A Protection Mechanism Against Pure Worms

by Dr. Frederick B. Cohen ‡

Abstract

In this paper, we propose a viable protection mechanism against 'pure worms' based on limited transitivity. We show that this method is effective, that it is feasible and practical in the current computing environment, and that it has little or no impact on current computing practice.

Search terms: Computer Viruses, Computer Worms

‡ Funded by ASP, PO Box 81270, Pittsburgh, PA 15217, USA

1 Introduction

A formal definition of worms was recently published [1], and many of the properties of computer viruses were show to be true of worms. [1] A particular case called 'pure worms' is of special interest because most of the current network based worms appear to be pure or very nearly pure worms [2].

One of the important results derived for worms from earlier work on viruses [3,4] is that the only 'perfect' defenses against worms are limited function, limited sharing, and limited transitivity of information flow. In modern computer systems, limited function is impractical in all but the simplest circumstances and in practice is rarely used. Limited sharing was shown, in general, to lead to a partially ordered set of information domains, and in almost all modern systems, there is only a single protection domain once transitivity is taken in to consideration, even though access controls are in place. In the general case, limited transitivity is impractical because the interaction of information from different sources in modern computer systems and networks is so prevalant and vital to operation, that any limitation we place on transitivity is very quickly reached, resulting in a rapid move toward isolation.

In this paper, we will show that in the case of 'pure worms', limited transitivity does not have these failings, and further, that a limited transitivity defense against pure worms is both practical and easily implemented in the current networked computing environment. We will give several examples of how to implement this worm defense in current networks, show that its time and space requirements are very small, and demonstrate that the mechanism as a whole provides protection even in cases where select computers in the network do not cooperate in the protection process.

[1] In simplest terms, a worm is defined as a program that automatically replicates and initializes interpretation of a replica.

2 Some Formalities

We begin by reciting the definition of worms and summarizing previous results [1]. Computing machines '$\mathcal{M}$' are defined as:

$$\forall M[M \in \mathcal{M}] \Leftrightarrow$$
$$M : (S_M, I_M, O_M : S_M \times I_M \rightarrow I_M, N_M : S_M \times I_M \rightarrow S_M, D_M : S_M \times I_M \rightarrow d)$$

where:

$\mathcal{N} = \{0 \ldots \infty\}$	(the 'natural' numbers)
$\mathcal{I} = \{1 \ldots \infty\}$	(the positive 'integers')
$S_M = \{s_0, ..., s_n\}, n \in \mathcal{I}$	($\mathcal{M}$ states)
$I_M = \{i_0, ..., i_j\}, j \in \mathcal{I}$	($\mathcal{M}$ tape symbols)
$d = \{-1, 0, +1\}$	($\mathcal{M}$ head motions)
$\$_M : \mathcal{N} \rightarrow S_M$	($\mathcal{M}$ state over time)
$\Box_M : \mathcal{N} \times \mathcal{N} \rightarrow I_M$	($\mathcal{M}$ tape contents over time)
$P_M : \mathcal{N} \rightarrow \mathcal{N}$	(current $\mathcal{M}$ cell at each time)

The 'history' of the machine H_M is given by $(\$, \Box, P)$,[2] the 'initial state' is described by $(\$_0, \Box_{0,i}, P_0), i \in \mathcal{N}$, and the set of possible $\mathcal{M}$ tape sub-sequences is designated by I^*. We say that; M is halted at time $t \Leftrightarrow \forall t' > t, H_t = H_{t'}, (t, t' \in \mathcal{N})$; that M halts $\Leftrightarrow \exists t \in \mathcal{N}, M$ is halted at time t; that p 'runs' at time $t \Leftrightarrow$ the 'initial state' occurs when P_0 is such that p appears at $\Box_{0,P_0}$; and that p runs $\Leftrightarrow \exists t \in \mathcal{N}, p$ runs at time t.

We define a 'Worm Set' $\mathcal{W}$ as a $\mathcal{V}$ in which, $\forall H_M$ in which any $w \in W$ appears under the tape head with $\$_M = \$_0$ at some move i, the tape head appears over some $w' \in W$ at some other place on the tape at some move i' with $\$_M = \$_0$, $(i' > i)$:

$$\forall M \forall W (M, W) \in \mathcal{W} \Leftrightarrow$$
$$[W \subset I^*] and [M \in \mathcal{M}] \ and \ \forall w \in W \forall H \forall t, j \in \mathcal{N}$$
$$[[P_t = j] \ and \ [S_t = S_0] \ and \ (\Box_{t,j}, ..., \Box_{t,j+|w|-1}) = w] \Rightarrow$$
$$\exists w' \in W \exists t', t'', t''', j' \in \mathcal{N}, t' > t$$
$$1)[[(j' + \mid w' \mid) \leq j] or [(j + \mid w \mid) \leq j']] \ and$$
$$2)[(\Box_{t',j'}, ..., \Box_{t',j'+|w'|-1}) = w'] \ and$$
$$3)\exists t''[t < t'' < t'] \ and \ [P_{t''} \in j', ..., j' + \mid w' \mid -1]$$
$$4)\exists t'''[t' < t'''] \ and \ [P_{t'''} = j'] and [S_{t'''} = S_0]$$

It was previously shown that $\mathcal{W}$ is a subset of the 'Viral Set' $\mathcal{V}$ [3], and we refer to members of $W, (M, W) \in \mathcal{W}$ for a given machine M as 'worms' on M. We typically drop the 'on M' when we are referring to a particular M, and make statements like "all worms

[2]For convenience, we drop the M subscript when we are dealing with a single machine except at the first definition of each term.

are viruses".

The 'Universal Protection Machine' (UPM) [2] can be used to model multiuser environments with protection. The UPM is implemented on a universal $\mathcal{M}$, and processes 'moves' from each of a set of $\mathcal{M}$s defined over the tape by using a 'scheduler' and a 'special state' which implements 'system calls'. This machine consists of an 'interpretation unit'; a set of 'subjects' $S = (s_0, \ldots, s_i), i \in \mathcal{I}$; a set of 'objects' $O = (o_0, \ldots, o_j), j \in \mathcal{I}$; a protection matrix $(S \times O \rightarrow R)$ which specifies a set of generic 'rights' $R = (r_0, \ldots, r_k), k \in \mathcal{I}$ of subjects over objects [8]; and a potentially infinite 'run sequence' $(\mathcal{R})$ of objects to be interpreted for each subject $(\mathcal{R} = ((s,o)_0, \ldots, (s,o)_l), l \in \mathcal{I})$.

When a subject interprets an object (i.e. $(s,o) \in \mathcal{R}$), M uses the rights of s for the duration of the interpretation of o. We are particularly interested in the rights r (read) and w (write), because these translate into the flow (f) of information between subjects. (i.e. $s_x w o_a \ and\ s_y r o_a \Rightarrow s_x f s_y$) [10]. This is alternatively expressed as:

$$(s_x, o_a) \in w \ and\ (s_y, o_a) \in r \Rightarrow (s_x, s_y) \in f.$$

Information flow is transitive (i.e. $s_x f s_y \ and\ s_y f s_z \Rightarrow s_x f s_z$) when M is a universal $\mathcal{M}$ [2], and using this model, a vital result that viruses can spread to the transitive closure of information flow from the source subject was derived [3]. This is because there exists an $\mathcal{R}$ in which each subject in the transitive closure of information flow, in turn, interprets an object modified by the virus interpreted by a subject previously in the information flow from the original virus source. In any 'fair' scheduler with unbounded work to be done and assuming that all accessible programs are run with some non-zero frequency, such an $\mathcal{R}$ will eventually be realized because there will always be a partial subsequence of $\mathcal{R}$ in which the each of the necessary objects will be interpreted in sequence. It turns out that the same result is, in general, true for worms, but transitivity doesn't result simply from running replicas. That is, if no object is modified by subjects running the worm, and if we ignore all other causal factors (e.g. a progenator of a worm is run by some other user independently of the actions of the worm under consideration), only subjects with direct access to the worm can run it. Mathematically:

$$(M,W) \in \mathcal{W}, \forall a \in W, \forall b \notin W, \forall \mathcal{R}, \forall s \in S; \nexists (s,b) \in \mathcal{R} \ and\ \forall (s,a) \in \mathcal{R}, (s,b) \notin w \Rightarrow \nexists H_M : b \in W$$

We will call these worms, 'pure worms' because when they are interpreted, they do not spread transitively by replicating into other objects. By definition, s runs $a \Rightarrow (s,a) \in r$, so

$$\text{Theorum B:} \forall \text{ pure worms } a \forall s \in S : (s,a) \notin r \Rightarrow \nexists H : s \text{ runs } a$$

We anthropomorphise objects containing worms by saying that a worm (w) has been granted the rights of a subject $(s) \Leftrightarrow s$ runs $w \in W, (M,W) \in \mathcal{W}$, and we express this as $w \leftarrow s$. If only the creator of a pure worm has direct access to it, it follows that the rights

[3] page 35 [2]

of all replicas will be limited to the rights of the originator, since only the originator can run it, and by definition, rights are only extended to objects by virtue of the subjects that interpret them. More generally, the rights granted to replicas of pure worms are limited to the rights of the subjects who run them:

Lemma B.1: $\forall$ pure worms $a, w \leftarrow s \Leftrightarrow s$ runs w

3 The Limited Transitivity Approach

Limiting the transitivity of information flow is at least NP-complete [5] because it requires tracking the history of the effect of every bit on every other bit throughout a computer system. In cases where general purpose operations on values in multiple POset domains are permitted, even some operations that should be permitted cannot be permitted while still maintaining protection [4], and in these cases, protection moves toward isolation. In a more practical sense, there is a great deal of uncontrolled mixing of information in most modern information systems, and limited transitivity very quickly moves us to a situation where all information flow ceases up or we have to solve NP-complete problems very often.

The reason that limited transitivity causes these problems is because of the mixing of information between domains and the need to accurately characterize the constituent domains of any particular piece of information. This is not the case for pure worms, however, because by definition, pure worms do not cause the mixing of information. Since no output that can be input by another domain is generated by a pure worm, we can treat the domain from which the pure worm was launched as the SUP of a POset. The transitive closure of information flow of the SUP of a POset is the SUP of the POset, and thus we have no complexity problem in computing transitive closure at the domain level. If we make the simplifying and relatively accurate assumption, with respect to pure worms, that each computer in the network corresponds to a single domain, we can then proceed as follows.

In modern computer networks such as the Internet, the spread of pure worms between machines is normally performed via a relatively well controlled remote program execution facility available at the operating system level, and the operating systems have adequate protection to prevent the vast majority of attackers from creating their own operating system calls. Thus, a single system call is normally involved in the spreading of a pure worm between machines.

The Internet remote program execution facility is not normally used by programs that perform network-wide operations through numerous remote machines, and the authorization granted by machines for incoming remote execution is limited to a very small portion of the total internet population. This makes the average minimum 'distance' between machines through successive remote execution relatively long, while making the maximum 'distance' normally used by legitimate users and programs relatively small.

Assuming the difference between the average minimum distance of domains and the legitimate distance is sufficient, we can exploit this difference to limit the impact of pure worms. The method is quite simple. We keep a distance count on each process, and increase the distance every time a remote execution call is performed. The current distance is sent with each remote execution call and the machine processing the call decides whether to accept the request as a function of the current distance and the local policy. This technique requires only a few bits per process and a few instructions per remote program execution, and is thus very low complexity and has no substantial impact on system or network performance.

The question remains of how to set transitivity thresholds. A network-wide maximum distance of 3, for example, might allow almost all normal use to proceed unhindered, while limiting the impact of a pure worm to a small subset of the network. This would avoid massive network collapse and make the task of tracking down the source of a pure worm far easier.

Let's take an example. Suppose we have a network with 100,000 nodes, 10 of which are available for remote program execution from any given user ID on any machine in the network. In the worst case, a pure worm could spread to 10 computers at distance 1, 100 computers at distance 2, and 1,000 computers at distance 3. Thus a pure worm could impact only 1.111% of the network in the worst case. Once such a worm is detected, we can easily track down the source by determining which of the network machines had a distance of 3 or less from each of the impacted nodes. This would dramatically limit the amount of searching required to find the machine from which the worm was launched, and thus speed the process of correcting the problem and tracking the attacker while protecting the privacy of people using machines that are not the source of the attack.

An important side effect of limited transitivity in remote login and execution is the reduction of transitive attacks by human attackers. For example, the attacks described by Stohl [6] and many others are based on exploiting weak systems to extend access to stronger systems. The extension is possible because weak systems are granted access to stronger systems through remote execution and other similar mechanisms. An attacker simply tries systems until a weak system is located, and then extends the privileges granted by this system to other systems. After extending as far as possible through this technique, other weak systems are sought through the newly acquired access paths, and the attack is extended.

By limiting transitivity, an attacker has to find far more direct paths to systems under attack, thus reducing the problem of tracking the attacker, dramatically limiting the scope of the attack, and forcing the attacker to penetrate far more systems directly rather than through privilege extension.

4 The Real Impact of Limited Transitivity

5 Summary, Conclusions, and Further Work

We have shown that limiting the transitivity of remote procedure calls is an efficient and effectivce means of limiting the impact of pure worms as well as many current human attacks on modern information networks. This technique also appears to have little or no impact on network operations performed by legitimate users and current automated systems for mail delivery and other automated network facilities.

We see no reason that this technique cannot or should not be implemented immediately on the vast majority of the worlds networks. The only impediment is the will to proceed.

In theory this idea works quite well, but from a practical standpoint, we must consider the real impact of such a mechanism on current network operations. The major unanswered questions are what the actual distances are and whether the difference between them is sufficient to provide effective protection. A study of this sort is beyond the scope of this paper, and since access to this sort of information is not currently made available to external researchers, it will, for the moment, remain the task of each administrator to perform their own analysis.

It appears that in the standard Unix environment, this idea is practical, but the remote file system sharing mechanism now being implemented in some portions of the worldwide computing environment tightens still further, the link between machines, and blurs the distinction between machines to the point where they are almost indistinguishable. We anticipate that these systems will continue to be vulnerable because they make the distinction between the legitimate activities of normal users and the illegitimate activities of pure worms nearly impossible.

In the most common DOS networks providing remote program execution, the present mechanism is also ineffective because the commercial programs that provided remote program execution, until very recently, provided no protection against unauthorized remote program execution, and the protection provided on modern PC based networks is simply inadequate to prevent even a pure worm from spreading unhindered.

The examples we have given here are only the tip of the iceburg. A variety of mechanisms should be studied to address such issues as network-wide defaults, special case exceptions, network analysis tools that yield local maps of reachable machines and machines that can reach the node under analysis, and the current situation with regard to the transitive distance of network privileges. Further studies would be useful to analyze the detailed behavior of existing networks so that minimal values for transitive flow could be used.

6 References

- **1)** F. Cohen, “A Formal Definition of Computer Worms and Some Related Results” IFIP-TC11, “Computers and Security”, 1992
- **2)** T. Longstaff and E Schultz. “ ” IFIP TC-11, ‘Computers and Security’, 1992
- **3)** F. Cohen, “Computer Viruses - Theory and Experiments”, DOD/NBS 7th Conference on Computer Security, originally appearing in IFIP-sec 84 (1984), also appearing as invited paper in IFIP-TC11, “Computers and Security”, V6#1 (Jan. 1987), pp 22-35 and other publications in several languages.
- **4)** F. Cohen, “Computer Viruses”, ASP Press, PO Box 81270, Pittsburgh, PA 15217 USA, 1985, subsequently approved as a dissertation at the University of Southern California, 1986.
- **5)** D. Denning, “Cryptography and Data Security”, Addison Wesley, 1982.
- **6)** C. Stohl, ‘The Cookoos Egg’