Peer Production of Privacy and Security Information

L Jean Camp*
Informatics
Indiana University
Bloomington, IN 47401

Allan Friedman
Harvard University
79 JFK Street
Cambridge, MA 02138
Allan_Friedman@ksgphd.harvard.edu

Abstract

If the public switched telephone system required human operators in the model of the turn of the twentieth century, as it required at the turn of the nineteenth, every man, woman ,and child in American would have to be a telephone switch operator. In fact, every individual who makes a phone call is a switch operator. The switch interface is a twelve button pad that fails almost every possible standard for highly usable design.

Today, the Internet has reached a point where every person is required provision security, usually for their home networks. Unlike telephony, however, users are failing systematically in this function.

How can individuals secure their own computers? What form might a system that encourages or enables users to generate security information take? How can an individual detect and repair a subversion in a machine, if the machine's reports about its own state are not trustworthy?

Introduction

Peer production of security information is a radical new paradigm in computer security. Rather than having centralized points that generate security information, individuals combine their own limited understanding to create a larger view. Peer production can be justified by economic arguments, history of network security failures, and a characterization of security. Current centralized vulnerability management will continue to be necessary. Yet it is inadequate. Not only does this paper present a general case for peer production, it presents a sketch of the type of mechanism that can be used to enable this production.

Theory & Motivation

Inevitability of Vulnerabilities

In this section we explain that the existence of vulnerabilities is an inherent element of innovative computing, and explain why neither a centralized security mechanism nor software engineering can be a total solution to the problem of network vulnerabilities. Thus an embedded system that assists end users in protecting themselves is a necessary element of any security system. Even if trusted computing were installed in every machine, the complexity and interoperation of the code could be leveraged to implement undesirable behaviors "on behalf" of the machine's user.

First, it must be understood that there will always be security failures. Better software engineering is desirable but not sufficient to remove all vulnerabilities. Carl Landwher et al (1994) have identified three classifications of software engineering errors that lead to security holes:

- Coding Errors are holes left in the code that permit exploitation of the process, the most common being the buffer overrun. Coding errors can arguably be prevented, or at least patched, by adding error-checking to the code, and some type-safe languages can mitigate this common flaw. Yet adding error-checking for all inputs in a complex program has proven to be extremely difficult. Coding errors are rightly the responsibility of the programming institution. While they may never be entirely eliminated, the frequency of coding errors can certainly be reduced.
- Logic Errors are design flaws where malicious interruption of code can enable an attacker to obtain privileges. There is no agreed-upon method for eliminating all potential logic errors.
- Emergent Errors result from the interaction of different software packages, and thus cannot be the responsibility of either interacting program. These errors may not arise until after many hours of use in a specific implementation of multiple software packages, or occur as a result of a particular hardware configuration. Emergent failures cannot be prevented at the production level.

Because of the existence of emergent errors, the lack of generally applicable formal methods for avoiding logic errors, and the impossibility of perfect quality control, a security layer must be made available to the end-users system.

Single, centralized solutions that tend to offer a single model of user security have consistently failed. The misaligned incentives discussed above are compounded by the high costs of large scale software verification procedures. Moreover, centralized implementations violate a central tenet of any security system in that each has a single point of failure. The malicious MSBlast worms attempted to disable the very server that provided the patch to protect against the worm.

Thus robust highly available technology offering protection must be distributed, and function in a heterogeneous environment. It must engage the user, and integrate the intelligence at the end point of the networks. It must not require simultaneous coordinated adoption, but rather be able to grow organically by one community at a time. Such requirements indicate a peer-to-peer solution.

The emergence of P2P systems empowered end users. Technically naïve computer owners have used P2P systems to rate, transfer, distribute and integrate information for results that would previously required significant technical expertise. P2P systems work by leveraging the uncertain processing power, communications bandwidth, and storage at the edge of the networks. Despite the recent backlash against P2P, and the fact that such systems have thus far been considered primarily as security risks, these distributed networks of users can, by their very structure help users easily maintain and improve the security of their systems. P2P security is complicated by its decentralized nature but the absence of a single critical system offers unique potential for securing networks (Friedman and Camp, 2004).

Necessity of Information and Incentives

Security is currently a lemons market. (Landwher, 2004) The lack of comprehensible information about security threats reduces user motivation. Given the publicity major attacks

receive, and their widespread impact, an observer might predict that users would spend time to protect themselves, or at least spend money on security products. Yet that is not the case. Few users make a $50 investment to protect a $2000 machine (and remove its proclivity to participate in the rapid and rampant spread of exploitative code). Those that do often fail to make the minimal effort to update their defenses. A mechanism is needed that both raises end-user awareness of individual own risks and lowers the barriers of effort in terms of price and cost. While risk communication may increase user awareness, this proposal can be implemented in conjunction with and is not in opposition to risk communication.

Users lack the full information, interest, ability, and, arguably, the incentives to defend their systems. Security information is scarce and not accessible, and users are focused on other aspects of their user experience; many security operations are also beyond the basic competency consumers possess. In fact, not everyone sees the direct benefits to secure their personal systems. (Camp and Lewis, 2004)

There have been a plethora of theoretical and corporate approaches. Apple implements push technology, whereby users are reminded upon start-up. RedHat was first to implement on its personal desktop software implemented an alert on the desktop management system that indicates green, yellow or red depending on the current status of the users machine in terms of available patches. Microsoft has traditionally implements a pull model, where users are expected to connect with Microsoft intermittently to upgrade their machines. Using the security-repairing code provided by Microsoft, Symantec and Computer Associates provide regular alerts to their end users, turning Microsoft's pull into a competitive push market. In addition, Windows XP includes a zombie detection mechanism that is subject to trade secret constraints.

Methods and proposals for solving the problem of chronic vulnerabilities include the creation of liability for end users, insurance, liability for software producers, self-proving code, type-safe code, better software engineering, and open code. Proposals closest to ours are arguably mechanism for sharing trust (Beth et al 1994; Ellison on PKI); reliance (Golberg, Hill & Shostack, 2001) and risk include careful calculations of transitive trust. Yet our proposal does not requires extending trust in the sense of increasing ones vulnerability, depending on the version of the proposal chosen (Camp, 2000). Our proposal requires extending trust in that an individual allows their machine to take actions the individual would not otherwise take. Our proposal arguably requires confidence rather than trust. That is to say it requires belief in the system to participate but participating does not make the user worse off; nor does it increase probability of harm. Of course, users can choose to participate by sharing access.

This proposal is targeted at home users of broadband. Home users of broadband are particularly susceptible to malicious code. Each machine has valuable processing capacity and connectivity, but may not be capable of understanding and using the necessary protection mechanisms. Their always-on capacity and slight defenses make them prime targets for intruders wishing to use others machines for malicious purposes. Corporate networks have designated security and information services employees; yet even this has proven inadequate for managing patching behavior of users. Home users have little time, and tend to rely on obscurity and anonymity to prevent attacks. Home users do not realize that their processing, storage, and communications

resources are valuable to attackers. It is clear to the average user that he or she can reach millions of people from their broadband-enabled machine. It is not equally obvious that millions of people can reach back.

In this paper we propose Good Neighbors, a P2P system that enhances user security. The proposed system will provide a framework for solving chronic security problems, but will not introduce new vulnerabilities. The P2P system will enable users to rate, transfer, and install appropriate software updates to close security vulnerabilities.

System Solution

There are three key observations with respect to this system. The first is peer production of information. End users need security, and most are aware, from experience or media, that computer security is important, yet they may lack the interest and competence to protect their systems. Using the wisdom of crowds (Surowiecki, 2005) this system evaluates and selects systems. Like the prevention of epidemics by immunization, this system uses some minimal user involvement to reduce the impact of the inevitable attacks.

The second key feature is highly scalable utilization of resources. No single course has the ability to evaluate all the machines on the network. In order to secure all the machines on the network; it is necessary to involve every machine on the network.

The third essential component is making state visible within a social network. Attackers can determine if individuals' machines are subverted or vulnerable. With Good Neighbors, individuals and their self-selected social networks can assist each other in protecting their own information. There is little visible incentive for users to patch their own machines. The damage each user does to others is deniable to the point of invisible. Vendors include materials in patching code (e.g., code intended to prevent interoperability, such as DRM).

Good Neighbors is aligned with incentives and user behavior. Requiring end users to actively patch systems has been proven to result in the existence of long standing, well-known vulnerabilities. Corporate networks have centers that evaluate patches, while users at the end point can obtain little information about which patches are aligned with their interests. While better software engineering is desirable, the work explains why this alone is not sufficient to remove all vulnerabilities.

Scanning and *Tabula Intuta*

A common means for detecting internet vulnerabilities is to "scan" them. The act of scanning entails activities ranging from seeking active ports or checking the version of communicating software to actively attempting to exploit known vulnerabilities. Scanning software for self-evaluation since SATAN was released on April 5, 1995. Such software is targeted at system administrators because of the difficulty in interpreting the results. Moreover, on larger systems, an increased number of services running leads to increased complexity of scanning feedback. Users need a simplified interface, unlike the standardized but highly technical SATAN or

SNORT outputs. Recent research has demonstrated that this information can be made more usable (Yurcik et al 2003) but the focus is still on the administrator of large networks.

Home users run fewer services, and require an interface that reflects the typical lack of services or informs them of the service. For example, most users do not host web servers on their machines, but being told that Apache is running on port 80 will not tell your average user that there is a malicious party using their machine as a server. Active user involvement in regular system functions should be minimal.

This work also assumes a descriptive mechanism for security conditions. This work assumes that such conditions exist, and are expressible in application of the various SAML-based security languages. At the very least, we assume that there exists a body of legitimate patches for vulnerabilities and at least one trusted site for distributing them. In order to simplify the description, assume the initial patch distribution point is CERT/CC. In theory, market forces develop when demand awareness is high enough to place consumer value on security information; that information could be paid for by the consumer seeking security, or a remote host seeking to accrue a reputation in order to earn the trust of customers. However, a full analysis of the economics and politics of the computer security industry is outside the scope of this research, it is adequate to note that there exists a party that validates and hosts patches.

Having a scanning client itself is not sufficient. Security is not applied from perfect starting conditions; we describe the initial state as *tabula intuta*. Unlike a world of blank slates, the computer security practitioner presumes that every system may be insecure and compromised in its initial state. *Tabula intuta,* (defenseless or unsafe slate) means that no one system can trust itself enough to scan itself. Once compromised, the owner of a system—especially one lacking security tools or experience—cannot be sure that any measures taken actually secure the system. The scanning tools, the patches, the feedback that a patch has occurred—all can be compromised.

The failure of traditional patch architectures has been clearly demonstrated in several recent attacks. Users will believe a system is patched if the system itself claims to be patched. A flaw in the Windows Update feature let many users—including the admins of a US Army server—believe that they were not vulnerable to MSBlast because they possessed the registry key to the file, but for some reason had not fully downloaded and installed the actual patching file (Kotadia, 2003). Moreover, the actual patching source itself can be vulnerable. The Microsoft Windows Update website, source for Windows users' security patches, reportedly fell victim to the Code Red defacement attack (Leydon 2001). More seriously, it was explicitly targeted for a denial of service attack by the MSBlast worm. The attackers used the IP address of the site as the explicit target, making evasion somewhat easy in that case, but the attack highlighted the vulnerability of relying on single sources for patch distribution and patch application. Even users who are sophisticated enough to protect themselves with consumer firewalls are not safe; the Witty worm attacked a vulnerability in BlackICE and other internet firewall products, corrupting the hard drives of those machines (Shannon and Moore, 2004). Under the conditions of *tabula intuta*, individual machines cannot reliable fix themselves, even with the help of a centralized server.

A third party scanning a system, however, is not as susceptible. Traditionally, third party scanning has a negative reputation in the security field. Administrators see scanning as an intrusion on their turf, and it is often perpetrated by malicious actors looking for weaknesses to exploit, or vigilantes who are often careless in their activism. From a systems defense perspective, however, third party scanning can be an invaluable and incorruptible information source. A system is either patched or not, and a connecting machine can report whether or not it successfully gained access. We thus take the rather unorthodox position that peers should, in fact, scan each other just as good neighbors watch each others' homes for unusual activity.

The Good Neighbors system combines the functional scanning of the "good worm" with the robustness of P2P networks and the embeddedness of social networks. Overlapping groups of known parties, with some degree of trust among them, scan each others' machine, looking for known security holes.

At this point there are a variety of ways to implement a good neighbors system. One is based on social networks. The other is based on herd immunity. Both have an underlying epidemiological model.

Epidemic Models

In the epidemic model, users scan neighbors machines when a vulnerability is addressed by the release of a "good worm". On finding one, they patch the system utilizing only the pre-existing vulnerability, and then leave record of having done so with the patchee's client. Each user's client maintains these networks, and requires little active intervention from the user, once set up. Basing worm propagation inside finite, linked groups precludes the possibility of traffic escalation, since the worm will only contact a small, established set of peers. This system thus capitalizes on the good worm's individual benefits without the devastating externality of unbounded growth.

The distribution of good patches depends on the assumptions about network topology and distribution rates.

The system has several design goals. First, decentralization can be a virtue, not a vice, in information security. End users do not fall into an easily administered category to begin with, and systems that reflect this have a greater chance of success. Since the end user is not a reliable actor as either a qualified security professional or the verifier of a secure starting platform, some group interaction is desirable. Social networks already exist, and provide some degree of inherent security, since users can rely on several disjoint social networks to implement the Good Neighbors networks. Below, we discuss how these networks can be formed, what regular operation looks like, and how the "good worms" propagate through the system. First, we present the underlying mental model that stresses society-wide end immunity over the security of any given

The Epidemic Model

The underlying mental model proposed in this paper is a model of the network as a dynamic system that can be roughly characterized as population that is susceptible to an infection (Kephart, Chess and White, 1993). Conceptually, the flow of new subversions is tied to the stock of vulnerable hosts. The overall effect of a large infection can be mitigated by immunizing a substantial number of potential hosts. The concept of biological model can be found in more than the description of rapidly reproducing malcode as worms, viruses or even reproducing. The biological systems model can be supported by examining the patterns of diffusion, the importance of heterogeneity, and the mathematical equivalence of the diffusion of viral code and biological viri.

Consider the classic epidemiological model with a total population (Pop) with a an infected population I(t) at time t and where

Pr is the probability of recovery of those ill at time t-1
Pd is the probability of death for those who are ill at time t-1
Pe is the probability of encountering an ill person and
Pi is the probability of infection

then the infection rate can be modeled as a conditional probability based on the number of infected parties:

$$I(t) = I(t-1)\,(1 - Pr) - I(t-1)Pd + (Pop - I(t-1))[(Pe)|(Pi)]$$

This description can be used for dynamics of computer attacks when the virus is not attacking a previously determined set of machines. The above model can be used to describe a network with vulnerable, patched and offline machines, where

Pr is the probability of patching of those computers subverted at time t-1
Pd is the probability of bringing down those computers subverted at time t-1
Pe is the probability of being attacked and
Pi is the probability of success, or the probability that the machine has the vulnerability targeted.

Thus if there is an adequate threshold of machines protected there will not be an epidemic. Good Neighbors can be thought of as a shared immune system, or a peer-to-peer vaccination scheme.

Zou, Gong and Towsley (2003) have also attempted to leverage the epidemic model discussed above, but they advocate a quarantine-driven system to temporarily remove at-risk users from the network. In contrast Good Neighbors relies on network access to spread the patch, but it may be possible to align these two systems. First, Zou et al.'s system is aimed at rapidly spreading worms, such as the Slammer worm, which infected 90% of all vulnerable machines on the planet in less than 10 minutes. In such an instance, their system can improve network balance by removing systems that act as if infected, allowing for a high false positive rate. On re-entering, a system cannot be counted on to ask its neighbors for a scan, however, since the attack may disable that feature. It is not clear how Good Neighbors could be altered to detect the temporary quarantine of its Neighbors. One option would be to have the same network conditions that

trigger any quarantine also alert all local Neighbors, but this adds an additional degree of complexity to the protocol and creates further potential for adverse interactions. Furthermore, Zou's system is inherently reactive, which ours is preventative, since it spreads patches across a potentially vulnerable network.

Network Assembly

This patch propagation mechanism needs to be structured to fulfill the goals of controlled propagation and moderate user awareness. A user must have a way of introducing the user's machine and the Good Neighbor's system to others. We use the circles of friends and colleagues that define existing social networks for several reasons. Social networks have *trust.* While the authors do not wish to attempt to define this highly over-loaded term, various versions of this concept have been prescribed for both social networking and security—some degree of confidence in future behavior is needed (Camp 2003). Moreover, users can feel more comfortable authenticating their friends and colleagues, since shared knowledge and familiarity can confirm identities. This particular authentication mechanism, while not invulnerable, does not rely on a trusted third party, and thus avoids the infrastructural costs and flaws that accompany. Note that this degree of trust is important but not critical: we show below that an attacker will gain little by asserting membership in a trusted circle. Finally, established social networks are likely to contain interconnection without excessive overlap. Contacts in an employer's committee are unlikely to significantly overlap with a PTA network, and a circle of personal friends might lie largely outside both. This maximizes the spread of helpful patches and minimizes the chance that everyone will get hit with the same variant through other contacts such as email. Social network topology is extremely unlikely to exactly mirror network geography exactly.

Email is the mechanism for bringing new members into the system. To introduce a user to a trusted circle, the initiating party:
1. Sends invitee "join" email
2. Obtains reply to "join" email
3. Confirms "join" reply
4. Receives a confirmation after the invitee joins.

Each party takes two actions to generate messages, as the invitee responds to the initial invitation and confirms his or her presence in the system. In such an exchange, encryption is optimal for authentication more than confidentiality as the system can tolerate eavesdropping by the active nature of the interaction.

By having an automated reply-to plug-in for email, the system uses the most reliable and most widely used mechanism on the network: email. That email is the new authenticator is argued in (Garfinkle, 2003). Offline contacts prevent a Man in the Middle spoofing attack, as such an attack would require read and write access to the email users' mailbox. It is also possible that some social mention might be made of this application.

The protocol will have this basic foundation, where the client machine MID invited the machine PID to join the network:

```
MID -> PID:     Join            MID HMID(t0, PID, MID, nonce)
PID -> MID      Confirm         PID HPID(t1, PID, MID, nonce)
MID ->PID       Acknowledge     MID HMID(t2, PID, MID, nonce)
```

Here, tn is a time signal and the nonce creates a degree of confidence in the hash on the sender's key. Trust is further embedded into the system by riding on top of the confidence in the sending party's key, either from prior contact or faith in the PKI. Of course the most common implementation is where the invitee is not a client of the system, and needs to download the software:

```
MID -> PID:     Join             MID HMID(t0, MID, nonce)
PID -> MID      NoClient
MID -> PID      NoClientConfirm MID HMID (t1, MID, nonce) URL(MID)
```

Here, URL is the location of the software to download. At this point, the invited user must make an explicit choice to download the software as requested by the inviter. The client machine PID then invited MID, since the direction of the invitation is immaterial in the pair-wise interactions.

Note that no internal state is kept of the exact structure or distinction of the social networks that grew this network of peers. Clients keep lists of those to whom they should be connected, but these are pooled together. In addition to being a simpler implementation, this reflects privacy concerns of "guilt by association" within a given context. Users themselves do little but extend invitations and accept or decline. By using the already familiar medium of email, we reduce user complexity. Unlike other trusted entry systems, it is not necessary to measure the degree of trust. The social network connection allows the requisite filtering, testing and rating, and the Boolean output of accept or reject is made in light of prior social knowledge.

Operation

To see the system in full operation, we first look at the interactions of two machines, a scanner and a scannee. For the moment, we assume that each client has a list of vulnerabilities for which to scan on the other. This list consists of known security holes in common end-user software and the "good worms" to subvert and then fix them. Clients scan by attempting various attacks on the specified machine. If a vulnerability is found--and thus an attack is successful—different actions may be taken.

At one the attack software patches the hole, preventing future code, malicious or friendly, from exploiting that vulnerability. The attack and patch are then recorded for both the scanner and the scanned system. As a basic metaphor, this can be thought of as neighbor checking your doors, and locking any that are unlocked.

Alternatively the neighbor learns about the subversion and insures it is itself patched. This is analogous to getting a flu shot after the neighbors become ill. In this case, informing the subverted machine owner is critical.

Every interaction occurs in these pairs, so we do not lose generality by examining a base case. From the user's perspective, this operation allows users to have better security with the automatic patching of known vulnerabilities. Of course, we consider larger-scale effects of such a system in wide-spread operation below. Potential hazards include malicious exploitative attacks, network traffic overload or undermining of the security system itself.

The most obvious concern raised might be that hostile code could be inserted, to allow for malicious use of these scans, rather than a software patch. This possibility, however, existed before the implementation of the system and, in fact, necessitates the use of this system. A structure to scan specific machines in a non-network-based geography is unique, but not a critical security exposure, since a machine under the control of a malicious actor can scan as many systems as desired. The basic model of the attack-and-patch is the "good worm," yet the client observes local traffic and is well-behaved, preventing a snow-balling denial-of-service attack. The chief harm of most worms to date has been the crippling network traffic spikes caused by the procreative efforts of the malware, not the malware itself.[1]

Good Neighbors is specifically designed to be sensitive to network traffic issues. At some point after sufficient idle system time, a system will initiate a scan on its fellows. We have considered the possibility of denial of service attack and decided to use the TCP fast back-off mechanism. In order to prevent interruption the initial timing setting is five minutes. Since it will only scan those in its trusted circle, and scanning only occurs during periods of relative system inactivity, this does not pose a threat to network stability, nor does it introduce new vulnerabilities that did not already exist. The client program will interact only with this system to seek timing permission to install a patch. If a good worm has been sent by another machine (notice this occurs when there was no record of this vulnerability being patched) then it waits for idle time to install. If it does not have an opportunity in 24 hours it forces installation using annoy-ware. That is, it will begin to attempt installation every half hour unless the user actively prevents the installation. Otherwise, information about a security flaw will not be an interrupt for the user, but a report that the user can view at leisure.

The system is relatively open, and thus, vulnerable to familiar spoofing attacks. However, identities are integrated with reputation, and the system is designed such that undermining the security efforts of an entire social network is very difficult. Each client keeps reputation records (confirmed via hash chains and signatures) that ensure that no subversion of a machine or false entry will go undetected. This coordination increases the difficulty of an undetected vulnerability existing in a group of peers. The following information is traded throughout the system so that a network of Neighbors is aware of network activity:

```
t: time of scan
```

[1] Probably need to cite this

L. Jean Camp and Allan Friedman, "Peer Production of Privacy and Security Information" Ethical Surveillance, 8-10 June 2005 (Austin TX).

v: 0 or record of vulnerability
MID: machine identifier
MID-T: type of machine identifier
SID: Identifier of scanner
EID: Evaluated machine (the SID of the machine that was scanned)

where v is set if a vulnerability is discovered. This can be a string containing some information about the vulnerability, and the patch, including the author of the patch, and the data of release. The record sent to the patched machine is

$H_{SID}(t, v, MID)$ MID_T

The record published is to other users is

$H_{SID}(t, v, MID)$ MID_T

An obvious attack would be to lie about MID and get another machine to interpret it, thereby leaving a vulnerability not patched while the host machine claims that it has been patched. This would be particularly useful to those who intend to sell access to machines or return and use those machines for their own illicit purposes (Schecter and Smith, 2003), as it would essentially prevent trustworthy machines from repairing subverted machines.

Note, however, that other machines in a circle have a chance of sending out the same patch. Since each machine can see the records of the scans made, this the falsification can be verified on other machines patch records, identifying the immediate need to patch the vulnerability the attacker is attempting to hide. Future research can explore the possibility of re-installing the good worms at some probability in order to detect false records.

Each machine evaluates a reputation based on the records of others' scan. In order to verify the reputation record the evaluated machine keeps this tuple:

$H_{EID}H_{SID}(t, MID)$ MID_T

A machine can claim its own reputation as total service parameter and a list of tuples:

$H_{SID}(t_0, v, MID)$ MID_T
$H_{SID}(t_1, v, MID)$ MID_T
$H_{SID}(t_2, v, MID)$ MID_T
.
.
.
$H_{SID}(t_{n-1}, v, MID)$ MID_T
$H_{SID}(t_n, v, MID)$ MID_T

This could be embedded into a larger P2P system. In addition to having an established namespace and connection mechanism, many P2P systems use reputation to reward good users who share some resource for the public benefit. While care has to be taken to ensure that reputation is not forged, reputation systems have great potential to provide intrinsic value on altruistic acts (Dingledine, 1999). Riding on top of an established P2P system, or social network software like Friendster can provide some intangible but still valued incentive to participate and participate responsibly. Such synergy would benefit Good Neighbors and the host reputation mechanism, as awareness of one can drive awareness of the other. The reputation aspect of Good Neighbors should be designed with an open interface to be used by other systems.

Generation and propagation of vulnerabilities and patches

Security vulnerabilities must be identified, of course, and patches written. With one exception, all incidents of widespread malicious code have utilized *known* vulnerabilities: reports existed from such institutions as CERT, and the software manufacturer made patches available. Users simply did not patch their systems. With a detailed description of the vulnerability, and availability of the patch, designing a tool that exploits the former to apply the latter does not require an inordinate amount of systems knowledge beyond what would be expected of a professional systems security expert. Only in the case of the Morris worm did the attacker wreak significant havoc with a previously unannounced vulnerability he himself identified, as a so-called "Zero day" worm, and most exploits appear a month or later after the original vulnerability announcement (Kienlze and Elder, 2003). It is worth noting that Morris is currently faculty at MIT. Few attackers have this level of skill and expertise. In fact, not only do incidents occur well after vulnerability disclosure, most continue to grow after the patch has been released (Browne, Arbaugh, McHugh and Fithen, 2001).

Since generation of new scanning patches depends only on widely available code, we assume a starting case of a central generator, either affiliated with a major security center, an open source project or the implementer of the Good Neighbors system. As is discussed below, over time multiple sources of patches might be available. A "good worm" requires three elements: exploitation code, patch code and generic Good Neighbors code. The generator must only develop the exploitation code, as the patch code will be developed in conjunction with those responsible for the vulnerable software.

Once generated, patches need to be propagated through the network. The network structure has two salient features, one of which aids rapid spread, the other of which impedes it. A system with the vulnerability is scanned by another machine, patched, and that "attack" is registered on the scanned machine. A copy of the patch is left on the newly secured machine, so the next time the patched machine scans its trusted circle, it will spread this patch to all vulnerable machines, and then onto their trusted circles, and so on. Each client sees the network as a circle of Neighbors; all the clients together form a connected network. If, for example, we assume a fully connected network and if each machine scans once each night, then the spread follows this geometric growth rate over n nights:

$$(f - d)^n$$

where *f* is the number of trusted friends in a circle, and *d* is the number of duplicates in circles, where a scanned machine was already scanned by the same machine that patched the current scanner. Any worm that is patching its neighbors must have already been patched; from this perspective, the network actually looks like a tree, so propagation is geometric. Geometric growth can get very large, but of course, there is a natural boundary on unpatched machines.

Still, if time is critical, and a patch must race against an attacking exploitation, we may wish faster propagation. In this case, a machine can be 'woken' when patched, and scan its peers in turn. That is, the system can have two modes of distribution, one critical and the other "casual." Critical mode can be based on the freshness of the patch, with the date of the release embedded in the record of the vulnerability `v` above. When a client receives a new record with some $v_{date} - t < critical\_time$, it can immediately scan its neighbors. While special care should be taken to listen to the network for traffic each trusted circle will only have a maximum of (1/2)*f(f-l)* cross-scans, assuming a fully connected circle. This would spread very fast over the course of a single evening. Of course, such a controlled distribution could never compete with the most efficient malware. The slammer worm, for example, infected over 90% of vulnerable hosts within 10 minutes (Moore et al, 2003). Good Neighbors is largely intended as a preventative measure, but speed optimization can help when an exploit is expected soon after a vulnerability is announced. [2]

The default case is for each machine to continue to try to patch every machine that knows. Over time, this will have decreasing returns, as the number of patches grows, and every machine in the circle has been tested multiple times. Age could be introduced for each patch on a given machine, counting from when the machine first acquired the patch. Note that this age of residence on a local machine is different than the freshness of a patch existing in the network, discussed above. Age would be inversely related to the probability of a client scanning a neighborhood with a given patch. Since any successful patch installation now places the patch with age 0 on a new machine in the vicinity, feedback can ensure that vulnerabilities do not reappear in the network through new arrivals or missed patches.

Note that, again, apart from initial introduction of the patch, spread occurs largely independent of active user involvement. Mechanisms can exist to inform the user as to the state of her system, but should not be intrusive to the point of annoyance, where the user turns off the system.

There will, of course, be some security threats that Good Neighbors cannot address, such as Zero Day exploits. (Espiner, 2005)

Risks and Benefits

A design principle applicable to security systems is the Hippocratic Oath: First Do No Harm. One possible harm is the granting of trust to untrustworthy parties. Suppose an invitation is sent to a colleague, whose untrustworthy roommate shares the machine. Now this untrusted user can

[2] Keep this paragraph?

scan machines. Of course, any attacker could do this before the installation of the software. Furthermore, although the malicious party can now sign hashes, other machines in the circle can detect inconsistencies, and eventually root out the malevolent actor. Even if they could not, no one is worse off than if they had not downloaded the software.

Note that this security system utilizes worms. The traditional distinction in the computer security field requires that a virus needs some human intervention to activate the malicious code and trigger reproduction, while a worm can infect a system and distribute duplicates without any user involvement. This latter category has been seen as a larger threat because they cannot be actively prevented through responsible user behavior, and can reproduce at devastating rates with crippling traffic volume. Viruses are at least bound by the speed of their human enablers. A "good virus" is far more problematic as a security concept, since it requires user involvement. At very least, it forces the user to take action with each patch, impeding patch flow speeds and reducing the chances that the Good Neighbors system will be widely and regularly used. In the worst case, it encourages users to regularly execute unknown code, thereby encouraging bad computer hygiene.

The current crisis in compute security can be thought of in economic terms, as a market failure. Currently, total investment in security is not optimal. There is some agreement that security is a public good problem (Varian, 2003). Private security decisions have a public externality, as the cost of an insecure system is accrued by other systems that are subsequently infected as the malcode spreads. Various solutions to this problem have been proposed, including liability, (Anderson 2001), insurance markets for business risks (Schnieier 2002) and enforcement mechanisms from environmental economics (Camp & Wolfram, 2000). These proposals would function for the larger forces in the security market, where decisions, at least in theory, are driven by cost-benefit analysis, but not for vulnerable end users. In fact, some proposals could increase the potential risks for individual computer owner/operators without providing any mechanisms for enabling them to avoid these risks. Individual computer owner/operators include home users, small businesses, and other who have no specific security-focused expertise.

Information, ability to process available information, and attention span are necessary for a functioning market. Currently security information is available, but not processed so that end users cannot evaluate the importance of a particular bulletin. A particularly egregious example is the Slammer worm. As Slammer attacked the underlying SQL database, most users were unaware that they might be vulnerable. Any announcements containing an accurate description of the worm declared that the worm attacked "Microsoft SQL Server 2000" but how many users know that their PC is, in fact, an SQL server? In fact, any technically useful report might be construed by the average user as identification that the worm did not apply to them. Some simplifying system is needed.[3]

The proposed security system is the means by which a user of information systems can be sure that information being sent and received is valid and can be trusted. There is a recognized need to inform users about security to enable them to make rational information management

[3] Not entirely sure these two paragraphs do much for the paper at the end, no matter how I rework them. Make a brief

decisions. Similarly, more responsible, informed views for better decision-making are needed for information security. Even when a value-driven information system is not actively used, its presence in the user's environment makes the user more aware of the issue (Cranor & Garfinkle 2005). Making security visible to the user will generate more public awareness of security, increase information, and thereby enable the security market to function in a way as to serve consumer interest. At the same time, making security automatic will ensure the distribution of patches at a faster rate than direct user participation. The automatic propagation saves user cognitive effort, ensures propagation and is still embedded within a controlled and trusted environment.

Conclusion

We have proposed a novel peer-to-peer patch security mechanism that relies on the automatic distribution of patches within a trusted circle of known associates. The cumulative effect of interlocking social circles is a large-scale network for the rapid but controlled distribution of these patches. The shape and scope of the network can eliminate the largest threat of "good worms" that still overwhelm networks with their reproduction and distribution. At the same time, involvement of users creates some awareness of security processes, raising sensitivity to this important policy issue, and highlighting its salience as a public good. Reputation can enhance both user salience and awareness of the security mechanism. At the same time, direct user intervention is not needed, nor does the system rely on a centralized point vulnerable to attack. Good Neighbors will not solve all end user security issues, but it is a step forward in reflecting both security demands and user needs.

References

R. Anderson. "Why information security is hard: an economic perspective" ,ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference, page 358, Washington, DC, USA, 2001. IEEE Computer Society.

W. A., Arbaugh, W. Fithen, and J. McHugh (2000) Windows of Vulnerability: A Case Study Analysis. Computer 33, 12 (Dec. 2000), 52-59.

T. Beth, M. Borcherding and B. Klein, (1994) "Valuation of trust in open networks", in D. Gollman, ed., Computer Security --- ESORICS '94 (Lecture Notes in Computer Science), Springer-Verlag Inc., New York, NY, 3-18.

L. Jean Camp (2000) Trust & Risk in Internet Commerce, MIT Press.

L. J. Camp (2003) "Design for Trust," Trust, Reputation and Security: Theories and Practice, ed. Rino Falcone, Springer-Verlang (Berlin).

L. J. Camp and S. Lewis (2004) The Economics of Information Security, Springer/Kluwer.

L J. Camp and C. Wolfram (2000) Pricing security. In Proceedings of the CERT Information Survivability Workshop, pages 31–39, Boston, MA, USA, October 2000. CERT.

A. Cavoukian, (2005) “The Market for Privacy”, Keynote: Workshop on the Economics of Information Systems, 1-2 June 2005, Cambridge, MA.

L. F. Cranor and S. Garfinkle, eds., 2005, Security and Usability: Designing Secure Systems That People Can Use, O'Reilly & Associates, Inc Sebastopol, CA.

M. Delio, (2003) "Are You a Good or a Bad Worm?", Wired News, 2003-08-19, http://www.wired.com/news/infostructure/1,60081-0.html, Last viewed 15 January 2006.

R. Dingledine (1999) “Chapter 16: Accountability”, Open codes: Voices from the Open code Revolution, DeBona, S. Ockman and M. Stone (eds) O'Reilly.

K. Dresser, (2005) “Patch Management”, CIRT Professional Day, Toronto CN, 10 February 2005.

T. Espiner (2005) Symantec flaw found by Tipping Point bounty hunters. ZDNET, October 2005, {http://news.zdnet.co.uk/0,39020330,39230317,00.htm}, last view January 2006.

A. Friedman and L. J. Camp (2004) “Security in Peer to Peer Systems,” The Handbook of Information Security ed. Hussein Bidgoli, John Wiley & Sons (Hoboken, New Jersey) 2004.

S. Garfinkel, (2003) “Email-Based Identification and Authentication: An Alternative to PKI?”, IEEE Security and Privacy, November/December 2003, Vol 1 No 6 pp. 20 -26.

I. Golberg, Hill and A. Shostack (2001) "Privacy Ethics and Trust", Boston University Law review, Vol. 81, N. 2 April. pp. 407 -422.

J. Kephart, G. Chess, and D. White (1993) Computer Networks as Biological Systems IEEE SPECTRUM May Vol. 30, No. 5, pages 20-173; 26 pages total.

D. M. Kienzle and M. C. Elder (2003) “Recent worms: a survey and trends.”, Proceedings of the 2003 ACM Workshop on Rapid Malcode (Washington, DC, USA, October 27 - 27, 2003). WORM '03. ACM Press, New York, NY,

M. Kotadia (2003) "Windows Update flaw 'left PCs open' to MSBlast" ,ZDNet UK, August 15, 2003 {http://news.zdnet.co.uk/software/windows/0,39020396,39115732,00.htm} last visited January 2006

C. Landwher (2004) “Information Flows in Computer Security”, in Camp & Lewis eds. Economics of Information Security, Springer-Verlag.

C. Landwher, Bull, McDermott & Choi (1994) "A Taxonomy of Computer Program Security Flaws, with Examples, ACM Computing Surveys, Vol. 26, Sept. pp. 3. -39.

R. Lemos (2003) "Good worm, new bug mean double trouble", CNET News.com Published: August 19, 2003 {http://news.com.com/Good+worm,+new+bug+mean+double+trouble/2100-1002_3-5065644.html} last viewed January 2006.

J. Leydon (2001) "MS Windows Update Site Falls to Code Red" Security Focus, September 2001, last viewed 7 August 2005.

B. Schneier (2002)We don't spend enough (on security). In Workshop on the Economics of Information Security, Berkeley, CA, USA, May 2002.

C. Shannon and D. Moore, (2004) "The spread of the Witty worm" IEEE Security & Privacy Magazine, July-Aug, Vol 2, No 4, pp. 46- 50

J. Surowiecki (2005) The Wisdom of Crowds, Anchor (Random House) NY, NY, 2005.

H. Varian (2003) "System reliability and free riding", N. Sadeh, editor, Proceedings of the ICEC 2003, pages 355–366, New York, NY, USA, 2003. ACM Press.

W. Yurcik, J. Barlow, K. Lakkaraju and M. Haberman (2003), "Two Visual Computer Network Security Monitoring Tools Incorporating Operator Interface Requirements" Workshop on Human-Computer Interaction and Security Systems, (Florida).

C. C. Zou, L. Gao, W. Gong, and D. Towsley (2003) "Monitoring and early warning for Internet". Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03), October 2003.