An Abstract Theory of Computer Viruses

Leonard M. Adleman*

Department of Computer Science
University of Southern California

1 Introduction

In recent years the detection of computer viruses has become common place. It appears that for the most part these viruses have been 'benign' or only mildly destructive. However, whether or not computer viruses have the potential to cause major and prolonged disruptions of computing environments is an open question.

Such basic questions as:

1. How hard is it to detect programs infected by computer viruses?
2. Can infected programs be 'disinfected'?
3. What forms of protection exist?
4. How destructive can computer viruses be?

have been at most partially addressed [Co1][Co2][1]. Indeed a generally accepted definition of computer virus has yet to emerge.

For these reasons, a rigorous study of computer viruses seems appropriate.

*Research supported by NSF through grant CCR 8519296

[1] It appears that F. Cohen is the first researcher in an academic setting to consider the practical and theoretical aspects of computer viruses. The formalism presented here differs considerably from that explored by Cohen [Co1][Co2].

S. Goldwasser (Ed.): Advances in Cryptology - CRYPTO '88, LNCS 403, pp. 354-374, 1990.

2 Basic Definitions

For the purpose of motivating the definitions which follow, consider this (fabricated) 'case study':

A text editor becomes infected with a computer virus. Each time the text editor is used, it performs the text editing tasks as it did prior to infection, but it also searches the files for a program and infects it. When run, each of these newly infected programs performs its 'intended' tasks as before, but also searches the files for a program and infects it. This process continues. As these infected programs pass between systems, as when they are sold, or given to others, new opportunities for spreading the virus are created. Finally, after Jan. 1, 1990, the infected programs cease acting as before. Now, each time such a program is run, it deletes all files.

Such a computer virus can easily be created using a program scheme (in an *ad hoc* language) similar to that found in [Co1]:

```
{main:=
     call injure;
     ...
     call submain;
     ...
     call infect;
}

{injure:=
     if condition then whatever damage is to be done and halt
}

{infect:=
     if condition then infect files
}
```

where for the 'case study virus':

```
{main:=
     call injure;
     call submain;
     call infect;
}

{injure:=
    if date $\geq$ Jan. 1, 1990 then
        while files $\neq \emptyset$:
            file = get-random-file;
            delete file;
        halt;
}

{infect:=
    if *true* then
    file = get-random-executable-file;
    rename main routine submain;
    prepend self to file;
}
```

By modifying the scheme above, a wide variety of viruses can be created. Even 'helpful' viruses may be created. For example the following minor variant of Cohen's [Col] compression virus which saves storage space:

```
{main:=
    call injure;
    decompress compressed part of program;
    call submain;
    call infect;
}

{injure:=
    if *false* then halt
}

{infect:=
    if executable-files $\neq \emptyset$ then
    file = get-random-executable-file;
    rename main routine submain;
    compress file;
    prepend self to file;
}
```

With the 'case study virus' and all of those which could be created by the scheme above, it appears that the following properties are relevant:

1. For every program, there is an 'infected' form of that program. That is, it is possible to think of the virus as a map from programs to ('infected') programs.

2. Each infected program on each input (where here by input is meant all 'accessible' information: e.g. the user's input, the system's clock, files containing data or programs) makes one of three choices:

   Injure:
   Ignore the 'intended' task and compute some other function. Note that in the case study, which inputs result in injury (i.e. those where the system clock indicates that the date is Jan. 1, 1990 or later), and what kind of injury occurs (file deletion) are the same whether the infected program is a text editor or a compiler or something else. Thus which inputs result in injury and what form the injury takes is independent of which infected program is running and is actually dependent solely on the virus itself.

   Infect:
   Perform the 'intended' task and if it halts, infect programs. Notice in particular that the clock, the user/program communications and all other 'accessible' information other than programs, are handled just as they would have been had the uninfected version of the program been run. Further, notice that whether the infected program is a text editor or a compiler or something else, when it infects a program the resulting infected program is the same. Thus the infected form of a program is independent of which infected program produces the infection.

   Imitate:
   Neither injure nor infect. Perform the 'intended' task without modification. This may be thought of as a special case of 'Infect', where the number of programs getting infected is zero. (In the case study, imitation only occurs when no programs are accessible for infection).

A formal definition of computer virus is presented next.

Notation 1

1. S denotes the set of all finite sequences of natural numbers.

2. *e denotes a computable injective function from $S \times S$ onto N with computable inverse.*
3. *For all $s, t \in S$, $< s, t >$ denotes $e(s, t)$.*
4. *For all partial $f : N \rightarrow N$, for all $s, t \in S$, $f(s, t)$ denotes $f(< s, t >)$.*
5. *e' denotes a computable injective function from $N \times N$ onto N with computable inverse such that for all $i, j \in N$, $e'(i, j) \geq i$.*
6. *For all $i, j \in N$, $< i, j >$ denotes $e'(i, j)$.*
7. *For all partial $f : N \rightarrow N$, for all $i, j \in N$, $f(i, j)$ denotes $f(< i, j >)$.*
8. *For all partial $f : N \rightarrow N$, for all $n \in N$, write $f(n) \downarrow$ iff $f(n)$ is defined.*
9. *For all partial $f : N \rightarrow N$, for all $n \in N$, write $f(n) \uparrow$ iff $f(n)$ is undefined.*

Definition 1 *For all partial $f, g : N \rightarrow N$, for all $s, t \in S$, $f(s, t) = g(s, t)$ iff either:*

1. *$f(s, t) \uparrow$ & $g(s, t) \uparrow$ or*
2. *$f(s, t) \downarrow$ & $g(s, t) \downarrow$ & $f(s, t) = g(s, t)$.*

Definition 2 *For all $z, z' \in N$, for all $p, p', q = q_1, q_2, ..., q_z, q' = q'_1, q'_2, ..., q'_{z'} \in S$, for all partial functions $h : N \rightarrow N$, $< p, q > \overset{h}{\sim} < p', q' >$ iff:*

1. *$z = z'$ and*
2. *$p = p'$ and*
3. *there exists an i, with $1 \leq i \leq z$ such that $q_i \neq q'_i$ and*
4. *for $i = 1, 2, ..., z$, either*
   - *(a) $q_i = q'_i$ or*
   - *(b) $h(q_i) \downarrow$ and $h(q_i) = q'_i$.*

Definition 3 *For all partial $f, g, h : N \rightarrow N$, for all $s, t \in S$, $f(s, t) \overset{h}{\sim} g(s, t)$ iff $f(s, t) \downarrow$ & $g(s, t) \downarrow$ & $f(s, t) \overset{h}{\sim} g(s, t)$.*

Definition 4 *For all partial $f, g, h : N \to N$ for all $s, t \in S$, $f(s,t) \overset{h}{\cong} g(s,t)$ iff $f(s,t) = g(s,t)$ or $f(s,t) \overset{h}{\sim} g(s,t)$.*

Definition 5 *For all Gödel numberings of the partial recursive functions $\{\phi_i\}$, a total recursive function v is a* virus *with respect to $\{\phi_i\}$ iff for all $d, p \in S$, either:*

1. Injure:

$$(\forall i, j \in N)[\phi_{v(i)}(d,p) = \phi_{v(j)}(d,p)]$$

2. Infect or Imitate:

$$(\forall j \in N)[\phi_j(d,p) \overset{v}{\cong} \phi_{v(j)}(d,p)]$$

Remark 1 *The choice of symbols d, p above is intended to suggest the decomposition of all 'accessible' information into 'data' (information not susceptible to infection) and 'programs' (information susceptible to infection).*

3 Types of Viruses

In this section the set of viruses is decomposed into the disjoint union of four principal types. The nature of so called 'Trojan horses' is considered.

Definition 6 *For all Gödel numberings of the partial recursive functions $\{\phi_i\}$, for all viruses v with respect to $\{\phi_i\}$, for all $i, j \in N$:*

i is pathogenic *with respect to v and j iff*

$$i = v(j) \ \& \\ (\exists d, p \in S)[\phi_j(d,p) \overset{v}{\ncong} \phi_i(d,p)]$$

i *is* contagious *with respect to* v *and* j *iff*

$$\begin{array}{l} i = v(j)\ \& \\ (\exists d, p \in S)[\phi_j(d,p) \overset{v}{\sim} \phi_i(d,p)] \end{array}$$

i *is* benignant *with respect to* v *and* j *iff*

$$\begin{array}{l} i = v(j)\ \& \\ i \textit{ is not pathogenic with respect to } j\ \& \\ i \textit{ is not contagious with respect to } j \end{array}$$

i *is a* Trojan horse *with respect to* v *and* j *iff*

$$\begin{array}{l} i = v(j)\ \& \\ i \textit{ is pathogenic with respect to } j\ \& \\ i \textit{ is not contagious with respect to } j \end{array}$$

i *is a* carrier *with respect to* v *and* j *iff*

$$\begin{array}{l} i = v(j)\ \& \\ i \textit{ is not pathogenic with respect to } j\ \& \\ i \textit{ is contagious with respect to } j \end{array}$$

i *is* virulent *with respect to* v *and* j *iff*

$$\begin{array}{l} i = v(j)\ \& \\ i \textit{ is pathogenic with respect to } j\ \& \\ i \textit{ is contagious with respect to } j \end{array}$$

When there exists a unique j such that $i = v(j)$ (e.g. when v is injective) then if i is pathogenic (contagious, benignant, a Trojan horse, a carrier, virulent) with respect to v and j, the reference to j will be dropped and i will be said to be pathogenic (contagious, benignant, a Trojan horse, a carrier, virulent) with respect to v.

Hence, if with respect to some virus an infected program is benignant, then it computes the same function as its uninfected predecessor. If it is a Trojan horse then it is incapable of infecting other programs. It can only imitate or injure, and under the right conditions it will do the latter. If it is a carrier, it is incapable of causing injury but under the right conditions it will infect other programs.

Definition 7 *For all Gödel numberings of the partial recursive functions* $\{\phi_i\}$, *for all viruses* v *with respect to* $\{\phi_i\}$:

v *is* benign *iff both:*

$$(\forall j \in N)[v(j) \textit{ is not pathogenic with respect to } v \textit{ and } j]$$
$$(\forall j \in N)[v(j) \textit{ is not contagious with respect to } v \textit{ and } j]$$

v *is* Epeian [2] *iff both:*

$$(\exists j \in N)[v(j) \textit{ is pathogenic with respect to } v \textit{ and } j]$$
$$(\forall j \in N)[v(j) \textit{ is not contagious with respect to } v \textit{ and } j]$$

v *is* disseminating *iff both:*

$$(\forall j \in N)[v(j) \textit{ is not pathogenic with respect to } v \textit{ and } j]$$
$$(\exists j \in N)[v(j) \textit{ is contagious with respect to } v \textit{ and } j]$$

v *is* malicious *iff both:*

$$(\exists j \in N)[v(j) \textit{ is pathogenic with respect to } v \textit{ and } j]$$
$$(\exists j \in N)[v(j) \textit{ is contagious with respect to } v \textit{ and } j]$$

The next theorem records some simple facts about types of viruses.

Theorem 1 *For all Gödel numberings of the partial recursive functions* $\{\phi_i\}$ *for all viruses* v *with respect to* $\{\phi_i\}$:

1. $(\exists j \in N)[v(j)$ *is benignant with respect to* v *and* $j]$

2. v *is benign iff*

$$(\forall j \in N)$$
$$[v(j) \textit{ is benignant with respect to } v \textit{ and } j]$$

[2]

Now shift your theme, and sing that wooden horse
Epeios built, inspired by Athena -
the ambuscade Odysseus filled with fighters
and sent to take the inner town of troy

The Odyssey of Homer, 8.492-495.
translation by Robert Fitzgerald
Doubleday & Co., NY, 1961

3. if v is Epeian then

$(\forall j \in N)$
$[[v(j)$ *is benignant with respect to* v *and* $j]$ *or*
$[v(j)$ *is a Trojan horse with respect to* v *and* $j]]$

4. if v is disseminating then

$(\forall j \in N)$
$[[v(j)$ *is benignant with respect to* v *and* $j]$ *or*
$[v(j)$ *is a carrier with respect to* v *and* $j]]$

Proof

Part 1 follows immediately from the recursion theorem.

All other parts follow immediately from the definitions.

□

Thus, all programs infected by a benign virus are benignant with respect to their uninfected predecessors. They function just as if they had never been infected. Viruses in this class appear to be the least threatening. This class includes many 'degenerate' viruses such as the identity function and 'padding' functions.

Programs infected by an Epeian virus can only be benignant or Trojan horses with respect to their uninfected predecessors. Further the latter option must sometimes occur. Epeian viruses will not be able to spread themselves; however, an infected program may imitate the 'intended' task of its uninfected predecessor until some 'trigger' causes it to do damage. Among the Epeian viruses are the 'degenerate' class of constant functions, which never imitate-or-infect but only injure.

Programs infected by a disseminating viruses can only be benignant or carriers with respect to their uninfected predecessors. Further the latter option must sometimes occur. Thus programs infected with such viruses are never pathogenic. However, it is worth noting that disseminating viruses may modify the size of programs or their complexity characteristics, and by this means become detectable or cause harm (or benefit as in the case of the compression virus). In fact, size and complexity may be important properties when considering viruses. An extension of the current theory to account for size and complexity seems appropriate (see §*further research*).

Malicious viruses can both spread and produce injuries. They appear to be the most threatening kind of virus. The 'case study virus' in §*basic definitions* is malicious.

Remark 2 *It may be appropriate to view contagiousness as a necessary property of computer viruses. With this perspective, it would be reasonable to define the set of viruses as the union of the set of disseminating viruses and the set malicious viruses, and to exclude benign and Epeian viruses altogether.*

4 Detecting The Set Of Viruses

The question of detecting viruses is addressed in the next theorem:

Theorem 2 *For all Gödel numberings of the partial recursive functions* $\{\phi_i\}$*:*

$$V = \{i|\phi_i \textit{ is a virus}\} \textit{ is } \Pi_2 - \textit{complete}$$

Proof

Let $T = \{i|\phi_i \text{ is a total}\}$. It is well known (§13 and §14 [Ro]) that T is Π_2- complete.

To establish that $T \leq_1 V$, let $j \in V$ (for example let j be an index for the identity function) and consider the function $g : N \rightarrow N$ such that for all $i, y \in N$:

$$g(i,y) = \begin{cases} \phi_j(y) & \text{if } \phi_i(y) \downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

Then g is a partial recursive function. Let k be an index for g, and let $f : N \rightarrow N$, be such that:

$$(\forall i \in N)[f(i) = s(k, 1, i)]$$

where s is as in the $s-m-n$ theorem [Ro].

Then f is a total recursive function and:

$$(\forall i, y \in N)[\phi_{f(i)}(y) = \phi_{s(k,1,i)}(y) = \phi_k(i,y) = g(i,y) = \begin{cases} \phi_j(y) & \text{if } \phi_i(y)\downarrow \\ \uparrow & \text{otherwise} \end{cases}\ ]$$

It follows that:

$$i \in T \Leftrightarrow f(i) \in V$$

Thus $T \leq_m V$. It follows, as in §7.2 [Ro], that $T \leq_1 V$ as desired.

To establish that $V \in \Pi_2$, consider the following formula for V which arises directly from the definition of virus:

$$\begin{array}{ll}
(\forall j)(\exists k,t) & [H(i,j,k,t)] \\
\& & \\
(\forall <d,p>) & [(\forall j_1,k_1,t_1) \\
 & [H(i,j_1,k_1,t_1) \Rightarrow \\
 & (\forall <e,q>,t_2)[\neg H(k_1,<d,p>,<e,p>,t_2)]] \\
 & \mathit{or} \\
 & (\forall j_1,k_1,t_1,j_2,k_2,t_2) \\
 & [[H(i,j_1,k_1,t_1)\ \&\ H(i,j_2,k_2,t_2)] \Rightarrow \\
 & (\exists <e,q>,t_3,t_4) \\
 & [H(k_1,<d,p>,<e,q>,t_3)\ \& \\
 & H(k_2,<d,p>,<e,q>,t_4)]] \\
 & \mathit{or} \\
 & (\forall j_1,k_1,t_1,<e,q>,t_2) \\
 & [[H(i,j_1,k_1,t_1)\ \&\ H(j_1,<d,p>,<e,q>,t_2)] \Rightarrow \\
 & (\exists <e',q'>,t_3,t_4) \\
 & [H(k_1,<d,p>,<e',q'>,t_3)\ \& \\
 & L(i,<e,q>,<e',q'>,t_4)] \\
 & \& \\
 & [H(i,j_1,k_1,t_1)\ \&\ H(k_1,<d,p>,<e,q>,t_2)] \Rightarrow \\
 & (\exists <e',q'>,t_3,t_4) \\
 & [H(j,<d,p>,<e',q'>,t_3)\ \& \\
 & L(i,<e',q'>,<e,q>,t_4)]]]
\end{array}$$

Where H is a 'step counting' predicate for $\{\phi_i\}$ such that:

$$\begin{array}{ll}
(\forall i,j,k) & \\
\text{if } \phi_i(j) = k \text{ then} & (\exists t)[H(i,j,k,t)] \\
\text{if } \phi_i(j) \neq k \text{ then} & (\forall t)[\neg H(i,j,k,t)]
\end{array}$$

And where L is a predicate for $\{\phi_i\}$ such that:

$$\begin{array}{ll} (\forall i, < e, q >, < e', q' >, t) & \\ \text{if } < e, q > \overset{\phi_i}{\sim} < e', q' > \text{ then} & (\exists t)[L(i, < e, q >, < e', q' >, t)] \\ \text{if } < e, q > \overset{\phi_i}{\not\sim} < e', q' > \text{ then} & (\forall t)[\neg L(i, < e, q >, < e', q' >, t)] \end{array}$$

Since for all acceptable Gödel numberings of the partial recursive functions $\{\phi_i\}$ it is easily seen that there exist recursive predicates H and L as above, it follows that $V \in \Pi_2$.

□

Thus detecting viruses is quite intractable, and it seems unlikely that protection systems predicated on virus detection will be successful.

5 Isolation As A Protection Strategy

As noted in [Col] isolating a computing environment from its surroundings is a powerful method of protecting it from viruses. For example, if no new programs can be introduced, no old programs can be updated, and no communication can occur, then it seems viruses are no threat.

Unfortunatly, such isolation is unrealistic in many computing environments. The next theorems explore the possibility of protecting computing environments with less severe forms of isolation.

Definition 8 *For all Gödel numberings of the partial recursive functions $\{\phi_i\}$, for all viruses v with respect to $\{\phi_i\}$, let:*

The infected set of v

$$I_v = \{i \in N | (\exists j \in N)[i = v(j)]\}$$

Definition 9 *For all Gödel numberings of the partial recursive functions $\{\phi_i\}$, for all viruses v with respect to $\{\phi_i\}$, v is* absolutely isolable *iff I_v is decidable.*

Clearly if a virus is absolutely isolable, then (at least in theory) it can be neutralized. Whenever a program becomes infected, it is detected and removed. The following is a simple fact about absolutely isolable viruses:

Theorem 3 *For all Gõdel numberings of the partial recursive functions $\{\phi_i\}$, for all viruses v with respect to $\{\phi_i\}$ if for all $i \in N$, $v(i) \geq i$ then v is absolutely isolable.*

Proof trivial.

□

Thus the case study virus, as implemented using the scheme in §*basic definitions* would be absolutely isolable. In fact, what little experience with viruses there is to date seems to suggest that in practice people who produce viruses begin by producing ones with the increasing property necessary for theorem 3 to apply. Unfortunately, not all viruses have this property. For example, with any reasonable compression scheme, the compression virus of §*basic definitions* would not have this property. Nonetheless, the compression virus is absolutely isolable. Given a program with the proper syntax, it is in the infected set if and only if decompressing the compressed part results in a legitimate program.

Is every virus absolutely isolable?

Regretably, the next theorem shows that the answer is no.

Theorem 4 *For all Gõdel numberings of the partial recursive functions $\{\phi_i\}$, there exists a total recursive function v such that:*

1. *v is a malicious virus with respect to $\{\phi_i\}$*
2. *I_v is Σ_1-complete.*

Proof

Let f be a total recursive function such that:

$$Rg(f) = K = \{i | \phi_i(i) \downarrow\}$$

Let $j_1 : N \rightarrow N$ be a $1-1$ total recursive function such that for all $i, x \in N$:

$$\phi_i = \phi_{j_1(i,x)} \qquad (1)$$

Such a function, known as a padding function, exists by Proposition 3.4.5 [MY].

Let $j_2 : N \rightarrow N$ be such that:

$$(\forall i, x \in N)[j_2(i, x) = j_1(i, y)]$$

where y is the least natural number such that, for all $i', x' \in N$ with $< i', x' ><< i, x >$, $j_2(i', x') < j_1(i, y)$.

Then j_2 is a monotonically increasing total recursive function and by (1), it follows that:

$$(\forall i, x \in N)[\phi_i = \phi_{j_2(i,x)}] \tag{2}$$

Let $j' : N \rightarrow N$ be such that for all $i \in N$:

$$j'(i) = \begin{cases} y + 1 & \text{if } i = j_2(1, y) \\ 0 & \text{otherwise} \end{cases}$$

Then since j_2 is monotonically increasing, it follows that j' is a total recursive function.

Consider the function $b_1 : N \rightarrow N$ such that for all $d, p \in S$ and $i, k \in N$:

$$\phi_{b_1(i,k)}(d, p) = \begin{cases} 0 & \text{if } d \text{ is even} \\ < e, [\phi_k(q)] > & \text{if } d \text{ is odd } \& \ \phi_i(d, p) =< e, [q] > \text{ and } \phi_k(q) \downarrow \\ \uparrow & \text{if } d \text{ is odd } \& \ \phi_i(d, p) =< e, [q] > \text{ and } \phi_k(q) \uparrow \\ \phi_i(d, p) & \text{otherwise} \end{cases}$$

where for all $q \in N$, $[q]$ denotes the one element sequence in S consisting only of q.

Then by standard arguments, b_1 is a total recursive function and:

$$(\forall i, x, k \in N)[\phi_{b_1(i,k)} = \phi_{b_1(j_2(i,x),k)}]. \tag{3}$$

Let $b_2 : N \rightarrow N$ be such that for all $i, k \in N$:

$$b_2(i, k) = \begin{cases} j_2(b_1(i, k), f(0)) & \text{if } j'(i) = 0 \\ j_2(b_1(1, k), f(y)) & \text{if } j'(i) = y + 1 \end{cases}$$

Then b_2 is a total recursive function and it follows from (2) and (3) that:

$$(\forall i, k \in N)[\phi_{b_2(i,k)} = \phi_{b_1(i,k)}]. \tag{4}$$

Applying the s-m-n theorem there exists a total recursive function g such that for all $i, k \in N$:

$$\phi_{g(k)}(i) = b_2(i, k)$$

By the recursion theorem, there exists an $h \in N$ such that for all $i \in N$:

$$\phi_h(i) = b_2(i, h)$$

Let $v = \phi_h$. Then v is a total recursive function since b_2 is.

Let $d, p \in S$, then using that fact that $v = \phi_h$ is a total recursive function and applying (4) gives:

$$\begin{aligned}
\phi_{v(i)}(d,p) &= \phi_{b_2(i,h)}(d,p) \\
&= \phi_{b_1(i,h)}(d,p) \\
&= \begin{cases} 0 & \text{if } d \text{ is even} \\ <e, [\phi_h(q)]> & \text{if } d \text{ is odd } \& \ \phi_i(d,p) =<e, [q]> \ \& \ \phi_h(q) \downarrow \\ \uparrow & \text{if } d \text{ is odd } \& \ \phi_i(d,p) =<e, [q]> \ \& \ \phi_h(q) \uparrow \\ \phi_i(d,p) & \text{otherwise} \end{cases} \\
&= \begin{cases} 0 & \text{if } d \text{ is even} \\ <e, [v(q)]> & \text{if } d \text{ is odd } \& \ \phi_i(d,p) =<e, [q]> \\ \phi_i(d,p) & \text{otherwise} \end{cases}
\end{aligned}$$

1 of the theorem now follows directly from the definition of malicious virus.

Since, for all total recursive functions m, $Rg(m)$ is recursively enumerable, it follows that $I_v = Rg(v) \in \Sigma_1$.

Let $c : N \to N$ be such that for all $x \in N$, $c(x) = j_2(b_1(1,h), x)$. Since j_2 is $1 - 1$ so is c. Then $x \in K$ implies the existence of a $y \in N$ such that $f(y) = x$. Let $i = j_2(1, y)$, then:

$$c(x) = j_2(b_1(1,h), x) = j_2(b_1(1,h), f(y)) = b_2(i,h) = v(i) \in I_v$$

On the other hand, assume $x \notin K$ and $c(x) \in I_v$. Then there exists an $i \in N$ such that:

$$j_2(b_1(1,h),x) = c(x) = v(i) = b_2(i,h) = \begin{cases} j_2(b_1(i,h),f(0)) & \text{if } j'(i) = 0 \\ j_2(b_1(1,h),f(y)) & \text{if } j'(i) = y+1 \end{cases}$$

Since j_2 is $1-1$, it follows that $x = f(y) \in K$. $\Rightarrow\Leftarrow$. Hence, $K \leq_1 I_v$ and 2 of the theorem holds.

□

Thus, for the viruses described in the previous theorem, protection cannot be based upon deciding whether a particular program is infected or not. Paradoxically, despite this, it is often possible to defend against such viruses. How such a defense could be mounted will be described below; however, a few definitions are in order first.

Definition 10 *For all Gōdel numberings of the partial recursive functions $\{\phi_i\}$, for all viruses v with respect to $\{\phi_i\}$, let:*

The germ set of v

$$G_v = \{i | i \in N \ \& \ (\exists j \in N)[\phi_i = \phi_{v(j)}]\}$$

Thus the germs of a virus are functionally the same as infected programs, but are syntactically different. They can infect programs, but cannot result from infection. They may start 'epidemics', but are never propagated with them.

Definition 11 *For all Gōdel numberings of the partial recursive functions $\{\phi_i\}$, for all viruses v with respect to $\{\phi_i\}$, v is* isolable within its germ set *iff there exists an $S \subseteq N$ such that:*

1. *$I_v \subseteq S \subseteq G_v$.*
2. *S is decidable.*

Notice that if a virus is isolable within its germ set by a decidable set S, then not allowing programs in the set S to be written to storage or to be communicated will stop the virus from infecting. Further, the isolation of some uninfected germs by this process appears to be an added benefit.

Returning now to the viruses described in the previous theorem: assume that the function b_1 above had the property that for all i, k, $b_1(i, k) >< i, k >$. The proof of the previous theorem could easily have been modified to assure this. Further, in Godel numberings derived in the usually fashion from natural programming languages, a b_1 constructed in a straightforward manner would have this property. Consider the set

$$S = \{j_2(b_1(i, h), y) | i, y \in Z_{>0}\}$$

By the monotonically increasing property of j_2 and the property of b_1 which is being assumed, S is decidable. On the other hand if $a \in I_v$ then there exists i such that

$$a = v(i) = b_2(i, h) = \begin{cases} j_2(b_1(i, h), f(0)) & \text{if } j'(i) = 0 \\ j_2(b_1(1, h), f(y)) & \text{if } j'(i) = y + 1 \end{cases}$$

And it follows that $a \in S$. On the other hand if $a \in S$ then there exist an y, i such that

$$a = j_2(b_1(i, h), y)$$

By (2) and (4):

$$\phi_a = \phi_{j_2(b_1(i,h),y)} = \phi_{b_1(i,h)} = \phi_{b_2(i,h)} = \phi_{v(i)}$$

And hence $a \in G_v$ as desired.

Thus viruses like the ones in theorem 4 demonstrate that decidability of I_v is sufficient but not necessary for neutralization. Apparently, more work needs to be done before a clear idea of the value of isolation will emerge. Are all viruses isolable within their germ set? The answer is no (proof omitted). Are all disseminating viruses isolable within their germ set? The answer is not known. Are there notions of isolation which provide significant protection at a reasonable cost?

6 Further Research

The study of computer viruses is embryonic. Since so little is known, virtually any idea seems worth exploring.

Listed below are a few avenues for further investigation.

1. *Complexity theoretic and program size theoretic aspects of computer viruses.*

Introduce complexity theory and program size theory into the study of computer viruses. As noted earlier, even disseminating viruses may affect the complexity characteristics and size of infected programs and as a result become detectable or harmful.

Complexity theory and program size considerations can be introduced at a abstract level (see for example [MY]) or a concrete level.

For example, viruses in the 'real world' would probably have the property that the running time of an infected program, at least while imitating or infecting, would be at most polynomial (linear) in the running time of its uninfected precursor. Does this class of 'polynomial (linear) viruses' pose a less serious threat? Do NP-completeness considerations, or cryptographic considerations come into play?

2. *Protection Mechanisms*

In this paper one form of protection mechanism, isolation, was briefly considered. In addition to considering isolation in greater depth, numerous other possibilities exist. For example:

Quarinteening
Is there value in taking a new program and running it in a safe environment for a while before introducing it into an environment were it could spread or do harm? For example, putting the new program on an isolated machine with dummy infectable programs and with a variety of settings of the system clock might evoke behavior indicative of infection. In particular would this be helpful with the class of polynomial viruses or linear viruses?

Disinfecting
Under what circumstances can an infected program be disinfected? Certainly when a virus is absolutely isolable there exists a procedure which when given an infected program will return a program which 'infects to' the original one. How general is this phenomena?

Certificates
Can some programs be given a 'clean bill of health'? For example, if it is know that a certain virus is about, would it be possible for a vendor to 'prove' that his program was not in the germ set? Would it be possible to prove that the software was not in the germ set of a large class of viruses?

Operating System Modification

Could modifications to the operating system provide some protection. For example, assume that the (secure) operating system required that the user 'initiate' all new programs by designating the files which the program is given the privilege to read and write. Then, for example, a simple program (e.g. a game) could be given only the privilege to read and write files it creates. If the program was uninfected it might perform satisfactorily under this constraint. If however the program was infected, this constraint might severely limit the damage due to the virus. (This example arose during joint work with K. Kompella).

3. *Other Models Of Computer Viruses.*

The notion of computer viruses presented here is not the only one possible. It was selected because it seemed to be an adequate place to begin an investigation. More general, and more restrictive notions are possible. Indeed it seems possible that no definition will conform to everyone's intuitions about 'computer viruses'.

More 'machine dependent' approaches could be considered. Approaches which take into account the communications channels over which viruses pass seem particularly important.

One interesting generalization of the current notion is inspired by [Col], where viruses are assumed to be capable of evolving. The 'Mutating Viruses' (μ-viruses) partially defined next are an attempt to capture this property.

Definition 12 *For all $z, z' \in N$, for all $p, p', q = q_1, q_2, ..., q_z, q' = q'_1, q'_2, ..., q'_{z'}$ S, for all sets H of partial functions from N to N, $< p, q > \overset{H}{\sim} < p', q' >$ iff:*

(a) $z = z'$ and

(b) $p = p'$ and

(c) there exists an i, with $1 \leq i \leq z$ such that $q_i \neq q'_i$ and

(d) for $i = 1, 2, ..., z$, either

i. $q_i = q'_i$ or

ii. there exists an $h \in H$ such that $h(q_i) \downarrow$ and $h(q_i) = q'_i$.

Definition 13 *For all sets of partial functions H from N to N, for all partial $f, g : N \rightarrow N$, for all $s, t \in S$, $f(s,t) \overset{H}{\sim} g(s,t)$ iff $f(s,t)\downarrow$ & $g(s,t)\downarrow$ & $f(s,t) \overset{H}{\sim} g(s,t)$.*

Definition 14 *For all sets of partial functions H from N to N, for all partial $f, g : N \rightarrow N$, for all $s, t \in S$, $f(s,t) \overset{H}{\cong} g(s,t)$ iff $f(s,t) = g(s,t)$ or $f(s,t) \overset{H}{\sim} g(s,t)$.*

Definition 15 *For all Gödel numberings of the partial recursive functions $\{\phi_i\}$, a set M of total recursive functions is a* mutating virus, *μ-virus, with respect to $\{\phi_i\}$ iff both:*

(a) for all $m \in M$, for all $d, p \in S$ either:

i. Injure:

$$(\forall i, j \in N)[\phi_{m(i)}(d,p) = \phi_{m(j)}(d,p)]$$

ii. Infect or Imitate:

$$(\forall j \in N)[\phi_j(d,p) \overset{M}{\cong} \phi_{m(j)}(d,p)]$$

Some computer viruses which have recently caused problems (e.g. the so called 'Scores virus' [Up] which attacked Macintosh computers) are μ-viruses and not just viruses. Hence this generalization of the notion of virus may be of more than theoretical interest.

This is only a partial definition because some notion of 'connectivity' is needed. That is, the union of two μ-viruses, neither of which 'evolves' into the other should not be a μ-virus. Many definitions of 'connectivity' can be defined, but further study will be required to choose those which are most appropriate. Once an appropriate choice is made, an important question will be whether the set of infected indices of a μ-virus can be harder to detect than those of a virus.

4. *Computer Organisms.*

This issue has evolved during joint work with K. Kompella.

There appear to be programs which can reproduce or reproduce and injure but which are not viruses (e.g. programs which just make copies of themselves but never 'infect'). These 'computer organisms' may be a serious security problem.

It may be appropriate to study 'computer organisms' and treat 'computer viruses' as special case.

7 Acknowledgments

I would like to thank Dean Jacobs, and Gary Miller for contributing their ideas to this paper.

I would also like to thank two of my students: Fred Cohen and Kireeti Kompella. Cohen brought the threat of computer viruses to my (and everyone's) attention. Kompella has spent many hours reviewing this work and has made numerous suggestions which have improved it.

References

[Co1] Cohen F. Computer Viruses. Ph.D. dissertation, University of Southern California, Jan. 1986.

[Co2] Cohen F. Computer Viruses - Theory and Experiments. Computers and Security 6 (1987) 22-35. North-Holland.

[MY] Machtey M, Young P. An introduction to the general theory of algorithms. North-Holland, NY 1978.

[Ro] Rogers, H Jr. Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Co., NY 1967.

[Up] Upchurch, H. The Scores Virus, unpublished manuscript , 1988.