

Good Neighbors Can Make Good Fences: A Peer-to-peer User Security System

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

Internet security depends largely on the security of end-user machines, and it has failed. Individuals with little technical expertise are required to guard their own machines and monitor their communications, resulting in personal harm and the social loss of downed networks. There exists little visible incentive for users to patch their own machines. We propose a system that is aligned with incentives and user behavior. Centralized top-down solutions can be targets of attack themselves, while a computer that has been subverted can mislead attempts to identify it as needing attention. The model presented here is based on the premises of social trust and the peer production of knowledge to disseminate security information in an automatic yet robust and accessible fashion.

From this model, we develop the Good Neighbors system. The core of this system is a social-network-based client that allows communities to determine if a party in that community has a vulnerability or is acting as if subverted. The network structure provides rapid but controlled distribution of important security information, while the social arrangements enable the trust between users and increase the security of the system. The presented system is not a panacea for all security issues, but by leveraging existing social structures, it can help remedy a market failure while complementing other policy solutions.

Introduction

In August of 2003, the computer security world, already weary after a summer of headline-grabbing security problems, rallied to defend systems against yet another internet worm, the Blaster. In this case, systems administrators had both assistance and additional troubles from a rapidly reproducing but arguably not malicious code -- worm_blastMS.D, or “AntiMSBlaster.” Similar to earlier Blaster strains (it exploited a Windows RPC vulnerability), this “good worm” gained permissions through the security vulnerability, and then fixed that vulnerability in the infected system. That is, the worm prevented further malicious code from being able to attack the system. Despite the positive results of the ‘good’ worm still managed to bring down several networks from the overwhelming traffic generated while the worm attempted to spread itself, even shutting down some operations at Air Canada and CSX, the security industry’s response was mixed (Lemos 2003). Some systems were undeniable saved, suffering only patching, in an environment where far too many users failed to protect themselves. Worm_blastMS.D was the first “good worm.”

Many internet security crises are preventable by proper end-user patching, but as the worm_blastMS.D story shows, the current mechanism by which we expect users to secure their machines is flawed. In this paper, we propose a system that addresses the underlying problems of end-user security, inspired by—but far less dangerous than—the “good worm.”

Secure machines which have been fully patched are not at risk from malicious code exploiting known vulnerabilities that can compromise the machine and fill the larger network with traffic as worm attempts to propagate. In order to address the chronic failures in network security, we propose to use the tools of malicious code, together with the trust-reinforcing forces of social networks. We present system in which end-user machines automatically look for and fix vulnerabilities can disseminate valid patches in an effect and secure fashion. By advocating a peer-to-peer protocol with the goal of monitoring and repairing the network, we enable the use of the vast resources at the end points.

We begin by discussing the motivation for our approach. Resource allocation, history of network security failures, and theoretical analysis of security argue that vulnerability management will continue to be necessary. We then introduce our solution, first conceptually, highlighting the underlying mental model, and then detailing certain design specifications. Finally, we review how the system will handle various attacks, so as not to introduce new vulnerabilities, and discuss the merits of this approach in the context of other information security policy issues.

Motivation

The end user will experience continuing risks with respect to computer security. Thus the end user must have security protection integrated into the technology. Several general approaches exist, from automatic security software to ignoring the problem and hoping it

goes away, to legal and economics responses. None has been terribly successful, in part because any solution must reflect the nature of computer security problem while still enabling the user to understand and react.

An end-user security crisis

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.

Corporate users are often more interested in performing a task and avoiding downtime. Detecting and remediation of non-compliant endpoints is difficult in a technical and organization sense. *Endpoint systems are vulnerable and represent the most likely point of infection from which a virus or worm can spread rapidly and cause serious disruption and economic damage.* (Burton Group, 2004). However, security is not the end goal for most users, who simply want to use their systems without too much hassle, and the immediate benefits of proper security practices are not always readily felt. The monitoring and repairing of vulnerabilities must minimize the interruption of the user and reward the user with some visible mechanism.

Users lack the full information, interest, ability, and, arguably, the incentives to defend their systems. Thus there was the perception that any improvement in the overall level of security among internet users was a good thing. 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, and indeed, there is a large externality. Since another user can be infected by a neglectful computer owner, failure to patch generates a social cost. Furthermore, much of the harm of rapidly-spreading worms, for example, is in the quantity of traffic they generate.¹ The user bears all the cost of patching her own machine, but shares the harms wrought by failing to do so, in a classic market failure.

There have been a plethora of theoretical and corporate approaches. Apple implements push technology, whereby users are reminded upon start-up. RedHat Linux was first to implement on its personal desktop software implemented an alert on the desktop

¹ The standard model for a worm that has successfully compromised as system is to cause the host machine to contact as many other (often random) machines as possible in order to spread itself. As the number of infected hosts grows, so does the network traffic from these attempts, sometimes crippling the network.

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 pushing responsibility onto some party by the creation of liability for end users, or liability and insurance for software producers, or mitigating the risk of attack with 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 2002); 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 2001). 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.

The inevitability of vulnerabilities

First, it must be understood that there will always be security failures. Better software engineering is desirable but not sufficient to remove all vulnerabilities. Landwehr et al (1993) 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.

The majority of security literature focuses on the administrator level, preventing attacks on the technical side with a level of expertise outside the range of most non-technically trained users. This is reasonable, since the administrators of professional-level systems face more threats and have a higher cost of failure. Yet if security is to be salient to average user experience, some accessible user-friendly mechanism should allow users to accept some responsibility for the security of their online interactions. Users, after all, possess the incentives to protect their own information assets more than any other party. Lowering the cost of detecting and patching vulnerabilities will reduce the number of vulnerable systems that pose private and social risk.

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. Millions of users do not download free Windows patches, even after endless news reports of internet security risks. Some 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.

“Tabula Intuta” and decentralization

Even if end-users had adequate incentives to easily secure their machines, two obstacles stand in the way of the most direct solutions. If we assume that there will be some significant portion of the population which is not interested in purchasing extant prophylactic software in an open market, then they might be encouraged to either examine themselves for specific vulnerabilities, or at least tolerate a centralized authority directly patching their machine. However, both these approaches have theoretical obstacles: a subverted vulnerable machine cannot be trusted to identify itself as vulnerable, and centralized mechanisms are vulnerable to attack.

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. There already exists scanning software for self-evaluation, but such devices are marketed to system administration because of the difficulty in interpreting the results. Recent research has demonstrated that this information can be made more usable (Yurcik 2003) but the focus is still on the administrator of large networks; users need a more simplified interface.

Having an easy-to-use self-scanning application itself is not sufficient, however. Security procedures cannot be assumed to initiate under perfect starting conditions. We present the idea of *tabula intuta*.² 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* 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. By analogy, if one suspects one's accountant of cheating, any documents supplied by the accountant indicating fidelity cannot be seen as themselves trustworthy without outside verification. Users cannot rely on themselves, nor have they the necessary data or expertise to verify the claims of a simple patching system.

Moreover, a centralized mechanism that simply handles patch management automatically shows little likelihood of working, either. We have seen the failure of centralized patch architectures 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 attack³ 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 reliably fix themselves, even with the help of a centralized server.

More generally, 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, but not pose technical barriers of entry. Such technologies are often called peer-to-peer.

² Literally, “a vulnerable slate”

³ In a denial-of-service attack, the attacker floods the victim with repeated requests to connect, preventing valid users from accessing the victim's resource.

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. P2P systems create an additional layer, a namespace, that does not require that the user have a DNS record to perform traditional server functions.⁴ 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 (Camp and Friedman, 2004). More generally, P2P enables what Benkler calls “commons-based peer production,” which can enable solutions to social coordination problems using a diverse set of motivations and behavioral incentives, rather than market or structured hierarchies (2002).

The peer behavior presented below is third party scanning. A third party scanning a system, however, is not as susceptible to the problems of *tabula intuta* or centralization. Traditionally, however, scanning by a third party 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.

In the next section, we present the Good Neighbors system. This 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.

The Good Neighbors System

This section describes how the system functions and explains the concepts underlying the basic design principle. The basic model of decentralized but moderated patch distribution is presented here. The mechanics of the system are further explained in more detail in the following section.

In the Good Neighbors system, users select a set of trusted friends to form social networks. When notified of a vulnerability by the release of a “good worm,” users scan neighbors’ machines. On finding a vulnerable system in their social network, they patch

⁴ A namespace is the mechanism by which each resource is identifiable, usually uniquely. For example, Napster notoriously had a centralized namespace coordinated by a central server, while the Gnutella namespace is distributed. That is, a Gnutella node does not have to consult a central server to communicate with another node. The Domain Name System (DNS) is a centralized namespace, and has been the subject of multiple attacks. See (Stewart 2003).

the system utilizing only the pre-existing vulnerability, and then leave record of having done so with the patchee's client. The user's client maintains these networks, requiring little active intervention from the user beyond the initial set up. Worm propagation is constrained inside the pre-specified, finite, linked groups, which capitalizes on the good worm's individual benefits without the devastating externality of unbounded growth.

This paper assumes some trust-worthy and effective source of the patches. A descriptive mechanism for security conditions would also be desirable to enable client interpretation of the patches with minimal user knowledge required. At the very least, the model depends on the existence of a body of legitimate patches for vulnerabilities and at least one trusted site for distributing them. In order to simplify the description, a single provider, such as CERT, is assumed. 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 paper.

The trusted network

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.

Trust inside a social network not only informs risk behavior, but tends to increase altruism and community-oriented behavior through social capital (Putnam 2000). 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 seldom mirrors network geography exactly.

Neighbor patching

The basic metaphor underlying Good Neighbors is that of a neighbor checking one's doors and locking any that are unlocked. The basic interaction is between the scanner and the scannee, who are acquainted through the social network and have some degree of trust between them. 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—intervention is necessary.

The most direct, automated response to vulnerability discovery is for 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. The client on the scanned system may also download the patch if it does not already possess it, and check its neighbors in turn.

Alternatively, if direct patching is not possible or desirable, the scanning machine will attempt to contact the vulnerable host, either through the Good Neighbor client, or through a more direct route such as an email. Some users may feel reluctant to apply a patch if they have unique system configurations. A rich descriptive language can help automate that decision. Machines that refuse a patch will continue to be scanned in the normal operation of the system. If there is evidence that a vulnerable machine has been subverted, informing the subverted machine owner is critical.

Patch propagation through the network

For Good Neighbors to work, of course, security vulnerabilities must be identified 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. 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. Each exploit must be designed for the specific vulnerability.

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 potential attack, 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 (see below). When a client receives a new record with some specified 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 $(1/2)f(f-1)$ cross-scans, assuming a fully connected circle. This would spread very fast over the course of a single evening.

It is important to note that, even when accelerated, 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 by scanning for vulnerable machines on the internet as fast as the infected machine’s connection would allow (Moore et al, 2003). No prophylactic distribution mechanism could respond so quickly: automated solutions are necessary at the ISP level to mitigate worm spread. 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.

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). . The concept of biological model can be found in more than the description of rapidly reproducing malware as worms, viruses or

even reproducing. 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. Reducing the number of potential victims prevents the worm from taking over the critical mass of machines it would need to scan and attack a pandemic quantity. Since most worms attack too quickly for a time-driven epidemic model with immunity introduced by Good Neighbors, the problem does not require a full-blown epidemic model. Still, this approach demonstrates the value of increasing the number of patched machines.

Just as rapid distribution may be necessary, after most machines are patched, peer scanning is wasteful. 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. At the same time, Good Neighbors maintains enough state information and metadata to allow advanced users to set client behavior as they see fit.

There will, of course, be some security threats that Good Neighbors cannot address, such as Zero Day exploits. Moreover, extensive patching can lead to less stable systems, especially for complex professional systems. Computer Security Officer Magazine editor Scott Berinato points out that more patches may not be the answer, but only offers the solution of more vigilance and proper implementation of a patch-management strategy (2003). This system allows for just those precautions.

System Design

Network Assembly

Email is the mechanism for bringing new members into the system. 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) yet it was being used in that manner as early as the 1980s (Sirbu, 1984). 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. Offline contacts prevent a Man in the Middle spoofing attack⁵, since it is likely that some out-of-band mention might be made of this application.

Both parties are assumed to have a public/private key pair. The protocol itself might look something like this, 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)
```

After this exchange, PID and MID have an asymmetric link in the social network. Here, t_n 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 invites 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 (e.g. Beth et al 1994), it is not necessary to measure the degree of trust. As discussed below, a single bad actor can do little damage in the larger network, and the social network approach prevents widespread insertion of malicious nodes.

⁵ In a Man-in-the-Middle attack, the attacker can intercept all communication between the two parties, with the ability to eavesdrop, alter messages or, as mentioned above, impersonate one party of the transaction.

Security Issues

Good Neighbor clients only interact with other machines in their prescribed social network. We consider larger-scale effects of such a system in wide-spread operation below. The system relies on an open exchange of client behavior information to prevent individual actors from behaving maliciously without detection. The system is designed to introduce minimal new traffic or new threats to the network of users.

Enforcement through reputation

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. Reputation information here means a record of claims of scans made that can be verified against the state of machines in the social network. 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
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 and published for the community is

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

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 (Schechter and Smith, 2003), as it would essentially prevent trustworthy machines from repairing subverted machines.

Note, however, that other machines in the victim's circle have the opportunity to send 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:

$$HEID_{HSID}(t, MID) \quad MID_T$$

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

$$\begin{aligned} &HSID(t_0, v, MID) \quad MID_T \\ &HSID(t_1, v, MID) \quad MID_T \\ &HSID(t_2, v, MID) \quad MID_T \\ &\cdot \\ &\cdot \\ &\cdot \\ &HSID(t_{n-1}, v, MID) \quad MID_T \\ &HSID(t_n, v, MID) \quad MID_T \end{aligned}$$

While this prevents undetectable fraud, the reputation system does not in itself provide incentives to verify the claims of other nodes in the network. The social capital from the personal ties between peers may induce the extra effort. Moreover, this system could easily be embedded in 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.

Traffic management

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.

New risks

A design principle applicable to security systems is the Hippocratic Oath: First Do No Harm. The problem addressed in this paper was how to distribute end-user security patches effectively and robustly. To do so, we introduce a mechanism that depends on releasing code that exploits important system vulnerabilities. This may give some cause for concern. 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.

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.

Even when users engage with a value-driven information system minimally, its presence in the user’s environment makes the user more aware of the issue (Cranor 2002). 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.

Many of the macro-view policy proposals for internet security, such as tort liability (Chandler 2004) or insurance (Anderson 1994), are actually complementary to the proposed Good Neighbors system. Companies will have an even greater incentive to generate patches if there is an automatic distribution mechanism, since it will further shift responsibility onto the end users to secure their system once a solution to the software vulnerability has been found. Similarly, insurance schemas are at risk of a market failure

from adverse selection, and a user's operation of this automated system can provide a more reliable signal than a claim of patching. Good Neighbors will not solve all end user security issues, but it is a step forward in reflecting both security demands and user needs.