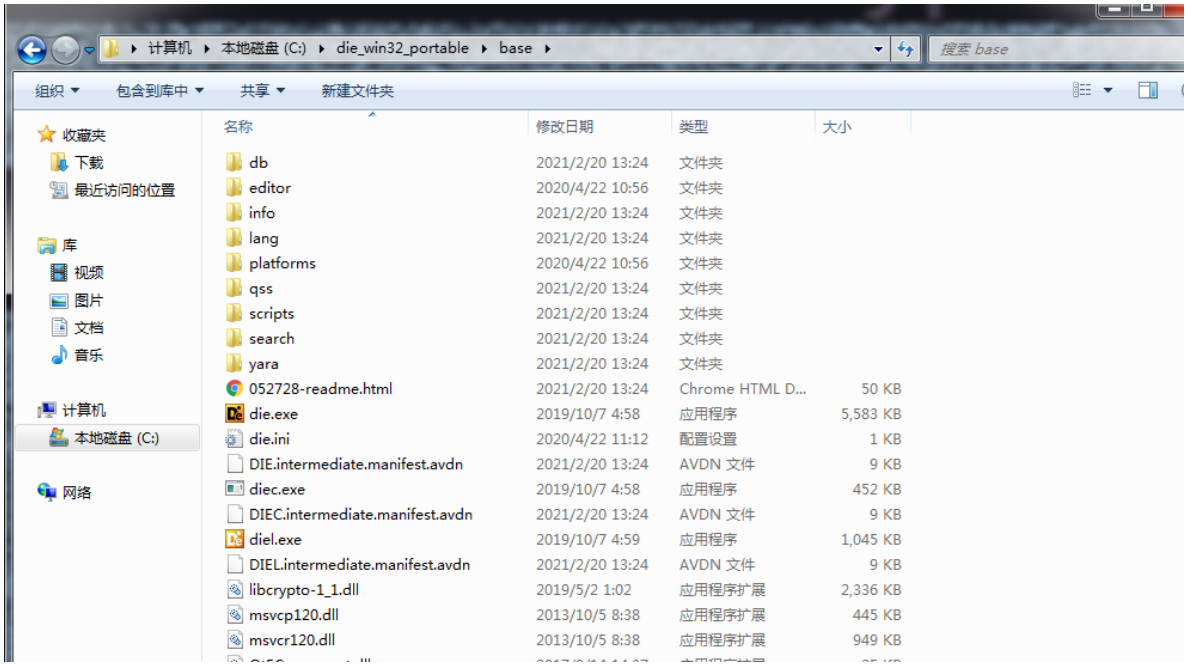
Avaddon勒索

该勒索病毒使用C++语言进行编写，采用RSA-2048和AES-256加密算法对文件进行加密，加密库使用的是Windows自带的CryptAPI



被该勒索加密后的文件后缀为avdn

解密工具

国外安全研究人员发布了一款Avaddon勒索病毒解密工具，解密工具源代码地址：

https://github.com/JavierYuste/AvaddonDecryptor

经过测试，这个工具确实是可以解密被Avaddon勒索加密的文件，下面是 我输出的解密时的日志

```
    Removing signature
Size of the encrypted file with signature: 156184
Size of the encrypted file without signature: 155648
    Decrypting file
    Decrypting file C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\1049\License_SysClrTypes.rtf.avdn with k
    Decrypting C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\1049\License_SysClrTypes.rtf.avdn
    Truncating to 152187
> Found file C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\2052\License_SysClrTypes.rtf.avdn
Signature: b'\n#\x02\x00\x00\x00\x00\x00\x00\x02\x00\x00\x01\x00\x00\x00\x07\x03\x03\x01\x01\x01\x89\x05'
Original file size unpacked: 140042 (0x2230a)
Hardcoded signature: 0x1030307
    Removing signature
Size of the encrypted file with signature: 147992
Size of the encrypted file without signature: 147456
    Decrypting file
    Decrypting file C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\2052\License_SysClrTypes.rtf.avdn with k
    Decrypting C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\2052\License_SysClrTypes.rtf.avdn
    Truncating to 140042
> Found file C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\3082\License_SysClrTypes.rtf.avdn
Signature: b'\xac\xaa\x01\x00\x00\x00\x00\x00\x00\x02\x00\x00\x01\x00\x00\x00\x07\x03\x03\x01\x01\x01\x89\x05'
Original file size unpacked: 109228 (0x1aaac)
Hardcoded signature: 0x1030307
    Removing signature
Size of the encrypted file with signature: 115224
Size of the encrypted file without signature: 114688
    Decrypting file
    Decrypting file C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\3082\License_SysClrTypes.rtf.avdn with k
    Decrypting C:\Program Files (x86)\Microsoft SQL Server\90\Shared\Resources\3082\License_SysClrTypes.rtf.avdn
    Truncating to 109228

--- SUMMARY ---
Total files: 141136
Decrypted files: 4212
Time: 182.6574169
```

想要解密被加密的文件 需要具备下面几个条件

1. 当前存活的勒索进程
2. 勒索进程的dump文件
3. 一份被加密的文件
4. 一份被加密文件的原始文件

然后调用下面这个命令

```
python3 main.py -f <encrypted_file> -o <original_file> -d <memory_dump> --folder
<folder_to_decrypt>
```

就可以解密机器上所有的被加密文件了（需要修改源码中写死的三个路径 才能把解密脚本跑起来）

解密工具原理

目前我的需求是把这个解密能力集成的到公司的工具里，再来分析一下代码

```python
def main():
    # 解析命令行参数
    args = p.parse_args()
    # 获取被加密的原始文件名
    original_file = os.path.abspath(args.original)
    # 获取被加密的文件名
    encrypted_file = os.path.abspath(args.file)
    # 分割被加密文件的文件名和后缀
    filename, file_extension = os.path.splitext(encrypted_file)
    # 获取dump文件路径
    dump = os.path.abspath(args.dump)
    # 这个是密钥特征码
    pattern = "10660000010000002000000"

    # 在dump文件中搜索特征码 返回偏移列表
    # Get a list of offsets of the matches with searchbin
    offsets = search_pattern(dump=dump, pattern=pattern)
    print(f"Offsets: {offsets}")

    # 从搜索到的列表中 获取每一个可能的key
    # Get each possible key from the list of offsets
    possible_keys, pointers_to_possible_keys = get_keys_from_offsets(dump=dump, offsets=offsets)
    print(f"Pointers to possible keys: {pointers_to_possible_keys}")
    print(f"Possible keys: {possible_keys}")
```

首先在dump文件中搜索所有可能的密钥，然后返回一个偏移列表，再根据这个偏移列表，去拿到所有的key

```
Offsets: [7465731, 32681323, 32687019, 32688443, 32691291, 32700547, 32705531, 35507403, 35508115, 35511675, 35512387, 35521643, 35524491, 35530899, 35537307, 35545851, 35556531, 35563
Structure found: 1066000001000000200000005011b700
Structure found: 1066000001000000200000000d0319704
Structure found: 1066000001000000200000010489704
Structure found: 1066000001000000200000000a04d9704
Structure found: 1066000001000000200000000c0589704
Structure found: 1066000001000000200000000e87c9704
Structure found: 1066000001000000200000060909704
Structure found: 1066000001000000200000030c1d604
Structure found: 1066000001000000200000000f8c3d604
Structure found: 1066000001000000200000000e0d1d604
Structure found: 1066000001000000200000000a8d4d604
Structure found: 1066000001000000200000000d0f8d604
Structure found: 1066000001000000200000000f003d704
```

输出的日志显示offset有90个，也就是说有90个AES的密钥

```
# 尝试每一个密钥 一直到解密成功
# Try each key till success
success = False
i = 0
possible_keys = list(dict.fromkeys(possible_keys))
while not success and i < len(possible_keys):
    # Decrypt file 解密文件 调用C++程序
    decrypted_file = decrypt_file(data['encrypted_truncated_file'], possible_keys[i])

    # Truncate to original size  截断为原始大小
    with open(decrypted_file, "r+b") as f:
        f.truncate(data["original_size"])
    # Compare with the original file  与原始文件比较
    success = filecmp.cmp(decrypted_file, original_file, shallow=True)

    #不成功则继续解密文件
    if not success:
        i = i + 1
```

然后利用搜索到的key去解密文件，每解密一次，都去和源文件进行比对，比对成功则说明密钥正确

```
# 如果成功 打印出正确的密钥 并开始解密整个文件
if success:
    print(f"[SUCCESS] Found the correct symmetric key: {possible_keys[i]}")
    os.remove(data['encrypted_truncated_file'])
    decrypt_whole_system(args.folder, possible_keys[i], file_extension)
else:
    shutil.copy(f"{encrypted_file}.backup_copy", encrypted_file)
    os.remove(f"{encrypted_file}.backup_copy")
    print("[FAIL] Did not find the correct symmetric key")
```

比对成功之后，开始解密整个系统的文件。

```
def decrypt_file(file, key):
    print(f"\tDecrypting file {file} with key {key}")
    with open("key_bytes", 'wb') as f:
        f.write(key)
    key_path = os.path.abspath("key_bytes")
    # Invoke C++ program, which decrypts a specified file with a given key  调用c++程序，用给定的密钥解密指定的文件
    abs_path = os.path.abspath(file)
    filename, file_extension = os.path.splitext(abs_path)
    print(f"\tDecrypting {abs_path}")
    subprocess.run(["DecryptFile.exe", abs_path, filename, key_path], stdout=subprocess.PIPE)
    if os.stat(abs_path).st_size > 0x100000:
        with open(abs_path, "r+b") as f:
            mm = mmap.mmap(f.fileno(), 0)
            with open(filename, "r+b") as f2:
                # Memory-map the file, size 0 means whole file
                mm.seek(0x100000)
                f2.seek(0, 2)
                while mm.tell() < mm.size() and (mm.size() - mm.tell()) > 0x2000:
                    f2.write(mm.read(0x2000))
                if mm.tell() < mm.size():
                    f2.write(mm.read(mm.size() - mm.tell()))
            mm.close()
    return filename
```

解密文件时，传入被加密的文件路径和解密后的文件路径以及密钥文件路径，然后调用DecryptFile.exe对文件进行解密

需要特殊处理的是，如果文件大小大于0x100000个字节，那么解密完成之后需要将0x10000字节后的数据全部复制到解密后的文件。

也就是说这个勒索实际上只会加密前0x10000个字节，这个细节在目前已有的分析报告中并没有提及。

```python
def decrypt_whole_system(rootdir, key, extension):
    print("Decrypting whole system")
    total_files = 0
    total_encrypted_files = 0
    start_time = time.perf_counter()
    for subdir, dirs, files in os.walk(os.path.abspath(rootdir)):
        for file in files:
            total_files += 1
            file_path = os.path.abspath(os.path.join(subdir, file))
            if (file_path.endswith(extension)) and "$Recycle.Bin" not in file_path:
                try:
                    print(f"> Found file {file_path}")
                    total_encrypted_files += 1
                    data = get_signature_data(file_path)
                    print(f"\tRemoving signature")
                    data['encrypted_truncated_file'] = remove_signature(file_path)
                    print("\tDecrypting file")
                    decrypted_file = decrypt_file(data['encrypted_truncated_file'], key)
                    print(f"\tTruncating to {data['original_size']}")
                    with open(decrypted_file, "r+b") as f:
                        f.truncate(data["original_size"])
                    os.remove(file_path)
                except OSError:
                    print("Permissions denied?")
                    pass

    print(f"\n--- SUMMARY ---"
```

解密完成之后，将文件截断为原始文件大小

那么这里其实有一个问题，为什么可以根据内置的特征码搜索到AES密钥？那个密钥的特征码是哪来的？

作者在代码中给出了这样一句注释

```
# Dump the process with procdump.exe -ma <PID>
# Pattern to search for (part of the key_data_s structure, in particular alg_id, flags and key_size):
# 106600000100000020000000
```

根据这个提示，找到了这个结构体，原文出处：https://forums.codeguru.com/showthread.php?79163-Structure-of-HCRYPTKEY-Data

```
struct key_data_s
{
    void *unknown; // XOR-ed
    uint32_t alg;
    uint32_t flags;
    uint32_t key_size;
    void* key_bytes;
};
```

勒索采用的是AES256，那 alg = 0x00006610,keysize=0x00000020, flags= 0x1

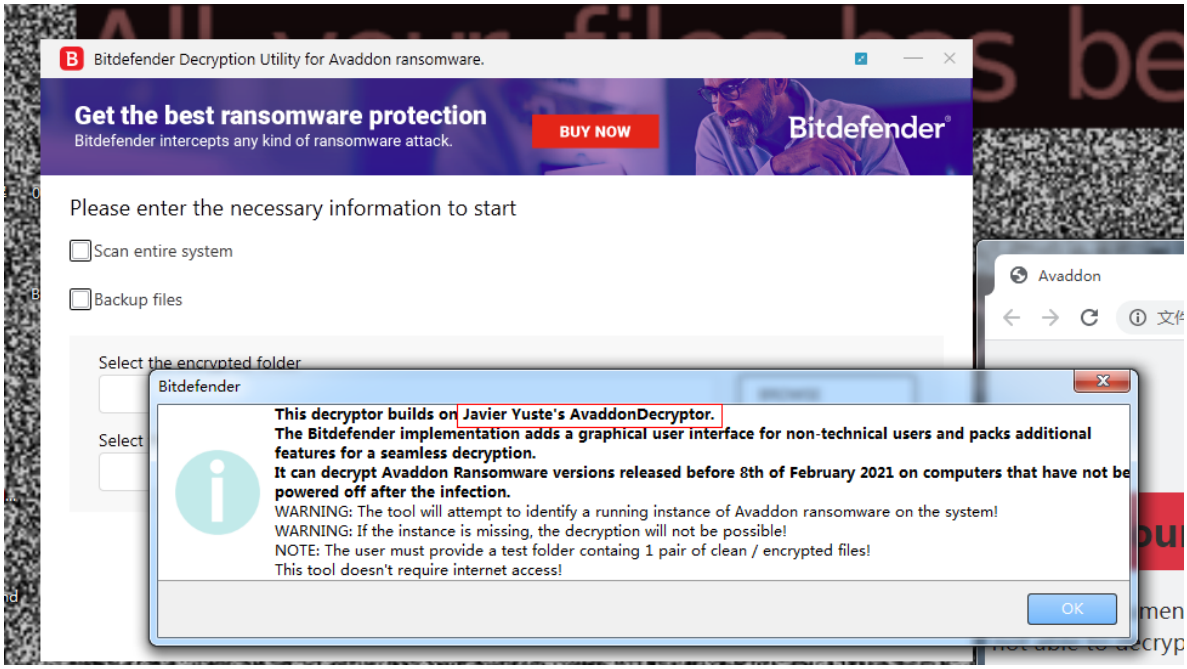
则特征值对应：106600000100000020000000

这个结构体来自于CryptApi，作者是逆向了cryptsp.dll和rsaenh.dll这两个dll得到的这个数据结构。

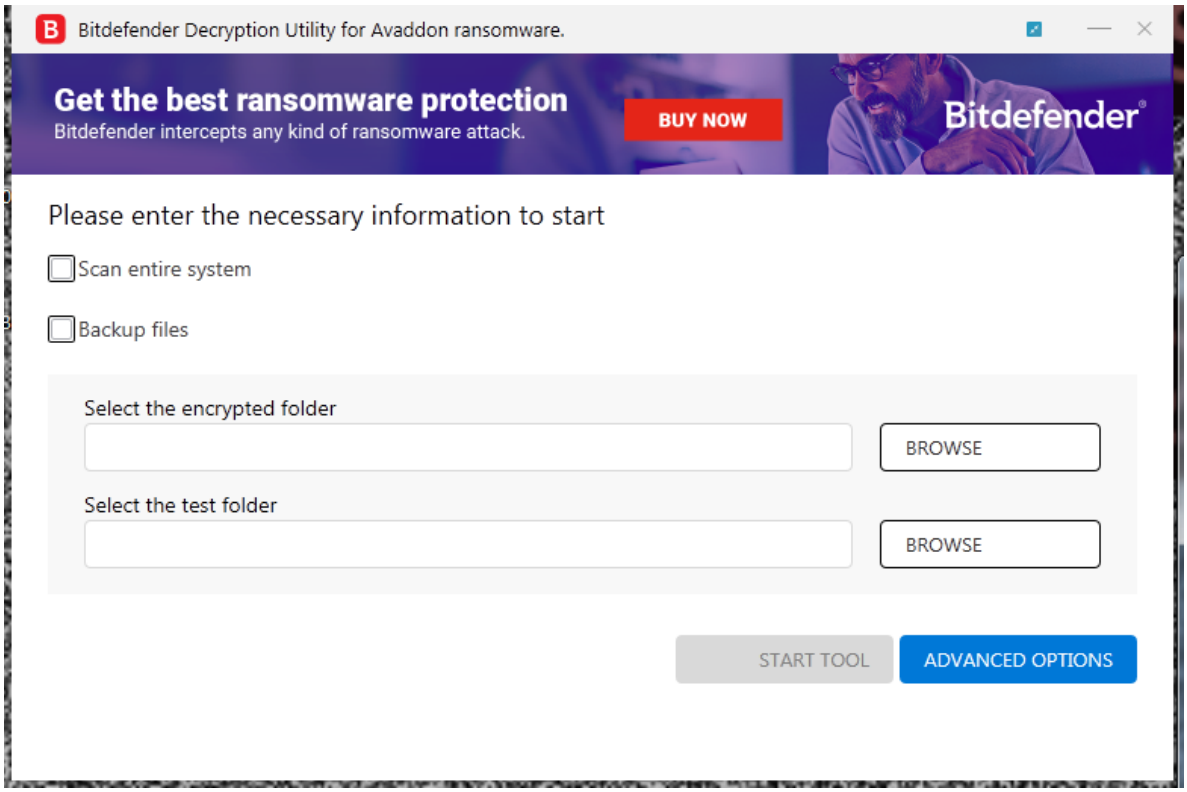
也就是说，这种在内存中暴力搜索密钥去解密被加密文件的方式，只适用于调用了CryptApi，并且随机生成密钥的情况。

那么有没有可能将作者的源码进行优化呢？答案是有

解密工具优化



在这之后，bd也针对该勒索发布了一款解密工具，根据提示，也是基于上面的代码做了一个图形化的工具而已。



工具只需要一个被加密文件和被加密前的源文件，不需要选择进程，不需要dump文件，就能对整个文件夹进行解密，但是我这里测试是解密失败的。分析一下这个工具有没有什么可借鉴的地方

```
if ( Process32FirstW(v4, &pe) )
{
  while ( !getAesKeys_sub_1000A460(v1, pe.th32ProcessID) )
  {
    if ( !Process32NextW(v4, &pe) )
    {
      v6 = 0;
      goto LABEL_26;
    }
  }
```

bd的做法是遍历整个系统的进程

```
{
  v8 = (const void **)v2[20];
  while ( 1 )
  {
    v26 = Buffer.RegionSize + v2[20];
    if ( (signed int)(v26 - (_DWORD)v8) < 12 )
      break;
    v9 = 8;
    v29 = (const void **)(v26 - 12);
    v10 = &v37;
    v11 = v8;
    while ( 1 )
    {
      v12 = *v11;
      if ( *v11 != (const void *)*v10 )
        break;
```

直接在内存里匹配密钥，

```
do
{
  v21 = *v20;
  v22 = (_BYTE *)sub_1000D392(32);
  lpAddress = v22;
  if ( ReadProcessMemory(hObject, v21, v22, 0x20u, &NumberOfBytesRead) && NumberOfBytesRead )
  {
    if ( *v22 || v22[1] || v22[2] )
    {
      v23 = v30 + 0x18;
      v24 = (_DWORD *)v30[25];
      if ( v24 == (_DWORD *)v30[26] )
      {
        sub_1000C3E0(v23, (unsigned int)v24, &lpAddress);
      }
      else
      {
        *v24 = v22;
        v23[1] += 4;
      }
    }
    else
    {
      sub_1000D1A8(v22);
    }
  }
  ++v20;
```

如果匹配完成，就直接读取，然后调用解密程序。

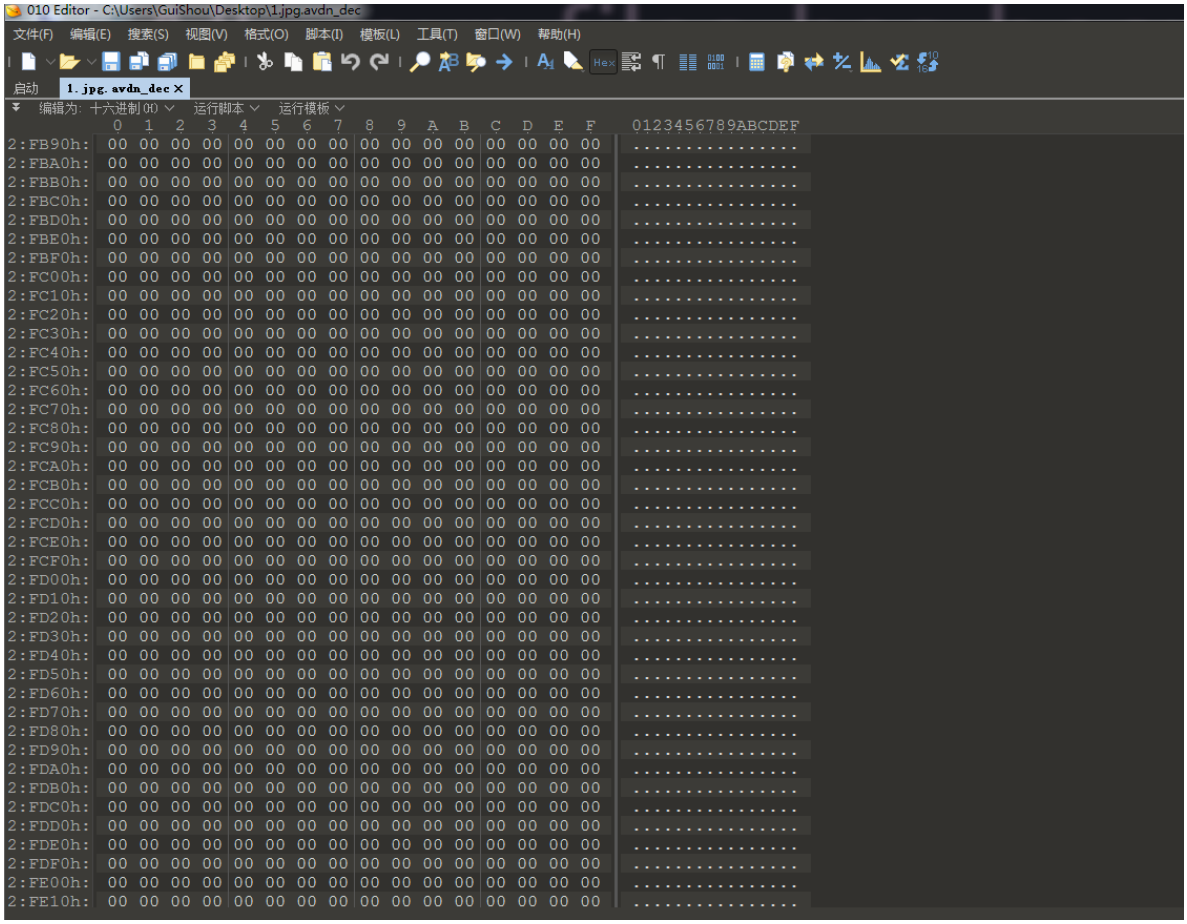
关于文件大小的疑惑

根据网上的分析报告提示

最后附加的24个字节中前4个字节是原始文件大小，但这个大小似乎不太对。如果原始文件大小是0x01EB0B，那么

```
0x01EB0B-512-24=0x1E8F3
```

但上图被加密文件数据的大小是0x20000，这个文件大小这么来看的话是对不上的。直到我即将完成我的解密工具的时候，解密出来的文件在末尾总是会出现一堆0。

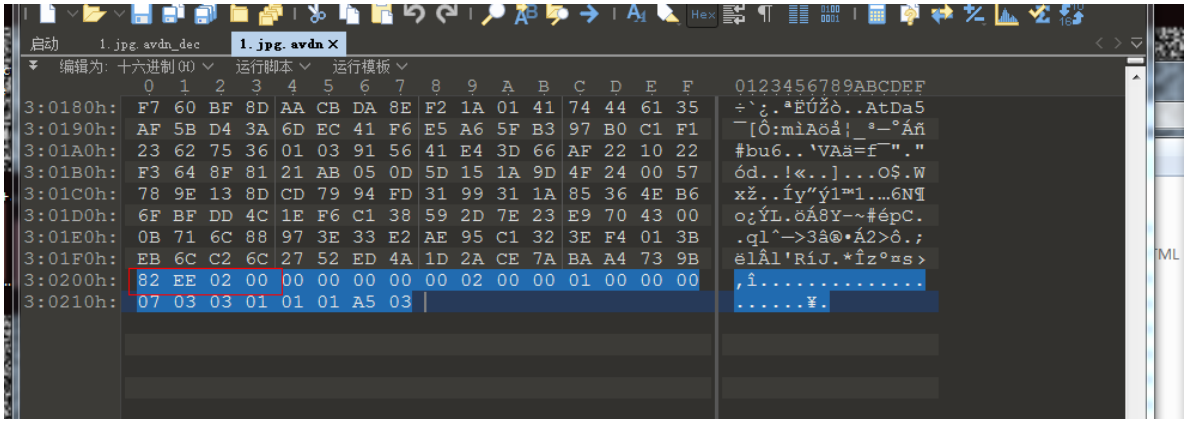


末尾的这一堆0，直接导致了解密后的文件和解密前的文件md5对比失败，一开始我以为是程序逻辑上的bug，但是后来发现并不是。实际上

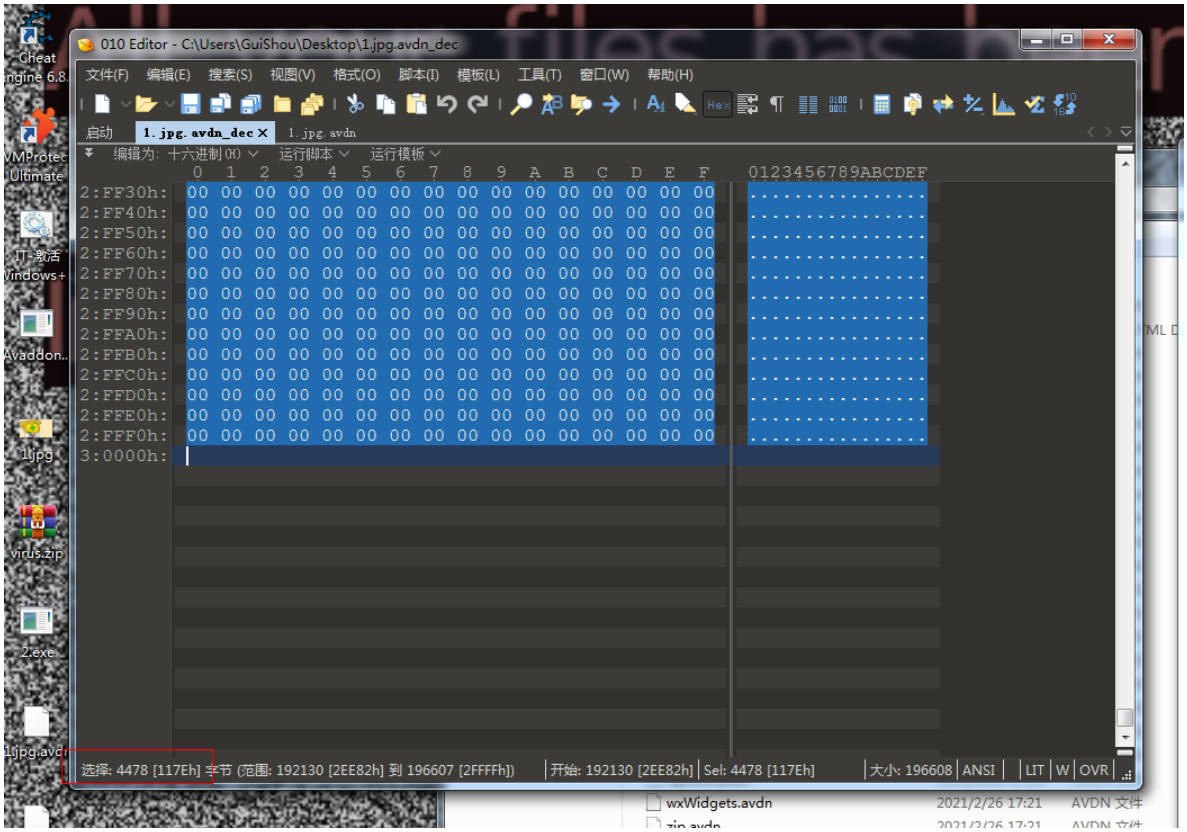
文件的计算方式如下：

> 加密后的文件大小(小于0x100000)=原始文件大小+填充0的字节数+512+24

以下面某个文件为例：



- 原始文件大小为：0x2EE82
- 加密后的文件大小为:0x30218



- 填充0的字节数为4478

```
0x30218=0x2EE82+4478+512+24
```

最后填充的0实际上是为了对齐到0x2000个字节，因为该样本每次会加密0x2000个字节，如果不足2000那么在加密的过程可能会导致异常退出。

Avaddon勒索加密流程补充

解决了文件大小的问题，这里对Avaddon勒索的AES加密流程做一个补充。

对于文件大小小于0x10000的文件，首先会在文件末尾填充0，将大小补齐到0x2000，然后将样本进行加密处理，每次加密0x2000个字节

对于文件大小小于0x10000的文件，不需要进行填充，直接加密前0x10000个字节，0x10000以后的部分不做加密处理

解密工具实现

那么到这里，已经填完了所有的坑，可以做一个相对来说最优化的解决方案。整个解密流程如下：

```
BOOL Check(LPCTSTR lpszFile);
```

首先判断是否是该家族的加密文件，判断条件有三个，两个末尾24字节写死的特征码和文件大小的计算是否满足条件

```
void GetValidPid();
```

首先获取有效进程的PID，遍历整个进程，并且获取进程映像文件的md5，将md5和注册表启动项中的映像文件做对比，如果对比成功，说明可能是潜在的勒索进程

```
ScanAddress(hProcess, (char*)"106600000100000020000000", 0);
```

接着遍历所有的有效进程，搜索特征码，获取到所有可能的Key

```
BOOL GetUniqueKey(LPCTSTR szSourceFile, LPCTSTR szEncryptedFile);
```

用所有可能的Key文件去解密被加密文件，如果解密出来的文件和源文件md5一致，那么视为密钥获取成功

```
void DecryptAllFiles(LPCTSTR FileDirectory);
```

开始解密整个需要解密的目录

```
//勒索解密函数
BOOL Decrypt(LPCWSTR lpszOldFile, LPCWSTR lpszNewFile);
//Aes解密
BOOL AesDecrypt(LPCWSTR lpszOldFile, LPCWSTR lpszNewFile);
//获取原始文件大小
DWORD GetOriginalFileSize(LPCWSTR lpszFile);
//特征码搜索
uintptr_t ScanAddress(HANDLE process, char *markCode, int nOffset, unsigned long dwReadLen = 4, uintptr_t StartAddr = 0, uintptr_t EndAddr
//从偏移中获取有效密钥
void GetValidKeyFromOffset(HANDLE hProcess);
//计算密钥
BOOL CalcKey(LPCTSTR szSourceFile, LPCTSTR szEncryptedFile);
string byteToHexStr(unsigned char byte_arr[], int arr_len);
void HexString2Bytes(const std::string& hex, BYTE* bytes);
//计算文件的md5
string CalcMd5(LPCTSTR lpFilePath);
//获取唯一密钥
BOOL GetUniqueKey(LPCTSTR szSourceFile, LPCTSTR szEncryptedFile);
//获取启动项md5
void GetBootInfoMd5(vector<string>& md5arr);
//获取有效的进程ID
void GetValidPid();
//判断是否为被加密文件
BOOL Check(LPCTSTR lpszFile);
//提升为Debug权限
BOOL EnableDebugPrivilege();
//挂起进程
void SuspendProcess(DWORD dwPid,BOOL isSuspend);
//重新设置文件大小
void ResetFileSize(LPCWSTR lpszFile, LONG lDistanceToMove, DWORD dwMoveMethod);
//解密所有文件
void DecryptAllFiles(LPCTSTR FileDirectory);
```

整个解密工具写了快1100行代码，花了6天左右。工程这里就不发了，记录一下整个过程和一些踩过的坑。

相关资料

勒索分析：https://www.freebuf.com/articles/others-articles/249109.html

解密工具: https://github.com/JavierYuste/AvaddonDecryptor

勒索解密工具分析：https://mp.weixin.qq.com/s?__biz=MzA4ODEyODA3MQ==&mid=2247486514&idx=1&sn=6464b9066980a6c045a33ef58dd5b6b0&chksm=902fa31aa7582a0c88242eb816fb466cbb7f94fe33b3e8b14f1f769d6d493961eaf9721cc87d#rd