WanaCryptor 파일 암복호화 분석

출처 : https://modexp.wordpress.com/2017/05/15/wanacryptor/(번역: @BlackFalcon)

서론

이번 주말 전 세계의 많은 네트워크를 파괴한 Wanna Crypt 랜섬웨어에 대한 빠른 포스트입니다. 모든 뉴스 보도를 통해, 대부분의 사람들은 이미 그것으로 인한 문제를 알고 있습니다.

시스템에서 실행되면, RSA 및 AES 암호화 알고리즘을 사용하여 파일을 복구하는 데 필요한 키와 교환하여 비용을 요구하기 전 파일을 암호화합니다. RSA 암호 시스템을 이해하려면 먼저 공개키(Public Key)와 개인키(Private Key)에 대한 모든 논의가 일부 독자에게는 혼란스러울 수 있으므로 위키피디아의 RSA를 읽어보십시오.

WanaFork

이 게시물과 함께 제공된 소스코드는 주로 암호화 및 암호 해독 프로세스를 이해하고자 하는 보안연구원을 대상으로 하며, 이는 랜섬웨어 제작자가 개인키를 공개하기로 결정한 이벤트에서 파일을 복구하는 데 도움이 될 수 있습니다.

여기 암호화/복호화 과정을 제외하고는 랜섬웨어의 동작에 대한 논의가 없으므로 다른 것에 대한 정보가 필요하면 Freenode IRC 서버의 #wannadecryptor 채널에 있는 다양한 보안연구원이 컴파일 한 이 파일(https://gist.github.com/rain-1/989428fa5504f378b993ee6efbc0b168)을 살펴보십시오.

wanafork에 대한 소스 코드는 C의 소스를 참조하십시오. MSVC로만 컴파일되었고, Windows에서 테스트되었습니다. 비록 MINGW가 최신 버전이라면 괜찮습니다.

암호화 프로세스

각 시스템은 Microsoft Crypto AP (CAPI)의 일부인 **CryptGenKey** API를 사용하여 2048비트의 RSA 키 쌍을 생성합니다.

공개키는 **CryptExportKey**를 사용하여 **00000000.pky**에 저장됩니다. 개인키는 **CryptExportKey**를 사용하여 **00000000.eky**에 저장되지만 디스크의 파일 암호화를 담당하는 DLL 내부에 포함 된 마스터 공개키로 **CryptEncrypt** API를 사용하여 저장 전에 암호화됩니다.

이 개인키의 암호화는 랜섬웨어의 작성자의 도움없이 파일을 복구하지 못하게 합니다.

암호화된 각 파일에 대해 **CryptGenRandom** API는 데이터를 암호화하기 위해 CBC 모드에서 AES-128과 함께 사용되는 16 바이트 값을 파생시키는 데 사용됩니다.

AES키는 사용자 공개 키로 암호화되어 AES 암호문과 함께 저장됩니다.

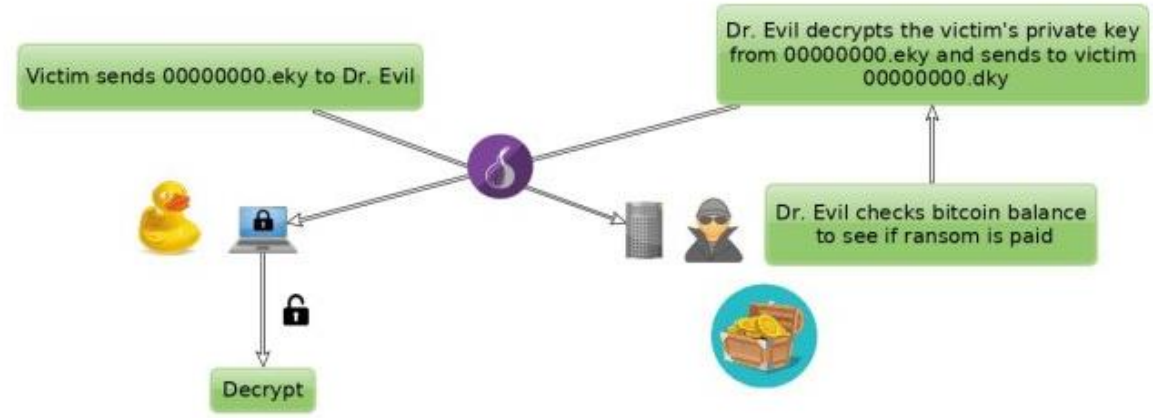
이 AES 키와 암호화 된 파일의 내용을 복구하는 유일한 방법은 개인키를 사용하는 암호 해독을 사용하는 것이며 이 작업을 수행하려면 마스터 개인키가 필요합니다.

몸값 지불 프로세스

필자가 전혀 조사하지는 않았지만, 암호화 모델을 기반으로 몇 가지 가정을 하는 것이 합리적입니다. 여기서 랜섬웨어의 일부 구성 요소는 00000000.eky에 저장된 암호화된 개인키를 TOR 네트워크를 통해 서버로 보냅니다. 랜섬웨어 작성자는 마스터 개인키를 사용합니다.

그런 다음 해독 된 개인키가 피해자 시스템으로 보내지고 00000000.dky로 저장되어 @WanaDecryptor@.exe가 파일을 복구할 수 있게 합니다.

물론 수정을 위해 열립니다. 랜섬웨어 제작자가 누가 지불 하는지를 식별 할 수 있는 방법이 없다는 지적이 있었습니다.



사용된 알고리즘 때문에 랜섬웨어 작성자의 도움없이 암호화된 파일에서 데이터를 복구 할 수 없습니다.

나머지 게시물은 개발자/연구원에게만 관심이 있을 수 있습니다.

정의(Definitions)

여기에 표시된 소스 코드에서 이 값을 볼 수 있습니다.

```
#define WC_RANSOM_KEY      "rw_public.bin" // public key blob belonging to authors

#define WC_PUBLIC_KEY      "00000000.pky"  // public key to encrypt AES keys
#define WC_PRIVATE_KEY     "00000000.dky"  // private key to decrypt AES keys
#define WC_ENCRYPTED_KEY   "00000000.eky"  // encrypted private key sent to remote server

#define WC_BUF_SIZE        1048576         // writes/reads data in 1MB chunks

#define WC_ARCHIVE_EXT     ".WNCRY"        // encrypted archive extension
#define WC_SIGNATURE       "WANACRY!"      // signature for encrypted archives
#define WC_SIG_LEN         8               // 64-bit length
#define WC_ENCKEY_LEN    256               // 2048-bit
#define WC_RSA_KEY_LEN  2048               // 2048-bit
#define WC_AES_KEY_LEN    16               // 128-bit
#define WC_DATA_OFFSET  WC_SIG_LEN + WC_ENCKEY_LEN + 4 + 4
```

WanaCryptor 아카이브 구조

암호화된 각각의 파일 또는 내가 호출 한 파일은 성공적인 복호화에 필요한 미리 정의된 구조를 가집니다.

```
// structure of wanacrypt archive
typedef struct _wc_file_t {
    char     sig[WC_SIG_LEN];     // 64-bit signature
    uint32_t keylen;              // length of encrypted key
    uint8_t  key[WC_ENCKEY_LEN]; // AES key encrypted with RSA
    uint32_t unknown;             // usually 4, unsure what exactly for
    uint64_t datalen;             // length of file before encryption
    uint8_t  data;                // AES-128 ciphertext
} wc_file_t;
```

- Signature
  64비트 시그니처. 현재 "WANACRY!"문자열로 설정됩니다.

- 키 길이
  다음 암호화된 AES 키의 길이(바이트)를 나타냅니다.

- 암호화된 AES 키
  00000000.pky에 저장된 사용자 RSA 공개키를 사용하여 암호화 된 128비트 AES키
  이 키는 CryptGenRandom에 의해 생성됩니다.

- 알려지지 않은 32비트 값
  아직 무엇인지 확신 할 수 없습니다. 일반적으로 3 또는 4로 설정됩니다

- 파일 길이
  파일의 원래 크기를 나타내는 GetFileSizeEx에서 얻은 64비트 값입니다.

- 암호문
  CBC 모드에서 AES-128을 사용하여 암호화된 데이터. Zero Padding을 사용합니다.

다음은 멀웨어에 의해 생성된 암호화 된 파일의 구조화 된 hex 덤프입니다.

```
  [ signature

0000    57 41 4e 41 43 52 59 21                          WANACRY!

  [ encrypted key

0000    59 64 5c 27 77 76 80 fc  c8 3e e1 b7 86 06 d3 ee   Yd\'wv...>......
0010    8d a9 17 c1 88 ce be f6  80 c3 f6 75 5a c6 3d 8e   ...........uZ.=.
0020    13 04 65 0f e7 3c 8e 0f  60 5d ea b5 94 2a c0 39   ..e..<..`]...*.9
0030    94 15 b5 be bb c4 60 79  9b 08 28 33 a9 ff b2 7f   ......`y..(3....
0040    02 22 f7 a0 2e 57 93 8c  89 14 be 9f 66 7f b1 98   ."...W......f...
0050    e4 d1 ec 8a 14 b1 43 f7  24 88 0e 27 70 bd 6d d9   ......C.$..'p.m.
0060    bc e9 ca 6a d3 df 1b 58  cf 64 f8 a4 03 7a 14 c4   ...j...X.d...z..
0070    0d b8 0f 94 d5 ee 7e bb  f9 b3 ce 55 49 85 4a ef   ......~....UI.J.
0080    af 62 a0 49 18 ba e2 86  f4 b2 98 17 45 45 b3 4e   .b.I........EE.N
0090    46 d8 bc a8 dd 8f 27 2c  fc 15 ab ad 28 56 fc 4b   F.....',....(V.K
00a0    29 cc f5 3c ee f0 b1 fb  93 27 c9 a9 ab 7d 7a a4   )..<.....'...}z.
00b0    ac 74 77 ef 0f 87 ee f2  59 10 6f 52 ef d1 cd e6   .tw.....Y.oR....
00c0    b1 ca 7c ef 54 52 f7 38  2f a5 30 e9 81 ae c7 ea   ..|.TR.8/.0.....
00d0    ce 41 a5 fc fa 2d fd 1c  9e c4 9b 34 25 33 32 b2   .A...-.....4%32.
00e0    e8 72 99 d5 17 5e f2 87  f7 aa 63 59 14 66 ad f0   .r...^....cY.f..
00f0    1d 0e d9 19 3c 46 76 22  d8 22 f7 05 e5 fe d6 31   ....<Fv".".....1

  [ unknown

0000    04 00 00 00                                      ....

  [ original file length

0000    12 00 00 00 00 00 00 00                          ........

  [ ciphertext

0000    37 66 60 4f d2 d7 8f 7b  87 95 41 26 0e 77 9a 95   7f`O...{..A&.w..
0010    f8 ac 05 db f8 2f b2 1f  42 68 99 bb 83 e0 79 f6   ...../..Bh....y.
```

RSA 키 생성

마찬가지로 생성된 RSA키는 각 시스템마다 고유합니다.

이 툴에서 공개키+개인키는 모두 평문으로 내보내집니다. 개인키는 또한 멀웨어가 이를 어떻게 하는지 설명하기 위해 암호화됩니다.

```
printf ("\n  [ generating RSA key pair");

// acquire a crypto provider context
if (CryptAcquireContext(&prov,
      NULL, MS_ENH_RSA_AES_PROV, PROV_RSA_AES,
      CRYPT_VERIFYCONTEXT));
{
  printf ("\n  [ acquired crypto provider");

  // generate 2048-bit RSA key pair
  if (CryptGenKey(prov, AT_KEYEXCHANGE,
    (WC_RSA_KEY_LEN << 16) | CRYPT_EXPORTABLE, &key))
  {
    // export the public key as blob
    printf ("\n  [ exporting public key");
    export_rsa_key(prov, key, "00000000.pky", PUBLICKEYBLOB,  FALSE);

    // export the private key as blob
    printf ("\n  [ exporting private key");
    export_rsa_key(prov, key, "00000000.dky", PRIVATEKEYBLOB, FALSE);

    // export the private key (encrypted with ransomware public key)
    printf ("\n  [ exporting private key (encrypted)");
    export_rsa_key(prov, key, "00000000.eky", PRIVATEKEYBLOB, TRUE);

    CryptDestroyKey(key);
  }
  CryptReleaseContext(prov, 0);
}
```

AES 키 생성

각 파일의 AES키는 CryptGenRandom API를 사용하여 생성됩니다. 이 API는 암호로 보호되어 공격에 취약하지 않습니다.

```
// generate new one
CryptGenRandom(prov, WC_AES_KEY_LEN, key.key);

// encrypt using public key
memcpy (aes_key->enc, key.key, WC_AES_KEY_LEN);

len = WC_AES_KEY_LEN;

if (!CryptEncrypt(rsa_key, 0, TRUE,
          0, aes_key->enc, &len, WC_ENCKEY_LEN)) {
  xstrerror ("CryptEncrypt");
  return 0;
}
```

아카이브를 해독할 때 00000000.dky에 저장된 RSA 개인키 BLOB를 사용하여 암호화된 AES 키를 해독해야합니다.

```
printf ("\n  [ importing AES key from archive");
// open archive
in = fopen(file, "rb");

if (in != NULL) {
  printf ("\n  [ skipping signature");

  // skip signature + enclen
  fseek(in, WC_SIG_LEN + sizeof(uint32_t), SEEK_SET);

  // read encrypted key
  key.len = fread(aes_key->enc, 1, WC_ENCKEY_LEN, in);

  // decrypt AES key using private key
  if (!CryptDecrypt(rsa_key, 0, TRUE, 0,
      aes_key->enc, &key.len))
  {
    xstrerror ("AES Key Decryption with CryptDecrypt");
    return 0;
  }
  // copy decrypted AES key to key header for importing
  memcpy (key.key, aes_key->enc, WC_AES_KEY_LEN);
  fclose(in);
}
```

두 시나리오 모두에서 AES용 Crypto API를 사용하기 때문에 키를 CAPI 키 객체로 가져오는 추가 단계가 필요합니다.

```
// set key parameters
key.hdr.bType     = PLAINTEXTKEYBLOB;
key.hdr.bVersion = CUR_BLOB_VERSION;
key.hdr.reserved = 0;
key.hdr.aiKeyAlg = CALG_AES_128;
key.len           = WC_AES_KEY_LEN;

// import it
if (CryptImportKey (prov, (PBYTE)&key, sizeof(key),
      0, CRYPT_NO_SALT, &aes_key->key))
{
  printf ("\n  [ key imported ok");

  mode = CRYPT_MODE_CBC;

  // set to use CBC mode
  CryptSetKeyParam(aes_key->key, KP_MODE, (PBYTE)&mode, 0);
}
```

AES를 사용하는 것이 Crypto API보다 사용하기 쉽다는 것을 추측할 수는 있지만, 내 생각에 불과합니다.

암호화

WanaCryptor는 Zero Padding을 사용하지만 Crypto API는 Zero Padding을 지원하지 않습니다.

마지막 블록을 암호화 할 때 bFinalize flag를 TRUE로 설정하는 대신 버퍼를 16바이트로 정렬하고, null 바이트로 채웁니다. bFinalize 플래그는 WanaCryptor와 호환되도록 FALSE로 유지되어야 합니다.

다음은 WanaCryptor 아카이브를 생성하고 Crypto API를 사용하여 파일 데이터의 AES-128-CBC 암호화를 수행하는 예제입니다.

```
// write the signature
fwrite(WC_SIGNATURE, 1, WC_SIG_LEN, out);

// write the encrypted key length
t = WC_ENCKEY_LEN;
fwrite(&t, 1, sizeof(t), out);

// write the encrypted AES key
fwrite(aes_key->enc, 1, WC_ENCKEY_LEN, out);

// write the unknown value
t = 4;
fwrite(&t, 1, sizeof(t), out);

// obtain 64-bit file size
_stat64(infile, &st);
// write size to file
fwrite(&st.st_size, 1, sizeof(uint64_t), out);

// encrypt data and write to archive
for (;;) {
  // read in 1MB chunks
  len = fread(buf, 1, WC_BUF_SIZE, in);

  // no more data?
  if (len==0) break;

  // encrypt block
  CryptEncrypt(aes_key->key, 0, len<WC_BUF_SIZE,
      0, buf, &len, WC_BUF_SIZE);

  // write to file
  fwrite(buf, 1, len, out);
}
```

복호화

암호화와 마찬가지로 bFinalize flag(Crypto API를 사용하는 경우)는 항상 FALSE로 해야합니다. 이는 Microsoft Crypto CSPS가 zero padding을 지원하지 않기 때문입니다.

```
for (total=0;; total += len) {
  // read in 1MB chunks
  len = fread(buf, 1, WC_BUF_SIZE, in);

  // no more data?
  if (len==0) break;

  // decrypt block but don't finalize
  if (!CryptDecrypt(aes_key->key, 0, FALSE,
      0, buf, &len))
  {
    xstrerror("Decryption: CryptDecrypt");
    break;
  }

  // last block?
  if (len < WC_BUF_SIZE) {
    len = dataLen - total;
  }

  // write to file
  fwrite(buf, 1, len, out);
}
```

요약

저자가 생성한 마스터 개인키가 없으면 기적적으로 누군가 AES 또는 RSA로 결함을 발견하지 않는 한 암호화된 파일에서 데이터를 복구할 수 없습니다.

주말 동안 이 악성 코드를 연구하기 위해 freenode의 게시물 및 모든 사람들을 돕는 0x4d_에 감사드립니다.

이 툴은 신뢰할 수 있는 것으로 간주되기 전에 더 많은 테스트가 필요하며 다른 사용자가 소스를 연구하고 자체적인 복호화 도구를 작성할 수 있도록 하기 위해 초기에 배포해야 하는 유일한 이유입니다.