COMPUTER VIROLOGY: SELF-DEFENSE SYSTEM

Hamid M. A. Abdul-Hussain & Hamed Mizher Shabib
Computer Engineering Dept.
College of Engineering, University of Baghdad

ABSTRACT

Based on the biological human models in defending human body against viruses, a new approach in designing the anti-virus system is introduced. This approach is called SDS(Self-Defence System). The principle of the SDS is that each executable program is responsible of defending itself against viral-attacks. In this system, each executable program is injected with basic anti-virus component which is called Self-Defence Routine. This routine, together with dedicated anti-virus loading program are used to construct the SDS which protects the computer system from virus invasion.

INTRODUCTION

Today, the computer virus pandemic becomes a serious security threat to causal home computers and large corporate networks. Over the years, the anti-virus industry has had to keep pace, as virus writers have become more sophisticated. Therefore, the effort to combat the computer virus must continue until an ideal, universal anti-virus system is designed. Researchers have taken many approaches, and some of the newest and most promising anti-virus technology is modeled on the way the human body fights viruses [1].

Based on the similarities between human and computer viruses (both types of viruses latch onto a host, use its resources to reproduce, and cause a range of symptoms). The objective of this work is to build computer immune system. The proposed system is called Self-Defence System (SDS). The SDS mimics the characteristics of human immune system by distributing the anti-virus components through the computer in the same way the human immune system distributes the anti-bodies. The concept of the SDS is to vaccinate each executable program with special anti-virus component that will detect and eradicate any foreign code attached to the executable. The vaccinated executable is loaded and executed by special centralized anti-virus program which is designed to prevent virus infection and damage to the computer system.

In part this research has been a follow-up on the two papers; the first titled “Computer Virology: Formal Analysis of Computer Viruses” [7], and the second titled “Computer Virology: Toward Designing an Ideal Anti-virus System” [4]. Some terms, classifications, and concepts presented in this research are thoroughly explained in the above papers. For example the efficiency of the SDS is determined according to the analysis criteria described in the second paper mentioned above.

SELF-DEFENCE CONCEPT

The self-defense concept is that: **“Every executable program must be capable of protecting itself against viral attacks by detecting and eradicating any virus attached to its body”.** Accordingly, special part of each executable must be reserved for the self-defence task. This part is called **“Self-Defence Routine (SDR)” and** defined as: **“The part of the**

executable that is specifically designed to protect the executable against viral attacks". SDR is the first basic anti-virus component in SDS, which is embedded, as integral part of an executable E. E will be called the **"protected executable"** along this research. The second basic anti-virus component in SDS is a special OS program called "Load and Execute Program (LEP)". LEP is used to load executables into memory and execute their SDRs secretly by using a special secret communication protocols. LEP and SDR cooperate to satisfy the requirements of the IAVS.

The communication between LEP and SDR consist of two phases (PHASE-ONE and PHASE-TWO). During PHASE-ONE, LEP must transfer control to SDR without giving the attached virus "if any" the opportunity to execute. If SDR detect a virus infection, it will try to eradicate the attached virus and repair the protected executable using the Foreign Block Eradication Algorithm (FBEA). If FBEA failed, SDR must alert LEP to switch to PHASE-TWO. In PHASE-TWO, LEP will execute the virus in a virtual computer system. After the virus execution is completed, SDR will try to eradicate the virus and repair the infected executable using Virus Follower Eradication Algorithm (VFEA). The following sections will discuss how SDR and LEP are designed, and the communication protocols they use in PHASE-ONE and PHASE-TWO.

SECRET ENTRY POINT

Most viruses tacks themselves into an executable program and ensure that they will execute before their host, therefore, they must redirect the executable standard entry point to point to the virus entry point. The standard entry point is the location to which the OS transfers control when deciding to execute the executable. The standard entry point is considered the most vulnerable spot for viral attacks.

In PHASE-ONE, LEP must execute SDR without giving the opportunity to any attached virus "if any" to execute. Clearly, this cannot be done by transferring control to the standard entry point of the executable because viruses know where the standard entry point of the executable is, and they always redirect it to ensure that they will get the opportunity to execute before the infected executable. There is only one way to ensure that SDR will get the opportunity to execute before any attached virus. That is, by using a nonstandard or secret entry point. This kind of entry point is called trapdoor: "**The Trapdoor is a secret entry point to the executable [2]".** Fig.1 shows how the trapdoor can be used.

In fact, the secret entry point will be used to solve the standard entry point problem. Therefore, the trapdoor location must vary from one executable to the other. For example, while the trapdoor of the executable 'E1' is located at offset 'X1', the trapdoor of the executable 'E2' must be located at offset 'X2', where '$X1 \neq X2$'. Because the same standard entry point problem will be faced again, if the trapdoor location is standardized for all executables. For example, if the trapdoor location is standardized to be at offset 'X1' for all executables, then: **"The virus will know that there is another entry point to the executable located at offset 'X1'. Consequently, it will redirect to this entry point to ensure that it will get the opportunity to execute before its host".** Therefore, a new technique that will enable LEP to find or mark the trapdoor while, at the same time, viruses cannot find the trapdoor. The following sections describe how LEP mark the trapdoor, the standard SDR module format, and how different LEPs can exchange protected executables between each other.

TRAPDOOR MARKING

LEP can find the trapdoor location of a given executable by using special type of information stored in the executable itself. This special type of information will be called the "**Trapdoor Mark"** and defined as "**A special code or string of characters whose location (LOC) or value (TMV "Trapdoor Mark Value") can be used to find the trapdoor".**

LEP must calculate the TMV and store it into LOC using criteria known by LEP only. This criterion differs from one LEP to the other. Therefore, viruses cannot estimate in advance which trapdoor mark is used by a randomly selected LEP. For example, if LEP1 and LEP2 are used in two different computers PC1 and PC2 respectively. LEP1 define (TMV= "HMS"), and define the first byte position which follow the 3'rd character as the trapdoor (Trapdoor=LOC+3), while LEP2 define (TMV=666) and (Trapdoor=LOC+2). Since the virus cannot estimate in advance in which computer it exist, in PC1, PC2, or some other computer, it cannot estimate in advance whether the TMV is, "HMS", 666, or any other number or string. The efficiency and reliability of the secret entry point depend on the TMV type and the method used to generate it. In general, two TMV types can be defined:

FIXED TMV

In this type, the same TMV used for all executables in the system. The generation and searching formulas are very simple in this case. There is, however, a vulnerable spot in this case. Since the TMV is used in all executables, it will be the common code between all executable. Viruses can determine the fixed TMV by comparing two or more executables to see which code is common between them. This code might be the TMV. Note that if each executable in the system uses the suggested self-defence technique efficiently, no virus will get the opportunity to execute and therefore it cannot perform the comparison described above.

Variable TMV

In this type, a different TMV used for each executable in the system. Only LEP knows how to generate and find this TMV for a given executable. Two general techniques can be used to generate a variable TMV:

A- Value-Based Technique: "In this technique, TMV is calculated using a predefined formula and stored in LOC". The general Value-Based generation formula can be defined as follow: **"TMV= f(g(EBV), LBC)".** Where:

- **EBV "Executable-Based Variable":** This value is constant with respect to a given executable. But it differ from one executable to the other. Therefore, LEP can ensure that TMV will change from one executable to the other.
- **LBC "Loader-Based Constant":** This value is constant with respect to a given LEP. But differ from one LEP to the other. This ensures that the TMV will change from one LEP to the other even if both use the same EBV, g, and f.
- **f and g :** Functions (arithmetic or logical relation).

In general, LEP must search for the trapdoor mark location (LOC) using TMV. The general Value-Based searching formula can be defined as follow: **"IF ([P]=TMV) THEN (LOC=P)".** Where:

- **P= Address Pointer**
- **[P]= Content of the location at address P.**

For example, the executable file name can be used as EBV, because it is constant for a given executable file. The TMV generation formula will be: **"TMV= f(g(Executable Name),LBC)".** Let us define the following, for LEP1 and LEP2:

Component	LEP1-Definition	LEP2-Definition
g	ASCII of the 1st Character + ASCII of the 2nd Character + ASCII of the 3rd Character	ASCII of the 2nd character - ASCII of the 4th character
LBC	6	305
f	g + LBC	g ⊕ LBC

Tab.1 shows the TMV generated for three different file names by using the definitions shown above. LEP1 can find LOC for HAMED.COM using the following algorithm:

```
  Calculate :TMV=72+65+77+6=220
  P=0
  IF ([P]=220) THEN (LOC=P) ; GO TO 6
  P=P+1
IF (P<Size of HAMED.COM) THEN 3
```

End.

B- Condition-Based Technique: "In this technique, TMV is selected so that a predefined condition satisfied". The general Condition-Based generation formula can be defined as follow: **"Select TMV To Satisfy: f(LOC,LBC1)=g(LOC,LBC2)".**

The general Condition-Based searching formula can be defined as follow: **"IF (f(P,LBC1)=g(P,LBC2)) THEN (LOC=P)".** For example, the following can be defined:

LBC1	f(P,LBC1)	LBC2	g(P,LBC2)	Condition
0	[P]+LBC1	0	P+LBC2	[P]=P
1	[P]+LBC1	2	P*LBC2	[P]+1=P*2
19	[P]-LBC1	0	P⊕[P-1]+LBC2	[P]-19=P⊕[P-1]

Tab.2 shows the TMVs generated to satisfy the condition "[P]=P+3" for the three executables shown in Tab.1. The following algorithm can be used by LEP to search for LOC, assuming that f, g, and LBC are defined as shown above:

```
 P=0
IF ([P]=P+3) THEN (LOC=P) ;  GO TO 5
P=P+1
 IF (P<Size of HAMED.COM) THEN 2
End.
```

However, it is very important to test the influence of virus infection on the selected condition. For example, if the condition [LOC]=LOC+3 used, and a virus inserts 256-bytes in the start of HAMED.COM and, hence, shift the contents of HAMED.COM by 256-byte. In this case the "TMV=503" will be shifted from location 500 to location 756. Clearly, the condition will be not satisfied in location 500 and location 756. Conditions that link the content of LOC with the contents of its predecessor or successor locations are not affected by the virus infection. For example, the condition:[LOC]=[LOC-1]+[LOC+1].

SDR Module Standard Format

The SDR module consists of the SDR code/data and trapdoor marking information. Two problems must be considered before defining the standard format of the SDR module:

Pseudo Trapdoor Mark: "The pseudo trapdoor mark is any value satisfy the searching formula and cause LEP to transfer control to an incorrect location". The increased probability of pseudo trapdoor mark in the value-based technique is the great disadvantage of this technique. For example, in Tab.1 the TMV associated with HAMED.COM is 220 with respect to LEP1. Clearly, this value may exist more than one time within the code or data of HAMED.COM. Moreover, viruses may consider this as a vulnerable spot for attack. For example, TMV will range from 0 to 255 if represented as one byte storage location. The virus can insert a table that contains all of the expected values (from 0 to 255) in the start of the infected executable. When LEP search the infected executable, it will find a pseudo trapdoor mark in the virus table and transfer control accordingly. This transfer will activate the virus to start its execution.

The probability of finding a pseudo trapdoor mark in the condition-based technique is less relative to the value-based technique. Because it searches for a **value that satisfy a condition.** For example, in Tab.2, even though the value 503 may exist many times within the code or data of HAMED.COM, it will be considered as a trapdoor mark only if it exists at offset address 500.

In general, the solution to the pseudo trapdoor mark problem is to use two TMVs associated with special integrity check information. Tab.3 shows the suggested standard format of the SDR module. According to Tab.3, LEP can find the SDR module as follows:

1. **P=0**

```
2.  IF (f11(P,LBC11)=g11(P,LBC12)) THEN 6
3.  P=P+1
4.  IF P<Executable_Size THEN go to 2
;Trapdoor not found,
5.  Display Alarm Message "Cannot Find the Trapdoor".
;Test if the second condition satisfied
6.  IF (f12(P,LBC13)=g12(P,LBC13)) THEN go to 8
;TMV2 Not exist, continue the search
7.  Go To 3
;Perform checksum test
8.  LEN= $f13^{-1}$([P+08H])
9.  Using CHKS-ALG1, Set C= Checksum of the block
    (P+10H+LEN)
10. IF C=[P+0CH] THEN go to 12
;Checksum Error, Continue The search
11. Go To 3
;Trapdoor Mark Found
12. LOC1=P
```

Destruction: Destruction can be caused by a virus infection. If the virus, for example, inserts part or all of its code between TMV11 and TMV12, between the header and the SDR module, or inside the SDR module. The solution to this problem is to use redundancy. That is, two SDR modules must be used so that if one of them destroyed the other one can be used. The first SDR module (shown in Tab.3) will be called the **"Primary SDR Module"** and the second SDR module will be called the **"Redundant SDR Module"**. The redundant SDR module is shown in Tab.4.

LEP-To-LEP Communication

As mentioned earlier, each LEP uses its own standard to generate and search for TMV and LOC. Users may ask what happens when the protected executable 'E' copied from the environment of LEP1 to the environment of LEP2. That is: **"How LEP2 can find LOC1 and LOC2 of 'E', without giving viruses the opportunity to find them".** Clearly, there must be a standard and secure communication protocol between LEP1 and LEP2. There are many ways to do this.

First, the communication can be done by using a global TMV, say "HMS". When the user asks LEP1 to copy 'E' to be used by LEP2, LEP1 must store "HMS" at LOC1 and LOC2. When the user asks LEP2 to execute 'E' for the first time, it will search 'E' for the global TMV (i.e. HMS) to find LOC1 and LOC2. Once "HMS" is found LEP2 will recalculate LOC1, LOC2 and reorganize the SDR locations according to its formula. The advantage of this method is its transparency for users. The great disadvantage of this method is its vulnerability. Viruses know that the global TMV used in copies is "HMS", therefore, they can easily find LOC1 and LOC2 of the protected executable.

Second, the communication can be done through the computer user. The user can ask LEP1 about the value of LOC1 and LOC2 of 'E' after copying the executable to a new diskette. The user must give the value of LOC1 and LOC2 to LEP2 (i.e. manually) when he wants to execute 'E' for the first time. The advantage of this method is that it is secure, because there is no way that a virus will know the value of LOC1 and LOC2. The disadvantage of this method is that it depends on the computer user responsibility. A serious problem arises when the user decides to copy a large number of executables. For example, if the user wants to copy 100-executable from DISK1 to DISK2, then, he must ask LEP1 about the value of LOC1 and LOC2 for each one of these executable, keep these (200-Value) values in his mind, and give them to LEP2.

Third, the best suitable method is to use Message-Files. When the user asks LEP1 to copy the 100-executable (E1, E2,… E100) from DISK1 to DISK2, LEP1 will ask the user about the message file name to store the values of LOC1 and LOC2 in it. Assuming that the user uses the name MESSF.LOC. LEP will create a message file with the name MESSF.LOC in DISK2, and insert 100-entry in this file. Each entry represent one executable. The standard entry format is shown in Tab.5. LEP2 can use the message file to find LOC1 and LOC2 of each executable individually or all executables at once. Regardless of the number of executables being copied, the user needs to remember only the message file name.

-The Communication Standards

Before describing the communication protocols used in PHASE-ONE and PHASE-TWO; and the operations of LEP and SDR in each phase, the standard data blocks, variables, and flags used in the system must be defined and described. Fig.2 shows an overview of the communication system and its individual components. Each data block, variable, and flag used for specific purpose. The following sections will describe these components and explain the purpose of using them.

- Image Information Block (IIB)

In PHASE-ONE, LEP must prepare the Image Information Block (IIB) before transferring control to the SDR. The IIB describes the status of the disk and memory images of E. The standard format of the IIB is shown in Tab.6.

MIS=DIS for COM and SYS files. But MIS≠DIS for EXE and OVL files, because the header exist in the disk image but not loaded with the memory image. MIS and DIS are used by the detection algorithm. TOA is used by the detection and eradication algorithms as will be explained later. EPN (Executable Private Number) is generated by LEP for each executable before executing it. EPN of E is considered, by LEP, as the “identifier” or “Secret Name” of E. While in PHASE-TWO, the SDR of E must pass the EPN of E associated with the other identification information to LEP, so that it can get the permission from LEP to access the disk image of E. LEP-TRAP is a pointer to a special trapdoor within LEP. This trapdoor is used to ensure that the SDR can communicate with LEP secretly while in PHASE-TWO.

- Detection and Eradication Information

The SDR must know the following information about the protected executable:

1- **Executable Critical Bytes: “The critical byte is any byte which is virtually guaranteed to be changed after a virus infection”.** The executable size, the first three bytes of a COM or SYS, and the EXE or OVL file header are considered as critical bytes. Determining the critical bytes of a given executable, require a deep understanding of how the computer virus infect it. In general, the following components are calculated and stored in the SDR of E during the protection process “see section 5”:
   1. **CMIS= Correct Memory Image Size (CMIS)**
   2. **CDIS= Correct Disk Image Size (CDIS)**
   3. **The correct values of any other critical byte. Such as the first three bytes of COM and SYS files and the header of EXE and OVL files.**

- **Integrity Check Information (Checksum):** The SDR is assumed to view the executable memory image as group of N-Blocks (BLK1,BLK2, ... BLKN) “see Fig.3”. The block size (BLKS) is equal for all blocks and stored in (BLKS). A CheckSum Number (CSN) calculated for each block using the algorithm CHKS-ALG, and stored in a special SDR table. Also, a given block ‘BLKi” is assumed to be valid (i.e. has a correct checksum) if the following condition satisfied: **“CHKS-ALG(BLKi, CSNi)=0”.**
- **Position Test Information:** The position test information (SOF and EOF) are used (with TOA, and MIS) by the FBEA to find the virus block position relative to the SDR module as show in Tab.7. Fig.3 shows how the protected executable appear in memory. SOF and EOF represent the position test information and defined as follow:

- ♦ **The Start Offset (SOF):** Is the offset of the SDR trapdoor relative to the start of the memory image**.**
- ♦ **The End Offset (EOF):** Is the backward offset of the SDR trapdoor relative to the end of memory image**.**

- SDR Internal Flags

SDR uses two internal flags:

1- Executable Infection Flag (EIF): SDR set this flag if it detects a virus infection. The status of this flag is returned to LEP at the end of PHASE-ONE.

2- Executable Repair Flag (ERF): SDR set this flag if it can repair the infected executable. The status of this flag is returned to LEP at the end of PHASE-ONE, and the end of PHASE-TWO through LEP-TRAP.

2.4 The Secret Identification Block (SIB)

The SIB passed by the SDR of E to LEP during PHASE-ONE. When LEP switch to PHASE-TWO, it can use SIB to distinguish the SDR of E from the SDRs of the other executables and viruses. The standard format of SIB is described in Tab.8.

3- PHASE-ONE Communication

PHASE-ONE starts when LEP receives a request to execute an executable. Assuming that LEP receives a request to execute the executable E, the following sections describe the sequence of operations:

Locating and Executing SDR

The following steps describe loading the executable by LEP and preparing the IIB:

Step-1: “Loading the executable memory image”

1. **Find the disk image of E.**
2. **Store DIS in the IIB.**
3. **Assign EPN to E and store it in the IIB.**
4. **Store LEP-TRAP in the IIB**
5. **Reserve a memory block to store the executable memory image. And store the start address (SMI) in the IIB.**
6. **Load the executable memory image into the reserved block.**
7. **Store the MIS in the IIB.**

Step-2: “Finding the Trapdoor”

1. **Search for the trapdoor of the primary SDR module.**
2. **IF (the trapdoor found) THEN go to 6**
3. **Search for the trapdoor of the redundant SDR module.**
4. **IF (the trapdoor found) THEN go to 6**
5. **LEP cannot find any one of the trapdoors. This can happen if the executable is not Self-Protected, the executable is new in the system, or both SDR modules are destroyed due to a virus infection. In either case, LEP must display alarm message to the user, and ask him what to do. Depending on the user response, LEP must proceed as follow:**
   - ♦ **If E is not self-protected, then execute it in a virtual computer (explained later).**
   - ♦ **If E is new, then prepare it using the message file.**
   - ♦ **If E destroyed by a virus infection, then, avoid executing it**
6. **Store the offset of the trapdoor in TOA.**
7. **Store TOA in the IIB.**

Step-3:”Execute and Wait”

LEP can transfer control to the SDR trapdoor using a far call instruction, and wait until the SDR return control again. What happens when the SDR receives control is explained in the next section.

SDR Operation in PHASE-ONE

The SDR operation can be described by two cycles: **“Self-Test Cycle, and Self-Repair Cycle”**. Both cycles are described in the following sections.

Self-Test Cycle

This cycle initiated each time SDR executed, in this cycle SDR must decide whether E is infected or not. If E is infected, SDR will set EIF, and clear it otherwise. The status of EIF returned from SDR to LEP as shown in Fig.2. The following algorithm describe the self-test cycle:

```
      ;Size test
   IF DIS≠CDIS THEN go to 13
1.  IF MIS≠CMIS THEN go to 13
      ;Critical byte test
2.  IF (Any critical byte changed) THEN go to 13
      ;Checksum test
3.  i=1
4.  C=CHKS-ALG(BLKi, CSNi)
5.  IF C≠0 THEN go to 13
6.  i=i+1
7.  IF i ≤ N THEN go to 5
      ;Position test
8.  IF SOF≠TOA THEN go to 13
9.  IF EOF≠MIS-TOA go to 13
      “The executable is clean”.
10.EIF=0
11.Return to LEP.
      “The executable is infected”.
12.EIF=1
13.Go to the Self-Repair Cycle.
```

Self-Repair Cycle

In this cycle, SDR will try to eradicate the virus and repair the protected executable. SDR will set ERF if the protected executable repaired properly and clear it otherwise. A new eradication algorithm (Foreign Block Eradication Algorithm “FBEA”) will be used. FBEA capability depends on how the virus distributes itself within the infected executable. The FBEA described below can eradicate SBD viruses efficiently, and can be upgraded to eradicate CBD viruses as well. However, because the CBD idea not used by viruses yet “see [3]”, the discussion will be limited for the SBD viruses only. Assuming a simple virus distribution, the eradication algorithm can be divided into the following steps:

Step-1: “Find the virus block position relative to the SDR”

In general, if the virus block inserted after SDR, the infected executable memory image will take the form of image ①,②, or ③ in Fig.4. Therefore, the search must start from BLK1. If the virus block inserted before SDR, the infected executable memory image will take the form of image ④,⑤, or ⑥. Therefore, the search must start from BLKN. Finding the virus block position relative to SDR can be done by using the position test information TOA, MIS, SOF, and EOF as shown in Tab.7.

Step-2: “Starting the block checksum test”

For example, let us assume that the virus is inserted after the SDR, therefore, the block checksum test must start from BLK1. The following algorithm can be used in this case:

```
1. i=1
2. IF BLKi contain any critical byte, then repair the critical bytes.
        “Only BLK1 in image ①, ②, and ③ affected by this step”
3. C=CHKS-ALG(BLKi, CSNi)          ;Calculate the checksum of BLKi
4. IF C≠0 THEN go to Step-3        ;Invalid block
5. Move BLKi into BUF              ;Valid block
6. i=i+1
7. IF i ≤ N THEN go to 2
        Arriving to this point means that the virus has inserted all of its added bytes at
          the end of the infected executable, as shown in image ①.
 Go to Step-6
```

‘Note the difference between (go to 3) and (go to Step-3)”

Step-3: “Reversing the block checksum test order”.

Arriving to this step means that the virus inserts its added bytes after SDR but not at the end of the infected executable. In this case, the infected executable memory image expected to take the form of image ② or ③. Because BLKi is not found, the searching processes must be reversed. The following algorithm can be used:

```
1. j=0
2. m=N-j
3. IF BLKm contain any critical byte, then, repair the critical bytes.
4. C=CHKS-ALG(BLKm, CSNm) ;Calculate checksum number of BLKm
5. IF C≠0 THEN go to Step-4      ;Invalid checksum
6. Move BLKm into BUF            ;Valid checksum
7. IF m=i THEN go to 10
8. j=j+1
9. Go to 2
        Arriving to this point means that the virus block inserted between two
          consecutive blocks (BLKi-1 and BLKi) without destroying any one of them, as
          shown in image ③. Because the blocks (BLK1, BLK2, ..., BLKi-1) are moved
          into BUF in Step-2. And the blocks (BLKi, BLKi+1, ... BLKN) are moved into
          BUF in Step-3. Therefore, all blocks are moved into BUF.
10.Go to Step-6
```

Step-4: “Repairing the damaged block”

Arriving to this point means that the virus has inserted its block inside BLKi and, hence, destroyed this block. This is shown in image ②. Two routines to repair the destroyed block BLKi will be discussed:

REPAIR1:

Inputs:

- **i= Block number**
- **S= Start offset address of BLKi which contain the virus block.**
- **E= End offset address of BLKi which contain the virus block.**
- **Z= MIS-CMIS=NAB**
- **P= Offset address of a byte that is guaranteed to exist in the virus block.**

Outputs:

- **C=0 ;BLKi cannot be repaired**
- **C=1 ;BLKi repaired properly and stored in the buffer BBK**

<u>**The Idea:**</u> REPAIR1 assumes that the virus block start at P and define: **"VS (Virus-Start)=P, and VE (Virus-End)= VS+Z-1".** And then, moves the bytes at block (S-To-(VS-1)) and block ((VE+1)-To-E) into the buffer BBK. If the checksum test on BBK fails, REPAIR1 assumes that P is not the actual start of the virus block. That is, there is at least one byte belong to the virus block and exist before P. Therefore, REPAIR1 shift VS and VE up by one byte position, and repeat the test process.

<u>**Algorithm:**</u>

```
1. VS=P
2. VE=VS+Z-1
3. IF {VE>(E-1)} THEN {[VS=VS-(VE-(E-1))] AND [VE=VS+Z-1]}
      "This ensures that VE≤(E-1). Note that, the byte at E must belong to BLKi,
         because, otherwise, all of the content of BLKi exists above the virus block.
         And this can happen only if BLKi wasn't destroyed by the virus infection".
4. Move the bytes at block (S-To-(VS-1)) into BBK
5. Move the bytes at block ((VE+1)-To-E) into BBK
6. C=CHKS-ALG(BBK, CSNi)
7. IF C= 0 THEN go to 12        ;Valid block

      "Arriving to this point means that VS is not the actual start of the virus. That is,
         there is at least one byte belong to the virus but exist above VS. Therefore, VS
         must be decremented and the checksum must be calculated again".
8. VS=VS-1
9. VE=VS+Z-1
10.IF VS>S THEN go to 4
11.C=0, and RETURN "BLKi cannot be repaired"
12.C=1, and RETURN "BLKi repaired properly and stored in BBK"
```

<u>REPAIR2:</u>

Inputs:

- ♦ **i= Block number**
- ♦ **S= Start offset address of BLKi which contain the virus block.**
- ♦ **E= End offset address of BLKi which contain the virus block.**
- ♦ **Z= MIS-CMIS=NAB**

Outputs:

- ♦ **C=0 ;BLKi cannot be repaired**
- ♦ **C=1 ;BLKi repaired properly and stored in the buffer BBK**

<u>**The Idea:**</u> REPAIR2 is based on the fact: **"If BLKi *destroyed* by one sequential virus block, then, the byte at S and the byte and E must belong to BLKi. Because otherwise, if the byte at S(E) belong to the virus block, the entire virus block must exist above (below) BLKi, and this can happen only if BLKi wasn't *destroyed* by the virus block insertion".** REPAIR2 moves one byte starting from S and (BLKS-1) bytes starting from E into BBK. If the checksum test on BBK fail, REPAIR2 repeat the process by moving 2-bytes starting from S and (BLKS-2) bytes starting from E into BBK.

<u>**Algorithm:**</u>

```
1. k=S
2. j=k+Z+1
3. Move the bytes at block (S-To-k) into BBK.
4. Move the bytes at block (j-To-E) into BBK.
5. C=CHKS-ALG(BBK, CSNi)
6. IF C=0 THEN go to 10;Valid block
7. k=k+1
8. IF k< (BLKS-1) THEN go to 2
```

9. **C=0, and RETURN “BLKi cannot be repaired”**
10. **C=1, and RETURN “BLKi repaired properly and stored in BBK”**

The advantage of REPAIR2 over REPAIR1 is that it can work without using P. However, the number of trails or the time needed by REPAIR1 is, in general, less than the time needed by REPAIR2. Which routine “REPAIR1 or REPAIR2” the SDR must use, depend on the prepared input arguments. The input arguments can be prepared as follow:

1. **S= (i-1)*(BLKS) ;The number of bytes in all blocks before BLKi.**
2. **E= S+Z+BLKS;To understand how S and E calculated see Fig.5.**
3. **The SDR can use REPAIR1 only if it can prepare P. Otherwise, it must use REPAIR2. Two ways are suggested to determine P:**
    - a) **The entry point of the virus code within BLKi can be found from the standard entry point of the executable. For example, for COM files the displacement of the near jump instruction which is found in the first three bytes of BLK1 can be used to find the virus entry point. After finding this entry point, set P= Virus code entry point.**
    - b) **If the virus code entry point cannot be found, the following fact can be used: “IF (Z>BLKS) THEN (The point at “S+(E-S)/2” must be in the virus block)”. Therefore: IF (Z>BLKS) THEN (P=S+(E-S)/2)”.**

If P prepared, the SDR can proceed as follows:

1. **CALL REPAIR1**
2. **If C=1 THEN go to 6 ;BLKi repaired properly using REPAIR1**
3. **CALL REPAIR2**
4. **IF C=1 THEN go to 6 ;BLKi repaired properly using REPAIR2**

BLKi cannot be repaired using REPAIR1 or REPAIR2.

5. **Go to Step-5**
6. **Move the contents of BBK into the gap of BLKi in BUF.**
7. **Go to Step-6**

Step-5: “Return Error Code and SIB to LEP”.

Arriving to this step means that the SDR cannot repair E properly. Therefore, the SDR must return the following information:

1. **EIF=1 ;The file is infected**
2. **ERF=0 ;The file is not repaired**
3. **SIB**
4. **Go to Step-7.**

Step-6: “Repair the Infected Disk Image of E”

Arriving to this step means that BUF contain all blocks (BLK1, ... BLKN) of the protected executable. The content of BUF represent a clean memory image of the executable, therefore, it can be used to repair the infected disk image of E. The following algorithm describes how this can be done.

1. **If the infected executable is an EXE or OVL file, then, put the correct file header in the start of BUF.**
2. **Overwrite the disk image of E by the content of BUF.**
3. **ERF=1 ; Executable Repaired Properly.**
4. **Go to Step-7**

Step-7: “Return To LEP”

The SDR can return control to LEP now. PHASE-ONE terminated at this point.

- PHASE-TWO Communication

LEP starts PHASE-TWO after the SDR execution, in PHASE-ONE, is completed. Once in PHASE-TWO, LEP will test the status of EIF and ERF returned from SDR. The executable is clean if (EIF=0). Therefore, LEP can execute the executable through its standard entry point. The executable was infected but repaired properly by SDR if (EIF=1 and ERF=1). Therefore, LEP must load the executable disk image again. And, then execute it through its standard entry point.

The executable is infected and SDR cannot repair it if (EIF=1 and ERF=0). Therefore, the Virus Follower Eradication Algorithm (VFEA) must be used to eradicate the virus. VFEA is based on the fact that shell type viruses, "See [7]", always repair the infected executable memory image before executing it. Therefore, VFEA use the following eradication approach: **"The infected executable must be executed through its standard entry point so that the attached virus will get the opportunity to execute. The virus will repair the memory image of the executable and execute it. SDR will receive control again and test the memory image. If the memory image repaired properly by the virus, SDR must alert LEP to use the current memory image of the executable to replace the infected disk image of the executable".**

However, giving the opportunity to the virus to execute is very critical. Therefore, before executing the virus, LEP must prepare a trusted environments "Virtual Computer System" to ensure that the virus cannot cause any damage or infection. Preparing the virtual computer system and executing the SDR in PHASE-TWO is described in the following sections:

The Virtual Computer System

The idea of the virtual computer system is described in [4]. In general, the virtual computer system must be designed to satisfy the following requirements:

1- **Protecting System Disks:** Prevent the virus from infecting or destroying any target site/cell in the system disks.

2- **Protecting System Memory:** Prevent the virus from reserving memory space and hide itself there, that is, stay resident in system memory.

3- **Deceiving the Virus:** The virtual computer system must give the virus the illusion that it is running in a normal system. Because, if the virus knows that it exists in a virtual computer system, it may try to use special methods to bypass or deceive the virtual computer system; or it may terminate its execution without repairing its host, therefore, VFEA cannot repair the protected executable.

Satisfying these requirements depend on the computer system hardware (i.e. Real-Mode or Protected-Mode PC) and software (i.e. DOS or WINDOWS). As a case study, preparing a virtual computer system in DOS machines will be explained in what follows:

1- **Protecting System Disks:** In order to prevent the virus from infecting/destroying target sites/cells in the system disks, it must be prevented from writing to these disks. First of all, LEP must ensure that all of the disk related interrupt vectors points to special handlers. Therefore, LEP must redirect the following vectors:

   1. **Vector 13H (BIOS Disk Interrupt INT 13H).**
   2. **Vector 21H (DOS-API Interrupt INT 21H).**
   3. **Vector 25H (DOS: Absolute Disk Read Interrupt INT 25H) "see [5]".**
   4. **Vector 26H (DOS: Absolute Disk Write Interrupt INT 26H).**

   Under DOS, if the virus knows that it exists in a virtual computer system, it may try to use direct hardware access method to bypass the virtual computer handlers. The virtual computer can use one of the following methods to prevent viruses from accessing the disk: First, the handler of the virtual computer can reject any write to disk request by returning some error code which indicates that the requested operation cannot be performed because, for example, the drive is not ready. Rejecting all writes to disk requests can help the virus in deciding whether it is working in a virtual computer system or not. Second, the virtual computer system can trick the virus to believe that the requested write to disk operation is performed while it is not. This can be done by returning a no error code that indicates the requested operation performed properly, without performing the actual operation. If this method used, the virus can decide whether it is working in a virtual computer system or not by using the following trick:

1. **Send a request to write the data block (BLK) to disk (C:).**
2. **Read the data block from disk (C:) into (BLK1)**
3. **If BLK≠BLK1, then, the data wasn't written to the disk. Therefore, the system is a virtual computer system.**

Finally, all disk read/write operation can be redirected to a special RAM disk instead of the actual disk. In this way, the virtual computer system can trick the virus to believe that the requested write to disk operation was performed and the data written to the disk properly, while the data was written to the RAM disk. If the virus request the data later, the virtual computer can read it from the RAM disk. This means that the virtual computer must handle both disk read and write operations, this explains why vector 25H was redirect above.

2- Protecting the IVT: The virtual computer system must save the content of the IVT before executing the virus, and restore it once the virus execution completed. This ensures that any redirection to the IVT vectors by a resident type virus is fixed once the virus execution is completed.

3- Protecting System RAM: Protecting the PC system RAM can be done as follow:

- **Save the amount of the available URAM (the word at 0:0413H) before executing the virus. And restore the content of 0:0413H once the virus execution completed. This ensures that the virus cannot decrement the amount of URAM and install itself at the end of the URAM.**
- **If the virus tries to allocate a memory block using function 48H of INT 21H, then, the virtual computer must store the address of the allocated memory block so that it can release this block once the virus execution completed. Therefore, it must redirect vector 21H.**
- **Save a map of all memory control blocks "see "[٥] before executing the virus. And restore them once the virus execution completed. This ensures that the virus cannot allocate memory by directly accessing the memory control blocks.**
- **Redirect vector 66H to special handler before executing the virus. INT 66H is used to access the Expanded Memory Manger functions. This interrupt must be handled only if the system uses expanded memory. If the virus allocate a page (or pages) in the expanded memory, the handler must store the handles reserved by the expanded memory system for the allocated pages. This handle can be used to release the reserved pages when the virus execution completed. Expanded memory expands RAM beyond the 640KB limit, for more information see [5].**
- **Extended memory exists in AT machines (the memory beyond the 1 MB limit "see [5]"). Because the eXtended Memory Manager functions are called through a FAR CALL instruction, instead of the special interrupt, the virtual computer cannot intercept requests to the extended memory function. Therefore, the virtual computer must ensure that all of the extended memory is free before executing the virus. And free it again after the virus execution completed, to ensure that viruses cannot allocate memory in the extended memory area.**

Execute SDR and Wait

After preparing the virtual computer system, LEP can execute E through its standard entry point and wait until it receives a special call which indicate that the virus execution was completed and SDR was executed. Note that LEP can switch from the virtual computer system back to the normal system only after ensuring that the virus execution was completed. Receiving the call is the clue that indicates to LEP that SDR is the currently active program. Because viruses always try to disable or circumvent the protection mechanism, the following requirements must be satisfied:

1- **The virus cannot mask the call:** Therefore, in the PC system the software interrupt mechanism cannot be used to perform the call. Because the virus can redirect all of the interrupt vectors to its own handler, therefore, it can mask the call and preventing it from arriving to LEP. The call must be direct, that is, by using jump or call instructions. This explains why LEP passes a pointer (LEP-TRAP) to its trapdoor to SDR in PHASE-ONE.

2- **The virus cannot deceive LEP:** In order to deceive LEP by a tricky call, the virus must know where to send the call and which information to pass with it. Therefore, the virus cannot deceive LEP through LEP-TRAP, because it doesn't know the correct values of LEP-TRAP, EPN, and SIB. Since these variables passed from LEP to SDR in PHASE-ONE and the virus was inactive at that time, therefore, there is no way to know these variables by the virus.

SDR Operation in PHASE-TWO

When the SDR receives control in PHASE-TWO it will proceed as follows:

```
1.  IF EIF=1 THEN go to 3
        ;E is clean
2.  Continue the execution of E normally
        ;E is infected, start the repair cycle using VFEA
        ;Perform checksum test
3.  i=1
4.  C=CHKS-ALG(BLKi, CSNi)
5.  IF C≠0 THEN go to 12
6.  i=i+1
7.  IF i≤N THEN go to 4
        ;All blocks are valid.
8.  ERF=1 ;The memory image is repaired properly
9.  SMI= Start of the repaired Memory Image
10. MIS= Size of the repaired Memory Image
11. Go to 15
        ;BLKi is invalid
12. ERF=0 ;The memory image cannot be repaired
13. Prepare SIB
14. EPN= Executable Private Number that was received from LEP in PHASE-ONE
        ;Transfer control to LEP through LEP-TRAP
15. JMP FAR PTR LEP-TRAP
```

Receiving Control from SDR

The virtual computer system prevents any program from writing to the system disk to ensure that viruses cannot replicate themselves. However, the question is how the SDR of E can repair the infected disk image of E. LEP can give the permission to the SDR to perform disk write operations. SDR must call LEP through LEP-TRAP with the proper EPN and SIB to get this permission. Giving this permission to SDR can be done in one of two ways:

1- **Filtering:** In this case, the virtual computer must be capable of distinguishing between the SDR write to disk requests and the other requests. This can be done, for example, by sending EPN and SIB with each request.

2- **Restoring the IVT:** LEP can restore the IVT content when receiving EPN and SIB through its trapdoor. Therefore, SDR can access the disk as desired by using DOS and BIOS services.

However, giving the permission to SDR to access the disk is not a good idea, because, the virus exist in system memory and may, in some way, interfere with the SDR operation, so there is a possibility that the virus will get the opportunity to access the disk. Therefore, the following method is suggested: **"Instead of giving the disk access permission to SDR, LEP can repair the disk image by itself after receiving the necessary information from SDR".**

Therefore, as shown above, SDR return SMI and MIS to LEP through LEP-TRAP. After receiving this information, LEP can proceed as follows:

1. **Compare the received EPN with the one that was given to SDR in PHASE-ONE, if not equal go to 6**
2. **Compare the received SIB with the one that was received from SDR at the end of PHASE-ONE, if not equal go to 6**
3. **IF ERF=0, THEN, switch to normal mode "SDR cannot prepare a clean backup image".**

   "A clean backup image is available and LEP must use it to replace the infected disk image, as follows:"
4. **Use the MIS-byte block that starts at (SMI) to replace the executable whose private number is EPN.**
5. **Switch to normal mode.**
6. **The SDR identification information (EPN or SIB) are invalid, therefore, reject the call, and display alarm message.**

4.5 Switching to Normal Mode

LEP can switch to normal mode after receiving the SDR request through LEP-TRAP and repairing the infected disk image if necessary. LEP must do the following so that it can switch to normal mode:

1. **Restore the IVT**
2. **Release any memory block allocated during the virus execution using DOS function 48H, URAM, memory control blocks, expanded memory, and extended memory.**
3. **Release the memory allocated for E, to ensure that the viral code which exist at this memory area will be destroyed.**
4. **Release the RAM disk memory area.**

5- Vaccination

There is a large number of executables in the world. All of these executables are developed without any built-in SDR. Clearly, some way must be found to convert these executables to self-protected executable. Vaccination can be used to do this. In this research, vaccination is defined as: **"The process of converting an executable into a self-protected executable by injecting the SDR module inside it".**

"**Note:** Some references (see [6]) refer to 'Vaccination' as one of the protection methods that was suggested by IBM. Some anti-virus programs are designed to inject themselves inside the executable files, and then operate like a '**Benign Virus**' when the vaccinated program executed. The injected anti-virus creates a signature (finger-print) of uninfected executable, and display an alarm if the signature or executable size changed. Unfortunately, the alarm generated after the virus execution. The difference between the suggested SDR and the Vaccine of IBM, is that the SDR executed before any attached virus and uses sophisticated techniques to detect and eradicate the virus".

In fact, the idea of vaccination is borrowed from viruses. A special program "called the INJECTOR" will inject the executable by the vaccine (i.e. SDR). However, there is no searching, or infection routine in this vaccine, therefore, the vaccine cannot replicate itself. The word "injection/inject" is used instead of the word "infection/infect" to distinguish vaccination from virus infection. The following steps can describe the algorithm used by INJECTOR to inject SDR into an executable E.

Step-1: "Injecting SDR in the executable E"

This is similar to what the virus infection routines do, that is:

1. **Store the contents of the standard entry point of E in SDR.**
2. **Attach SDR to E.**
3. **Redirect the standard entry point of E to point to the entry point of SDR.**

After this injection, E will take the form of image ② in Fig.6. From now, image ② is considered the correct executable image. If SDR receives control through the standard entry point and after the completion of its execution, it can repair the image of E to take the form of image ① so that it can execute properly.

Step-2: "Preparing the SDR Module"

As shown in Fig.6, the correct image of E is image ② from the SDR point of view. Therefore, INJECTOR must prepare SDR with respect to this image. After calculating and storing SOF, EOF, CDIS, CMIS, and the other critical bytes, INJECTOR must divide image ② into block (BLK1, ... BLKN) and store the checksum numbers in the SDR table see fig. 3. Preparing the SDR header can be done easily if INJECTOR knows which algorithms and functions are used by LEP. Otherwise, INJECTOR must store the values of LOC1 and LOC2 in a message file so that LEP can prepare the SDR module header.

- Efficiency Analysis

The efficiency of SDS must be analyzed to see whether it can satisfy all of the requirements of IAVS or not. The efficiency of the SDS is determined according to the analysis criteria presented in [٤]. In what follows, SDS will be analyzed with respect to detection, prevention, eradication, and damage control.

1- Detection: SDR classified as a Target-Based detection.

a- VGS= Max.: Because SDR knows every thing about the protected executable, it can detect any change to this executable. Viruses cannot avoid detection by the SDR, because, they cannot infect the executable without changing its contents. Even stealth type viruses "see [7]" cannot deceive the SDR, because, they cannot get the opportunity to execute before the SDR. Therefore, SDR detection capability is independent from the virus generation date and VDC. Also, if LEP cannot find any one of SDR modules, it concludes that the protected executable was changed, and it might be infected by a virus.

b- TGS=1: Because SDR designed to detect viruses attached to the protected executable only.

c- Capability: The following is concluded from a & b: **"SDS can satisfy the requirement of ideal Target-Based detection".**

- Prevention: Both VEP-Based and VIP-Based prevention can be used in SDS.

♦ **VEP-Based Prevention:** SDR classified as a VEP-Based prevention anti-virus, because, it can detect the infection without giving viruses the opportunity to execute.

a- VGS= Max.: SDR prevention is independent from the virus generation date and VDC.

b- TGS= Max.: SDR will prevent the detected virus from infecting any target site within its environment.

c- Capability: The following is concluded from a, b, and the fact that the SDR is ideal with respect to detection: **“SDS can satisfy the requirement of ideal VEP-Based prevention”.**

♦ **VIP-Based Prevention:** LEP classified as a VIP-Based prevention anti-virus, because, it can prevent the detected virus from infecting a new target site when executed in PHASE-TWO by using the virtual computer system.

a- VGS: It was mentioned in section 4.1 that the direct hardware access is the most vulnerable spot that the virus can use to penetrate the virtual computer system. Maximizing VGS thoroughly depends on the system that will implement the SDS. VGS can be maximized, if SDS is implemented on a computer system that provides hardware protection level (i.e. protected mode PC) with a permission or privilege level that will prevent an unauthorized program to direct access the hardware resources. Otherwise, VGS cannot be maximized.

B- TGS= Max.: LEP will try to prevent the detected virus from infecting any target site within its environment.

c- Capability: The following is concluded from a & b:

♦ **“SDS can satisfy the requirement of ideal VIP-Based prevention if used on a system that provides hardware protection level (i.e. Protected Mode PC)”.**

♦ “**SDS cannot satisfy the requirement of ideal VIP-Based prevention if used on a system that lacks the hardware protection level (i.e. Real Mode PC)”.**

- Eradication: As shown above, two eradication algorithms (FBEA and VFEA) are used by SDS:

♦ **FBEA:** FBEA uses Analysis-Based eradication technique. It analyzes the protected executable and eradicates any foreign block.

a- VGS: FBEA eradication capability depend on the virus type, shell or intrusive “see [7]”, and NAB-distribution method (SBD or CBD). Therefore, the FBEA described above can eradicate only shell type viruses that use SBD method. Even though, FBEA can be upgraded to eradicate CBD viruses, the virus designers can design new strains of CBD-viruses that cannot be eradicated by FBEA. Therefore, the FBEA eradication capability depends on the virus generate date. This means that: **“FBEA cannot maximize VGS”.**

<u>**NOTE:**</u> FBEA is a good enhancement in the design of eradication algorithms. It is very efficient eradication algorithm at the present time, because, it can eradicate, almost, all of the currently available viruses.

b- TGS=1: Because FBEA designed to eradicate viruses from the protected executable only.

c- Capability: The following is concluded from a & b: **“SDS cannot satisfy the requirement of ideal Analysis-Based eradication”.**

♦ **VFEA:** VFEA uses Backup-Based eradication technique. In this case, the infected disk image represents the vulnerable copy, and the repaired (i.e. by the virus) memory image represents the backup copy.

a- VGS: VFEA can eradicate shell type viruses only and its eradication capability depend on the virus generation date, this means that: **“VFEA cannot maximize VGS”.**

b- TGS=1: VFEA designed to eradicate viruses from the protected executable only.

c- Capability: The following is concluded from a & b: **“SDS cannot satisfy the requirement of ideal Backup-Based eradication”.**

NOTE: VFEA is a good enhancement in the design of eradication algorithms. It is very efficient eradication algorithm at the present time, because, it can eradicate, almost, all of the currently available shell-type viruses.

- **Damage Control:** SDS system can use Virus-Based, Cell-Based, and Backup-Based damage control techniques.
  - ♦ **Virus-Based Damage Control:** In PHASE-ONE, SDR classified as a Virus-Based damage control anti-virus. Because it detects the virus before executing it. Clearly, the virus cannot cause any damage if not executed. Virus-Based damage control is similar to VEP-Based prevention, therefore, VGS= Max, TGS= Max, and: "**SDS can satisfy the requirement of ideal Virus-Based damage control".**
  - ♦ **Cell-Based Damage Control:** In PHASE-TWO, LEP classified as a Cell-Based damage control anti-virus, because, it tries to prevent viruses from destroying any target cell within its environment by using the virtual computer system. Clearly, direct hardware access is still a vulnerable spot. Cell-Based damage control is similar to VIP-Based prevention, therefore:
    - ♦ **"SDS can satisfy the requirement of ideal Cell-Based damage control, if used on a system with hardware protection level".**
    - ♦ **"SDS cannot satisfy the requirement of ideal Cell-Based damage control, if used on a system with no hardware protection level.**
  - ♦ **Backup-Based Damage Control:** In PHASE-TWO, LEP can store a backup copy of the virus target cells such as FAT, RD, BPS, and CMOS RAM before executing the virus. After the completion of the virus execution, LEP can repair any target cell that was destroyed by the virus, by using its backup. This technique is similar to saving the IVT before executing the virus and restoring it after the completion of the virus execution. In this case, LEP classified as a Backup-Based damage control anti-virus.
    - **a- VGS= Max.:** LEP protection is independent from the virus generation date and VDC.
    - **b- TGS:** TGS depend on the number of target cells selected by LEP.
    - **c- Capability:** The following is concluded from a & b: LEP can protect any target cell in the selected T-group against destruction by any virus, therefore: **"SDS can satisfy the requirement of ideal Backup-Based damage control".**

- SDR Size Optimization

SDR will increase the protected executable size and the time needed to load and execute it. Therefore, minimizing the SDR size must be one of the design goals. One way to minimize the code of SDR is to implement the FBEA as a LEP (or OS) service. As mentioned earlier, the SDR calls the algorithm 'CHKS-ALG' with two arguments "Block number 'BLKi' and Checksum number 'CSN'". Clearly, it is not necessary to implement CHKS-ALG inside each SDR. Instead CHKS-ALG can be implemented as a global OS service that can be used by all SDRs. Note that the fact that the block size is different for different SDRs ensures that the general trend to avoid standard protection is not violated. Assuming that CHKS-ALG uses a standard CRC method, then, it can be implemented as a global service as follows:

<u>CHKS-ALG</u>

INPUT:

- ♦ BLKi= Start address of the block
- ♦ BLKS= Block Size in bytes
- ♦ CSN= The CRC checksum of the block.

OUTPUTS:

- ♦ C=0 ;Valid checksum
- ♦ C=1 ;Invalid checksum

In the same way, FBEA can be implemented as a global service. In this case, CHKS-ALG can be implemented inside the global FBEA. The standard Input/Output of the global FBEA can be defined as follows:

FBEA:

INPUTS:

- ♦ SMI, MIS, TOA, SOF, EOF
- ♦ N= Number of blocks in the memory image
- ♦ BLKS= Block Size
- ♦ Checksum Number Table: The table format is shown in Tab.9

OUTPUTS:

- ♦ C=0 ;Repair Succeed
- ♦ C=1 ;Repair Failed

- Conclusions and Discussion

A new approach in designing anti-virus system is presented in this paper. The proposed system is called Self-Defence System (SDS). The purpose of this research is to design an anti-virus system that complies with the requirements of ideal anti-virus system (IAVS) “see [4]. In [4], it was mentioned that Dependent-Defence System (DDS) couldn’t satisfy the requirement of the IAVS because of four problems. Now let us consider these problems from the SDS point of view:

1- **Lack of Knowledge:** In principle, SDR must be implemented as integral part of the protected executable during the development process and it must know every thing about the protected executable. Therefore, SDS can satisfy the requirement of ideal detection. However, if the SDR injected inside an existing executable using vaccination, then, it is important to ensure that the executable is clean before injecting the SDR. If a virus exists in the executable prior to the injection, then, SDR cannot detect the presence of this virus.

2- **Standard Protection:** In principle, each SDR is designed and implemented by a different manufacturer. Therefore, the code and data of SDR will be different from one SDR to the other. For example, even though more than one SDR may use the FBEA, the block size (BLKS) selected by each SDR and the location of the checksum number table might be different. Clearly, viruses cannot deceive SDR without knowing this information.

3- **Intended Vulnerability:** In SDS, the SDR capability in coping with viruses is considered one of the executable quality factors. Good programs are those that can protect themselves against viruses efficiently. Therefore, any failure in the SDR operation and any undetectable infection have bad affects on the trustiness between the program developer and his customers. They may, simply, do not purchase his programs if they find that the SDRs associated with these programs are vulnerable.

4- **Users Responsibility:** In SDS, user responsibility is no longer a requirement or a factor of efficiency, because, the SDR designer cannot claim that his program cannot protect itself against the computer virus because the user do not use the protection program properly. SDR is embedded inside the protected program and its operation is, or must be, transparent for the computer user.

As a summary: **“SDS system solves the problems (Lack of knowledge, Standard Protection, Intended Vulnerability, and User Responsibility) found in DDS”.**

Also, SDS provides a good support to design and use new sophisticated and efficient eradication algorithms. Even though FBEA is not ideal, it is considered a good enhancement in the design of Analysis-Based eradication programs, relative to the eradication method used in DDSs. In DDS, the eradication program can eradicate only ‘Known’ viruses. Any detected virus must be ‘Identified’ by security professionals, the identification information stored in a database, and a special eradication program designed to eradicate the virus. In SDS, FBEA eradicate any foreign block that exists between the protected executable blocks. FBEA doesn’t care whether the foreign block belong to a known virus, unknown virus, logic bomb, time bomb, or a Trojan Horse.

Even though VFEA is not ideal, it is considered a good enhancement in the design of Backup-Based eradication programs, relative to the methods used by DDS. In DDS, for each target site (vulnerable copy), a backup copy must be stored in the disk. In SDS, VFEA depend on the virus to generate the backup-copy and then use it to repair the vulnerable copy, therefore, it has the following advantages relative to the method used by DDS:

- ♦ No disk space needed to store backups.
- ♦ There is no way that the virus will reside in the backup copy.
- ♦ There is no need to update the backup copy

References

-Garber, L., **“Antivirus Technology Offers New Cures”**, IEEE Computer Magazine, February, 1998.

-Pfleeger, C., P., **“SECURITY IN COMPUTING”**, Prentice-Hall Inc., 1989.

-Kaspersky, E., **“AVP Virus Encyclopedia”**, Version 1.3 (1992-1997).

-Hamid M. A. Abdul-Hussain, **“Computer Virology: Toward Designing An Ideal Anti-Virus System”.** Engineering Journal, College of Engineering, University of Baghdad. Vol.8, No.3, 2002.

Tischer,M., **“PCINTERN SYSTEM PROGRAMING”**, Abacus, 1992

- د. عامر نزار فايز، **"فيروسات الكمبيوتر"**، جامعة الإسراء-عمان-الأردن, 1995

- Hamid M. A. Abdul-Hussain & Hamed M. Shabib, **“Computer Virology: Formal Analysis of Computer Viruses”.** Engineering Journal, College of Engineering, University of Baghdad. Vol.8, No.1, 2002.

List of Abbreviations

AT	Advanced Technology
BPS	Boot (or) Partition Sector
CBD	Complex Block Distribution
CDIS	Correct Disk Image Size
CMIS	Correct Memory Image Size
CRC	Cyclical Redundancy Check
CSN	CheckSum Number
DDS	Dependent Defense System
DIS	Disk Image Size

DOS	Disk Operating System
EBV	Executable Based Variable
EIF	Executable Infection Flag
EOF	End Offset
EPN	Executable Private Number
ERF	Executable Repair Flag
FAT	File Allocation Table
FBEA	Foreign Block Eradication Algorithm
IAVS	Ideal Anti-Virus System

IBM	International Business Machine
IIB	Image Information Block
IVT	Interrupt Vector Table
LBC	Loader Based Constant
LEP	Load and Execute Program
LOC	Trapdoor mark LOCATION
Max.	Maximum
Min.	Minimum
MIS	Memory Image Size
NAB	Number of Added Bytes
NCB	Number of Changed Bytes
OS	Operating System
PC	Personal Computer
RD	Route Directory
SBD	Simple Block Distribution
SDS	Self-Defense System
SDR	Self-Defense Routine
SIB	Secret Identification Block
SMI	Start of Memory Image
SOF	Start Offset
TGS	Target site/cell Group Size
TMV	Trapdoor Mark Value
TOA	Trapdoor Offset Address
VDC	Virus Deception Capability
VEP	Virus Execution Probability
VFEA	Virus Follower Eradication Algorithm
VGS	Virus Group Size
VIP	Virus Infection Probability

Tab.1

	LEP1		LEP2	
File Name	g	TMV=g+6	g	TMV= g⊕305
HAMED.COM	72+65+77=214	220	65-69= -4	-307
MUNTHER.EXE	77+85+78=240	246	85-84= 1	304
KARIM.SYS	75+65+82=222	228	65-73= -8	-311

Tab.2

Filename	LOC	[LOC]=TMV
HAMED.COM	500	503
MUNTHER.EXE	1024	1027
KARIM.SYS	856	859

Tab.3 Primary SDR Module Standard Format

LOC1+	Size(Byte)	Content	Function/Algorithm Name
00H	4	TMV11	f11(LOC1, LBC11)=g11(LOC1,LBC12)
04H	4	TMV12	f12(LOC1,LBC13)=g12(LOC1,LBC14)
08H	4	CLEN	f13(LEN)
0CH	4	CSN	CHKS-ALG1
10H	LEN	SDR	SDR code/data

Where:
LEN= SDR Module Length In bytes
CLEN= Coded Length
CSN= CheckSum Number
CHKS-ALG= CheckSum Algorithm

Tab.4 Redundant SDR Module Standard Format

LOC2+	Size(byte)	Content	Function/Algorithm Name
00H	4	TMV21	f21(LOC2,LBC21)=g21(LOC2,LBC22)
04H	4	TMV22	f22(LOC2,LBC23)=g22(LOC2,LBC24)
08H	4	CLEN	f23(LEN)
0CH	4	CSN	CHKS-ALG2
10H	LEN	SDR	SDR code/data

Tab.5 Message File Entry Format

Offset	Size (Byte)	Content
00H	11	Executable Name (E1, E2, ...)
0BH	4	Value of LOC1
0FH	4	Value of LOC2

Tab.6 IIB Standard Format

Offset	Contents	Description
00H	SMI	Start Address of the executable Memory Image
04H	MIS	Memory Image Size
08H	DIS	Disk Image Size
0CH	EPN	Executable Private Number
0EH	TOA	The SDR Trapdoor Offset Address relative to SMI
12H	LEP-TRAP	Far pointer to the trapdoor of LEP.

Tab.7

Conditions	Description
(TOA=SOF) AND (MIS-TOA=EOF)	Clean executable
(TOA>SOF) AND (MIS-TOA=EOF)	Virus block before the SDR
(TOA=SOF) AND (MIS-TOA>EOF)	Virus block after the SDR
(TOA>SOF)AND (MIS-TOA>EOF)	CBD Virus

Tab.8 SIB Standard Format

Address	Size (Byte)	Content
00	2	Length of SIB in bytes
02	?	SIB contents

Tab.9 “SDR Checksum Numbers Table”

Block Number	CRC-Checksum (CSN)
1	CSN1
2	CSN2
.	.
.	.
N	CSNn

Fig.1 Secret Vs. Standard Entry Point

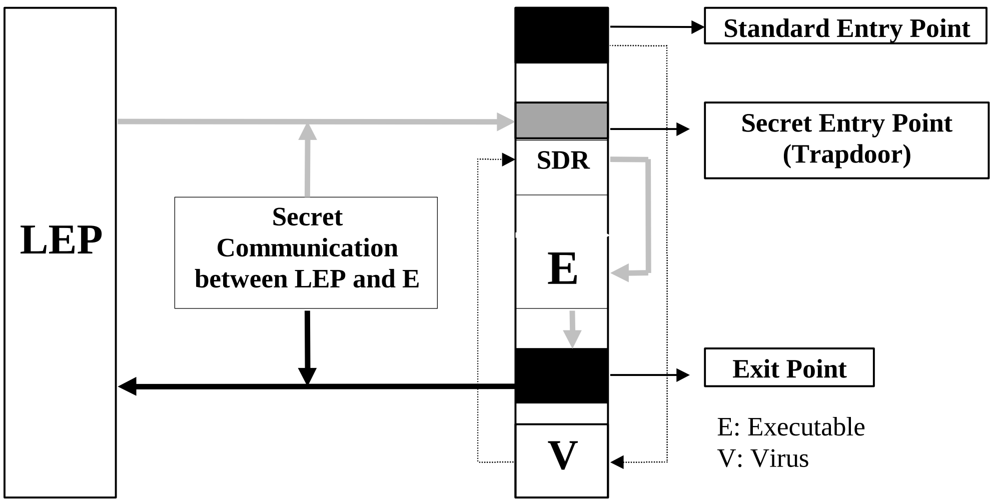

LEP
Standard Entry Point
Secret Entry Point (Trapdoor)
SDR
Secret Communication between LEP and E
E
Exit Point
E: Executable
V: Virus
V

Fig.2 Communication between LEP and SDR.

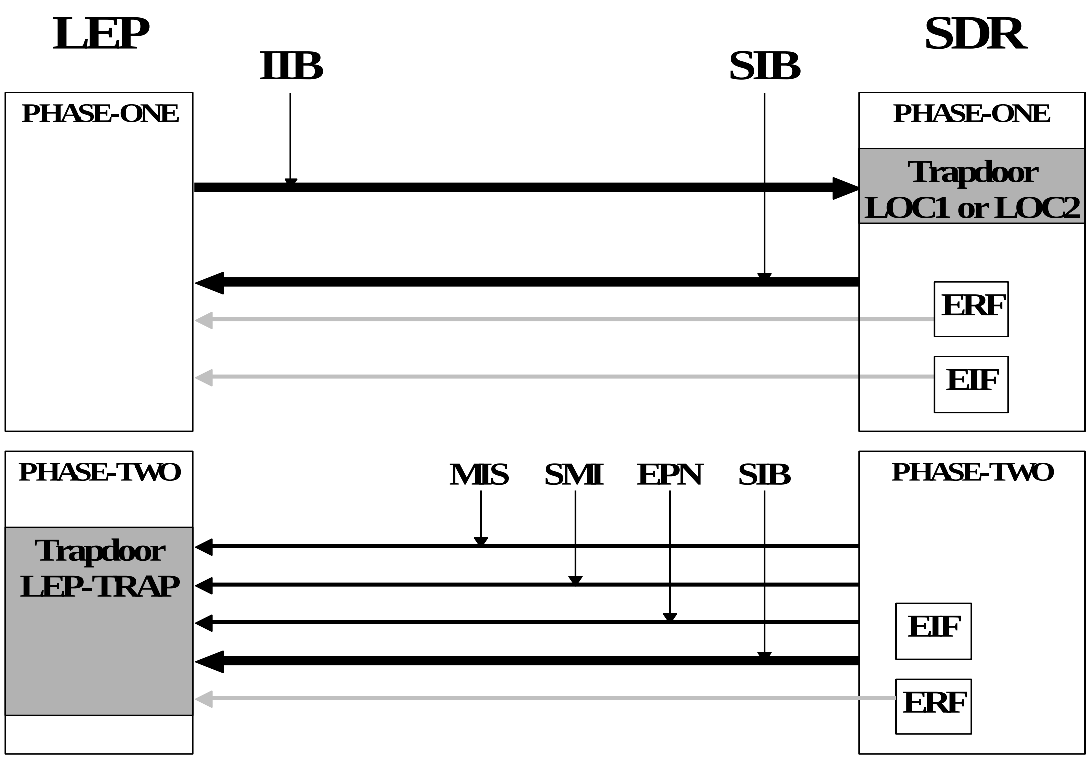


Fig.3 The Protected Executable In Memory

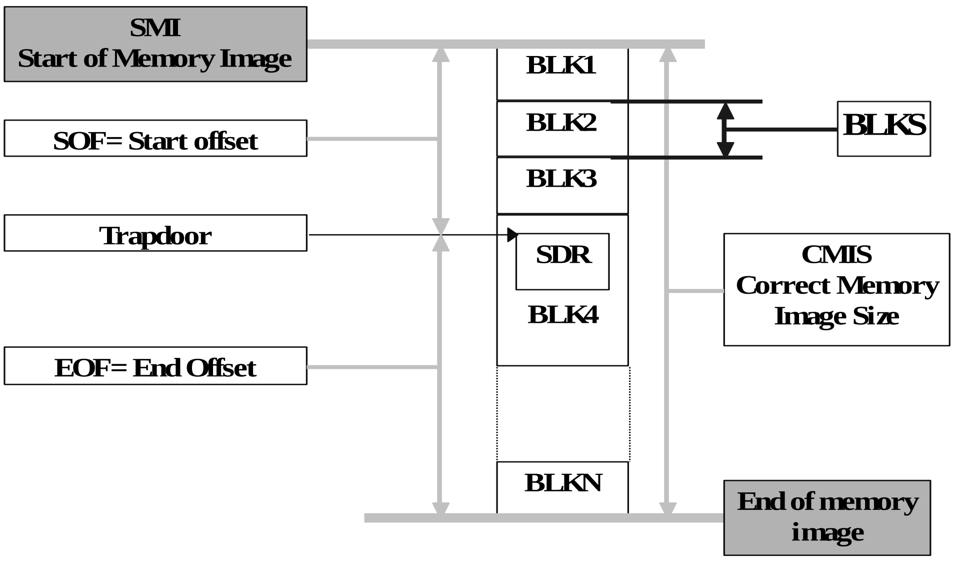


Fig.4 Simple Virus Distribution

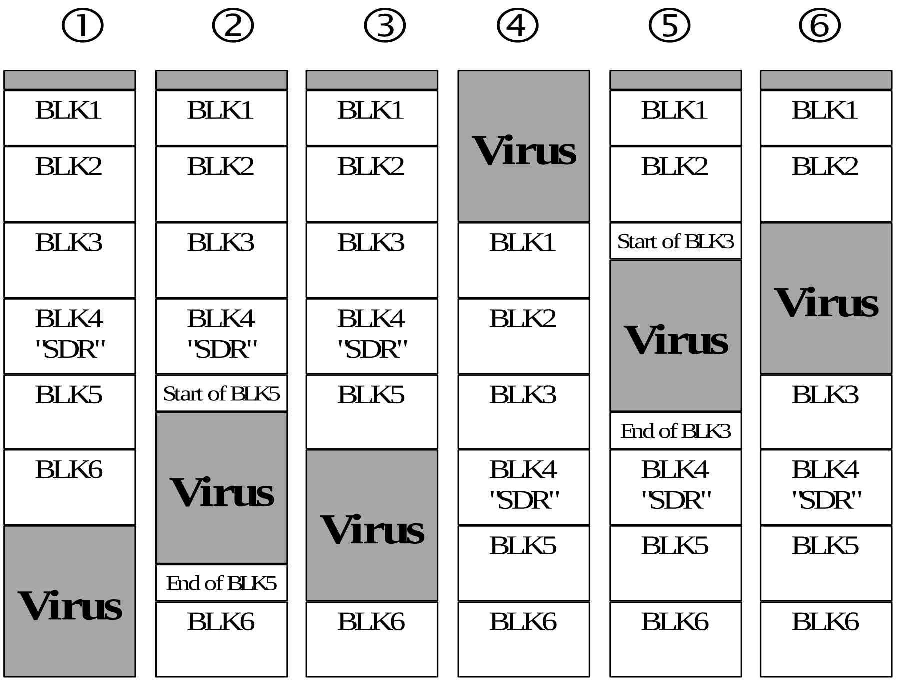


Fig.5 Determining S and E

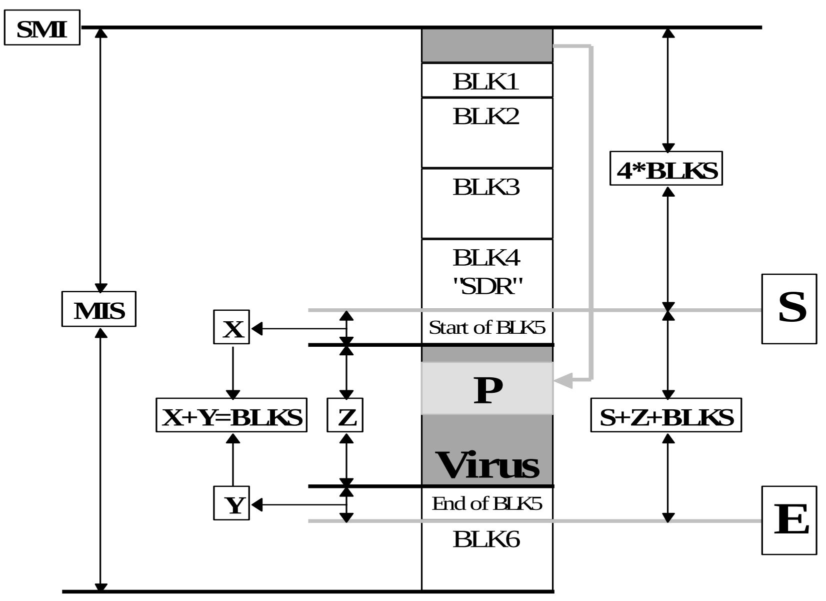
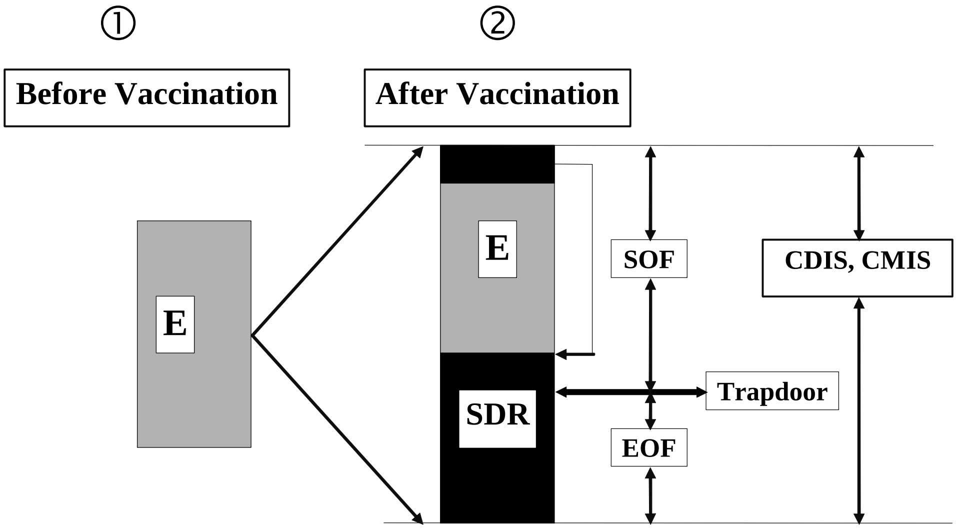


Fig.6 The Executable E Before and After Vaccination.


①
②
Before Vaccination
After Vaccination
E
E
SOF
CDIS, CMIS
SDR
Trapdoor
EOF