Need advice on Anti-Viral CRC schem

31 May 1990, 23:47:54

Gordon LETWIN

to

Microsoft is working on new versions of DOS and one of the things that I'm interested in is virus protection. Naturally, it will be impossible to produce iron-clad virus protection in a non-protected DOS environment, but I think we can produce some tools which - when properly applied - can be a big help.
I've been thinking about using CRCs to see if a file has been diddled. I want to consider just the subset problem of determining if a file has been modified; we can consider seperately how we'll apply this in real life and how we'll keep a virus from compromosing the checker program itself or it's list of "correct values".

It's easy for a virus to modify a file and - given a known CRC algorithm - preserve the CRC. But what about preserving two different CRCs simultaneously? What about preserving 16?

Specifically, what if the checker program knows - say - 4096 different CRC-32 polynomials. And what if the program - on a per user basis at install time - randomly picks 16 of those polynomials, computes them for the file, and stores them in some as-yet-unspecified "safe" way.

I know what a CRC is, but my math is too weak to know if there is an easy attack on the problem of changing the file so that 16 CRC-32s are the same. I hope that someone out there can help me with this problem. Can we consider 16 CRC-32s to be the same as 1 CRC-256? A CRC-256 solves the "add 1 to a location, compute the CRC, repeat until match" problem by making it too slow. And the program can't precompute the change, of course, because each user has a DIFFERENT CRC-256. However, if one can write a program which figures out the corresponding CRC-256 given the 16 CRC-32s, and then can use math - rather than brute force - to figure out the proper file "diddle values" to make that CRC-256 be unchanged, the scheme would be vulnerable.

Or, are 16 CRC-32s a different creature then a CRC-256? Please help with this analysis.

Regardless of this specific detail, the approach that I'm exploring is this:

1) in an unprotected world, a smart guy can unravel anything.

2) An attacking program, however, is not nearly as smart as a smart guy. If you can design your system so that every user is protected differently, in such a fashion that the entire set of protection mechanisms can't be "solved" in advance by this smart guy, then he's forced to have his program itself do the analysis, and given the state of AI it's not too hard to fool a program forever.

3) in all cases, we assume that the "smart [virus writing] guy" knows all

the possible CRCs, but he doesn't know which set will be used on
any particular user's machine. Assume that the virus program can
find out which ones are being used.

comments?

Thanks

Gordon Letwin
Microsoft

---

1 Jun 1990, 19:05:38
Bennet Yee
to

In article <54...@microsoft.UUCP>, gor...@microsoft.UUCP (Gordon

LETWIN) writes:
|> Microsoft is working on new versions of DOS and one of the things that
|> I'm interested in is virus protection.
|> ...

|> I've been thinking about using CRCs to see if a file has been diddled.
|> I want to consider just the subset problem of determining if a file has
|> been modified; we can consider seperately how we'll apply this in
real life and
|> how we'll keep a virus from compromosing the checker program itself or it's
|> list of "correct values".
|>
|> It's easy for a virus to modify a file and - given a known CRC algorithm -
|> preserve the CRC. But what about preserving two different CRCs
simultaneously?
|> What about preserving 16?
|>
You should read the papers (the references for which follows) about
fingerprinting using random irreducible polynomials and computing residues
in the field GF(2^p). The technique of fingerprinting allows you detect
modifications to a file. Since you can generate irreducible polynomials on
the fly, you don't need to have a static table of CRC polynomials and
corresponding checksums -- you could generate new moduli/residue pairs
yourself once you know that your copy is clean. If your source of random
numbers is cryptographically secure, then the tables of irreducible/residues
will differ -- unpredictably -- among all copies of your software, making
the attack where an attacker takes your table off-line to crack impractical.

Like CRCs, the irreducible polynomial that you're using should be kept
secret, but that's not always practical. If your system model admits a
trusted operator & a secure machine room, then you're okay; if you're just
talking about PCs where the user has physical access, you can't keep
secrets. Even if you can't keep the irreducible moduli secret, however,
using random irreducibles still wins over CRCs for the reasons I mentioned
above.

The algorithms can be implemented efficiently. I've hacked up a version in C for our Strongbox project, and have gotten very good throughput: with an IBM APC/RT, I can get 800 or 900 kbytes/sec, depending on the size of tables you want to use. This is disk I/O speeds or better (at least for the current generation of disks), so if your system would otherwise be blocked waiting for the transfer to complete, the cost is entirely hidden.

Doug Tygar (my advisor) and I gave a talk at Microsoft about a year ago about our research; I don't remember off hand who we talked to though.

-bsy

Bennet S. Yee	b...@cs.cmu.edu
School of Computer Science	bsy%cs.cm...@vma.cc.cmu.edu
Carnegie Mellon	bsy%cs.cm...@relay.cs.net
Pittsburgh, PA 15213-3890	+1 412 268 7571

@TechReport(Rabin,
Author="Michael Rabin",
Institution="Center for Research in Computing Technology, Aiken Laboratory, Harvard University",
Title="Fingerprinting by Random Polynomials",
Key="Rabin",
Year=1981,
Month="May",
Number="TR-81-15")

@article(RabinFF,
Key="Rabin",
Author="Michael O. Rabin",
Title="Probabilistic Algorithms in Finite Fields",
Year=1980,
Volume=9,
Journal="SIAM Journal on Computing",
Pages="273-280")

@TechReport(StrongboxTR,
Key = "Yee et al.",
Author = "Bennet S. Yee and J. Douglas Tygar and Alfred Z. Spector",
Title = "Strongbox: A Self-Securing Protection System for Distributed P rograms",
Institution = "Carnegie Mellon University",
Number = "CMU-CS-87-184",
Month = January, Year = 1988)

---

1 Jun 1990, 05:38:05
A J Annala

to
In article <54987> gor...@microsoft.UUCP (Gordon LETWIN) writes:
>Microsoft is working on new versions of DOS and one of the things that
>I'm interested in is virus protection.
>
Don't bother trying to use CRC's as a protection mechanism. My company
is working on a fully encrypted system for running DOS where code is only
decrypted when it is ready to be run (and when it is in a protected cache
memory) ... the same goes for data ... a virus would be hard put to get
into this system in the first place ... and it would be hard put to try
to modify disk resident code without knowing the master and subsidiary
crypto keys used to encode the boot block/FAT/directories/files/raw disk
space. Our approach is somewhat slower then traditional unencrypted raw
disk access; however, it is considerably more secure against viruses and
unauthorized intruders. Current generation 20-25 Mhz PC's hide the crypto
overhead pretty well. Hardware assist can be made available to hide the
crypto overhead on slower machines.

---

1 Jun 1990, 21:45:08
Jeffrey M. Keller
to
In article <54...@microsoft.UUCP> gor...@microsoft.UUCP (Gordon LETWIN) writes:
[bulk of message deleted]

>1) in an unprotected world, a smart guy can unravel anything.
Well, the "smart guy" may be able to subvert any executable code to
which he has access, but he cannot break a secure encryption scheme.
Unfortunately, protecting a program is more involved than protecting
data; perhaps this is what you meant.

[more points deleted]

>comments?
>
> Thanks
>
> Gordon Letwin
> Microsoft

It seems that what you really want is an efficiently computable,
foolproof hash function by which to verify the integrity of a file.
(By "foolproof", I mean that it is "infeasible" to produce an altered
message with the same hash value, i.e. to fool the hash function.)
CRC is certainly efficient; that's why it's used. But it is not (and
is not designed to be) foolproof, and it is probably a mistake to try
to make it so.

An example of such a hash function is RSA's MD4 Message Digest Algorithm,
described on this group a couple months ago. It may be more foolproof
and less efficient than you want, but it looks like a good example.
(Actually, Rivest indicates that it takes ~15 instructions/byte on a

32-bit machine, which is quite respectable.)

For communications purposes, a nice thing to do with such hash values
is to public-key encrypt them, thereby authenticating the message to
anybody with the public key without the overhead of encrypting/decrypting
the whole message (MD4 being much faster than most encryption schemes).
At first glance, this would seem perfect for your task. Unfortunately,
all keys, hash-values, and verification routines must be available for use,
somewhere. If you have devised a scheme to store these "safely enough",
then you win.

In article <25...@usc.edu> ann...@neuro.usc.edu (A J Annala) writes:
Don't bother trying to use CRC's as a protection mechanism.

>...My company

>is working on a fully encrypted system for running DOS where code is only
>decrypted when it is ready to be run (and when it is in a protected cache
>memory) ... the same goes for data ... a virus would be hard put to get
>into this system in the first place ... and it would be hard put to try
>to modify disk resident code without knowing the master and subsidiary
>crypto keys used to encode the boot block/FAT/directories/files/raw disk
>space. Our approach is somewhat slower then traditional unencrypted raw
>disk access; however, it is considerably more secure against viruses and
>unauthorized intruders. Current generation 20-25 Mhz PC's hide the crypto
>overhead pretty well. Hardware assist can be made available to hide the
>crypto overhead on slower machines.

Again, if one only cares about the integrity of the data, and not the
privacy of same, encrypting everything seems wasteful. (However, Annala
may well care about both.) Also, there remains the problem of protecting
the protection scheme, but it sounds like Annala is dealing with this,
somehow.

--
Jeff Keller kel...@saturn.ucsc.edu (408)425-5416
THIS LIFE IS A TEST. IT IS ONLY A TEST. HAD THIS BEEN A REAL LIFE,
YOU WOULD HAVE BEEN GIVEN INSTRUCTIONS ON WHERE TO GO AND WHAT TO DO.

---

2 Jun 1990, 23:34:11
Bennet Yee
to

In article <39...@darkstar.ucsc.edu>, kel...@saturn.ucsc.edu (Jeffrey M.

Keller) writes:
|> In article <54...@microsoft.UUCP> gor...@microsoft.UUCP (Gordon
LETWIN) writes:
|> [bulk of message deleted]
|> >1) in an unprotected world, a smart guy can unravel anything.
|>

|> Well, the "smart guy" may be able to subvert any executable code to
|> which he has access, but he cannot break a secure encryption scheme.
|> Unfortunately, protecting a program is more involved than protecting
|> data; perhaps this is what you meant.
You can not do encryption unless you have a trusted operator who knows the key and can keep it secret. In an unprotected environment, where are you going to keep the decryption key? If it's on a disk or on a ROM, it can be read.

|> It seems that what you really want is an efficiently computable,
|> foolproof hash function by which to verify the integrity of a file.
|> (By "foolproof", I mean that it is "infeasible" to produce an altered
|> message with the same hash value, i.e. to fool the hash function.)
|> CRC is certainly efficient; that's why it's used. But it is not (and
|> is not designed to be) foolproof, and it is probably a mistake to try
|> to make it so.

Whether a one-way hash function exists is an open question in complexity theory.

|>
|> An example of such a hash function is RSA's MD4 Message Digest Algorithm,
|> described on this group a couple months ago. It may be more foolproof
|> and less efficient than you want, but it looks like a good example.
|> (Actually, Rivest indicates that it takes ~15 instructions/byte on a
|> 32-bit machine, which is quite respectable.)

MD4 may be efficient, but like DES its status is not known. There is no proof that DES is a trapdoor function, and there is no proof that MD4 is a one-way hash function. Unlike DES, fewer people have looked at MD4. [Then again, NSA didn't have a hand in its design. We think. ;-)] Where's the theoretical justification?

| Bennet S. Yee | b...@cs.cmu.edu |

| School of Cucumber Science | bsy%cs.cm...@vma.cc.cmu.edu |
| Cranberry Melon | bsy%cs.cm...@relay.cs.net |

| Pittsburgh, PA 15213-3890 | +1 412 268 7571 |
---

4 Jun 1990, 18:46:00
Jim Williams
to
> Can we consider
> 16 CRC-32s to be the same as 1 CRC-256? A CRC-256 solves the "add 1
> to a location, compute the CRC, repeat until match" problem by making
> it too slow.
I believe that 16 CRC-16s is the equivalent to 1 CRC-256. Just multiply the 16th order polynomials together to get a 256th order polynomial.

The results will not be bitwise equivalent but can be converted using the
Chinese remainder theorum.
More generally, though, CRC's are a bitwise linear operator, and forcing
any linear operator to a specific value involves solving simultaneous linear
equations. This is worst case an n-cubed problem, not a 2**n problem.
(n is number of bits.)

> 3) in all cases, we assume that the "smart [virus writing] guy" knows all
> the possible CRCs, but he doesn't know which set will be used on
> any particular user's machine. Assume that the virus program can
> find out which ones are being used.

It sounds like the security of this system depends on the virus not knowing
which CRC's are being used. (If both the file and the CRC check-sum are
known, could the virus calculate the polynomial used? Don't know exactly
how to do this, but it sounds like a solvable problem, particularly if
multiple files have been processed by the same CRC.)

A one way hash function like MD4 would solve the problem if its output
could be stored somewhere which was write protected from the virus. In
an open system where the virus can simply recompute the check-sum and
overwrite the old value, this scheme doesn't have much value.

An interesting alternative is to choose a random polynomial (by sufficiently
secure method) and compute a CRC using that polynomial. Then store the
polynomial and the CRC result in encrypted form. You wouldn't need to
worry too much about the cryptographic security of the encryption function;
the chalange would be hide the code implementing the function within the
OS code such that the virus couldn't figure it out. Ideally it should
be randomized from installation to installation so the virus would
not know in advance what it would be. Obviously the code shouldn't be
in a subroutine that the virus could call.

Virus protection sounds like an interesting and important problem.
I hope you can come up with a workable solution.

---

5 Jun 1990, 04:00:38
Wrig...@cc.curtin.edu.au
to
In article <54...@microsoft.UUCP>, gor...@microsoft.UUCP (Gordon LETWIN) writes:
> Microsoft is working on new versions of DOS and one of the things that
> I'm interested in is virus protection. Naturally,
> it will be impossible to produce iron-clad virus protection in a
> non-protected DOS environment, but I think we can produce some tools
> which - when properly applied - can be a big help.
>
> I've been thinking about using CRCs to see if a file has been diddled.
> I want to consider just the subset problem of determining if a file has
> been modified; we can consider seperately how we'll apply this in real life and
> how we'll keep a virus from compromosing the checker program itself or it's
> list of "correct values".
> ...

Why not use something like Rivest's MD4 algorithm? While not proven, the jury is still out on whether it is possible to produce two files with the same signature.

/----------\ /----------\
Rob Wright	psi%050529452300030::CROBW
Curtin University	CROBW%acad.curt...@uunet.uu.net
Perth, Western Australia	CROBW%acad.curtin.edu.au%munna...@cunyvm.bitnet
Voice:+61 9 351 7385	munnari!acad.curtin.edu.au!CR...@uk.ac.ukc
FAX: 09-351-2673	{backbones}!munnari!acad.curtin.edu.au!CROBW
\----------/ \----------/

---

11 Jun 1990, 14:50:09
Y Radai
to

Gordon Letwin raised the question of file-modification detection for anti-viral purposes by means of a CRC-comparing program. Others have suggested more sophisticated algorithms for this purpose. I shall refer to all programs of this type as "checksum" programs. (Others call them "message digest", "hash", "signature", or "fingerprint" programs.)
I'm an anti-viral person, not a crypto-person, but I stay tuned in on this frequency to learn what I can. In any case, I have discussed this question extensively over the past two years. And from my experience, people with a cryptographic background tend to make several incorrect assumptions when discussing anti-viral matters:
(1) that the most important ingredient in a checksum type of program is the algorithm;
(2) that the virus has an unlimited amount (or at least a lot of) time to do its "dirty work";
(3) that the major threat facing a checksum program is that a virus might be able to forge a checksum, i.e. to create an infected file having the same checksum as the original file, or to alter the checksum in the user's database so that it corresponds to the infected file.
I think some of those who have replied to Gordon are making one or more of these assumptions. I shall explain why, in my opinion, these assumptions are incorrect.

Assumption (1): Suppose we have a cryptographically very sophisticated algorithm for use on a computer running MS-DOS or PC-DOS. If we construct a program based on this algorithm which checksums all (executable) files on the disk, both when the files are first created and at sufficiently frequent intervals thereafter, will we detect all viral infections? No, because in addition to files we must take into account that the boot sector and partition record (= master boot record) could also get infected.
OK, so we checksum these too. Now for maximum convenience we run our checksum program from our hard disk. But what if there's a virus somewhere in memory when we do this? It might intercept our reads and cause us to checksum the original file or boot sector instead of the modified one, so that we don't notice the modification. (There are several viruses in the PC world which do this.) The surest way to prevent this from happening is (a) to put the checksum program and its data base on a diskette, after formatting it with the /S option (using our original DOS diskette from which we have never removed the

write-protect tab), and (b) to run our program only from that diskette and only immediately after cold-booting from that diskette. (This is not as inconvenient as it may sound; this process doesn't have to be performed on every execution or every boot; it can be done once a day or even less frequently.)
Now with our super-duper forge-proof algorithm run from a clean diskette and used to check all executable code on the disk, no viral infection can go undetected. Right? Wrong! There are at least three "loopholes" in DOS, i.e. peculiarities which a virus writer could exploit if the checksum program is not carefully written, all of which are independent of the checksum algorithm and do not depend on memory being infected at the time the checksum program is run. (One of these loopholes has already been exploited in a test virus created by a Bulgarian virus writer known to us by his initials "T. P.", who has written at least 50 PC viruses.) Against such a virus, sophistication of the algorithm is of no help whatsoever.
Thus even a program based on the most sophisticated checksum algorithm in the world cannot be depended on to detect all infections. Whether a given algorithm is secure depends heavily on how it's implemented as a *program* in a particular *system*.

Assumption (2): In the situation presented by Gordon, i.e. determining whether a file on the user's disk has been altered since it was first written there, the virus writer does not have in advance a single file-checksum pair which he can analyze before unleashing his virus. Hence a virus must perform all its dirty work at the time the program containing it is executed, and in so short a time that the user will not notice anything unusual. Any modification in a program which causes it to take much more time than usual is likely to be noticed by the user, causing him to suspect a virus.

Finally, Assumption (3): In the situation described above, if the virus, after infecting a file, is supposed to forge a checksum, it will have to gain access to the checksum base so that it can determine either the key or the checksum corresponding to its target file. But if that's kept isolated on a diskette along with the checksum program, it will never be readable by a virus. (Even if the checksum base is kept online, I think that by various techniques such as encryption of the checksums, forging of a checksum could probably be made impossible for all practical purposes.)
On the other hand, a virus writer can exploit the fact that people often execute their checksum program without ensuring that the system is uninfected at the time. Or he can exploit the loopholes mentioned above. Either of these threats is much more practical (from his point of view) than forging checksums, since they are *independent of the checksum algorithm* and *do not require any calculations (of checksums, keys, etc.)*.

The question still remains: What algorithm should we choose? One answer is to say that it doesn't matter as long as there are a lot of different algorithms in use. But this argument makes sense only if checksum-forging is considered the only threat. Moreover, if we're speaking of an algorithm released as part of DOS, it's going to have widespread use. I therefore think that the question is relevant.
I trust that most people in this group will agree that the algorithm, con-

sidered as a function F, should satisfy the following two conditions:
(A) For any given file f, F(f) is (in general) different for each user (or computer or disk). (This condition amounts essentially to requiring that the key be chosen *randomly* or *personally* by each user.)
(B) Given a file f, it is computationally infeasible (without knowledge of the key) to construct a file f' from it containing the desired addition of (or replacement of ordinary code by) viral code such that F(f') = F(f). (Note that the phrase "computationally infeasible" is relative to the environment. In the present environment it is assumed that the virus and its writer cannot gain access to the user's checksums and that the virus has very little time to act.)

There are a lot of algorithms which satisfy these requirements: MD4, Snefru, etc. etc. But if these turn out to be too slow then people just aren't going to bother using them. So we should use the fastest algorithm satisfying these conditions. I have not yet had the opportunity to make speed comparisons, but my impression is that the fastest such algorithm is CRC.
Note: "CRC" does not refer to any particular polynomial, but to a certain algorithm or function, roughly: the remainder after dividing the file by a given polynomial. In communications, this polynomial is a standard one; for antiviral purposes it is not, or at least it shouldn't be. Bennet Yee seems to be against CRCs, yet judging by his reference to Rabin's "Fingerprinting" paper, he's actually in favor of them. For Rabin's scheme *is* a CRC scheme; he merely selects an irreducible polynomial (of chosen prime degree) at random instead of using a standard polynomial.

Now what should be the length of the CRC polynomial and how many CRCs should be used simultaneously? Gordon talks of using 16 selected CRC-32 polynomials or one CRC-256 polynomial. I say *one* 32-bit polynomial is quite sufficient, provided (a) the polynomial is chosen randomly or personally, (b) the checksum base is kept offline, and (c) the checksum program is activated only when there is no chance that memory is infected.

Concerning Rabin's scheme: Bennet mentioned that he has implemented it. It has also been implemented for the PC in a commercial program called "V-Analyst" written in Israel. To the best of my knowledge, it's the most secure and flexible checksum program available for the PC. In particular, it's the only checksum program I've ever heard of which blocks all three loopholes mentioned above. (Of course, it's conceivable that there are more such loopholes.) It's probably faster than any program which uses a more sophisticated algorithm, though it's not the fastest among CRC programs. McAfee's shareware program Sentry is much faster, but it's very inflexible, and not as secure since (among other things) it checksums only the beginning of each file.

BTW, no one has yet given me a convincing demonstration that Rabin's restriction of the choice of CRC polynomials to irreducible ones and requiring the degree of the polynomial to be prime have any *practical* anti-forging value (as opposed to merely making it easier for him to state and prove certain theorems). If Bennet or someone else has such a demonstration, I'd like to hear it.

Finally, turning to the more general situation which Gordon has raised:

>Microsoft is working on new versions of DOS and one of the things that
>I'm interested in is virus protection. Naturally,
>it will be impossible to produce iron-clad virus protection in a
>non-protected DOS environment, but I think we can produce some tools
>which - when properly applied - can be a big help.

In my opinion, this is a very positive development on the part of Microsoft. I first heard of it a few months ago, and was somewhat surprised to hear that they were the least bit interested in such matters as security. I'm glad to find that my previous impression was inaccurate.
In any case, Microsoft seems to be interested in more than just a modification detector, and in my opinion, the best forum for anti-viral discussion in general is comp.virus (= Virus-L). So I would suggest that Gordon or someone else from Microsoft place a notice in comp.virus, asking for suggestions on how to make DOS more secure in other respects as well. I promise that you'll get more suggestions than you'll know what to do with.

Y. Radai
Hebrew Univ. of Jerusalem, Israel
RA...@mush.huji.AC.IL
RA...@HUJIVMS.BITNET

---

18 Jun 1990, 06:19:37
Bennet Yee
to
This article argues that some of Y Radai's points are incorrect (partially due to incorrect assumptions), points out the threat of off-line attacks, clarifies the difference between CRCs and fingerprinting, analyses the use of reducible polynomials as moduli in calculating fingerprints, and shows why it provides less protection than using irreducible polynomials.
There is a bit of math at the end, and I've taken the liberty to write it in LaTeX. It is still readable without, but running it through TeX first makes it much simpler. The quotations from Radai remains in the common ">"-prefixed format.

I will be on vacation for a couple of weeks, so will not be able to respond to comments immediately. You may want to email responses since news articles may expire before I get back.

In article <7...@shuldig.Huji.Ac.IL>, ra...@mush.UUCP (Y Radai) writes:

> Gordon Letwin raised the question of file-modification detection for

> antiviral purposes by means of a CRC-comparing program. Others have

> suggested more sophisticated algorithms for this purpose. I shall refer to
> all programs of this type as "checksum" programs.

> ...

>
> ... And from my experience, people with a
> cryptographic background tend to make several incorrect assumptions when
> discussing anti-viral matters:

>
> (1) that the most important ingredient in a checksum type of program is the
> algorithm;
>
> (2) that the virus has an unlimited amount (or at least a lot of) time to do
> its "dirty work";
>
> (3) that the major threat facing a checksum program is that a virus might be
> able to forge a checksum, i.e. to create an infected file having the same
> checksum as the original file, or to alter the checksum in the user's
> database so that it corresponds to the infected file.
>

> Assumption (1): ... in

> addition to files we must take into account that the boot sector and
> partition record (= master boot record) could also get infected.
>
> OK, so we checksum these too. Now for maximum convenience we run our

> checksum program from our hard disk. But what if there's a virus somewhere
> in memory when we do this? It might intercept our reads and cause us to

> checksum the original file or boot sector instead of the modified one, so

> that we don't notice the modification. ...

> Now with our super-duper forge-proof algorithm run from a clean diskette and
> used to check all executable code on the disk, no viral infection can go

> undetected. Right? Wrong! There are at least three "loopholes" in DOS,
> i.e. peculiarities which a virus writer could exploit if the checksum

> program is not carefully written, all of which are independent of the
> checksum algorithm and do not depend on memory being infected at the time
> the checksum program is run. (One of these loopholes has already been
> exploited in a test virus created by a Bulgarian virus writer known to us by
> his initials "T. P.", who has written at least 50 PC viruses.) Against such
> a virus, sophistication of the algorithm is of no help whatsoever.
>
> Thus even a program based on the most sophisticated checksum algorithm in
> the world cannot be depended on to detect all infections. Whether a given

> algorithm is secure depends heavily on how it's implemented as a *program*
> in a particular *system*.

One assumption that you are making about the system model (as I understand it) is that the system in question is PC-DOS/MS-DOS. I believe this to be incorrect. The new OSes to worry about is OS/2, AIX, OSF/1, Mach, various flavors of Unix (SysVRx/SunOS, BSD) etc. Real operating systems. With protected address spaces. With a file system abstraction that can not be bypassed. Gordon/Microsoft is probably more interested in OS/2 or perhaps AIX, but that is immaterial as long as the general solution is applicable. The ideas can extend to DOS as well, but it would take some extra work -- and can not be as user-friendly since the ``operator'' will have to be careful with ``clean'' disks and have to follow a manual procedure at boot time.

Whenever you have a system without inter-task protection, you can not run untrusted programs without possibly losing system integrity. And once compromised, there is no way to detect that with certainly unless you start from some outside, trusted base (see Ken Thompson's Turing Award lecture). Yes, maintaining system integrity is difficult. Yes, you can check only once a day or once a week -- you really absolutely must perform integrity checks when you use potentially untrustworthy programs on your system. If you download possibly trojan-horsed code or are on a multi-user system not all of whom can be trusted to be careful, then you need more frequent integrity checks. And your integrity check should be done from a complete reboot from a clean disk.

(1) _One_ of the most important ingredient _is_ the algorithm. Another is your system model. Clearly, reductio ad absurdum, using parity as the detection scheme is not the right approach. If your basic algorithm does not have the necessary security properties, then the system that you build up around it isn't going to be secure either. My claim earlier is that normal CRC does not have the needed properties for security.

You can't pull yourself up by your bootstraps unless you're wearing boots.

> Assumption (2): In the situation presented by Gordon, i.e. determining

> whether a file on the user's disk has been altered since it was first

> written there, the virus writer does not have in advance a single
> file-checksum pair which he can analyze before unleashing his virus. Hence
> a virus must perform all its dirty work at the time the program containing
> it is executed, and in so short a time that the user will not notice
> anything unusual. Any modification in a program which causes it to take
> much more time than usual is likely to be noticed by the user, causing him
> to suspect a virus.
>
> Finally, Assumption (3): In the situation described above, if the virus,

> after infecting a file, is supposed to forge a checksum, it will have to
> gain access to the checksum base so that it can determine either the key or
> the checksum corresponding to its target file. But if that's kept isolated
> on a diskette along with the checksum program, it will never be readable by

> a virus. (Even if the checksum base is kept online, I think that by various
> techniques such as encryption of the checksums, forging of a checksum could
> probably be made impossible for all practical purposes.)

>
> On the other hand, a virus writer can exploit the fact that people often

> execute their checksum program without ensuring that the system is

> uninfected at the time. Or he can exploit the loopholes mentioned above.
> Either of these threats is much more practical (from his point of view) than
> forging checksums, since they are *independent of the checksum algorithm*
> and *do not require any calculations (of checksums, keys, etc.)*.

(2/3) A particular virus may not have ``unlimited'' resources to crack your system. PC virii are indeed restricted to on-line algorithms, more or less. Virus writers, however, are NOT. In particular, if you're worried that a virus writer plans to attack a specific set of application -- say, ones that is sold, ones for which there are many identical copies -- the virus writer may choose to spend much more resources in the construction of the virus. The virus writer can easily make the virus such that an infected application would have the same checksums with respect to a standard set of polynomials. Off-line searches by the virus writer _is_ practical.

Furthermore, once we discard the assumption that we're dealing with minimalist OSes such as DOS, virii can spawn low priority tasks/threads and can do their dirty work in the background where you won't notice the additional load/activity.

Yes, people are careless and virus writers can exploit that. However, you can make it very difficult by performing system integrity checks on system programs at boot time automatically. If a user don't have to remember to actively check his environment, the user can't forget. The user's own files, on the other hand, can't be checked entirely automatically, but with proper design you can make it very easy.

> The question still remains: What algorithm should we choose? One answer is
> to say that it doesn't matter as long as there are a lot of different

> algorithms in use. But this argument makes sense only if checksum-forging
> is considered the only threat. Moreover, if we're speaking of an algorithm

> released as part of DOS, it's going to have widespread use. I therefore

> think that the question is relevant.
>
> I trust that most people in this group will agree that the algorithm,

> considered as a function F, should satisfy the following two conditions:

>
> (A) For any given file f, F(f) is (in general) different for each user (or
> computer or disk). (This condition amounts essentially to requiring that
> the key be chosen *randomly* or *personally* by each user.)
>
> (B) Given a file f, it is computationally infeasible (without knowledge of
> the key) to construct a file f' from it containing the desired addition of
> (or replacement of ordinary code by) viral code such that F(f') = F(f).
> (Note that the phrase "computationally infeasible" is relative to the
> environment. In the present environment it is assumed that the virus and

> its writer cannot gain access to the user's checksums and that the virus has

> very little time to act.)
>
>
> There are a lot of algorithms which satisfy these requirements: MD4, Snefru,
> etc. etc. But if these turn out to be too slow then people just aren't
> going to bother using them. So we should use the fastest algorithm
> satisfying these conditions. I have not yet had the opportunity to make
> speed comparisons, but my impression is that the fastest such algorithm is
> CRC.

Neither MD4 nor Snefru satisfy your conditions. There are no keys. The
standard terminology, by the way, is that the algorithm specifies a family
of functions indexed by a key. The choice of key is what makes the specific
function different for each user.

> Note: "CRC" does not refer to any particular polynomial, but to a certain
> algorithm or function, roughly: the remainder after dividing the file by a

> given polynomial. In communications, this polynomial is a standard one; for
> antiviral purposes it is not, or at least it shouldn't be. Bennet Yee seems

> to be against CRCs, yet judging by his reference to Rabin's "Fingerprinting"
> paper, he's actually in favor of them. For Rabin's scheme *is* a CRC

> scheme; he merely selects an irreducible polynomial (of chosen prime degree)

> at random instead of using a standard polynomial.
>

You misunderstand me. CRCs are different from ``fingerprinting''. Again, standard terminology: CRCs do not refer to random irreducible polynomials. Until Karp and Rabin came along and did their work, nobody knew how to generate irreducible polynomials, and known ``good'' polynomials are published in books. Perhaps it sounds like an appeal to authority, but the importance of the work should be hinted at from the fact that this technique was one of the topics in Karp's Turing Award lecture.

>
> Now what should be the length of the CRC polynomial and how many CRCs should
> be used simultaneously? Gordon talks of using 16 selected CRC-32
> polynomials or one CRC-256 polynomial. I say *one* 32-bit polynomial is
> quite sufficient, provided (a) the polynomial is chosen randomly or
> personally, (b) the checksum base is kept offline, and (c) the checksum
> program is activated only when there is no chance that memory is infected.
>

Have you made the probability calculations? Do you know what is the maximum length file that a 32-bit polynomial can protect safely is? Don't make claims without facts supporting them.

>
> Concerning Rabin's scheme: Bennet mentioned that he has implemented it. It
> has also been implemented for the PC in a commercial program called
> "V-Analyst" written in Israel. To the best of my knowledge, it's the most

> secure and flexible checksum program available for the PC. In particular,

> it's the only checksum program I've ever heard of which blocks all three
> loopholes mentioned above. (Of course, it's conceivable that there are more
> such loopholes.) It's probably faster than any program which uses a more
> sophisticated algorithm, though it's not the fastest among CRC programs.
> McAfee's shareware program Sentry is much faster, but it's very inflexible,
> and not as secure since (among other things) it checksums only the beginning
> of each file.
>
>
> BTW, no one has yet given me a convincing demonstration that Rabin's

> restriction of the choice of CRC polynomials to irreducible ones and
> requiring the degree of the polynomial to be prime have any *practical*

> anti-forging value (as opposed to merely making it easier for him to state

> and prove certain theorems). If Bennet or someone else has such a

> demonstration, I'd like to hear it.

>

Here goes.

CRC checksums and fingerprinting are two very different beasts. The
critical difference is that CRC checksums rely on known, published
polynomials that any good hacker can look up. Fingerprints rely on
randomized algorithms; unless the virus writer can predict your random bits,
s/he's out of luck.

The fact that the degree of the irreducible is prime simply makes the
calculations easier, yes, but it also maximizes the number of irreducibles:
the number of irreducible polynomials of degree n, denoted $m(n)$ is
bounded by
\[{{ p^n - p^{n/2} \log n } \over n} \le m(n) \le {p^n \over n} \]

To see why the irreducibility of Rabin's fingerprinting polynomials is
useful, consider the following. Suppose the modulus is a simple polynomial
such as x^{31}. Well, the last word of the data determines the residue
completely! When the modulus is irreducible, the induced homomorphism
$\phi: Z_2[x] \rightarrow R$ has the field GF(2^d) as its image R. If
the modulus is reducible, the image R is just a ring with zero divisors.

Why do we care whether R is a field or a ring? If we restrict ourselves
to the case where there are no repeated irreducible factors, we'd see that
this is equivalent to mod'ing out by each of the irreducible factors
separately and applying Chinese Remainder Theorem to reconstruct the residue
-- the ring is isomorphic to the direct product of the various image fields.
(This is not true, btw, if there are repeated factors.) So, a closer
examination of the proof for the irreducible case tells us what happens when
the polynomial is not so nice.

When the polynomial is irreducible, we figure the probability of two inputs
having the same residue as follows:

Call the inputs $f(x)$ and $g(x)$, $f \ne g$. $f \bmod m \equiv g \bmod m$
is equivalent to $f-g \bmod m \equiv 0$, or $m \mid f-g$. How many divisors
of $f-g$ are there of degree $\deg m$? Clearly, it is bounded by
$\deg (f-g)/\deg m$ since m is irreducible. Since we know that there are
$(2^{\deg m}-2)/\deg m$ irreducible polynomials of degree $\deg m$, the
probability that a random irreducible dividing $f-g$ is
$\deg (f-g)/(2^{\deg m}-2)$.

What happens if instead of a single $\deg m$ irreducible polynomial we had
the product of two irreducibles? I will simply state without proof that the
best case happens when $m(x) = p(x)q(x)$ where $\deg p = \deg q = {\deg m
\over 2}$, $p \ne q$. The probability that the polynomial $f-g$ is
divisible by m is now calculated as follows: $f-g$ is divisible by m if
$p \mid f-g$ and $q \mid f-g$. The number of $\deg m \over 2$ degree
divisors are $\deg (f-g)/({\deg m \over 2})$, so the probability of being
divisible by a random $\deg m/2$ degree irreducible is $2

\deg (f-g)/(2^{\deg m \over 2}-2)$, and the probability of being divisible
by two of them is simply $\approx 4 (\deg (f-g))^2/2^{\deg m}$, which is
clearly much larger than $\deg (f-g)/2^{\deg m}$.

I won't go into the case where factors are repeated.

Using reducible polynomials still gives you {\em some} protection, but using
irreducibles is ideal since you can minimize the probability that two inputs
will have the same residue.

| Bennet S. Yee | b...@cs.cmu.edu |

| School of Computer Science | bsy%cs.cm...@vma.cc.cmu.edu |

| Carnegie Mellon | bsy%cs.cm...@relay.cs.net |

| Pittsburgh, PA 15213-3890 | +1 412 268 7571 |
---

21 Jun 1990, 22:22:34
Robert English
to
> / b...@PLAY.MACH.CS.CMU.EDU (Bennet Yee) / 10:19 pm Jun 17, 1990 /
> One assumption that you are making about the system model (as I understand
> it) is that the system in question is PC-DOS/MS-DOS. I believe this to be
> incorrect. The new OSes to worry about is OS/2, AIX, OSF/1, Mach, various
> flavors of Unix (SysVRx/SunOS, BSD) etc. Real operating systems. With
> protected address spaces. With a file system abstraction that can not be
> bypassed. Gordon/Microsoft is probably more interested in OS/2 or perhaps
> AIX, but that is immaterial as long as the general solution is applicable.

It seems to me that this general problem is similar to that of attaching
"digital signatures" to documents. Someone running a program wants to
be able to authenticate the author of the program as a trusted vendor,
and the authentication must validate both the author and the program. I
know there's literature on this subject (at least there was 10 years
ago), and some proposed systems based on public key cryptography.
Unfortunately, I don't remember how well they worked.

--bob--
renglish@hpda

4 Jul 1990, 14:33:29
Y Radai
to
5...@cs.cmu.edu>
Sender: ne...@shuldig.huji.ac.il
Reply-To: ra...@mush.huji.ac.il (Y Radai)
Organization: The Hebrew University of Jerusalem, Israel
Lines: 164
Apparently-To: post-...@ucbvax.berkeley.edu.bitnet

In his article of 18 June, Bennet Yee writes:
>This article argues that some of Y Radai's points are incorrect (partially
>due to incorrect assumptions), points out the threat of off-line attacks,
>clarifies the difference between CRCs and fingerprinting, analyses the use
>of reducible polynomials as moduli in calculating fingerprints, and shows
>why it provides less protection than using irreducible polynomials.

When I first took a look at Bennet's long response quoting so much of my article, I thought I must have made a lot of errors. But I soon discovered that the passages which he actually criticized constituted far less than what he quoted. And when I investigated each of his criticisms, I came to the conclusion that I made no errors in anything the least bit essential to my main position. Let's take his criticisms in turn.

>One assumption that you are making about the system model (as I understand
>it) is that the system in question is PC-DOS/MS-DOS. I believe this to be
>incorrect. The new OSes to worry about is OS/2, AIX, OSF/1, Mach, various
>flavors of Unix (SysVRx/SunOS, BSD) etc.

> Gordon/Microsoft is probably more interested in OS/2 or perhaps
>AIX, but that is immaterial as long as the general solution is applicable.

The person who opened this discussion was Gordon, and he explicitly focused

the discussion on DOS when he wrote:
>Microsoft is working on new versions of DOS and one of the things that
>I'm interested in is virus protection. Naturally,
>it will be impossible to produce iron-clad virus protection in a
>non-protected DOS environment, but I think we can produce some tools
>which - when properly applied - can be a big help.

Thus the system in question definitely *is* DOS. (I might not have bothered mentioning this point were it not for the fact that Bennet makes this same claim again later.)

> PC virii are indeed restricted to on-line algorithms, more or less.
>Virus writers, however, are NOT. In particular, if you're worried that a
>virus writer plans to attack a specific set of application -- say, ones that
>is sold, ones for which there are many identical copies -- the virus writer
>may choose to spend much more resources in the construction of the virus.
>The virus writer can easily make the virus such that an infected application
>would have the same checksums with respect to a standard set of polynomials.
>Off-line searches by the virus writer _is_ practical.

Bennet has overlooked one point here which is so fundamental that it's worth going into in detail. A virus is a segment of code which propagates by surreptitiously copying itself to other files when the program which contains it is executed. But since the "virus" which Bennet has described is constructed to attack *a specific application*, and since it is not customary to keep two copies of the same application online at the same time, it follows that for all

practical purposes, the "virus" which Bennet has described *has no way of propagating*. True, one can distribute such a "virus" by sending (or making available on a BBS, etc.) copies of the entire altered program to various users and hoping they will execute it, but that is not propagation. Anti-viral people wouldn't even call it a virus at all, but rather a Trojan Horse. Now regardless of what names are given to these two "critters", there is a very important distinction between them. The whole strategy behind using a checksum program in the way Gordon Letwin is interested depends heavily on the fact that a virus (as opposed to a Trojan) PROPAGATES.
In the case of the problem considered by Gordon, by Prof. Rabin (in the section "Protection of Files" of his "Fingerprinting" paper), and by me, a checksum program is installed on each computer, whose task is to detect alterations in files which occur *after* they are initially put onto the user's disk. This idea works only because the purpose of a virus is to propagate to as many users as possible, and in order to achieve this, the damage which the virus performs (if any) is delayed to some future date. However, in the case which Bennet describes, there is no propagation, i.e. no previously checksummed file on the disk will get altered, hence no checksum program (implemented as described at the beginning of this paragraph) can be effective.
What can be done against a Trojan? First, it should be noted that since the advent of viruses, Trojans constitute only a very small percentage of malware (a word I just now coined for Trojans, viruses, worms, etc.). And there are good reasons for this: (1) Since a Trojan doesn't propagate, the number of users to which damage can be caused is much smaller than in the case of a virus; (2) it's almost impossible to trace the source of a virus since viruses are not attached to any particular program.
Still, Trojans do exist so we have to decide what to do about them. An obvious defense is to verify, by means of a hash function, the integrity of each file from the time of its release by its original author until the time when it arrives on our disks. This is similar to using a digital signature. But it is a quite different problem from that which Gordon raised. (The idea of a checksum or hash function is common to both situations, but the details are different.)

>Furthermore, once we discard the assumption that we're dealing with
>minimalist OSes such as DOS, virii can spawn low priority tasks/threads and
>can do their dirty work in the background where you won't notice the
>additional load/activity.

Very true, but irrelevant to the present discussion. As I mentioned above, both Gordon and I were speaking only of DOS. (BTW, there is no such word as "virii". The plural of "virus" is "viri" in Latin and "viruses" in English.)

>Neither MD4 nor Snefru satisfy your conditions. There are no keys.

Thanks for the correction. But since I merely gave MD4 and Snefru as examples, this mistake does not affect any of my central claims. On the other hand, it does concern those who recommended MD4 as a solution to the anti-viral problem. For I assume that the implication of no keys is that all users will obtain the same checksum for any given file. If so, I consider the use of such programs to be insufficiently secure in the anti-viral context, despite their value for

authenticating digital signatures.

> CRCs are different from ``fingerprinting''. Again,
>standard terminology: CRCs do not refer to random irreducible polynomials.
>Until Karp and Rabin came along and did their work, nobody knew how to
>generate irreducible polynomials, and known ``good'' polynomials are
>published in books. Perhaps it sounds like an appeal to authority, but the
>importance of the work should be hinted at from the fact that this technique
>was one of the topics in Karp's Turing Award lecture.

I have no doubt that Karp's work was important. (BTW, can you tell me where I can find his lecture?) But I contest the claim that this is standard terminology. I have seen several articles which speak of using randomly chosen CRC polynomials or even of using CRCs for fingerprinting purposes. For Bennet, the crucial difference between these terms seems to be whether the choice of polynomial is randomized or not. I, on the other hand, was interested in the fact that Rabin's algorithm is the same as the CRC algorithm. "CRC" seemed to me as good a word for that common algorithm as any, so I chose it. But obviously it's not worth quibbling over.

>Have you made the probability calculations? Do you know what is the maximum
>length file that a 32-bit polynomial can protect safely is?

No, I haven't and I don't. But in addition to the three incorrect assumptions which I mentioned as being made by crypto-people when dealing with virus problems, I think I should have added a fourth one: that they tend to over-emphasize the importance of probability considerations. Perhaps the best place to see this is in Bennet's answer to my question of what *practical* value there is in the fact that Rabin stipulates that the degree of the generating polynomial should be prime and that the polynomial itself must be irreducible.
Bennet gave a mathematical reply to my question, leading up to a pair of formulas expressing the probability that two files will have the same checksum under the alternative assumptions that the generating polynomial is (a) irreducible and (b) reducible, and pointed out that the former is smaller than the latter. Although I wasn't able to check all his formulas, I do wish to thank him for presenting them.
However, to say that one probability is smaller than the other does not answer my question of what PRACTICAL VALUE there is in the use of irreducible polynomials. I do not doubt that there are *some* contexts in which a smaller probability would be highly significant. But Rabin is assuming that each user has a different generator and that *the virus has no access to the checksum base*, i.e. no knowledge of either the checksum corresponding to the target file or of the generator. As he writes in the section "Protection of Files" of his "Fingerprinting" paper, "The fingerprinting and updating computations must be performed securely so that none other than the guardian can access [the generating polynomial] p or the [fingerprints] ...."
So in order to answer my question, Bennet will have to explain how a virus writer, working in *such* an environment, could *exploit* the fact that the probability of two files having the same checksum is larger when reducible polynomials are used. For sake of argument, I'm willing to make the (not very realistic) assumption that the virus has unlimited time for its trials with-

out being noticed. Still, if the checksum base is inaccessible, how could the virus ever know when it has *succeeded* in producing a suitably altered file with the same checksum?

Finally, I wish to say that despite the apparent differences between us, I suspect that Bennet and I agree on far more than we disagree.

Y. Radai
Hebrew Univ. of Jerusalem, Israel

ra...@mush.huji.AC.IL
RA...@HUJIVMS.BITNET
RAD...@HBUNOS.BITNET

---

7 Jul 1990, 05:18:35
Bennet Yee
to
This article clarifies some points of contentions/misunderstanding between Y. Radai and myself, points out a simple conceptual error, and argues against ad hoc design.
[Quasi-editorial note: I use the LaTeX-style notation of \\begin{quotation}...\\end{quotation} rather than the usual ``> '' prefix since prefixing Y. Radai's text would make an 80-column terminal wrap some of the quoted lines. His formatting style makes it difficult to simply rejustify his text because of the hyphenation. Also, I'm sorry if I over-quote Y. Radai -- I prefer to provide too much context rather than too little; it is easier for readers to skip over quoted material than for them to search back for the original articles. The Message-ID of Y. Radai's article quoted below is 8...@shuldig.Huji.Ac.IL]

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

>One assumption that you are making about the system model (as I understand
>it) is that the system in question is PC-DOS/MS-DOS. I believe this to be
>incorrect. The new OSes to worry about is OS/2, AIX, OSF/1, Mach, various
>flavors of Unix (SysVRx/SunOS, BSD) etc.
> Gordon/Microsoft is probably more interested in OS/2 or perhaps
>AIX, but that is immaterial as long as the general solution is applicable.

The person who opened this discussion was Gordon, and he explicitly focused the discussion on DOS when he wrote:
>Microsoft is working on new versions of DOS and one of the things that
>I'm interested in is virus protection. Naturally,
>it will be impossible to produce iron-clad virus protection in a
>non-protected DOS environment, but I think we can produce some tools
>which - when properly applied - can be a big help.

Thus the system in question definitely *is* DOS. (I might not have bothered mentioning this point were it not for the fact that Bennet makes this same claim again later.)

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

In my previous article, I described a general solution to the problem of detecting viruses/unauthorized file modifications that is best suited for a ``real'' operating system that has protected address spaces, and also pointed out that this solution can still be implemented (albeit less automated since a cold boot is needed) for an environment such as PC-DOS/MS-DOS. I made the mistake of not checking the original article that started the thread, and misidentified the type of the target OS. This has no impact, however, on the usefulness of the technique.

Also, BTW, note that since OS/2 allows concurrent DOS windows (likewise Xenix), thinking about these detection techniques in general terms is useful even if we restrict ourselves to just DOS.

As for your response to my points:

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

> PC virii are indeed restricted to on-line algorithms, more or less.
>Virus writers, however, are NOT. In particular, if you're worried that a
>virus writer plans to attack a specific set of application -- say, ones that
>is sold, ones for which there are many identical copies -- the virus writer
>may choose to spend much more resources in the construction of the virus.
>The virus writer can easily make the virus such that an infected application
>would have the same checksums with respect to a standard set of polynomials.

>Off-line searches by the virus writer is practical.

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

Here is a simple conceptual error that can be easily cleared up. Note that I described viruses that attack a {\em set} of application. For example, it is simple to write a virus that can specifically attack just the two programs WordPerfect and Lotus 1-2-3. Using the imperfect medical analogy, you can think of WordPerfect as the vector used to infect 1-2-3. The idea is that a virus can a priori know about a small set of applications --- and copying any one of these applications to an uninfected system (and running that application) will cause all other applications on the otherwise uninfected system also in this set to become infected as well. This is how such viruses can propagate. The rate of propagation depends on the size of the set of target applications and what these applications are.

Just because a computer virus is specific to certain programs do not mean it can not propagate. I'd argue [this part is not based on theoretical understanding but rather on engineering common sense] that such viruses will be harder to detect --- instead of blinding infecting/modifying all programs and thus causing some of them to fail at the ``wrong'' time, the infection

can be completely benign --- at least until some predetermined environmental
trigger comes along. Greater specificity should not be automatically be
equated with lesser power to damage.

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

[text about Trojan horses deleted]

>Furthermore, once we discard the assumption that we're dealing with
>minimalist OSes such as DOS, virii can spawn low priority tasks/threads and
>can do their dirty work in the background where you won't notice the
>additional load/activity.

Very true, but irrelevant to the present discussion. As I mentioned above,
both Gordon and I were speaking only of DOS. (BTW, there is no such word as
"virii". The plural of "virus" is "viri" in Latin and "viruses" in English.)

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

Thanks for the spelling correction. Note that even minimalist OSes can hide
activity if necessary --- with DOS, you can hack the clock interrupt to run
a coroutine. Thus even with DOS, it's relevant.

BTW, since my main purpose in responding to these postings is not to argue
with you but to disseminate information, in some way whether you and Gordon
are speaking only of DOS is of little concern to me. ;-) The techniques
that I'm describing works for all systems, including DOS; the analysis

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

>Neither MD4 nor Snefru satisfy your conditions. There are no keys.

Thanks for the correction. But since I merely gave MD4 and Snefru as examples,
this mistake does not affect any of my central claims. On the other hand, it
does concern those who recommended MD4 as a solution to the anti-viral problem.
For I assume that the implication of no keys is that all users will obtain the
same checksum for any given file. If so, I consider the use of such programs
to be insufficiently secure in the anti-viral context, despite their value for
authenticating digital signatures.

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

If you believe that MD4 or Snefru are one-way hash functions, then you can
use them to {\em construct} a family of checksum functions. Assume f^m is
a family of one-way hash functions, $f^m = \{ f_n^m \}$, $n \in Z+$, where
$f_n^m: Z_2^n \rightarrow Z_2^m$, i.e., n is the length of the input bits
and m is the length of the output hash values. Let t is the text of a

file expressed in binary. To construct a family of checksum functions $F_k^m = \{ F_{k,n,m} \}$ where $F_{k,n,m}: Z_2^n \rightarrow Z_2^m$, k is the key, simply form \[F_{k,n,m}(t) = f_{(\left| k \right| + n)}^m(k 2^n + t) \] where $\left| \cdot \right|$ denote the length operator.

Again, this works only if you have a one-way hash, and neither MD4 nor Snefru have been proven to be one-way hashes. So, if you believe in the claims about these algorithms, you can use them to construct a family of checksum functions that would satisfy your criteria (quoted from Radai's first article):

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

I trust that most people in this group will agree that the algorithm,

considered as a function F, should satisfy the following two conditions:

(A) For any given file f, F(f) is (in general) different for each user (or computer or disk). (This condition amounts essentially to requiring that the key be chosen *randomly* or *personally* by each user.)

(B) Given a file f, it is computationally infeasible (without knowledge of the key) to construct a file f' from it containing the desired addition of (or replacement of ordinary code by) viral code such that F(f') = F(f). (Note that the phrase "computationally infeasible" is relative to the environment. In the present environment it is assumed that the virus and

its writer cannot gain access to the user's checksums and that the virus has

very little time to act.)

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

The algorithms by themselves do not satisfy them. [Again, your notation that F is a {\em function} where F(f) is different for different people is contradictory --- F must either take an additional argument (the person/key) or, as I've written, a family of functions indexed by a key.]

{\bf NOTE}: If you really have a one-way hash, there's no need for criterion A, and the entire exercise of building checksum functions from one-way hashes is silly. The idea behind one-way functions is that computing the inverse is infeasible, and in particular with one-way hash functions it is infeasible to find an input given an arbitrary hash (output) value (and thus B is satified). The reason that people working with cryptographic checksum functions use a family of functions that are indexed by a key is because there does NOT exist a provably one-way hash --- so, instead of using a one-way hash, we use a family of checksum functions with

secret keys. Instead of assuming that a function is a one-way hash and thus non-invertible, we opt to keep the key (index) secret and chose our family of functions such that inversion includes guessing this key.

It certainly would be nice if there were provably one-way hashes, but none has been proven to exist. The existence of one-way hashes is still an open question.

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

> CRCs are different from ``fingerprinting''. Again,
>standard terminology: CRCs do not refer to random irreducible polynomials.
>Until Karp and Rabin came along and did their work, nobody knew how to
>generate irreducible polynomials, and known ``good'' polynomials are
>published in books.

I have no doubt that Karp's work was important. (BTW, can you tell me where I

can find his lecture?) But I contest the claim that this is standard terminology. I have seen several articles which speak of using randomly chosen CRC polynomials or even of using CRCs for fingerprinting purposes. For Bennet, the crucial difference between these terms seems to be whether the choice of polynomial is randomized or not. I, on the other hand, was interested in the fact that Rabin's algorithm is the same as the CRC algorithm. "CRC" seemed to me as good a word for that common algorithm as any, so I chose it. But obviously it's not worth quibbling over.

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

You can find Karp's lecture in the book {\it ACM Turing Award Lectures} available from ACM press. Karp and Rabin's technique hinges on finding random irreducible polynomials. They made it possible to randomly find irreducibles, and their algorithm is for finding irreducibles, not computing residues. If I sometimes clump both the generation of an irreducible and the residue computation together and thus confused the issue, I'm sorry. The fact remains, however, that the crucial difference is the random generation of moduli. The ad hoc method proposed of using a small fixed set of CRC moduli is a very bad way of building a protection system.

\begin{quotation} %>>>>>>>>>>>>>>>>>>>>

>Have you made the probability calculations? Do you know what is the maximum
>length file that a 32-bit polynomial can protect safely is?

No, I haven't and I don't. But in addition to the three incorrect assumptions which I mentioned as being made by crypto-people when dealing with virus problems, I think I should have added a fourth one: that they tend to over-empha-

size the importance of probability considerations. Perhaps the best place to see this is in Bennet's answer to my question of what *practical* value there is in the fact that Rabin stipulates that the degree of the generating polynomial should be prime and that the polynomial itself must be irreducible. Bennet gave a mathematical reply to my question, leading up to a pair of formulas expressing the probability that two files will have the same checksum under the alternative assumptions that the generating polynomial is (a) irreducible and (b) reducible, and pointed out that the former is smaller than the latter. Although I wasn't able to check all his formulas, I do wish to thank him for presenting them.
However, to say that one probability is smaller than the other does not answer my question of what PRACTICAL VALUE there is in the use of irreducible polynomials. I do not doubt that there are *some* contexts in which a smaller probability would be highly significant. But Rabin is assuming that each user has a different generator and that *the virus has no access to the checksum base*, i.e. no knowledge of either the checksum corresponding to the target file or of the generator. As he writes in the section "Protection of Files" of his "Fingerprinting" paper, "The fingerprinting and updating computations must be performed securely so that none other than the guardian can access [the generating polynomial] p or the [fingerprints] ...."

So in order to answer my question, Bennet will have to explain how a virus writer, working in *such* an environment, could *exploit* the fact that the probability of two files having the same checksum is larger when reducible polynomials are used. For sake of argument, I'm willing to make the (not very realistic) assumption that the virus has unlimited time for its trials without being noticed. Still, if the checksum base is inaccessible, how could the virus ever know when it has *succeeded* in producing a suitably altered file with the same checksum?

\end{quotation} %<<<<<<<<<<<<<<<<<<<<

First, we should note that the calculations that I made was the probabilities of two input having the same residue when mod'ing out by (1) an irreducible polynomial of degree d and (2) by a product of two irreducible polynomials, each of degree $d/2$ (thus the product would have the same degree as the irreducible in (1)). To calculate the probability of such collisions for the case where a random polynomial is chosen, you'd have to compute the weighted average of the probabilities of collision for all possible random moduli (for all possible number of irreducible factors of various degrees) --- I claim without proof here that this probability is worse that the value given for case (2).

Let me explain the reason why doing the calculations is important.

The practical value of using irreducible polynomials is that you have assurances as to the chances that a modification will go undetected. If you just pick a random polynomial that is probably reducible, you have no such assurance. And if you don't go through the calculations, you have no idea what kind of assurance claims you can make. Mechanical engineers have to

estimate probabilities of structural failure; electrical engineers have to estimate system downtime; it's only reasonable that probability calculations are needed in the area of computer security. And unlike the other disciplines, in security there are {\em active} adversaries, systems must be very carefully studied. Just hand-waving without calculation is not enough.

I argued for the use of randomly chosen irreducible polynomial moduli rather than a fixed set of moduli or just random polynomials because it is a good design decision. If the moduli are reducible, your system is weaker. Why build a weaker system? I don't want to have to worry that it used x^{31} (last word), x (last bit), $x+1$ (parity), or similarly weak choices, just as DES users should avoid the known weak keys.

Since Rabin's algorithm for generating irreducible polynomials can be very cheaply implemented, the assurance is worth the engineering tradeoff. The question isn't whether a virus can exploit the fact that your protection system is weak; the question is: Why would you want to deliberately build a weaker protection system when building a strong one has negligible additional cost?

I figured that this was simple common sense and omitted making it explicit.

Further, engineering common sense aside, we should pay attention to a lesson that we have learned from security work over and over again: just because we can't think of a way to break a system does not mean there is {\em no} way to break it. Just because we can not currently imagine a loophole doesn't mean there isn't one, and trusting a security system just because we don't think it's breakable sans proof is a bad idea. That's why the probability calculations were necessary. That's why computer security should be grounded in Theory.

The original claim that I disagreed with was that, restricting ourselves to worrying about checksums, what algorithms to use is of little consequence as long as many different ones are used. I think I've made it clear that the choice of algorithms is important, shown that the choice should be based on carefully analysing the algorithm, and sketched the mathematics for why you would want to use irreducible polynomials in practice.

Bennet S. Yee	internet: bs...@cs.cmu.edu
School of Cucumber Science	bitnet: bsy+%cs.cmu.edu@cmuccvma
Cranberry Melon	csnet: bsy+%cs.cm...@relay.cs.net
Schenley Park	uunet: ...!seismo!cs.cmu.edu!bsy+
Pittsburgh, PA 15213-3890	phone: +1 412 268 7571

---

19 Jul 1990, 12:57:13
Y Radai
to
.IL> <1990Jul7.0...@cs.cmu.edu>

Reply-To: ra...@mush.huji.ac.il (Y Radai)

Organization: The Hebrew University of Jerusalem, Israel
Lines: 319
Apparently-To: post-...@ucbvax.berkeley.edu

I will divide my reply to Bennet Yee's article of 7 July into four sections:

I. Application-specific Viruses
II. Why MD4 and Snefru are Insecure for Anti-Viral Purposes
III. Irreducible Polynomials
IV. Conclusions

I. Application-specific Viruses

Bennet writes:
>Here is a simple conceptual error that can be easily cleared up. Note that
>I described viruses that attack a {\em set} of application. For example, it
>is simple to write a virus that can specifically attack just the two
>programs WordPerfect and Lotus 1-2-3. Using the imperfect medical analogy,
>you can think of WordPerfect as the vector used to infect 1-2-3. The idea
>is that a virus can a priori know about a small set of applications --- and
>copying any one of these applications to an uninfected system (and running
>that application) will cause all other applications on the otherwise
>uninfected system also in this set to become infected as well. This is how
>such viruses can propagate. The rate of propagation depends on the size of
>the set of target applications and what these applications are.
>
>Just because a computer virus is specific to certain programs do not mean it
>can not propagate. I'd argue [this part is not based on theoretical
>understanding but rather on engineering common sense] that such viruses will
>be harder to detect --- instead of blinding infecting/modifying all programs
>and thus causing some of them to fail at the ``wrong'' time, the infection
>can be completely benign --- at least until some predetermined environmental
>trigger comes along. Greater specificity should not be automatically be
>equated with lesser power to damage.

There are a lot of things wrong here; I'll start with the less important ones:

First of all, application-specific viruses are *not* harder to detect, regardless of whether the detection is by a checksum program, a RAM-resident monitoring program, or a scanning program for known viruses.

Secondly, there aren't many viruses which cause programs to "fail at the wrong time", and to the extent that there are, it's not because of lack of specificity of the host program.

Thirdly, if "power to damage" means the amount of damage to any *one* user, then I don't understand the point of Bennet's last sentence above since I never said otherwise. What I did say in my previous article was that it is almost impossible for Bennet's virus to *propagate* (in the sense in which viruses propagate) if it is specific to *a single* program.

Fourth, even when we take into consideration that his virus can propagate to *several* applications, propagation would be confined almost entirely to other applications of *the same* user. Now suppose that its damage is to format the disk on which it resides if a program containing it is executed on or after some particular date. No matter how many applications it has spread to on *the same* disk, the damage will be the same. (At most the damage might take place a few days sooner than if only one application were infected.) What malware writers are interested in is in causing damage to as many *different* users as possible. As evidence for how unimportant propagation on the *same* disk is, note that many PC viruses are boot-sector viruses. By their very nature, they cannot spread to anywhere else on the same disk. But they *can* spread to the boot sectors of *other* disks, and this is what the author is interested in. Bennet's virus, on the other hand, would not propagate to other disks. True, any of the infected applications could be *copied* to another user's disk, but that's a very *inefficient* way to spread malware compared to what a genuine virus can achieve. Moreover, that's true of a Trojan also. In short, *such a limited virus would be no more successful than a Trojan*. So why should anyone bother to compute checksums and add propagation code when practically the same thing could be achieved much more simply by a Trojan? Moreover, a Trojan has the advantage over a virus of not being detectable by *any* checksum program activated after it is first installed on the disk. (Detecting Trojans requires a *different* strategy; see my previous article.)

Fifth, recall that Bennet originally mentioned the use of application-specific viruses in the context of a very specific algorithm:

>The virus writer can easily make the virus such that an infected application
>would have the same checksums with respect to a standard set of polynomials.

>Off-line searches by the virus writer _is_ practical.

With respect to a *standard* set of polynomials? Of course! But a virus writer is very rarely in a position to know where his virus will spread to. So despite the inefficiency of its propagation, Bennet can't be sure that his virus will be confined to checksum programs based on CRC with a standard polynomial. Even if there is only one user in the vicinity with an "intelligent" checksum program, the virus will get detected and the word will spread much faster than the virus itself. Bennet's argument makes sense only on the assumption that the virus can be confined to an environment in which several checksum programs based on a standard polynomial are used, but no checksum programs using any smarter program are used. That's not a very realistic assumption, and is ordinarily the exact opposite of what a virus writer desires.

Well then, maybe this particular virus writer, in contrast to the great majority, is interested in attacking only the users in a specific "closed" locality where (a) he *knows* that they use only CRC with standard generators, (b) he doesn't care that the virus will spread to other users only by programs being copied from one user to another, and (c) he doesn't care whether the virus gets detected elsewhere. *Now* is his virus practical? No, for the reason I

mentioned previously: He could obtain the same result by simply writing a Tro-
jan, without need for checksum calculations or propagation code.

Moreover, don't forget my Condition (A):

>> For any given file f, F(f) is (in general) different for each user (or
>> computer or disk).

Admittedly, my notation wasn't rigorous here, but my intention was quite clear:
For any given file, the algorithm must yield a different checksum for each
user. *That definitely rules out a CRC with a standard polynomial.*

Finally, just what is it that Bennet is trying to prove? Apparently it's
that
> Off-line searches by the virus writer _is_ practical.

Well, I never said otherwise. Remember that the discussion between Bennet and

me started because I wrote:
>> And from my experience, people with a cryp-
>>tographic background tend to make several incorrect assumptions when discus-
>>sing anti-viral matters:
>> (1) that the most important ingredient in a checksum type of program is the
>>algorithm;
>> (2) that the virus has an unlimited amount (or at least a lot of) time to do
>>its "dirty work";
>> (3) that the major threat facing a checksum program is that a virus might be

>>able to forge a checksum ....

Note that I made no absolute statements such as "Offline searches are *never*
practical". The closest I came to this was when (in my discussion of Assump-
tion (2)) I mentioned that a virus writer does not have in advance a single
file-checksum pair for analysis (i.e. for calculation of the key). That state-
ment has not been contradicted.

In stating (1)-(3) above, I was merely expressing my opinion that (according
to what I have observed) when crypto-people start to attack the virus problem,
they tend to overemphasize the things mentioned above because they think that
viral problems can be solved by the same means which they're used to using.
And I was thinking in particular of those means used for authenticating digital
signatures. Indeed, Robert English made precisely this claim:

>It seems to me that this general problem is similar to that of attaching
>"digital signatures" to documents. Someone running a program wants to
>be able to authenticate the author of the program as a trusted vendor,
>and the authentication must validate both the author and the program.

And I suspect that those who suggested MD4 were also of the opinion that the

virus problem is similar to the Digital Signatures problem.

Well, as I mentioned in my previous article, the Digital Signatures problem is much closer to the problem of verifying that a file has not been altered between the time it is written and the time it arrives on our disk than it is to the viral problem which Gordon Letwin raised (verifying that it is not altered *after* the file arrives on our disk).

II. Why MD4 and Snefru are Insecure for Anti-Viral Purposes

In my previous article I wrote, concerning Bennet's point that MD4 and Snefru do not use keys, as follows:

>>For I assume that the implication of no keys is that all users will obtain the
>>same checksum for any given file. If so, I consider the use of such programs
>>to be insufficiently secure in the anti-viral context, despite their value for
>>authenticating digital signatures.

Bennet's only comment on this passage was to tell us how to construct a family of checksum functions from such algorithms, and he mentions that "this works only if you have a one-way hash". But later he claims that "If you really have a one-way hash, ... the entire exercise of building checksum functions from one-way hashes is silly." Hmmm.
In any case, I would like to push my above point a bit farther and explain just why I considered such algorithms to be insecure for anti-viral purposes:
The only type of checksum forging considered by Bennet so far is that in which a program attempts to create an altered file having the same checksum as the original file. Now offline forging by the virus writer is, as he has mentioned, possible with certain trivial algorithms, e.g. if it is known that the user is using CRC with one of a small number of standard polynomials. But it is unlikely to work with any more intelligent scheme. And if we consider forging by the virus itself, then necessary conditions for such forging to be possible in the anti-viral context are that the user (ignoring our recommendations) keeps his CSB (CheckSum Base) on a disk which can be read by the virus and that the virus knows how to interpret the information which it reads.
If it is felt that exploitation of this possibility by a virus is feasible, then it's worth thinking a bit about the following argument. Under DOS, any file which can be read from can also be written onto (unless hardware protection, which is rather expensive, is used). This raises the possibility of a *different* kind of forging: Instead of modifying the file to fit the checksum, the virus modifies the checksum to fit the file. I.e, it alters the checksum in the user's CSB so that it corresponds to that of the infected file. If the former type of forging is possible, then so is the latter. And it is much simpler and faster than the above type, since there is no need to use trial and error.
Now if we use an algorithm (such as MD4 or Snefru) for which all users obtain the same checksum for any given file, and if the CSB is online, then after infecting a file, the virus can easily compute that unique checksum of the file (by incorporating the algorithm into its code) and write it into the user's CSB. This is why I feel that MD4 and Snefru are not suitable for anti-viral pur-

poses. (Of course, the fact that cryptographic methods are generally slow is another factor in the same direction.)

Now let's take a look at what Bennet writes:
> {\bf NOTE}: If you really have a one-way hash, there's no need for
> criterion A

where he characterizes a one-way hash as one in which

> it is infeasible to find an input given an arbitrary hash (output) value

and my Criterion (A) is given above.

From the above discussion it seems to me that Bennet is wrong in saying that with a true one-way hash function there's no need for Criterion (A). Such a function would not allow forging by the first method, but it *would* allow forging by the *second* method if Criterion (A) is not satisfied. Criterion (A) seems to be necessary for the viral problem *even with a provably one-way hash function*.

III. Irreducible Polynomials

>The practical value of using irreducible polynomials is that you have
>assurances as to the chances that a modification will go undetected. If you
>just pick a random polynomial that is probably reducible, you have no such

>assurance. ....
>Further, ... trusting a security system just because we

>don't think it's breakable sans proof is a bad idea. That's why the
>probability calculations were necessary. That's why computer security
>should be grounded in Theory.

I have nothing against theory as such. Some of my best friends use it. The question is whether the theory which Bennet is using has been *correctly applied in the present situation*. Suppose that for a certain degree of polynomial we find that the probability that there exists another file having the same checksum as a given file is 2^(-60) when the generator is chosen randomly among *irreducible* polynomials and 2^(-12) when it is chosen randomly among *all* polynomials. This difference is significant when trial-and-error methods are feasible. However, in our case, the relevant probabilities are not those of such a file merely *existing*, but those of finding such a file and of *knowing* when it has been found. If, as Rabin assumes, the CSB isn't accessible, then these probabilities are not 2^-60 and 2^-12, but ZERO and ZERO. The "theory" required to draw this conclusion is rather minimal, to say the least. No talk of rings or fields or Chinese Remainder theorems. How inelegant! But *that's what's appropriate to the question which I posed in my previous article*.

>The question isn't whether a virus can exploit the fact that your protection

>system is weak; the question is: Why would you want to deliberately build a
>weaker protection system when building a strong one has negligible
>additional cost?

Just where in my articles do you find that I "want to deliberately build a weaker protection system"?? I merely asked, given the theoretical advantage mentioned by you, how it could be exploited by the virus writer (in the environment described by Rabin) to give him a *practical* advantage. I am well aware that Rabin's algorithm for generating irreducible polynomials can be very cheaply implemented, and therefore it never occurred to me to advocate *not* using it. However, when one finds a portion of an algorithm whose purpose is unclear to him, isn't it natural to ask what it's there for? Can't you imagine a person asking this question out of pure curiosity without coming to the conclusion that he is advocating a weaker system and without telling him that what he asked isn't the question?

IV. Conclusions

I sincerely appreciate and thank Bennet for his contributions, from which I have learned quite a bit. Nevertheless, I'm now more convinced than ever before that my original contentions (see (1)-(3) in Section I above) were, on the whole, valid.

(1) I wrote that it's a mistake to think that the most important ingredient in a checksum type of program is the algorithm. From my explanations, it's clear that I was not talking of trivial schemes like CRC with a standard polynomial, but of the choice between more intelligent ones such as CRC with a random polynomial, MD4 and Snefru.
And what I was most referring to when I said that the algorithm is not the most important factor was the fact that even the most sophisticated algorithm is useless if it is not implemented very carefully, e.g. if it is activated in an infected environment or if the implementing program does not take into account the possibility of exploiting what I called "loopholes" in the system. (BTW, I previously mentioned the fact that there was a "test" virus which exploits one of these loopholes. Well, now there is also a "public" virus which exploits it.)
I do, however, have a slight amendment to make to (1). It seems that there *is* a difference between these algorithms, only it's in the opposite direction from what most crypto-people think! I have shown that if online forging is possible, then a keyless algorithm (like MD4 and Snefru) is *inferior* to a program which uses a key (such as CRC with random polynomial).

(2) I wrote that it's a mistake to think that the virus has an unlimited amount (or at least a lot of) time to do its "dirty work". Bennet mentioned a way in which a virus could have time under DOS (by hacking the clock interrupt to run a coroutine), and he discussed how a virus *writer* could have lots of time. The former *might* be practical, but the latter, in my opinion, is not (see the discussion above and the last sentence of (3) below).

(3) I wrote that it's a mistake to think that the major threat facing a checksum program is that a virus might be able to forge a checksum. There are simply too many obstacles to overcome: the CSB may not be accessible; if it is accessible it may be encrypted and/or stored under an unknown name; what works against one type of checksum program won't work against others; etc. Moreover, the above point about loopholes shows that under DOS *there are much simpler and more universal ways of getting a virus through than forging checksums*.
All Bennet has done is to describe a very specific situation in which a virus writer could forge checksums. But his example is based on the assumption that all checksum programs use a particular scheme (viz. CRC with one of a small set of standard polynomials) and that a particular type of virus (viz. application-specific) is used. This has no effect on my argument since my criteria exclude such a scheme, since the environment which he describes is not very realistic, and since the number of users to which his virus can spread is no greater than in the case of a Trojan.

Y. Radai
Hebrew Univ. of Jerusalem, Israel

ra...@mush.huji.AC.IL
RA...@HUJIVMS.BITNET
RAD...@HBUNOS.BITNET

P.S. Bennet's article began with the following

>[Quasi-editorial note: I use the LaTeX-style notation of
>\\begin{quotation}...\\end{quotation} rather than the usual ``> '' prefix
>since prefixing Y. Radai's text would make an 80-column terminal wrap some
>of the quoted lines. His formatting style makes it difficult to simply

>rejustify his text because of the hyphenation. ....]

None of my lines exceeds 79 characters. If you were to prefix simply ">" (as many if not most people do) instead of "> ", everything would fit quite nicely. Even if the reply software of your mailer automatically prefixes lines with "> ", is your editor so primitive that you can't do a global replacement of "> " by ">"?