.

Check out [this video on YouTube](#) where we demonstrate the use of a package available to commercial licenses to decode the BinaryFormatter encoded payload in a malicious Microsoft HTML Application.

```
46 public void Decode(string finalUrl, string avUrl, string doc, string documentName)
47 {
48     locals: int local_0;
49     System.Exception local_1;
50
51     try
52     {
53         try
54         {
55             /* 000002AC 02 */ idarg_0 // string finalUrl
56             /* 000002AD 7B 02 00 00 04 */ idrid location // string
57             /* 000002B0 0E 04 */ idarg_1 arg_4
58             /* 000002B4 28 12 00 00 0A */ call System.IO.Path::Combine(string, string) // returns string
59             /* 000002B9 05 */ idarg_2 // string documentName
60             /* 000002BA 28 13 00 00 0A */ call System.Convert::FromBase64String(string) // returns byte []
61             /* 000002BF 28 04 00 00 06 */ call Decompress(byte [] data) // returns byte []
62             /* 000002C4 28 14 00 00 0A */ call System.IO.File::WriteAllBytes(string, byte []) // returns void
63             /* 000002C9 02 */ idarg_3 // string finalUrl
64             /* 000002CA 7B 02 00 00 04 */ idrid location // string
65             /* 000002CF 0E 04 */ idarg_4 arg_4
66             /* 000002D1 28 12 00 00 0A */ call System.IO.Path::Combine(string, string) // returns string
67             /* 000002D6 28 15 00 00 0A */ call System.Diagnostics.Process::Start(string) // returns System.Diagnostics
68             /* 000002DB 26 */ pop
69             /* 000002DC DE 03 */ leave_s loc_53
70         }
71     }
```


INTERNAL PROJECT FILES

With version 5.6 of Cerbero Suite we introduced a new major core feature, namely the capability to generate files which do not exist on disk and store them in the analysis report.

While this feature doesn't seem so important, it has countless real-world applications. For instance, an unpacker may unpack a file during the scanning process and store the resulting file as an internal file. When the unpacked file is requested, the operation bypasses the unpacker and directly accesses the internal file.

In the following example a dummy internal file is generated for a scanned file and is added as an embedded object to the generated report.

```
from Pro.Core import *

def scanning(sp, ud):
    # skip if it's a nested scan: avoid
    #     ↪ recursion
    if sp.isNestedScan():
        return
    # a global report is needed to store
    #     ↪ internal files
    r = sp.getGlobalReport()
    if not r:
```

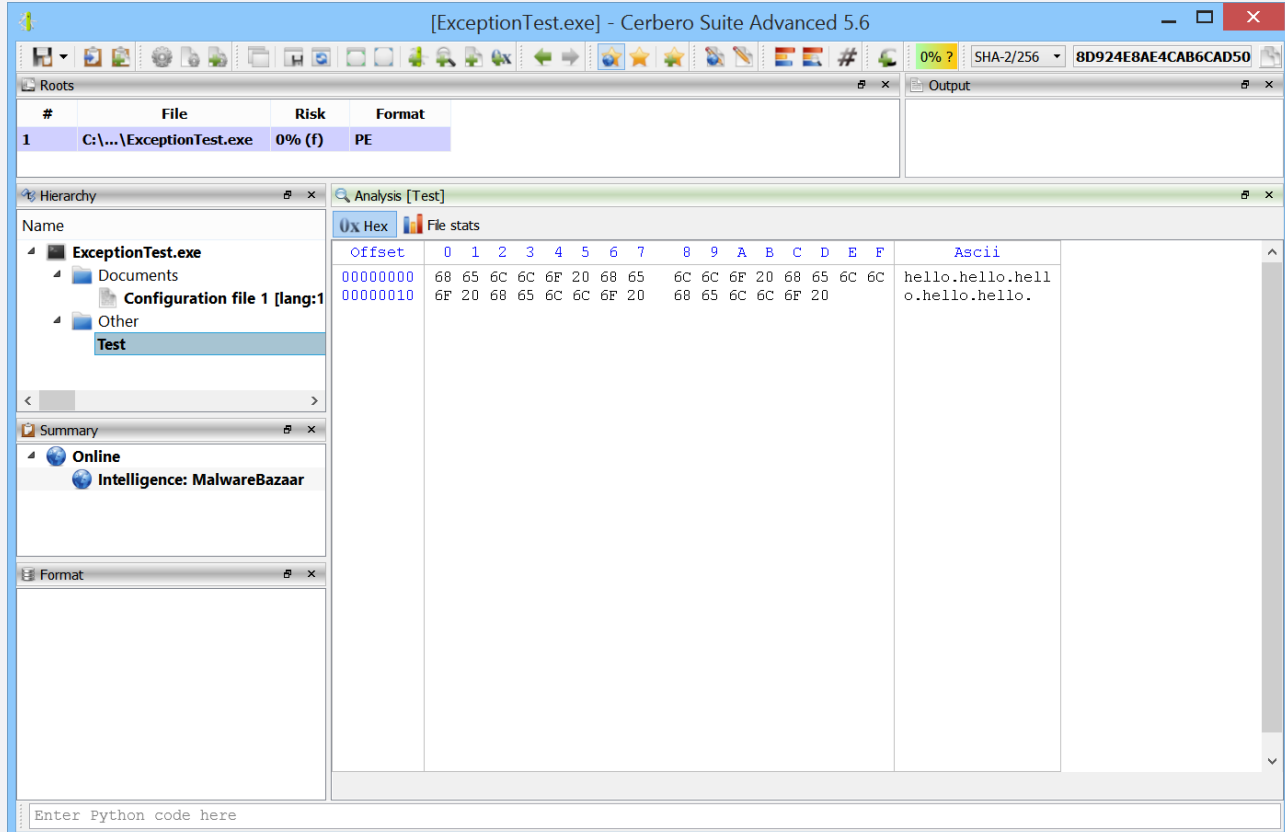
```

        return
# generate an internal file id
uid = r.newInternalFileUID()
if not uid:
    return
# retrieve the path on disk for the
    ↳ internal file
path = r.newInternalFilePath(uid)
# generate the content of the
    ↳ internal file
with open(path, "w") as f:
    f.write("hello " * 5)
# save the internal file
r.saveInternalFile(uid, "TEST FILE")
# add the internal file as embedded
    ↳ object
sp.addInternalFile(uid, "", "Test")

```

The lines in the 'hooks.cfg' configuration file:

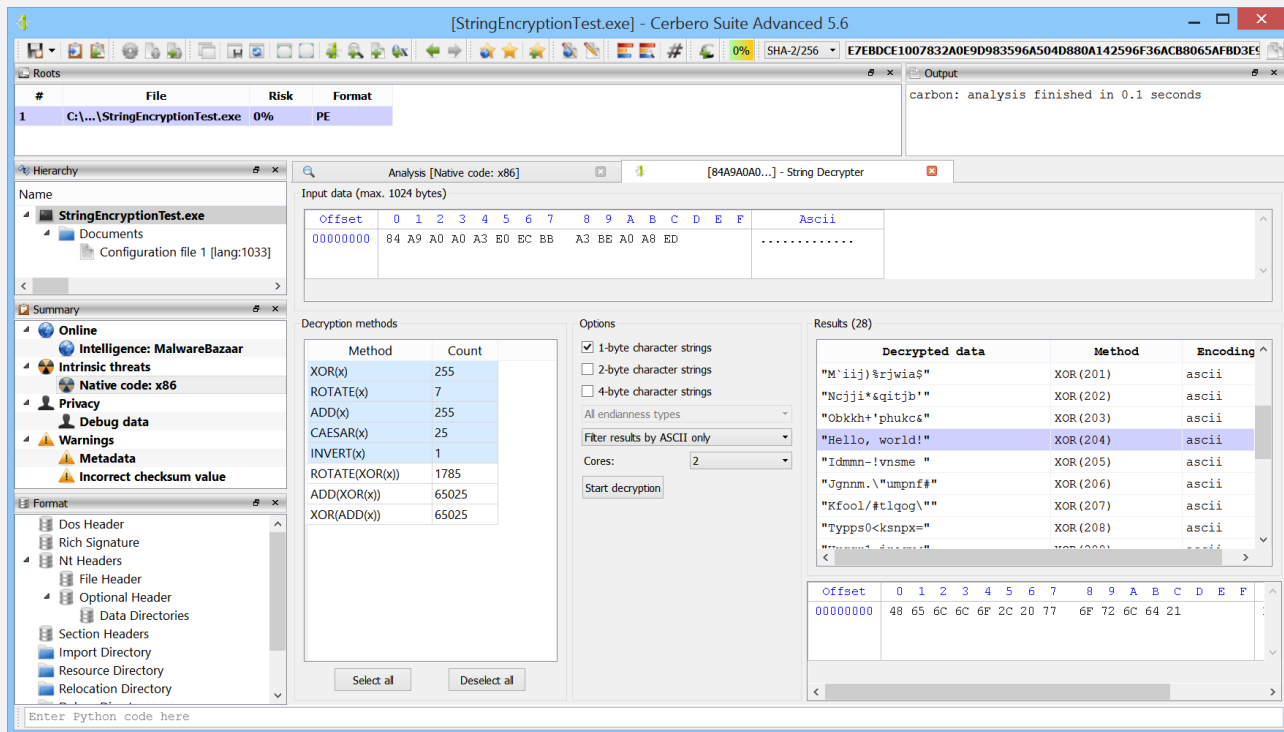
```
[IntFileTest_1]
label = Internal file test
file = intfile_hook.py
scanning = scanning
enable = yes
```



The internal file created by our code.

HAVING FUN DOING CTFs

... continued from page 1.



Example of a decrypted string.

Do you need to brute-force the decryption of a string in a CTF challenge? Our 'String Decrypter' package comes to the rescue!

The string decryption tool can be invoked as an action from a hex view or a Carbon disassembly view and can be used to brute-force the decryption of strings and byte-arrays.

The tool supports various algorithms and string encodings combined with endianness. Moreover, it can filter decoded strings with the following options:

- Don't filter (includes raw byte-arrays)
- Include only decoded strings

- Include only strings with ASCII characters
- Include only string matching a regular expression provided by the user

Parallel execution is also supported, as it will make a difference if more algorithms are added to the list. Also, for every decryption method the number of combinations is displayed.

For every decrypted entry, String Decrypter shows the performed operation along with the string encoding.

While the 'String Decrypter' package is useful during CTFs, it is equally so when reversing malware!

CHALLENGE: CTF-LIKE MALWARE

Not all malware is tedious. In fact, some of it resembles CTF challenges! From a [Twitter post](#) by InQuest, we discovered an interesting malware.

A few of the things encountered in this sample: encrypted MS Office document, VBA, Windows Link file (LNK), OLE objects, Windows help files (CHM), PNG steganography and PowerShell.

The analysis should take about 15-20 minutes in Cerbero Suite if you feel like testing your skills!

Sample SHA-256: 46AFA83E0B43FDB9062DD3E5FB7805997C432DD96F09DDF81F2162781DAAF834

If you get stuck in the analysis, [visit our blog](#) for some helpful screenshots (spoiler alert).

word/embeddings/oleObject1.bin	0%	Document	CFBF	Embedded
HelpDocument.docm	0%	Document	CFBF	Embedded
(decrypted) HelpDocument.docm	0%	Archive	ZIP	Embedded
Content_Type.xml	0%	Document	XML	Embedded
rels/rels	0%	Document	XML	Embedded
word/rels/document.xml.rels	90%	Document	XML	Embedded
word/document.xml	0%	Document	XML	Embedded
word/footnotes.xml	0%	Document	XML	Embedded
word/endnotes.xml	0%	Document	XML	Embedded
word/rels/vbaProject.bin.rels	0%	Document	XML	Embedded
word/vbaProject.bin	100%	Document	CFBF	Embedded
word/theme/theme1.xml	0%	Document	XML	Embedded
word/embeddings/oleObject12.bin	40%	Document	CFBF	Embedded
Helpfile.jpg	0%	Document	ITSF	Embedded
System.lnk	0%	System	LNK	Embedded
word/embeddings/oleObject1.bin	0%	Document	CFBF	Embedded

API SOLVER

Our 'API Solver' package is available on Cerbero Store for all commercial licenses of Cerbero Suite Advanced.

The purpose of this package is to convert CRC-like hashes contained in places such as shellcode back to their API names. The package features dozens of built-in hashing methods commonly found in malware.

Once installed from Cerbero Store, the API Solver user-interface will be available as an action. If the action is executed in the context of a Carbon disassembly, additional functions are available: API Solver can detect API hashes in code instructions and comment solved hashes in the Carbon disassembly.

You can choose which group of modules to use to solve API names. In each module group you can select one or more modules to populate the API solver database. If the hash method is set to 'all', API Solver tries to figure out the hashing method.

You can solve individual values and select a custom hashing method, as well as a built-in hashing method. When choosing a built-in hashing method, you can inspect its pseudo-code.

The solver can also be used programmatically:

```
from Pkg.APISolver import APISolver
```

```
solver = APISolver("win32", ("kernel32",
    ↳ "urlmon"))
for h in (0xEC0E4E8E, 0x702F1A36, 0
    ↳ xE8AFE98, 0x73E2D87E):
```

```
print(solver.solve(h))
```

The output of the code:

```
['KERNEL32.LoadLibraryA' (ror13_add_32)]
['URLMON.URLDownloadToFileA' (
    ↳ ror13_add_32)]
['KERNEL32.WinExec' (ror13_add_32)]
['KERNEL32.ExitProcess' (ror13_add_32)]
```

The solver can also be invoked with one or more built-in methods:

```
solver = APISolver("win32", ("kernel32",
    ↳ "urlmon"), ("ror13_add_32",))
```

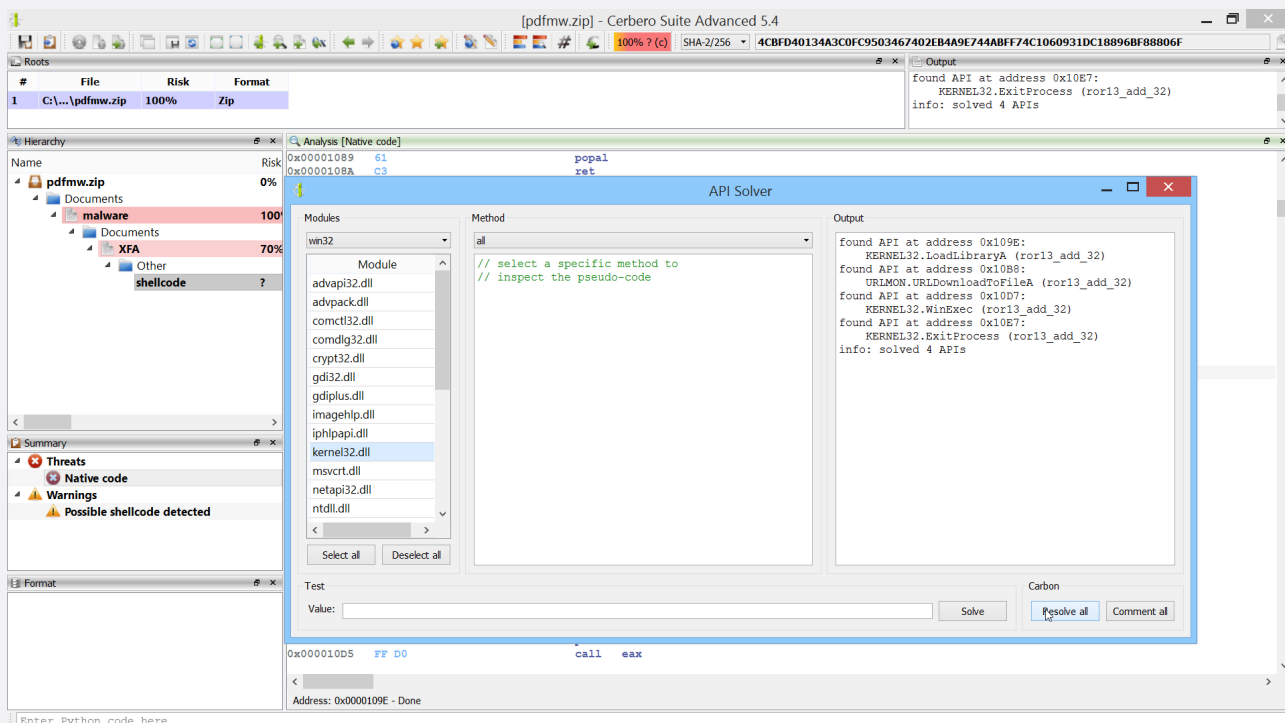
As well as one or more custom methods (built-in names and custom methods can be mixed together):

```
def custom_hash(name):
    h = 0
    for c in name:
        h += c
    return h
```

```
solver = APISolver("win32", ("kernel32",
    ↳ "urlmon"), (custom_hash,))
```

If the 'custom_hash' function has an additional argument, it will also be passed the name of the module.

You can watch a full introduction to the package on [YouTube!](#)



Some solved API hashes contained in a shellcode inside of a PDF document.

TIPS & TRICKS

As the 'Pro.PE' module is not yet documented on our [SDK web-page](#), we recently had a customer contact us to know whether it was possible to programmatically modify a .NET metadata table field.

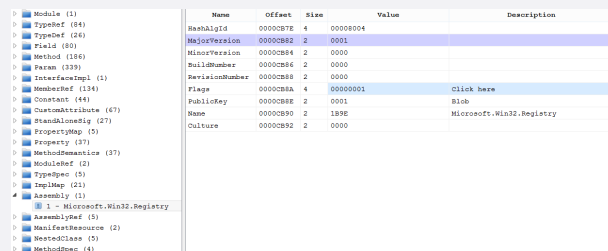
Parsing .NET metadata tables is **not trivial**, because in order to know the offset of a specific table it is necessary to calculate the size of all the tables which precede that table. The difficulty in the size calculation is that .NET metadata tables contain fields whose size is dynamically calculated.

However, parsing .NET metadata tables becomes trivial when using our SDK. The following function modifies the value of a field in the 'Assembly' table.

```
from Pro.Core import *
from Pro.PE import *

def editDotNETTable(fname):
    # open the file with write privileges
    f = NT_OpenFile(fname,
        ↳ NTFileOpt_Write)
    if not f:
        return
    # encapsulate the file in a container
    c = NTContainer()
    c.setData(f, True)
    # load the container as a PE binary
```

```
pe = PEObject()
if not pe.Load(c):
    return
# retrieve the .NET tables
tables = pe.MDTables("#~")
# retrieve the array of Assembly
↳ entries
assembly = tables.at(Assembly_t)
if assembly.IsNull():
    return
# modify a field in the structure
print("Original MajorVersion:",
    ↳ assembly.Uns("MajorVersion"))
assembly.Set("MajorVersion", 1)
print("Modified MajorVersion:",
    ↳ assembly.Uns("MajorVersion"))
```



Name	Offset	Size	Value	Description
HashAlgId	0000007E	4	00000004	
MajorVersion	00000082	2	0001	
MinorVersion	00000084	2	0000	
BuildNumber	00000086	2	0000	
RevisionNumber	00000088	2	0000	
Flags	0000008A	4	00000001	Click here
PublicKey	0000008E	2	0001	Blob
Name	00000090	2	189E	Microsoft.Win32.Registry
Culture	00000092	2	0000	

The edited field in the .NET binary.

CERBERO LABS



If you have any questions, feel free to contact us at: info@cerbero.io

You can follow us on [Twitter](#) to be notified about the latest updates!

If you're familiar with command-line scripting in Cerberero Suite, you might know that by running a script without the '-c' argument all output is redirected to the output view in the main window.

In certain cases, however, it might be desirable to avoid the creation of a main window.

For this purpose we have introduced the '-g' argument.

For example:

```
cerpro.exe -g -r foo.py
```

If the script doesn't create an output view, then the output of the 'print' function isn't visible.

Furthermore, since version 5.5 of Cerberero Suite we have added terminal support for Windows.

On Windows running scripts with the '-c' argument results in not being able to see the stdout output. The reason for this is that the 'cerpro' binary is built as a GUI application and therefore is not attached to a terminal.

To overcome this limitation we have created a launcher for Windows called 'cerpro_console.exe'.

For example, the following code asks the user to enter a string and prints it back:

```
cerpro_console.exe -e "t=input('Enter a
↳ string: ');print(t)"
```

If you're interested in learning more about command-line scripting in Cerberero Suite, you can read our dedicated [SDK documentation page](#).