Secure Software Updates: Disappointments and New Challenges

Anthony Bellissimo, John Burgess, Kevin Fu
{*twon, jburgess, kevinfu*}*@cs.umass.edu*
Department of Computer Science, University of Massachusetts Amherst
http://prisms.cs.umass.edu/

Abstract

A client can use a content distribution network to securely download software updates. These updates help to patch everyday bugs, plug security vulnerabilities, and secure critical infrastructure. Yet challenges remain for secure content distribution: many deployed software update mechanisms are insecure, and emerging technologies pose further hurdles for deployment. Our analysis of several popular software update mechanisms shows that deployed systems often rely on trusted networks to distribute critical software updates — despite the research progress in secure content distribution. We demonstrate how many deployed systems are susceptible to weak man-in-the-middle attacks. Furthermore, emerging technologies such as mobile devices, sensors, medical devices, and RFID tags present new challenges for secure software updates. Sporadic network connectivity and limited power, computation, and storage require a rethinking of traditional approaches for secure content distribution on embedded devices.

1 Introduction

Every day, millions of computer users update software — some manually, some automatically, and some unknowingly. Indeed, 69 of the last 71 CERT Technical Cyber Security Alerts[1] suggest applying patches, upgrades, or updates to resolve security vulnerabilities [33]. Corporations reportedly spent more than $2 billion in 2002 on patch management for operating systems alone [3]. Surprisingly, many deployed systems do not make use of well-understood techniques from secure content distribution (Table 1).

At the same time, emerging technologies such as mobile devices, sensors, and RFID tags sporadically connect to the edge of the Internet. These emerging technologies bring additional challenges for securely updating embedded software. For instance, the FDA has recently relaxed rules on embedded software in medical devices [11, 13]. The design requirements are now less stringent for mechanical/electrical failsafes to act as backups to software. One implantable infusion pump resulted in two overdose deaths and several injuries because the software in the wireless programmer allowed a clinician to transpose the hours and minutes field [5]. While it is a challenge to design user interfaces to prevent accidents, even a sound user interface will not prevent malicious updates generated by a wireless adversary.

We first report on the state of the art in secure automatic updates. The results are disappointing. Many software update mechanisms lack basic security measures such as verification of digital signatures. Left open and unprotected, these update channels serve as an ideal backdoor for spreading malicious code.

Embedded devices such as mobile phones, sensors, medical implants, and advanced RFID tags increasingly run more sophisticated software. One could apply techniques from secure content distribution for updating software on these new technologies. However, traditional approaches in secure content distribution often assume a well-connected network or a well-provisioned client. Thus, we enumerate several of the new challenges for updating software on embedded devices.

2 Survey of Deployed Update Systems

We begin by analyzing the resistance of several existing software update systems to man-in-the-middle attacks (MITM). Surprisingly, several systems lack protection against weak MITM attacks (Table 1).

Apple MacOS Software Update. Apple signs its binary updates to ensure software integrity and authenticity. Each update includes a file named "signature" containing a 1,024-byte signature of the hash of the accompanying installation executable. Each installation binary is checked against its signature which may only be signed by the private key held by Apple Computer Corp. (whose public key is included on the operating system's installation media). No encrypted connections are needed, nor

To appear at the USENIX Hot Topics in Security Workshop (HotSec), July 2006, Vancouver, Canada.

[1]Two of the 71 alerts do not suggest applying updates because updates were not yet available.

verification that Apple manages the servers providing the downloads. System administrators running MacOS Server can mirror updates locally for network clients to install without providing attackers an opportunity to introduce malicious updates.

Microsoft Windows Update. A Windows Update client first determines which packages require an update. The client then downloads signed packages from a Web server, verifies the authenticity, then extracts and installs the updates. Microsoft automatic updates cover a large number of services and software. Thus, the update methods must be extremely flexible. While we believe Microsoft has invested considerable time in designing a sound update architecture, we did not personally examine the Windows Update system [36]. Microsoft explains that:

> In general, Windows Update uses the same Authenticode code-signing technology, including indirect signing via catalog files, that Microsoft recommends and supports for ISVs and IHVs. All signatures (whether embedded Authenticode or catalog files) are RSA-SHA1 signatures chaining to specific, known Microsoft roots (at least 1024-bit RSA for end-entity signing keys, at least 2048-bit RSA for intermediate and root CAs). Catalog files contain SHA1 hashes of components that don't support embedded signatures (like text files) and are in turn RSA-SHA1 signed. The client-side WU code checks all signatures for cryptographic and cert chain validity before installing any updates. [16]

Measurements of the Windows Update service [12] indicate that decentralized content distribution systems like peer-to-peer services can greatly reduce inter-ISP traffic. Thus, it is likely that distribution of future updates will travel over less trusted infrastructure.

Adobe Acrobat Reader. Acrobat Reader's update mechanism also appears secure (we examined the OS X version). The software gets the locations of specially-formatted update files from a Web server, then downloads the update files. Through extensive examination of the update file format, we determined that Adobe includes both a OS X-native disk image containing the update itself and a corresponding signature. Though we were unable to determine the precise algorithm and key length used to generate the signature, we located the corresponding public key in the Acrobat Reader application package.

Microsoft Office Update. Microsoft's update mechanism for their MacOS Office products downloads an XML-like list of available product updates and product versions whose presence should trigger their installation. Software certificates signed by Microsoft are included in the XML-like file for each update that can be downloaded. Binaries downloaded for update are checked against the signature in the corresponding certificate and not executed if the signature does not match. Microsoft's public key for verification of signatures is signed by Verisign, whose root public key is included on the MacOS installation media. However, users are often given the advice to download and execute unsigned disk images from unauthenticated Web sites [23].

Mozilla Firefox 1.0 & 1.5. Firefox 1.0 downloads its update packages from a mirror server over unencrypted HTTP, but the list of updates and their locations is downloaded over HTTPS. We experimented briefly with SSL MITM, and Firefox notified the user in no uncertain terms that someone was doing something nasty with the update process. It did eventually allow the user to accept our certificates and retrieve the list through the middleman, but only after clicking through two warnings about untrusted certificates. Firefox 1.5 was released during the course of our survey, and now prevents a user from ignoring warnings about suspicious certificates. However, the list of updates and locations itself is not authenticated. Thus, DNS spoofing of these metadata servers may open an avenue for MITM attacks.

Fugu. Fugu is a commonly-used free sftp/scp agent for MacOS X. Fugu has a "Check for Updates" function on its menu bar. The program downloads an XML document from a predefined location detailing the newest available application version along with the location where the program may download the latest version. Although a spoofed DNS response or XML interception-and-modification could spread malicious code, only a disk image is mounted to the user's desktop. Execution of the downloaded program must be initiated manually by copying the binary from the disk image into the "Applications" folder and then calling the executable as normally required when running Fugu. The user may execute malicious code, but it will be restricted to the user's privileges unless the malicious application requested and obtained a root password from the user. An naïve user might be fooled into producing the root password for malicious code, but this attack is not substantially easier than posting a malicious executable on a Web site for a user to download and execute manually.

Software	Platform	Authenticated Connection?	Authenticated Binaries?
Apple Software Update	MacOS	no	yes
Windows Update	Windows	partially	yes
Adobe Acrobat	MacOS	no	yes
Microsoft Office	MacOS	no	yes
Mozilla Firefox	Windows	partially	no
Fugu	MacOS	no	no
McAfee VirusScan	Windows	no	no
McAfee VirusScan Enterprise	Windows	unknown	yes
McAfee Virex	MacOS	no	no*
Debian	Linux	no	yes

Table 1: A security analysis of several software update systems. For each software package, we analyzed network traces and mounted MITM attacks to determine whether connections were authenticated and whether binary software updates were authenticated (i.e., appearing to have a digital signature). *In one study, we were able to install root-privileged code via DNS spoofing. Authenticated binaries without authenticated connections are sufficient for end-to-end secure content distribution.

McAfee VirusScan. When VirusScan is first installed, it installs and registers a number of ActiveX components. This must be done with the user's explicit consent, though once the controls are installed they may be re-instantiated freely. The main application's dialogs are built from HTML and VBScript, and rendered using Microsoft's HTML engine. To start the update process, the client makes a HTTP POST request with version numbers of the component DLLs, executables, and virus definition file. It then parses the reply generated by the server to determine whether any components are out-of-date. If so, the client requests a number of documents from the server over HTTP, including VBScript instructions that control the behavior of the ActiveX components that the user authorized when the software was installed.

Although we cannot say whether the ActiveX controls expose more functionality than necessary, they certainly expose enough to be dangerous. One of the functions exported by one of the controls is a simple wrapper around the Windows API ShellExecute function (analogous to the POSIX fork/exec*/waitpid or spawn* family of functions). By injecting commands into the VBScript returned by the server, the McAfee Updater can be forced to execute arbitrary commands on the Windows machine with the access level of the updating user. This problem is compounded by the fact that the user must have administrative privileges to update the software. We exploited this functionality to download and execute code of our choice. We injected the commands using a simple program that sits between the victim and a Squid Web proxy/cache which has the ability to modify the content of messages in either direction. We modify the inbound data to reflect that the client is out-of-date (whether or not this is the case), which causes the client to request the exploitable VBScript from the server. We then inject our commands into the desired place in the returned VBScript in order to trigger the desired behavior.

We briefly examined the "Enterprise" edition of McAfee's VirusScan software, intended to be deployed internally at institutions such as corporations and universities. When this product performs an update, no script is downloaded from the network to drive the update mechanism. Instead, it simply retrieves named files from an administrator-configurable location, determined at install time. Additionally, the updates themselves are encrypted and signed using public-key cryptography. We are unsure why this markedly more secure design was chosen for one product but not the other.

ActiveX considerations. When an ActiveX control is installed, it is typically bound to a domain from which it is allowed to be instantiated. We were unsuccessful in invoking the vulnerable ActiveX control on another Web page, which indicates that this part of the installation process was done properly. However, malicious DNS replies remain a problem. We successfully pointed "www.mcafee.com" at our attacker, then used the root page to instantiate the vulnerable controls to run the exploit. A more sophisticated attack could use Apache's native URL Rewriting to reduce suspicion by users. The code installed and executed by the control would already be working silently on the user's machine.

Despite the amount of bad press Microsoft's ActiveX technology has received with respect to security, the cen-

tral problem with the technology is that, even though it is being instantiated and scripted over an untrustworthy medium (the network), users must trust the control not to abuse the very low-level control it has over their machine. In addition to our exploitation of McAfee's update control, there have been other recent ActiveX-related exploits. One involved Sony's XCP DRM software [29]; the original uninstaller came packaged as an ActiveX control that remained on the user's machine after the uninstallation of the DRM software. Not only could this control be instantiated from pages on an arbitrary domain (unlike the McAfee control), it also exposed the machine to executing arbitrary code without verification that it came from a trusted source.

McAfee Virex 7 for MacOS. Virex update manager asks the user to authenticate as an administrator, then proceeds to log on anonymously to McAfee's FTP server and checks the time of the files in a particular directory. This time is given in the filename, and is independent of the timestamp on the server. If the client finds an update newer than the last one retrieved, the client downloads and unpacks the update. The update file contains an OS X installer package, which can contain arbitrary code, as well as pre- and post-installation instructions. Since Virex has already obtained the root access needed by the installer, the client silently runs the install package in the background after retrieving it.

Employing DNS spoofing, we successfully convinced the Virex updater to install and run our arbitrary code. We mirrored the directory structure of the real server on our spoofed server, and placed a package dated January 1, 2006 (V7060101.gz) at the location that the updater searches for updates. The updater downloaded and installed the files in the package without complaint. We used the post-install script in the package to execute our code, which ran with root privileges.

Debian Linux. Debian distributes signed packages, but the distributed developers are having difficulty signing software in a manner convenient for users. In 2005, many users were unable to install updates because of an elusive public key. Debian changes its signing key each year [6], but until recently there was no procedure to securely locate the latest public key. Signed software distributions are common in the open-source community. For instance, RedHat issues packages in the form of signed RPMs [30].

3 What's Next for Secure Updates?

The following section analyzes the present-day methods for secure updates. We also enumerate the new challenges for secure updates in the context of embedded devices (e.g., RFID tags and sensors).

3.1 Authenticity Now!

Researchers have largely failed to transfer technology from secure content distribution into software updates for standalone applications, as demonstrated by our survey of deployed software.

Incapable standalone applications. Our study indicates that operating systems tend to have better designed update methods — as compared to the methods of standalone software applications. Operating systems have the luxury of having more centralized control. For instance, Microsoft and Apple tightly control the distribution of signed software under well-known public keys. Debian and other open-source operating systems also tend to have moderately tight controls over signed packages, but the public keys are sometimes elusive.

Operating systems are so monolithic, that an integrated software update mechanism is a necessity for smooth operation. Updates appear regularly, and operating systems live or die depending on the quality of package maintenance. On the other hand, standalone applications have the weakest update methods. We suspect that small applications do not have the resources to support an advanced software update system. Whereas an operating system can leverage a single update system for many thousands of packages, a standalone application must bear the entire cost.

Unawareness. Though the principles of secure, authenticated software distribution are well-understood, we have shown that these principles are not necessarily followed, even in popular software. We believe one cause to be unawareness both by consumers and producers of software. To a consumer, a secure software update system and an insecure software update system are indistinguishable in non-hostile environments. On the producer side, MITM attacks could be mitigated by following best practices such as signed mobile code [31] and secure content distribution [9, 17, 19, 27]. We believe that the lack of deployment of secure updates is also a result of time-to-market priorities: first get the updates to work, then later secure the update channel. We hope

to bring more awareness to the unfinished security objectives.

Apathy. Perhaps users and producers of software simply do not care about authenticity. One rationalization is that because much software is downloaded insecurely from an untrusted source, it doesn't matter if the updates are secure. The code is already insecure. An approach to securing such untrustworthy software is to confine the code to a sandbox or virtual machine [34]. Thus, secure update services will be most effective for software initially obtained from a secure source (e.g., trusted installation media or code signed by a trusted party). Unfortunately, confinement does not work for all applications. For instance, anti-virus scanners require access to restricted resources.

For software initially installed from untrustworthy sources, secure updates will at best give assurance that the updated software came from the same untrusted source. The initial install is taken on faith, but all future actions are tied to reputation and the initial leap of faith.

3.2 Secure Updates for Embedded Devices

An amazing number of emerging technologies make use of software updates. Software runs in medicinal implants [22], digital video recorders, cars [10], mobile devices [21], delay-tolerant networks [4], RFID tags [14], and secure sensors [15, 26]. Several constraints exist for such embedded devices.

Untrusted infrastructure. Nomadic devices often give an administrator no chance to consent to a software update. For instance, an RFID tag has no user interface. An implanted medicinal pump cannot provide a dialog box to a patient. Moreover, the result of a medicinal pump is rarely idempotent. Thus, any software update mechanism should remain secure without a user's direct involvement.

No application should trust the network. But in embedded systems, there are even more opportunities for untrusted components to interfere with secure updates. For instance, RFID tags communicate entirely through untrusted readers. There is no opportunity for a user to give or deny consent to an update.

Sporadic network connectivity. RFID tags and human implants connect to networks only when in range. Moreover, most automatic updates work in the background when a computer or network is idle. With mobile devices, it will be more difficult to work in the background because it is also when the device is offline.

Network throughputs are likely to remain low on embedded devices because the cost of deploying a spare network of devices is much cheaper than deploying a dense network (e.g., disruption tolerant networking [4]). Moreover, push-based approaches alone will not work for nomadic devices that attach to networks only sporadically. Instead, devices will need to pull for updates.

Limited local resources. RFID tags lack the local resources for advanced cryptographic protocols. Embedded devices often have limited working memory, making the most common protection mechanisms challenging to implement. Thus, it is more likely that embedded devices will offload computations to more powerful, semi-trusted third parties such as RFID readers. Sensors have limited power and the most widely deployed RFID tags have no local power. Medicinal implants and heart pacemakers operate on limited batteries. Modern RFID tags measure storage in the thousands of bits. Merely storing client software for verifying secure updates is difficult.

3.3 Recommendations

Secure updates have many technical, economic, and social challenges. While we doubt that any approach can address all the challenges perfectly, we make several recommendations for ensuring reasonably secure updates.

Develop a standard for secure updates. Small-time software houses tend to implement their own homebrew update methods. The cost of developing a sound and secure update system is too great for any individual software project to bear. The community would be much healthier if software publishers were to momentarily set aside differences and establish an open set of standards for secure updates. Security is too important to be peddled as a proprietary system. Imagine if every company invented its own secure channels instead of using one of a few well-understood systems like SSL/TLS or IPSec. A standards body should take a stand for the collective interests of all parties. We all need secure updates.

Secure notification. From the client's perspective, software updates consist of two operations: notification of the existence of an update, and then the installation of the update itself. We suspect that many notification systems are not resistant to attackers that trick a client into thinking no updates are available. Imagine a MITM attack that simply responds to all update requests with,

"No updates are available at this time." We are not aware of this attack in the wild yet, but it's relatively simple to stop. For instance, the SFS read-only file system is fast enough to support short expiration times on large quantities of signed content [9]. SFSRO also includes an authenticated directory of all file handles in the file system. Thus, a client can verify that servers responding with "No updates available" are actually speaking the truth.

Follow the open design principle [32]. The open design principle roughly states that it is usually better to design a secure system with publicly scrutinized technology rather than with proprietary homebrew technology.

Assume an untrusted infrastructure The larger the Internet becomes, the more infrastructure joins the network. Yet trust is eroding. File systems can operate on untrusted servers [18], but a server is just one of many untrusted components. To ensure secure updates in the future, designs should not trust the infrastructure. Rather, the update mechanisms should verify end-to-end authenticity of updates.

4 Related Work

Worm containment. Software updates on servers are as of yet an ineffective first-line defense against worms [28, 35] because administrators are wary of installing updates — leaving the software vulnerable. We believe that automated updates will forcefully become more commonplace, with software publishers installing updates without user consent. If this prediction comes to pass, a secure update mechanism will be necessary to prevent the update channel itself from becoming a new vector of attack.

Replication on untrusted hosts. Several systems use Merkle hash trees [20] for efficient and secure signing of content. For instance, the SFS read-only file system (SFSRO) [9] uses signed Merkle hash trees so that a single publisher can make content available at high throughput to many readers who download content from untrusted servers. Unfortunately, the SFSRO client software is too large to fit on a embedded device such as an RFID tag.

Given a small set of trusted hosts, a content distribution system can detect and discipline badly behaving replicas [27]. In embedded devices, corrections will be more difficult to realize because of sporadic network connectivity.

Secure HTTP servers [8] provide authenticated connections, but do not provide end-to-end authentication of content mirrored on untrusted hosts as is common with open source software. Secure DNS [7] signs individual DNS records, but requires a separate PKI to support basic features such as revocation and public key discovery. Techniques from TLS and secure DNS may play a larger role in securing databases associated with RFID tags [25], where there are as of yet no legacy systems.

Revere distributes signed updates over an overlay network for scalability [17]. Overlay techniques and aggressive pre-positioning of content at the edge of the network (e.g., RFID readers) have potential for overcoming the sporadic network connectivity challenge.

Managing software updates. Many tools provide dependency checking in software updates, but ultimately trust the network for the secure distribution of content (Table 1). Managing the updates themselves pose many additional challenges for distributed systems [1] and dynamic software updating [24].

5 Conclusion

In deployed software update systems, insecurity results from incapable standalone applications, unawareness, and apathy. In this paper, we demonstrate how to compromise a computer via a software update (it happens to be a type of computer immunodeficiency weakness). We believe that software engineers will need to more seriously embrace secure content distribution as more attackers — bored of traditional attack vectors — begin to exploit such widely-deployed software update mechanisms.

Our position is that embedded devices will also need a mechanism for secure software updates. While existing approaches help in secure distribution of software for general purpose computers, the benefits do not transfer well to the constrained environment of embedded devices with sporadic network connectivity and limited resources.

Secure content distribution provides one layer in the infrastructure for software updates. Good mobile code hygiene can provide further security. But by designing secure update mechanisms for the challenging environment of untrusted infrastructure, we can better tackle problems such as secure evolution of software and updates of unattended critical infrastructure. Our full survey and video demonstration appear on [2].

Acknowledgments

We thank the anonymous reviewers, Andy Ellis, Edwin Foo, and Adam Stubblefield for their helpful feedback on early versions of this paper. We thank McAfee for fixing the MITM vulnerabilities we identified in an earlier term project [2].

References

[1] S. Ajmani, B. Liskov, and L. Shrira. Scheduling and simulation: How to upgrade distributed systems. In *HotOS IX*, May 2003.

[2] A. Bellissimo and J. Burgess. Authentication failure attacks in software update mechanisms, Dec. 2005. http://www-edlab.cs.umass.edu/cs691i/projects.html decrypt with password "Ihsac".

[3] G. Brandman. Patching the interprise. *ACM Queue*, Mar. 2005.

[4] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for vehicle-based disruption-tolerant networking. In *Proceedings IEEE Infocom 2006*, Apr. 2006.

[5] Class I medical device recall of the medtronic 8870 software application card version aaa 02, 2004. http://www.fda.gov/cdrh/recalls/recall-082404b-pressrelease.html Last Viewed July 3, 2006.

[6] Debian SecureApt. http://wiki.debian.org/SecureApt Last Viewed July 5, 2006.

[7] D. Eastlake and C. Kaufman. Domain name system security extensions. RFC 2065, Network Working Group, January 1997.

[8] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0. Internet draft (draft-freier-ssl-version3-02.txt), Network Working Group, November 1996. Work in progress.

[9] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM Transactions on Computer Systems*, 20(1):1–24, February 2002. A version appeared in OSDI 2000.

[10] S. Garfinkel. History's worst software bugs. *Wired News*, Nov. 2005.

[11] General principles of software validation; final guidance for industry and FDA staff, Jan. 2002. http://www.fda.gov/cdrh/comp/guidance/938.html Last Viewed July 5, 2006.

[12] C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic. Planet scale software updates. In *ACM/SIGCOMM'06*, Sept. 2006. To appear.

[13] Guidance for industry - cybersecurity for networked medical devices containing off-the-shelf (OTS) software, Jan. 2005. http://www.fda.gov/cdrh/comp/guidance/1553.html Last Viewed July 5, 2006.

[14] A. Juels. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Computing*, 24(2):381–394, Feb. 2006.

[15] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.

[16] B. LaMacchia. Personal communication, June 2006.

[17] J. Li. *Revere – disseminating security updates at Internet scale*. PhD thesis, UCLA, 2002.

[18] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 91–106, San Francisco, CA, December 2004.

[19] U. Maheshwari and R. Vingralek. How to build a trusted database system on untrusted storage. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, San Diego, October 2000.

[20] R. C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *Advances in Cryptology—CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Berlin, 1987. Springer-Verlag.

[21] D. Mettler. Secure software updates for mobile devices. Master's thesis, Department of Information Technology, University of Zurich, July 2002.

[22] Microchips, Inc. http://www.mchips.com/ Last Viewed June 30, 2006.

[23] Microsoft Office 2004 - 11.2.4 Mac OS X version tracker. http://www.versiontracker.com/dyn/moreinfo/macosx/14980 Last Viewed June 29, 2006.

[24] I. Neamtiu, M. Hicks, G. Stoyle, and M. Oriol. Practical dynamic software updating for C. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*, June 2006. http://www.cs.umd.edu/projects/dsu/.

[25] Auto-ID Object Name Service (ONS) 1.0, 12 Aug. 2003. Auto-ID Working Draft. M. Mealling, editor. Available to members at develop.autoidcenter.org/TR/ons-1.0.pdf.

[26] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept. 2002.

[27] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Secure data replication over untrusted hosts. In *HotOS IX*, May 2003.

[28] E. Rescorla. Security holes... who cares? In *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[29] W. Roush. Inside the spyware scandal. *MIT Technology Review*, May 2006. http://www.technologyreview.com/read_article.aspx?id=16812 Last Viewed July 5, 2006.

[30] *RPM software packaging tool*. www.rpm.org Last Viewed July 5, 2006.

[31] A. D. Rubin and D. E. G. Jr. Mobile code security. *IEEE Internet Computing*, 2(6):30–34, 1998.

[32] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. In *Fourth ACM Symposium on Operating System Principles*, Oct. 1973. Revised version in Communications of the ACM 17, 7 (July 1974).

[33] Technical cyber security alerts. http://www.us-cert.gov/cas/techalerts/ Last Viewed June 29, 2006.

[34] VMware GSX server. http://www.vmware.com/ Last Viewed July 5, 2006.

[35] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *SIGCOMM '04*, 2004.

[36] Windows update and automatic updates. http://tinyurl.com/r8kl5 Last Viewed June 26, 2006.