

Secure Programming for Linux and Unix HOWTO

David A. Wheeler

Secure Programming for Linux and Unix HOWTO

by David A. Wheeler

v3.50 Edition

Published v3.50, 22 August 2004

Copyright © 1999, 2000, 2001, 2002, 2003 by David A. Wheeler

This book provides a set of design and implementation guidelines for writing secure programs for Linux and Unix systems. Such programs include application programs used as viewers of remote data, web applications (including CGI scripts), network servers, and setuid/setgid programs. Specific guidelines for C, C++, Java, Perl, PHP, Python, Tcl, and Ada95 are included. For a current version of the book, see <http://www.dwheeler.com/secure-programs>

This book is Copyright (C) 1999-2003 David A. Wheeler. Permission is granted to copy, distribute and/or modify this book under the terms of the GNU Free Documentation License (GFDL), Version 1.1 or any later version published by the Free Software Foundation; with the invariant sections being "About the Author", with no Front-Cover Texts, and no Back-Cover texts. A copy of the license is included in the section entitled "GNU Free Documentation License". This book is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1. Introduction.....	1
2. Background	5
2.1. History of Unix, Linux, and Open Source / Free Software.....	5
2.1.1. Unix	5
2.1.2. Free Software Foundation	5
2.1.3. Linux.....	6
2.1.4. Open Source / Free Software.....	6
2.1.5. Comparing Linux and Unix.....	7
2.2. Security Principles	7
2.3. Why do Programmers Write Insecure Code?.....	8
2.4. Is Open Source Good for Security?	9
2.4.1. View of Various Experts	9
2.4.2. Why Closing the Source Doesn't Halt Attacks	11
2.4.3. Why Keeping Vulnerabilities Secret Doesn't Make Them Go Away	12
2.4.4. How OSS/FS Counters Trojan Horses	13
2.4.5. Other Advantages	14
2.4.6. Bottom Line.....	14
2.5. Types of Secure Programs.....	15
2.6. Paranoia is a Virtue	16
2.7. Why Did I Write This Document?.....	16
2.8. Sources of Design and Implementation Guidelines	17
2.9. Other Sources of Security Information	19
2.10. Document Conventions	19
3. Summary of Linux and Unix Security Features	21
3.1. Processes	22
3.1.1. Process Attributes	22
3.1.2. POSIX Capabilities	23
3.1.3. Process Creation and Manipulation.....	24
3.2. Files	24
3.2.1. Filesystem Object Attributes	24
3.2.2. POSIX Access Control Lists (ACLs)	26
3.2.2.1. History of POSIX Access Control Lists (ACLs)	27
3.2.2.2. Data used in POSIX Access Control Lists (ACLs).....	27
3.2.3. Creation Time Initial Values.....	28
3.2.4. Changing Access Control Attributes	29
3.2.5. Using Access Control Attributes	29
3.2.6. Filesystem Hierarchy	29
3.3. System V IPC.....	29
3.4. Sockets and Network Connections.....	30
3.5. Signals	31
3.6. Quotas and Limits	32
3.7. Dynamically Linked Libraries	33
3.8. Audit.....	34
3.9. PAM	34
3.10. Specialized Security Extensions for Unix-like Systems	34

4. Security Requirements	37
4.1. Common Criteria Introduction	37
4.2. Security Environment and Objectives	39
4.3. Security Functionality Requirements	40
4.4. Security Assurance Measure Requirements	41
5. Validate All Input.....	45
5.1. Command line	47
5.2. Environment Variables	47
5.2.1. Some Environment Variables are Dangerous	47
5.2.2. Environment Variable Storage Format is Dangerous	48
5.2.3. The Solution - Extract and Erase	48
5.2.4. Don't Let Users Set Their Own Environment Variables	49
5.3. File Descriptors	51
5.4. File Names	51
5.5. File Contents	52
5.6. Web-Based Application Inputs (Especially CGI Scripts)	52
5.7. Other Inputs.....	53
5.8. Human Language (Locale) Selection.....	54
5.8.1. How Locales are Selected.....	54
5.8.2. Locale Support Mechanisms	54
5.8.3. Legal Values	55
5.8.4. Bottom Line.....	55
5.9. Character Encoding.....	56
5.9.1. Introduction to Character Encoding	56
5.9.2. Introduction to UTF-8	57
5.9.3. UTF-8 Security Issues	57
5.9.4. UTF-8 Legal Values.....	58
5.9.5. UTF-8 Related Issues	59
5.10. Prevent Cross-site Malicious Content on Input.....	60
5.11. Filter HTML/URIs That May Be Re-presented	60
5.11.1. Remove or Forbid Some HTML Data	60
5.11.2. Encoding HTML Data	61
5.11.3. Validating HTML Data.....	61
5.11.4. Validating Hypertext Links (URIs/URLs).....	63
5.11.5. Other HTML tags	67
5.11.6. Related Issues	67
5.12. Forbid HTTP GET To Perform Non-Queries	68
5.13. Counter SPAM	69
5.14. Limit Valid Input Time and Load Level.....	70
6. Avoid Buffer Overflow.....	71
6.1. Dangers in C/C++	71
6.2. Library Solutions in C/C++.....	73
6.2.1. Standard C Library Solution.....	73
6.2.2. Static and Dynamically Allocated Buffers	75
6.2.3. strcpy and strcat.....	76
6.2.4. libmib.....	77
6.2.5. Safestr library (Messier and Viega)	77

6.2.6. C++ std::string class	77
6.2.7. Libsafe	77
6.2.8. Other Libraries.....	78
6.3. Compilation Solutions in C/C++.....	79
6.4. Other Languages	80
7. Design Your Program for Security	81
7.1. Follow Good Security Design Principles	81
7.2. Secure the Interface.....	82
7.3. Separate Data and Control	82
7.4. Minimize Privileges	82
7.4.1. Minimize the Privileges Granted	83
7.4.2. Minimize the Time the Privilege Can Be Used	85
7.4.3. Minimize the Time the Privilege is Active	85
7.4.4. Minimize the Modules Granted the Privilege.....	86
7.4.5. Consider Using FSUID To Limit Privileges.....	87
7.4.6. Consider Using Chroot to Minimize Available Files	87
7.4.7. Consider Minimizing the Accessible Data	89
7.4.8. Consider Minimizing the Resources Available	89
7.5. Minimize the Functionality of a Component	89
7.6. Avoid Creating Setuid/Setgid Scripts.....	89
7.7. Configure Safely and Use Safe Defaults	89
7.8. Load Initialization Values Safely	90
7.9. Minimize the Accessible Data	91
7.10. Fail Safe	91
7.11. Avoid Race Conditions.....	91
7.11.1. Sequencing (Non-Atomic) Problems	92
7.11.1.1. Atomic Actions in the Filesystem.....	92
7.11.1.2. Temporary Files	93
7.11.2. Locking.....	99
7.11.2.1. Using Files as Locks	99
7.11.2.2. Other Approaches to Locking.....	101
7.12. Trust Only Trustworthy Channels	101
7.13. Set up a Trusted Path.....	102
7.14. Use Internal Consistency-Checking Code	103
7.15. Self-limit Resources	104
7.16. Prevent Cross-Site (XSS) Malicious Content	104
7.16.1. Explanation of the Problem	104
7.16.2. Solutions to Cross-Site Malicious Content.....	105
7.16.2.1. Identifying Special Characters	106
7.16.2.2. Filtering.....	107
7.16.2.3. Encoding (Quoting)	107
7.17. Foil Semantic Attacks	109
7.18. Be Careful with Data Types	109
7.19. Avoid Algorithmic Complexity Attacks	110

8. Carefully Call Out to Other Resources.....	111
8.1. Call Only Safe Library Routines.....	111
8.2. Limit Call-outs to Valid Values.....	111
8.3. Handle Metacharacters.....	111
8.4. Call Only Interfaces Intended for Programmers	114
8.5. Check All System Call Returns	114
8.6. Avoid Using vfork(2)	114
8.7. Counter Web Bugs When Retrieving Embedded Content	115
8.8. Hide Sensitive Information	116
9. Send Information Back Judiciously	117
9.1. Minimize Feedback.....	117
9.2. Don't Include Comments	117
9.3. Handle Full/Unresponsive Output.....	117
9.4. Control Data Formatting (Format Strings/Formatation)	117
9.5. Control Character Encoding in Output	119
9.6. Prevent Include/Configuration File Access.....	120
10. Language-Specific Issues.....	123
10.1. C/C++.....	123
10.2. Perl	126
10.3. Python	127
10.4. Shell Scripting Languages (sh and csh Derivatives).....	128
10.5. Ada	129
10.6. Java.....	130
10.7. Tcl	133
10.8. PHP	137
11. Special Topics	143
11.1. Passwords.....	143
11.2. Authenticating on the Web.....	143
11.2.1. Authenticating on the Web: Logging In	145
11.2.2. Authenticating on the Web: Subsequent Actions	146
11.2.3. Authenticating on the Web: Logging Out.....	148
11.3. Random Numbers	148
11.4. Specially Protect Secrets (Passwords and Keys) in User Memory	149
11.5. Cryptographic Algorithms and Protocols	150
11.5.1. Cryptographic Protocols.....	151
11.5.2. Symmetric Key Encryption Algorithms	153
11.5.3. Public Key Algorithms	154
11.5.4. Cryptographic Hash Algorithms.....	155
11.5.5. Integrity Checking	155
11.5.6. Randomized Message Authentication Mode (RMAC)	155
11.5.7. Other Cryptographic Issues	156
11.6. Using PAM.....	156
11.7. Tools.....	157
11.8. Windows CE.....	159
11.9. Write Audit Records	159
11.10. Physical Emissions.....	160

11.11. Miscellaneous.....	160
12. Conclusion	163
13. Bibliography	165
A. History.....	173
B. Acknowledgements.....	175
C. About the Documentation License	177
D. GNU Free Documentation License.....	179
E. Endorsements	185
F. About the Author.....	187

11.5. Cryptographic Algorithms and Protocols

Often cryptographic algorithms and protocols are necessary to keep a system secure, particularly when communicating through an untrusted network such as the Internet. Where possible, use cryptographic techniques to authenticate information and keep the information private (but don't assume that simple encryption automatically authenticates as well). Generally you'll need to use a suite of available tools to secure your application.

For background information and code, you should probably look at the classic text "Applied Cryptography" [Schneier 1996]. The newsgroup "sci.crypt" has a series of FAQ's; you can find them at many locations, including <http://www.landfield.com/faqs/cryptography-faq>. Linux-specific resources include the Linux Encryption HOWTO at <http://marc.mutz.com/Encryption-HOWTO/>. A discussion on how protocols use the basic algorithms can be found in [Opplinger 1998]. A useful collection of papers on how to apply cryptography in protocols can be found in [Stallings 1996]. What follows here is just a few comments; these areas are rather specialized and covered more thoroughly elsewhere.

Cryptographic protocols and algorithms are difficult to get right, so do not create your own. Instead, where you can, use protocols and algorithms that are widely-used, heavily analyzed, and accepted as secure. When you must create anything, give the approach wide public review and make sure that professional security analysts examine it for problems. In particular, do not create your own encryption algorithms unless you are an expert in cryptology, know what you're doing, and plan to spend years in professional review of the algorithm. Creating encryption algorithms (that are any good) is a task for experts only.

A number of algorithms are patented; even if the owners permit "free use" at the moment, without a signed contract they can always change their minds later, putting you at extreme risk later. In general, avoid all patented algorithms - in most cases there's an unpatented approach that is at least as good or better technically, and by doing so you avoid a large number of legal problems.

Another complication is that many countries regulate or restrict cryptography in some way. A survey of legal issues is available at the "Crypto Law Survey" site, <http://rechten.kub.nl/koops/cryptolaw/>.

Often, your software should provide a way to reject "too small" keys, and let the user set what "too small" is. For RSA keys, 512 bits is too small for use. There is increasing evidence that 1024 bits for RSA keys is not enough either; Bernstein has suggested techniques that simplify brute-forcing RSA, and other work based on it (such as Shamir and Tromer's "Factoring Large Numbers with the TWIRL device") now suggests that 1024 bit keys can be broken in a year by a \$10 Million device. You may want to make 2048 bits the minimum for RSA if you really want a secure system, and you should certainly do so if you plan to use those keys after 2015. For more about RSA specifically, see RSA's commentary on Bernstein's work. For a more general discussion of key length and other general cryptographic algorithm issues, see NIST's key management workshop in November 2001.

11.5.1. Cryptographic Protocols

When you need a security protocol, try to use standard-conforming protocols such as IPSec, SSL (soon to be TLS), SSH, S/MIME, OpenPGP/GnuPG/PGP, and Kerberos. Each has advantages and disadvantages; many of them overlap somewhat in functionality, but each tends to be used in different areas:

- Internet Protocol Security (IPSec). IPSec provides encryption and/or authentication at the IP packet level. However, IPSec is often used in a way that only guarantees authenticity of two communicating hosts, not of the users. As a practical matter, IPSec usually requires low-level support from the

operating system (which not all implement) and an additional keyring server that must be configured. Since IPsec can be used as a "tunnel" to secure packets belonging to multiple users and multiple hosts, it is especially useful for building a Virtual Private Network (VPN) and connecting a remote machine. As of this time, it is much less often used to secure communication from individual clients to servers. The new version of the Internet Protocol, IPv6, comes with IPsec "built in," but IPsec also works with the more common IPv4 protocol. Note that if you use IPsec, don't use the encryption mode without the authentication, because the authentication also acts as integrity protection.

- Secure Socket Layer (SSL) / TLS. SSL/TLS works over TCP and tunnels other protocols using TCP, adding encryption, authentication of the server, and optional authentication of the client (but authenticating clients using SSL/TLS requires that clients have configured X.509 client certificates, something rarely done). SSL version 3 is widely used; TLS is a later adjustment to SSL that strengthens its security and improves its flexibility. Currently there is a slow transition going on from SSLv3 to TLS, aided because implementations can easily try to use TLS and then back off to SSLv3 without user intervention. Unfortunately, a few bad SSLv3 implementations cause problems with the backoff, so you may need a preferences setting to allow users to skip using TLS if necessary. Don't use SSL version 2, it has some serious security weaknesses.

SSL/TLS is the primary method for protecting http (web) transactions. Any time you use an "https://" URL, you're using SSL/TLS. Other protocols that often use SSL/TLS include POP3 and IMAP. SSL/TLS usually use a separate TCP/IP port number from the unsecured port, which the IETF is a little unhappy about (because it consumes twice as many ports; there are solutions to this). SSL is relatively easy to use in programs, because most library implementations allow programmers to use operations similar to the operations on standard sockets like `SSL_connect()`, `SSL_write()`, `SSL_read()`, etc. A widely used OSS/FS implementation of SSL (as well as other capabilities) is OpenSSL, available at <http://www.openssl.org>.

- OpenPGP and S/MIME. There are two competing, essentially incompatible standards for securing email: OpenPGP and S/MIME. OpenPGP is based on the PGP application; an OSS/FS implementation is GNU Privacy Guard from <http://www.gnupg.org>. Currently, their certificates are often not interchangeable; work is ongoing to repair this.
- SSH. SSH is the primary method of securing "remote terminals" over an internet, and it also includes methods for tunnelling X Windows sessions. However, it's been extended to support single sign-on and general secure tunnelling for TCP streams, so it's often used for securing other data streams too (such as CVS accesses). The most popular implementation of SSH is OpenSSH <http://www.openssh.com>, which is OSS/FS. Typical uses of SSH allows the client to authenticate that the server is truly the server, and then the user enters a password to authenticate the user (the password is encrypted and sent to the other system for verification). Current versions of SSH can store private keys, allowing users to not enter the password each time. To prevent man-in-the-middle attacks, SSH records keying information about servers it talks to; that means that typical use of SSH is vulnerable to a man-in-the-middle attack during the very first connection, but it can detect problems afterwards. In contrast, SSL generally uses a certificate authority, which eliminates the first connection problem but requires special setup (and payment!) to the certificate authority.
- Kerberos. Kerberos is a protocol for single sign-on and authenticating users against a central authentication and key distribution server. Kerberos works by giving authenticated users "tickets", granting them access to various services on the network. When clients then contact servers, the servers can verify the tickets. Kerberos is a primary method for securing and supporting authentication on a

LAN, and for establishing shared secrets (thus, it needs to be used with other algorithms for the actual protection of communication). Note that to use Kerberos, both the client and server have to include code to use it, and since not everyone has a Kerberos setup, this has to be optional - complicating the use of Kerberos in some programs. However, Kerberos is widely used.

Many of these protocols allow you to select a number of different algorithms, so you'll still need to pick reasonable defaults for algorithms (e.g., for encryption).

11.5.2. Symmetric Key Encryption Algorithms

The use, export, and/or import of implementations of encryption algorithms are restricted in many countries, and the laws can change quite rapidly. Find out what the rules are before trying to build applications using cryptography.

For secret key (bulk data) encryption algorithms, use only encryption algorithms that have been openly published and withstood years of attack, and check on their patent status. I would recommend using the new Advanced Encryption Standard (AES), also known as Rijndahl -- a number of cryptographers have analyzed it and not found any serious weakness in it, and I believe it has been through enough analysis to be trustworthy now. In August 2002 researchers Fuller and Millar discovered a mathematical property of the cipher that, while not an attack, might be exploitable and turned into an attack (the approach may actually have serious consequences for some other algorithms, too). However, heavy-duty worldwide analysis has yet to provide serious evidence that AES is actually vulnerable (see [Landau 2004] for more technical information on Rijndael). It's always worth staying tuned for future work, of course. A good alternative to AES is the Serpent algorithm, which is slightly slower but is very resistant to attack. For many applications triple-DES is a very good encryption algorithm; it has a reasonably lengthy key (112 bits), no patent issues, and a very long history of withstanding attacks (it's withstood attacks far longer than any other encryption algorithm with reasonable key length in the public literature, so it's probably the safest publicly-available symmetric encryption algorithm when properly implemented). However, triple-DES is very slow when implemented in software, so triple-DES can be considered "safest but slowest." Twofish appears to be a good encryption algorithm, but there are some lingering questions - Sean Murphy and Fauzan Mirza showed that Twofish has properties that cause many academics to be concerned (though as of yet no one has managed to exploit these properties). MARS is highly resistant to "new and novel" attacks, but it's more complex and is impractical on small-ability smartcards. For the moment I would avoid Twofish - it's quite likely that this will never be exploitable, but it's hard to be sure and there are alternative algorithms which don't have these concerns. Don't use IDEA - it's subject to U.S. and European patents. Don't use stupid algorithms such as XOR with a constant or constant string, the ROT (rotation) scheme, a Vigenere ciphers, and so on - these can be trivially broken with today's computers. Don't use "double DES" (using DES twice) - that's subject to a "man in the middle" attack that triple-DES avoids. Your protocol should support multiple encryption algorithms, anyway; that way, when an encryption algorithm is broken, users can switch to another one.

For symmetric-key encryption (e.g., for bulk encryption), don't use a key length less than 90 bits if you want the information to stay secret through 2016 (add another bit for every additional 18 months of security) [Blaze 1996]. For encrypting worthless data, the old DES algorithm has some value, but with modern hardware it's too easy to break DES's 56-bit key using brute force. If you're using DES, don't just use the ASCII text key as the key - parity is in the least (not most) significant bit, so most DES algorithms will encrypt using a key value well-known to adversaries; instead, create a hash of the key and set the parity bits correctly (and pay attention to error reports from your encryption routine).

So-called “exportable” encryption algorithms only have effective key lengths of 40 bits, and are essentially worthless; in 1996 an attacker could spend \$10,000 to break such keys in twelve minutes or use idle computer time to break them in a few days, with the time-to-break halving every 18 months in either case.

Block encryption algorithms can be used in a number of different modes, such as “electronic code book” (ECB) and “cipher block chaining” (CBC). In nearly all cases, use CBC, and do *not* use ECB mode - in ECB mode, the same block of data always returns the same result inside a stream, and this is often enough to reveal what’s encrypted. Many modes, including CBC mode, require an “initialization vector” (IV). The IV doesn’t need to be secret, but it does need to be unpredictable by an attacker. Don’t reuse IV’s across sessions - use a new IV each time you start a session.

There are a number of different streaming encryption algorithms, but many of them have patent restrictions. I know of no patent or technical issues with WAKE. RC4 was a trade secret of RSA Data Security Inc; it’s been leaked since, and I know of no real legal impediment to its use, but RSA Data Security has often threatened court action against users of it (it’s not at all clear what RSA Data Security could do, but no doubt they could tie up users in worthless court cases). If you use RC4, use it as intended - in particular, always discard the first 256 bytes it generates, or you’ll be vulnerable to attack. SEAL is patented by IBM - so don’t use it. SOBER is patented; the patent owner has claimed that it will allow many uses for free if permission is requested, but this creates an impediment for later use. Even more interestingly, block encryption algorithms can be used in modes that turn them into stream ciphers, and users who want stream ciphers should consider this approach (you’ll be able to choose between far more publicly-available algorithms).

11.5.3. Public Key Algorithms

For public key cryptography (used, among other things, for signing and sending secret keys), there are only a few widely-deployed algorithms. One of the most widely-used algorithms is RSA; RSA’s algorithm was patented, but only in the U.S., and that patent expired in September 2000, so RSA can be freely used. Never decrypt or sign a raw value that an attacker gives you directly using RSA and expose the result, because that could expose the private key (this isn’t a problem in practice, because most protocols involve signing a hash computed by the user - not the raw value - or don’t expose the result). Never decrypt or sign the exact same raw value multiple times (the original can be exposed). Both of these can be solved by always adding random padding (PGP does this) - the usual approach is called Optimal Asymmetric Encryption Padding (OAEP).

The Diffie-Hellman key exchange algorithm is widely used to permit two parties to agree on a session key. By itself it doesn’t guarantee that the parties are who they say they are, or that there is no middleman, but it does strongly help defend against passive listeners; its patent expired in 1997. If you use Diffie-Hellman to create a shared secret, be sure to hash it first (there’s an attack if you use its shared value directly).

NIST developed the digital signature standard (DSS) (it’s a modification of the ElGamal cryptosystem) for digital signature generation and verification; one of the conditions for its development was for it to be patent-free.

RSA, Diffie-Hellman, and El Gamal’s techniques require more bits for the keys for equivalent security compared to typical symmetric keys; a 1024-bit key in these systems is supposed to be roughly equivalent to an 80-bit symmetric key. A 512-bit RSA key is considered completely unsafe; Nicko van Someren has demonstrated that such small RSA keys can be factored in 6 weeks using only already-available office

hardware (never mind equipment designed for the job). In the past, a 1024-bit RSA key was considered reasonably secure, but recent advancements in factorization algorithms (e.g., by D. J. Bernstein) have raised concerns that perhaps even 1024 bits is not enough for an RSA key. Certainly, if your application needs to be highly secure or last beyond 2015, you should use a 2048 bit keys.

If you need a public key that requires far fewer bits (e.g., for a smartcard), then you might use elliptic curve cryptography (IEEE P1363 has some suggested curves; finding curves is hard). However, be careful - elliptic curve cryptography isn't patented, but certain speedup techniques are patented. Elliptic curve cryptography is fast enough that it really doesn't need these speedups anyway for its usual use of encrypting session / bulk encryption keys. In general, you shouldn't try to do bulk encryption with elliptic keys; symmetric algorithms are much faster and are better-tested for the job.

11.5.4. Cryptographic Hash Algorithms

Some programs need a one-way cryptographic hash algorithm, that is, a function that takes an “arbitrary” amount of data and generates a fixed-length number that hard for an attacker to invert (e.g., it's difficult for an attacker to create a different set of data to generate that same value). For a number of years MD5 has been a favorite, but recent efforts have shown that its 128-bit length may not be enough [van Oorschot 1994] and that certain attacks weaken MD5's protection [Dobbertin 1996]. Indeed, there are rumors that a top industry cryptographer has broken MD5, but is bound by employee agreement to keep silent (see the Bugtraq 22 August 2000 posting by John Viega). Anyone can create a rumor, but enough weaknesses have been found that the idea of completing the break is plausible. If you're writing new code, use SHA-1 instead of MD5. Don't use the original SHA (now called “SHA-0”); SHA-0 had the same weakness that MD5 does. If you need more bits in your hash algorithm, use SHA-256, SHA-384, or SHA-512; you can get the specifications in NIST FIPS PUB 180-2.

11.5.5. Integrity Checking

When communicating, you need some sort of integrity check (don't depend just on encryption, since an attacker can then induce changes of information to “random” values). This can be done with hash algorithms, but don't just use a hash function directly (this exposes users to an “extension” attack - the attacker can use the hash value, add data of their choosing, and compute the new hash). The usual approach is “HMAC”, which computes the integrity check as

$$H(k \text{ xor opad}, H(k \text{ xor ipad}, \text{data})).$$

where H is the hash function (typically MD5 or SHA-1) and k is the key. Thus, integrity checks are often HMAC-MD5 or HMAC-SHA-1. Note that although MD5 has some weaknesses, as far as I know MD5 isn't vulnerable when used in this construct, so HMAC-MD5 is (to my knowledge) okay. This is defined in detail in IETF RFC 2104.

Note that in the HMAC approach, a receiver can forge the same data as a sender. This isn't usually a problem, but if this must be avoided, then use public key methods and have the sender “sign” the data with the sender private key - this avoids this forging attack, but it's more expensive and for most environments isn't necessary.

11.5.6. Randomized Message Authentication Mode (RMAC)

NIST has developed and proposed a new mode for using cryptographic algorithms called Randomized Message Authentication Code (RMAC). RMAC is intended for use as a message authentication code technique.

Although there's a formal proof showing that RMAC is secure, the proof depends on the highly questionable assumption that the underlying cryptographic algorithm meets the "ideal cipher model" - in particular, that the algorithm is secure against a variety of specialized attacks, including related-key attacks. Unfortunately, related-key attacks are poorly studied for many algorithms; this is not the kind of property or attack that most people worry about when analyzing with cryptographic algorithms. It's known triple-DES doesn't have this property, and it's unclear if other widely-accepted algorithms like AES have this property (it appears that AES is at least weaker against related key attacks than usual attacks).

The best advice right now is "don't use RMAC". There are other ways to do message authentication, such as HMAC combined with a cryptographic hash algorithm (e.g., HMAC-SHA1). HMAC isn't the same thing (e.g., technically it doesn't include a nonce, so you should rekey sooner), but the theoretical weaknesses of HMAC are merely theoretical, while the problems in RMAC seem far more important in the real world.

11.5.7. Other Cryptographic Issues

You should both encrypt and include integrity checks of data that's important. Don't depend on the encryption also providing integrity - an attacker may be able to change the bits into a different value, and although the attacker may not be able to change it to a specific value, merely changing the value may be enough. In general, you should use different keys for integrity and secrecy, to avoid certain subtle attacks.

One issue not discussed often enough is the problem of "traffic analysis." That is, even if messages are encrypted and the encryption is not broken, an adversary may learn a great deal just from the encrypted messages. For example, if the presidents of two companies start exchanging many encrypted email messages, it may suggest that the two companies are considering a merger. For another example, many SSH implementations have been found to have a weakness in exchanging passwords: observers could look at packets and determine the length (or length range) of the password, even if they couldn't determine the password itself. They could also determine other information about the password that significantly aided in breaking it.

Be sure to not make it possible to solve a problem in parts, and use different keys when the trust environment (who is trusted) changes. Don't use the same key for too long - after a while, change the session key or password so an adversary will have to start over.

Generally you should compress something you'll encrypt - this does add a fixed header, which isn't so good, but it eliminates many patterns in the rest of the message as well as making the result smaller, so it's usually viewed as a "win" if compression is likely to make the result smaller.

In a related note, if you must create your own communication protocol, examine the problems of what's gone on before. Classics such as Bellovin [1989]'s review of security problems in the TCP/IP protocol suite might help you, as well as Bruce Schneier [1998] and Mudge's breaking of Microsoft's PPTP implementation and their follow-on work. Again, be sure to give any new protocol widespread public review, and reuse what you can.

11.6. Using PAM

Pluggable Authentication Modules (PAM) is a flexible mechanism for authenticating users. Many Unix-like systems support PAM, including Solaris, nearly all Linux distributions (e.g., Red Hat Linux, Caldera, and Debian as of version 2.2), and FreeBSD as of version 3.1. By using PAM, your program can be independent of the authentication scheme (passwords, SmartCards, etc.). Basically, your program calls PAM, which at run-time determines which “authentication modules” are required by checking the configuration set by the local system administrator. If you’re writing a program that requires authentication (e.g., entering a password), you should include support for PAM. You can find out more about the Linux-PAM project at <http://www.kernel.org/pub/linux/libs/pam/index.html>.

11.7. Tools

Some tools may help you detect security problems before you field the result. They can’t find all such problems, of course, but they can help catch problems that would otherwise slip by. Here are a few tools, emphasizing open source / free software tools.

One obvious type of tool is a program to examine the source code to search for patterns of known potential security problems (e.g., calls to library functions in ways are often the source of security vulnerabilities). These kinds of programs are called “source code scanners”. Here are a few such tools:

- Flawfinder, which I’ve developed; it’s available at <http://www.dwheeler.com/flawfinder>. This is also a program that scans C/C++ source code for common problems, and is also licensed under the GPL. Unlike RATS, flawfinder is implemented in Python. The developers of RATS and Flawfinder have agreed to find a way to work together to create a single “best of breed” open source program.
- RATS (Rough Auditing Tool for Security) from Secure Software Solutions is available at <http://www.securesw.com/rats>. This program scans C/C++ source code for common problems, and is licensed under the GPL.
- ITS4 from Cigital (formerly Reliable Software Technologies, RST) also statically checks C/C++ code. It is available free for non-commercial use, including its source code and with certain modification and redistribution rights. Note that this isn’t released as “open source” as defined by the Open Source Definition (OSD) - In particular, OSD point 6 forbids “non-commercial use only” clauses in open source licenses. ITS4 is available at <http://www.rstcorp.com/its4>.
- Splint (formerly named LCLint) is a tool for statically checking C programs. With minimal effort, splint can be used as a better lint. If additional effort is invested adding annotations to programs, splint can perform stronger checking than can be done by any standard lint. For example, it can be used to statically detect likely buffer overflows. The software is licensed under the GPL and is available at <http://www.splint.org>.
- cqual is a type-based analysis tool for finding bugs in C programs. cqual extends the type system of C with extra user-defined type qualifiers, e.g., it can note that values are “tainted” or “untainted” (similar to Perl’s taint checking). The programmer annotates their program in a few places, and cqual performs qualifier inference to check whether the annotations are correct. cqual presents the analysis results using Program Analysis Mode, an emacs-based interface. The current version of cqual can detect potential format-string vulnerabilities in C programs. A previous incarnation of cqual, Carillon, has been used to find Y2K bugs in C programs. The software is licensed under the GPL and is available from <http://www.cs.berkeley.edu/Research/Aiken/cqual>.

- Smatch is a general-purpose static analysis tools for C/C++ programs. It generates output describing statically-determined states in a program, and has an API to let you define queries for potentially bad situations. It was originally designed to analyze the Linux kernel. More information is at <http://smatch.sourceforge.net>.
- Cyclone is a C-like language intended to remove C's security weaknesses. In theory, you can always switch to a language that is "more secure," but this doesn't always help (a language can help you avoid common mistakes but it can't read your mind). John Viega has reviewed Cyclone, and in December 2001 he said: "Cyclone is definitely a neat language. It's a C dialect that doesn't feel like it's taking away any power, yet adds strong safety guarantees, along with numerous features that can be a real boon to programmers. Unfortunately, Cyclone isn't yet ready for prime time. Even with crippling limitations aside, it doesn't yet offer enough advantages over Java (or even C with a good set of tools) to make it worth the risk of using what is still a very young technology. Perhaps in a few years, Cyclone will mature into a robust, widely supported language that comes dangerously close to C in terms of efficiency. If that day comes, you'll certainly see me abandoning C for good." The Cyclone compiler has been released under the GPL and LGPL. You can get more information from the Cyclone web site.

Some tools try to detect potential security flaws at run-time, either to counter them or at least to warn the developer about them. Much of Crispin Cowan's work, such as StackGuard, fits here.

There are several tools that try to detect various C/C++ memory-management problems; these are really general-purpose software quality improvement tools, and not specific to security, but memory management problems can definitely cause security problems. An especially capable tool is Valgrind, which detects various memory-management problems (such as use of uninitialized memory, reading/writing memory after it's been free'd, reading/writing off the end of malloc'ed blocks, and memory leaks). Another such tool is Electric Fence (efence) by Bruce Perens, which can detect certain memory management errors. Memwatch (public domain) and YAMD (GPL) can detect memory allocation problems for C and C++. You can even use the built-in capabilities of the GNU C library's malloc library, which has the `MALLOC_CHECK_` environment variable (see its manual page for more information). There are many others.

Another approach is to create test patterns and run the program, in attempt to find weaknesses in the program. Here are a few such tools:

- BFBTester, the Brute Force Binary Tester, is licensed under the GPL. This program does quick security checks of binary programs. BFBTester performs checks of single and multiple argument command line overflows and environment variable overflows. Version 2.0 and higher can also watch for tempfile creation activity (to check for using unsafe tempfile names). At one time BFBTester didn't run on Linux (due to a technical issue in Linux's POSIX threads implementation), but this has been fixed as of version 2.0.1. More information is available at <http://fbftester.sourceforge.net/>
- The fuzz program is a tool for testing other software. It tests programs by bombarding the program being evaluated with random data. This tool isn't really specific to security.
- SPIKE is a "fuzzer creation kit", i.e., it's a toolkit designed to create "random" tests to find security problems. The SPIKE toolkit is particularly designed for protocol analysis by simulating network protocol clients, and SPIKE proXy is a tool built on SPIKE to test web applications. SPIKE includes a few pre-canned tests. SPIKE is licensed under the GPL.

There are a number of tools that try to give you insight into running programs that can also be useful when trying to find security problems in your code. This includes symbolic debuggers (such as gdb) and trace programs (such as strace and ltrace). One interesting program to support analysis of running code is Fenris (GPL license). Its documentation describes Fenris as a “multipurpose tracer, stateful analyzer and partial decompiler intended to simplify bug tracking, security audits, code, algorithm or protocol analysis - providing a structural program trace, general information about internal constructions, execution path, memory operations, I/O, conditional expressions and much more.” Fenris actually supplies a whole suite of tools, including extensive forensics capabilities and a nice debugging GUI for Linux. A list of other promising open source tools that can be suitable for debugging or code analysis is available at <http://lcamtuf.coredump.cx/fenris/debug-tools.html>. Another interesting program along these lines is Subterfuge, which allows you to control what happens in every system call made by a program.

If you’re building a common kind of product where many standard potential flaws exist (like an ftp server or firewall), you might find standard security scanning tools useful. One good one is Nessus; there are many others. These kinds of tools are very useful for doing regression testing, but since they essentially use a list of past specific vulnerabilities and common configuration errors, they may not be very helpful in finding problems in new programs.

Often, you’ll need to call on other tools to implement your secure infrastructure. The Open-Source PKI Book describes a number of open source programs for implementing a public key infrastructure (PKI).

Of course, running a “secure” program on an insecure platform configuration makes little sense. You may want to examine hardening systems, which attempt to configure or modify systems to be more resistant to attacks. For Linux, one hardening system is Bastille Linux, available at <http://www.bastille-linux.org>.

Another list of security tools is available at <http://www.insecure.org/tools.html>.

11.8. Windows CE

If you’re securing a Windows CE Device, you should read Maricia Alforque’s “Creating a Secure Windows CE Device” at <http://msdn.microsoft.com/library/techart/winsecurity.htm>.

11.9. Write Audit Records

Write audit logs for program startup, session startup, and for suspicious activity. Possible information of value includes date, time, uid, euid, gid, egid, terminal information, process id, and command line values. You may find the function syslog(3) helpful for implementing audit logs. One awkward problem is that any logging system should be able to record a lot of information (since this information could be very helpful), yet if the information isn’t handled carefully the information itself could be used to create an attack. After all, the attacker controls some of the input being sent to the program. When recording data sent by a possible attacker, identify a list of “expected” characters and escape any “unexpected” characters so that the log isn’t corrupted. Not doing this can be a real problem; users may include characters such as control characters (especially NIL or end-of-line) that can cause real problems. For example, if an attacker embeds a newline, they can then forge log entries by following the newline with the desired log entry. Sadly, there doesn’t seem to be a standard convention for escaping these characters. I’m partial to the URL escaping mechanism (%hh where hh is the hexadecimal value of the escaped byte) but there are others including the C convention (\ooo for the octal value and \X where X is a special

symbol, e.g., `\n` for newline). There's also the caret-system (^I is control-I), though that doesn't handle byte values over 127 gracefully.

There is the danger that a user could create a denial-of-service attack (or at least stop auditing) by performing a very large number of events that cut an audit record until the system runs out of resources to store the records. One approach to counter to this threat is to rate-limit audit record recording; intentionally slow down the response rate if "too many" audit records are being cut. You could try to slow the response rate only to the suspected attacker, but in many situations a single attacker can masquerade as potentially many users.

Selecting what is "suspicious activity" is, of course, dependent on what the program does and its anticipated use. Any input that fails the filtering checks discussed earlier is certainly a candidate (e.g., containing NIL). Inputs that could not result from normal use should probably be logged, e.g., a CGI program where certain required fields are missing in suspicious ways. Any input with phrases like `/etc/passwd` or `/etc/shadow` or the like is very suspicious in many cases. Similarly, trying to access Windows "registry" files or `.pwl` files is very suspicious.

Do not record passwords in an audit record. Often people accidentally enter passwords for a different system, so recording a password may allow a system administrator to break into a different computer outside the administrator's domain.

11.10. Physical Emissions

Although it's really outside the scope of this book, it's important to remember that computing and communications equipment leaks a lot information that makes them hard to really secure. Many people are aware of TEMPEST requirements which deal with radio frequency emissions of computers, displays, keyboards, and other components which can be eavesdropped. The light from displays can also be eavesdropped, even if it's bounced off an office wall at great distance [Kuhn 2002]. Modem lights are also enough to determine the underlying communication.

11.11. Miscellaneous

The following are miscellaneous security guidelines that I couldn't seem to fit anywhere else:

Have your program check at least some of its assumptions before it uses them (e.g., at the beginning of the program). For example, if you depend on the "sticky" bit being set on a given directory, test it; such tests take little time and could prevent a serious problem. If you worry about the execution time of some tests on each call, at least perform the test at installation time, or even better at least perform the test on application start-up.

If you have a built-in scripting language, it may be possible for the language to set an environment variable which adversely affects the program invoking the script. Defend against this.

If you need a complex configuration language, make sure the language has a comment character and include a number of commented-out secure examples. Often `'#'` is used for commenting, meaning "the rest of this line is a comment".

If possible, don't create `setuid` or `setgid` root programs; make the user log in as root instead.

Sign your code. That way, others can check to see if what's available was what was sent.

In some applications you may need to worry about timing attacks, where the variation in timing or CPU utilization is enough to give away important information. This kind of attack has been used to obtain keying information from Smartcards, for example. Mauro Lacy has published a paper titled Remote Timing Techniques, showing that you can (in some cases) determine over an Internet whether or not a given user id exists, simply from the effort expended by the CPU (which can be detected remotely using techniques described in the paper). The only way to deal with these sorts of problems is to make sure that the same effort is performed even when it isn't necessary. The problem is that in some cases this may make the system more vulnerable to a denial of service attack, since it can't optimize away unnecessary work.

Consider statically linking secure programs. This counters attacks on the dynamic link library mechanism by making sure that the secure programs don't use it. There are several downsides to this however. This is likely to increase disk and memory use (from multiple copies of the same routines). Even worse, it makes updating of libraries (e.g., for security vulnerabilities) more difficult - in most systems they won't be automatically updated and have to be tracked and implemented separately.

When reading over code, consider all the cases where a match is not made. For example, if there is a switch statement, what happens when none of the cases match? If there is an "if" statement, what happens when the condition is false?

Merely "removing" a file doesn't eliminate the file's data from a disk; on most systems this simply marks the content as "deleted" and makes it eligible for later reuse, and often data is at least temporarily stored in other places (such as memory, swap files, and temporary files). Indeed, against a determined attacker, writing over the data isn't enough. A classic paper on the problems of erasing magnetic media is Peter Gutmann's paper "Secure Deletion of Data from Magnetic and Solid-State Memory". A determined adversary can use other means, too, such as monitoring electromagnetic emissions from computers (military systems have to obey TEMPEST rules to overcome this) and/or surreptitious attacks (such as monitors hidden in keyboards).

When fixing a security vulnerability, consider adding a "warning" to detect and log an attempt to exploit the (now fixed) vulnerability. This will reduce the likelihood of an attack, especially if there's no way for an attacker to predetermine if the attack will work, since it exposes an attack in progress. In short, it turns a vulnerability into an intrusion detection system. This also suggests that exposing the version of a server program before authentication is usually a bad idea for security, since doing so makes it easy for an attacker to only use attacks that would work. Some programs make it possible for users to intentionally "lie" about their version, so that attackers will use the "wrong attacks" and be detected. Also, if the vulnerability can be triggered over a network, please make sure that security scanners can detect the vulnerability. I suggest contacting Nessus (<http://www.nessus.org>) and make sure that their open source security scanner can detect the problem. That way, users who don't check their software for upgrades will at least learn about the problem during their security vulnerability scans (if they do them as they should).

Always include in your documentation contact information for where to report security problems. You should also support at least one of the common email addresses for reporting security problems (security-alert@SITE, secure@SITE, or security@SITE); it's often good to have support@SITE and info@SITE working as well. Be prepared to support industry practices by those who have a security flaw to report, such as the Full Disclosure Policy (RFPolicy) and the IETF Internet draft, "Responsible Vulnerability Disclosure Process". It's important to quickly work with anyone who is reporting a security flaw; remember that they are doing you a favor by reporting the problem to you, and that they are under no obligation to do so. It's especially important, once the problem is fixed, to give proper credit to the reporter of the flaw (unless they ask otherwise). Many reporters provide the information solely to gain the credit, and it's generally accepted that credit is owed to the reporter. Some vendors argue that people

should never report vulnerabilities to the public; the problem with this argument is that this was once common, and the result was vendors who denied vulnerabilities while their customers were getting constantly subverted for years at a time.

Follow best practices and common conventions when leading a software development project. If you are leading an open source software / free software project, some useful guidelines can be found in Free Software Project Management HOWTO and Software Release Practice HOWTO; you should also read The Cathedral and the Bazaar.

Every once in a while, review security guidelines like this one. At least re-read the conclusions in Chapter 12, and feel free to go back to the introduction (Chapter 1) and start again!

Chapter 12. Conclusion

The end of a matter is better than its beginning, and patience is better than pride.

Ecclesiastes 7:8 (NIV)

Designing and implementing a truly secure program is actually a difficult task on Unix-like systems such as Linux and Unix. The difficulty is that a truly secure program must respond appropriately to all possible inputs and environments controlled by a potentially hostile user. Developers of secure programs must deeply understand their platform, seek and use guidelines (such as these), and then use assurance processes (such as inspections and other peer review techniques) to reduce their programs' vulnerabilities.

In conclusion, here are some of the key guidelines in this book:

- Validate all your inputs, including command line inputs, environment variables, CGI inputs, and so on. Don't just reject "bad" input; define what is an "acceptable" input and reject anything that doesn't match.
- Avoid buffer overflow. Make sure that long inputs (and long intermediate data values) can't be used to take over your program. This is the primary programmatic error at this time.
- Structure program internals. Secure the interface, minimize privileges, make the initial configuration and defaults safe, and fail safe. Avoid race conditions (e.g., by safely opening any files in a shared directory like /tmp). Trust only trustworthy channels (e.g., most servers must not trust their clients for security checks or other sensitive data such as an item's price in a purchase).
- Carefully call out to other resources. Limit their values to valid values (in particular be concerned about metacharacters), and check all system call return values.
- Reply information judiciously. In particular, minimize feedback, and handle full or unresponsive output to an untrusted user.

Chapter 13. Bibliography

*The words of the wise are like goads,
their collected sayings like firmly
embedded nails--given by one Shepherd.
Be warned, my son, of anything in
addition to them. Of making many books
there is no end, and much study wearies
the body.*

Ecclesiastes 12:11-12 (NIV)

Note that there is a heavy emphasis on technical articles available on the web, since this is where most of this kind of technical information is available.

[Advosys 2000] Advosys Consulting (formerly named Webber Technical Services). *Writing Secure Web Applications*. <http://advosys.ca/tips/web-security.html>

[Al-Herbish 1999] Al-Herbish, Thamer. 1999. *Secure Unix Programming FAQ*. <http://www.whitefang.com/sup>.

[Aleph1 1996] Aleph1. November 8, 1996. "Smashing The Stack For Fun And Profit". *Phrack Magazine*. Issue 49, Article 14. <http://www.phrack.com/search.phtml?view&article=p49-14> or alternatively <http://www.2600.net/phrack/p49-14.html>.

[Anonymous 1999] Anonymous. October 1999. *Maximum Linux Security: A Hacker's Guide to Protecting Your Linux Server and Workstation* Sams. ISBN: 0672316706.

[Anonymous 1998] Anonymous. September 1998. *Maximum Security : A Hacker's Guide to Protecting Your Internet Site and Network*. Sams. Second Edition. ISBN: 0672313413.

[Anonymous Phrack 2001] Anonymous. August 11, 2001. Once upon a free(). *Phrack*, Volume 0x0b, Issue 0x39, Phile #0x09 of 0x12. <http://phrack.org/show.php?p=57&a=9>

[AUSCERT 1996] Australian Computer Emergency Response Team (AUSCERT) and O'Reilly. May 23, 1996 (rev 3C). *A Lab Engineers Check List for Writing Secure Unix Code*. ftp://ftp.auscert.org.au/pub/auscert/papers/secure_programming_checklist

[Bach 1986] Bach, Maurice J. 1986. *The Design of the Unix Operating System*. Englewood Cliffs, NJ: Prentice-Hall, Inc. ISBN 0-13-201799-7 025.

[Beattie 2002] Beattie, Steve, Seth Arnold, Crispin Cowan, Perry Wagle, Chris Wright, Adam Shostack. November 2002. *Timing the Application of Security Patches for Optimal Uptime*. 2002 LISA XVI, November 3-8, 2002, Philadelphia, PA.

[Bellovin 1989] Bellovin, Steven M. April 1989. "Security Problems in the TCP/IP Protocol Suite" *Computer Communications Review* 2:19, pp. 32-48. <http://www.research.att.com/~smb/papers/ipext.pdf>

[Bellovin 1994] Bellovin, Steven M. December 1994. *Shifting the Odds -- Writing (More) Secure Software*. Murray Hill, NJ: AT&T Research. <http://www.research.att.com/~smb/talks>

[Bishop 1996] Bishop, Matt. May 1996. "UNIX Security: Security in Programming". *SANS '96*. Washington DC (May 1996). <http://olympus.cs.ucdavis.edu/~bishop/secprog.html>

[Bishop 1997] Bishop, Matt. October 1997. "Writing Safe Privileged Programs". *Network Security 1997* New Orleans, LA. <http://olympus.cs.ucdavis.edu/~bishop/secprog.html>

- [Blaze 1996] Blaze, Matt, Whitfield Diffie, Ronald L. Rivest, Bruce Schneier, Tsutomu Shimomura, Eric Thompson, and Michael Wiener. January 1996. "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security: A Report by an Ad Hoc Group of Cryptographers and Computer Scientists." <ftp://ftp.research.att.com/dist/mab/keylength.txt> and <ftp://ftp.research.att.com/dist/mab/keylength.ps>.
- [CC 1999] *The Common Criteria for Information Technology Security Evaluation (CC)*. August 1999. Version 2.1. Technically identical to International Standard ISO/IEC 15408:1999. <http://csrc.nist.gov/cc/ccv20/ccv2list.htm>
- [CERT 1998] Computer Emergency Response Team (CERT) Coordination Center (CERT/CC). February 13, 1998. *Sanitizing User-Supplied Data in CGI Scripts*. CERT Advisory CA-97.25.CGI_metachar. http://www.cert.org/advisories/CA-97.25.CGI_metachar.html.
- [Cheswick 1994] Cheswick, William R. and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Full text at <http://www.wilyhacker.com>.
- [Clowes 2001] Clowes, Shaun. 2001. "A Study In Scarlet - Exploiting Common Vulnerabilities in PHP" <http://www.securereality.com.au/archives.html>
- [CMU 1998] Carnegie Mellon University (CMU). February 13, 1998 Version 1.4. "How To Remove Meta-characters From User-Supplied Data In CGI Scripts". ftp://ftp.cert.org/pub/tech_tips/cgi_metacharacters.
- [Cowan 1999] Cowan, Crispin, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade". Proceedings of DARPA Information Survivability Conference and Expo (DISCEX), <http://schafercorp-ballston.com/discex> SANS 2000. <http://www.sans.org/newlook/events/sans2000.htm>. For a copy, see <http://immunix.org/documentation.html>.
- [Cox 2000] Cox, Philip. March 30, 2001. *Hardening Windows 2000*. <http://www.systemexperts.com/win2k/hardenW2K11.pdf>.
- [Crosby 2003] Crosby, Scott A., and Dan S Wallach. "Denial of Service via Algorithmic Complexity Attacks" Usenix Security 2003. <http://www.cs.rice.edu/~scrosby/hash>.
- [Dobbertin 1996]. Dobbertin, H. 1996. The Status of MD5 After a Recent Attack. RSA Laboratories' CryptoBytes. Vol. 2, No. 2.
- [Felten 1997] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. *Web Spoofing: An Internet Con Game* Technical Report 540-96 (revised Feb. 1997) Department of Computer Science, Princeton University <http://www.cs.princeton.edu/sip/pub/spoofing.pdf>
- [Fenzi 1999] Fenzi, Kevin, and Dave Wrenski. April 25, 1999. *Linux Security HOWTO*. Version 1.0.2. <http://www.tldp.org/HOWTO/Security-HOWTO.html>
- [FHS 1997] Filesystem Hierarchy Standard (FHS 2.0). October 26, 1997. Filesystem Hierarchy Standard Group, edited by Daniel Quinlan. Version 2.0. <http://www.pathname.com/fhs>.
- [Filipski 1986] Filipski, Alan and James Hanko. April 1986. "Making Unix Secure." *Byte (Magazine)*. Peterborough, NH: McGraw-Hill Inc. Vol. 11, No. 4. ISSN 0360-5280. pp. 113-128.
- [Flake 2001] Flake, Havlar. *Auditing Binaries for Security Vulnerabilities*. <http://www.blackhat.com/html/win-usa-01/win-usa-01-speakers.html>.
- [FOLDOC] Free On-Line Dictionary of Computing. <http://foldoc.doc.ic.ac.uk/foldoc/index.html>.

- [Forristal 2001] Forristal, Jeff, and Greg Shipley. January 8, 2001. Vulnerability Assessment Scanners. Network Computing. <http://www.nwc.com/1201/1201f1b1.html>
- [FreeBSD 1999] FreeBSD, Inc. 1999. "Secure Programming Guidelines". *FreeBSD Security Information*. <http://www.freebsd.org/security/security.html>
- [Friedl 1997] Friedl, Jeffrey E. F. 1997. Mastering Regular Expressions. O'Reilly. ISBN 1-56592-257-3.
- [FSF 1998] Free Software Foundation. December 17, 1999. *Overview of the GNU Project*. <http://www.gnu.ai.mit.edu/gnu/gnu-history.html>
- [FSF 1999] Free Software Foundation. January 11, 1999. *The GNU C Library Reference Manual*. Edition 0.08 DRAFT, for Version 2.1 Beta of the GNU C Library. Available at, for example, http://www.netppl.fi/~pp/glibc21/libc_toc.html
- [Fu 2001] Fu, Kevin, Emil Sit, Kendra Smith, and Nick Feamster. August 2001. "Dos and Don'ts of Client Authentication on the Web". Proceedings of the 10th USENIX Security Symposium, Washington, D.C., August 2001. <http://cookies.lcs.mit.edu/pubs/webauth.html>.
- [Gabrilovich 2002] Gabrilovich, Evgeniy, and Alex Gontmakher. February 2002. "Inside Risks: The Homograph Attack". Communications of the ACM. Volume 45, Number 2. Page 128.
- [Galvin 1998a] Galvin, Peter. April 1998. "Designing Secure Software". *Sunworld*. <http://www.sunworld.com/swol-04-1998/swol-04-security.html>.
- [Galvin 1998b] Galvin, Peter. August 1998. "The Unix Secure Programming FAQ". *Sunworld*. <http://www.sunworld.com/sunworldonline/swol-08-1998/swol-08-security.html>
- [Garfinkel 1996] Garfinkel, Simson and Gene Spafford. April 1996. *Practical UNIX & Internet Security, 2nd Edition*. ISBN 1-56592-148-8. Sebastopol, CA: O'Reilly & Associates, Inc. <http://www.oreilly.com/catalog/puis>
- [Garfinkle 1997] Garfinkle, Simson. August 8, 1997. 21 Rules for Writing Secure CGI Programs. <http://webreview.com/wr/pub/97/08/08/bookshelf>
- [Gay 2000] Gay, Warren W. October 2000. Advanced Unix Programming. Indianapolis, Indiana: Sams Publishing. ISBN 0-67231-990-X.
- [Geodsoft 2001] Geodsoft. February 7, 2001. Hardening OpenBSD Internet Servers. <http://www.geodsoft.com/howto/harden>.
- [Graham 1999] Graham, Jeff. May 4, 1999. *Security-Audit's Frequently Asked Questions (FAQ)*. <http://lsap.org/faq.txt>
- [Gong 1999] Gong, Li. June 1999. *Inside Java 2 Platform Security*. Reading, MA: Addison Wesley Longman, Inc. ISBN 0-201-31000-7.
- [Gundavaram Unknown] Gundavaram, Shishir, and Tom Christiansen. Date Unknown. *Perl CGI Programming FAQ*. <http://language.perl.com/CPAN/doc/FAQs/cgi/perl-cgi-faq.html>
- [Hall 1999] Hall, Brian "Beej". Beej's Guide to Network Programming Using Internet Sockets. 13-Jan-1999. Version 1.5.5. <http://www.ecst.csuchico.edu/~beej/guide/net>
- [Howard 2002] Howard, Michael and David LeBlanc. 2002. Writing Secure Code. Redmond, Washington: Microsoft Press. ISBN 0-7356-1588-8.
- [ISO 12207] International Organization for Standardization (ISO). 1995. Information technology -- Software life cycle processes ISO/IEC 12207:1995.

[ISO 13335] International Organization for Standardization (ISO). ISO/IEC TR 13335. Guidelines for the Management of IT Security (GMITS). Note that this is a five-part technical report (not a standard); see also ISO/IEC 17799:2000. It includes:

- ISO 13335-1: Concepts and Models for IT Security
- ISO 13335-2: Managing and Planning IT Security
- ISO 13335-3: Techniques for the Management of IT Security
- ISO 13335-4: Selection of Safeguards
- ISO 13335-5: Safeguards for External Connections

[ISO 17799] International Organization for Standardization (ISO). December 2000. Code of Practice for Information Security Management. ISO/IEC 17799:2000.

[ISO 9000] International Organization for Standardization (ISO). 2000. Quality management systems - Fundamentals and vocabulary. ISO 9000:2000. See http://www.iso.ch/iso/en/iso9000-14000/iso9000/selection_use/iso9000family.html

[ISO 9001] International Organization for Standardization (ISO). 2000. Quality management systems - Requirements ISO 9001:2000

[Jones 2000] Jones, Jennifer. October 30, 2000. "Banking on Privacy". InfoWorld, Volume 22, Issue 44. San Mateo, CA: International Data Group (IDG). pp. 1-12.

[Kelsey 1998] Kelsey, J., B. Schneier, D. Wagner, and C. Hall. March 1998. "Cryptanalytic Attacks on Pseudorandom Number Generators." Fast Software Encryption, Fifth International Workshop Proceedings (March 1998), Springer-Verlag, 1998, pp. 168-188. http://www.counterpane.com/pseudorandom_number.html.

[Kernighan 1988] Kernighan, Brian W., and Dennis M. Ritchie. 1988. *The C Programming Language*. Second Edition. Englewood Cliffs, NJ: Prentice-Hall. ISBN 0-13-110362-8.

[Kim 1996] Kim, Eugene Eric. 1996. *CGI Developer's Guide*. SAMS.net Publishing. ISBN: 1-57521-087-8 <http://www.eekim.com/pubs/cgibook>

[Kiriansky 2002] Kiriansky, Vladimir, Derek Bruening, Saman Amarasinghe. "Secure Execution Via Program Shepherding". Proceedings of the 11th USENIX Security Symposium, San Francisco, California, August 2002. <http://cag.lcs.mit.edu/commit/papers/02/RIO-security-usenix.pdf>

Kolsek [2002] Kolsek, Mitja. December 2002. Session Fixation Vulnerability in Web-based Applications http://www.acros.si/papers/session_fixation.pdf.

[Kuchling 2000]. Kuchling, A.M. 2000. Restricted Execution HOWTO. <http://www.python.org/doc/howto/rexec/rexec.html>

[Kuhn 2002] Kuhn, Markus G. Optical Time-Domain Eavesdropping Risks of CRT displays. Proceedings of the 2002 IEEE Symposium on Security and Privacy, Oakland, CA, May 12-15, 2002. <http://www.cl.cam.ac.uk/~mgk25/ieee02-optical.pdf>

[Landau 2004] Landau, Susan. Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard. 2004. American Mathematical Monthly. <http://research.sun.com/people/slandau/maa1.pdf>

- [LSD 2001] The Last Stage of Delirium. July 4, 2001. *UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes*. <http://lsd-pl.net/papers.html#assembly>.
- [McClure 1999] McClure, Stuart, Joel Scambray, and George Kurtz. 1999. *Hacking Exposed: Network Security Secrets and Solutions*. Berkeley, CA: Osbourne/McGraw-Hill. ISBN 0-07-212127-0.
- [McKusick 1999] McKusick, Marshall Kirk. January 1999. "Twenty Years of Berkeley Unix: From AT&T-Owned to Freely Redistributable." *Open Sources: Voices from the Open Source Revolution*. <http://www.oreilly.com/catalog/opensources/book/kirkmck.html>.
- [McGraw 1999] McGraw, Gary, and Edward W. Felten. December 1998. Twelve Rules for developing more secure Java code. Javaworld. <http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules.html>.
- [McGraw 1999] McGraw, Gary, and Edward W. Felten. January 25, 1999. Securing Java: Getting Down to Business with Mobile Code, 2nd Edition John Wiley & Sons. ISBN 047131952X. <http://www.securingsjava.com>.
- [McGraw 2000a] McGraw, Gary and John Viega. March 1, 2000. Make Your Software Behave: Learning the Basics of Buffer Overflows. <http://www-4.ibm.com/software/developer/library/overflows/index.html>.
- [McGraw 2000b] McGraw, Gary and John Viega. April 18, 2000. Make Your Software Behave: Software strategies In the absence of hardware, you can devise a reasonably secure random number generator through software. <http://www-106.ibm.com/developerworks/library/randomsoft/index.html?dwzone=security>.
- [Miller 1995] Miller, Barton P., David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl. 1995. Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services. ftp://grilled.cs.wisc.edu/technical_papers/fuzz-revisited.pdf.
- [Miller 1999] Miller, Todd C. and Theo de Raadt. "strncpy and strlcat -- Consistent, Safe, String Copy and Concatenation" *Proceedings of Usenix '99*. <http://www.usenix.org/events/usenix99/millert.html> and http://www.usenix.org/events/usenix99/full_papers/millert/PACKING_LIST
- [Mookhey 2002] Mookhey, K. K. The Unix Auditor's Practical Handbook. <http://www.nii.co.in/tuaph.html>.
- [MISRA 1998] Guidelines for the use of the C language in Vehicle Based Software April 1998 The Motor Industry Software Reliability Association (MISRA) <http://www.misra.org.uk>
- [Mudge 1995] Mudge. October 20, 1995. *How to write Buffer Overflows*. 10pht advisories. <http://www.10pht.com/advisories/bufero.html>.
- [Murhammer 1998] Murhammer, Martin W., Orcun Atakan, Stefan Bretz, Larry R. Pugh, Kazunari Suzuki, and David H. Wood. October 1998. TCP/IP Tutorial and Technical Overview IBM International Technical Support Organization. <http://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf>
- [NCSA] NCSA Secure Programming Guidelines. <http://www.ncsa.uiuc.edu/General/Grid/ACES/security/programming>.
- [Neumann 2000] Neumann, Peter. 2000. "Robust Nonproprietary Software." Proceedings of the 2000 IEEE Symposium on Security and Privacy (the "Oakland Conference"), May 14-17, 2000, Berkeley, CA. Los Alamitos, CA: IEEE Computer Society. pp.122-123.
- [NSA 2000] National Security Agency (NSA). September 2000. Information Assurance Technical Framework (IATF). <http://www.iatf.net>.

- [Open Group 1997] The Open Group. 1997. *Single UNIX Specification, Version 2 (UNIX 98)*. <http://www.opengroup.org/online-pubs?DOC=007908799>.
- [OSI 1999] Open Source Initiative. 1999. *The Open Source Definition*. <http://www.opensource.org/osd.html>.
- [Opplinger 1998] Oppliger, Rolf. 1998. *Internet and Intranet Security*. Norwood, MA: Artech House. ISBN 0-89006-829-1.
- [Paulk 1993a] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. Capability Maturity Model for Software, Version 1.1. Software Engineering Institute, CMU/SEI-93-TR-24. DTIC Number ADA263403, February 1993. <http://www.sei.cmu.edu/activities/cmm/obtain.cmm.html>.
- [Paulk 1993b] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis, and Marilyn W. Bush. Key Practices of the Capability Maturity Model, Version 1.1. Software Engineering Institute. CMU/SEI-93-TR-25, DTIC Number ADA263432, February 1993.
- [Peteanu 2000] Peteanu, Razvan. July 18, 2000. Best Practices for Secure Web Development. <http://members.home.net/razvan.peteanu>
- [Pfleeger 1997] Pfleeger, Charles P. 1997. *Security in Computing*. Upper Saddle River, NJ: Prentice-Hall PTR. ISBN 0-13-337486-6.
- [Phillips 1995] Phillips, Paul. September 3, 1995. *Safe CGI Programming*. <http://www.go2net.com/people/paulp/cgi-security/safe-cgi.txt>
- [Quintero 1999] Quintero, Federico Mena, Miguel de Icaza, and Morten Welinder GNOME Programming Guidelines <http://developer.gnome.org/doc/guides/programming-guidelines/book1.html>
- [Raymond 1997] Raymond, Eric. 1997. *The Cathedral and the Bazaar*. <http://www.catb.org/~esr/writings/cathedral-bazaar>
- [Raymond 1998] Raymond, Eric. April 1998. *Homesteading the Noosphere*. <http://www.catb.org/~esr/writings/homesteading>
- [Ranum 1998] Ranum, Marcus J. 1998. *Security-critical coding for programmers - a C and UNIX-centric full-day tutorial*. <http://www.clark.net/pub/mjr/pubs/pdf/>.
- [RFC 822] August 13, 1982 *Standard for the Format of ARPA Internet Text Messages*. IETF RFC 822. <http://www.ietf.org/rfc/rfc0822.txt>.
- [rfp 1999] rain.forest.puppy. 1999. "Perl CGI problems". *Phrack Magazine*. Issue 55, Article 07. <http://www.phrack.com/search.phtml?view&article=p55-7> or <http://www.insecure.org/news/P55-07.txt>.
- [Rijmen 2000] Rijmen, Vincent. "LinuxSecurity.com Speaks With AES Winner". http://www.linuxsecurity.com/feature_stories/interview-aes-3.html.
- [Rochkind 1985]. Rochkind, Marc J. *Advanced Unix Programming*. Englewood Cliffs, NJ: Prentice-Hall, Inc. ISBN 0-13-011818-4.
- [Sahu 2002] Sahu, Bijaya Nanda, Srinivasan S. Muthuswamy, Satya Nanaji Rao Mallampalli, and Venkata R. Bonam. July 2002 "Is your Java code secure -- or exposed? Build safer applications now to avoid trouble later" <http://www-106.ibm.com/developerworks/java/library/j-staticsec.html?loc=dwmain>
- [St. Laurent 2000] St. Laurent, Simon. February 2000. *XTech 2000 Conference Reports*. "When XML Gets Ugly". <http://www.xml.com/pub/2000/02/xtech/megginson.html>.
- [Saltzer 1974] Saltzer, J. July 1974. "Protection and the Control of Information Sharing in MULTICS". *Communications of the ACM*. v17 n7. pp. 388-402.

- [Saltzer 1975] Saltzer, J., and M. Schroeder. September 1975. "The Protection of Information in Computing Systems". *Proceedings of the IEEE*. v63 n9. pp. 1278-1308.
<http://www.mediacity.com/~norm/CapTheory/ProtInf>. Summarized in [Pfleeger 1997, 286].
- [Schneider 2000] Schneider, Fred B. 2000. "Open Source in Security: Visting the Bizarre." Proceedings of the 2000 IEEE Symposium on Security and Privacy (the "Oakland Conference"), May 14-17, 2000, Berkeley, CA. Los Alamitos, CA: IEEE Computer Society. pp.126-127.
- [Schneier 1996] Schneier, Bruce. 1996. *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*. New York: John Wiley and Sons. ISBN 0-471-12845-7.
- [Schneier 1998] Schneier, Bruce and Mudge. November 1998. *Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)* Proceedings of the 5th ACM Conference on Communications and Computer Security, ACM Press. <http://www.counterpane.com/pptp.html>.
- [Schneier 1999] Schneier, Bruce. September 15, 1999. "Open Source and Security". *Crypto-Gram*. Counterpane Internet Security, Inc. <http://www.counterpane.com/crypto-gram-9909.html>
- [Seifried 1999] Seifried, Kurt. October 9, 1999. *Linux Administrator's Security Guide*.
<http://www.securityportal.com/lasg>.
- [Seifried 2001] Seifried, Kurt. September 2, 2001. WWW Authentication
<http://www.seifried.org/security/www-auth/index.html>.
- [Shankland 2000] Shankland, Stephen. "Linux poses increasing threat to Windows 2000". CNET.
<http://news.cnet.com/news/0-1003-200-1549312.html>
- [Shostack 1999] Shostack, Adam. June 1, 1999. *Security Code Review Guidelines*.
<http://www.homeport.org/~adam/review.html>.
- [Sibert 1996] Sibert, W. Olin. Malicious Data and Computer Security. (NIST) NISSC '96.
<http://www.fish.com/security/maldata.html>
- [Sitaker 1999] Sitaker, Kragen. Feb 26, 1999. *How to Find Security Holes*
<http://www.pobox.com/~kragen/security-holes.html> and
<http://www.dnaco.net/~kragen/security-holes.html>
- [SSE-CMM 1999] SSE-CMM Project. April 1999. *Systems Security Engineering Capability Maturity Model (SSE CMM) Model Description Document*. Version 2.0. <http://www.sse-cmm.org>
- [Stallings 1996] Stallings, William. Practical Cryptography for Data Internetworks. Los Alamitos, CA: IEEE Computer Society Press. ISBN 0-8186-7140-8.
- [Stein 1999]. Stein, Lincoln D. September 13, 1999. *The World Wide Web Security FAQ*. Version 2.0.1
<http://www.w3.org/Security/Faq/www-security-faq.html>
- [Swan 2001] Swan, Daniel. January 6, 2001. comp.os.linux.security FAQ. Version 1.0.
<http://www.linuxsecurity.com/docs/colsfaq.html>.
- [Swanson 1996] Swanson, Marianne, and Barbara Guttman. September 1996. Generally Accepted Principles and Practices for Securing Information Technology Systems. NIST Computer Security Special Publication (SP) 800-14. <http://csrc.nist.gov/publications/nistpubs/index.html>.
- [Thompson 1974] Thompson, K. and D.M. Richie. July 1974. "The UNIX Time-Sharing System". *Communications of the ACM* Vol. 17, No. 7. pp. 365-375.

- [Torvalds 1999] Torvalds, Linus. February 1999. "The Story of the Linux Kernel". *Open Sources: Voices from the Open Source Revolution*. Edited by Chris Dibona, Mark Stone, and Sam Ockman. O'Reilly and Associates. ISBN 1565925823. <http://www.oreilly.com/catalog/opensources/book/linus.html>
- [TruSecure 2001] TruSecure. August 2001. Open Source Security: A Look at the Security Benefits of Source Code Access. http://www.trusecure.com/html/tspub/whitepapers/open_source_security5.pdf
- [Unknown] *SETUID(7)* <http://www.homeport.org/~adam/setuid.7.html>.
- [Van Biesbrouck 1996] Van Biesbrouck, Michael. April 19, 1996. <http://www.csclub.uwaterloo.ca/u/mlvanbie/cgisec>.
- [van Oorschot 1994] van Oorschot, P. and M. Wiener. November 1994. "Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms." Proceedings of ACM Conference on Computer and Communications Security.
- [Venema 1996] Venema, Wietse. 1996. Murphy's law and computer security. <http://www.fish.com/security/murphy.html>
- [Viega 2002] Viega, John, and Gary McGraw. 2002. Building Secure Software. Addison-Wesley. ISBN 0201-72152-X.
- [Watters 1996] Watters, Arron, Guido van Rossum, James C. Ahlstrom. 1996. Internet Programming with Python. NY, NY: Henry Hold and Company, Inc.
- [Wheeler 1996] Wheeler, David A., Bill Brykczynski, and Reginald N. Meeson, Jr. Software Inspection: An Industry Best Practice. 1996. Los Alamitos, CA: IEEE Computer Society Press. IEEE Copmuter Society Press Order Number BP07340. Library of Congress Number 95-41054. ISBN 0-8186-7340-0.
- [Witten 2001] September/October 2001. Witten, Brian, Carl Landwehr, and Michael Caloyannides. "Does Open Source Improve System Security?" IEEE Software. pp. 57-61. <http://www.computer.org/software>
- [Wood 1985] Wood, Patrick H. and Stephen G. Kochan. 1985. *Unix System Security*. Indianapolis, Indiana: Hayden Books. ISBN 0-8104-6267-2.
- [Wreski 1998] Wreski, Dave. August 22, 1998. *Linux Security Administrator's Guide*. Version 0.98. <http://www.nic.com/~dave/SecurityAdminGuide/index.html>
- [Yoder 1998] Yoder, Joseph and Jeffrey Barcalow. 1998. Architectural Patterns for Enabling Application Security. PLoP '97 <http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/yoder.pdf>
- [Zalewski 2001] Zalewski, Michael. May 16-17, 2001. Delivering Signals for Fun and Profit: Understanding, exploiting and preventing signal-handling related vulnerabilities. Bindview Corporation. <http://razor.bindview.com/publish/papers/signals.txt>
- [Zoebelein 1999] Zoebelein, Hans U. April 1999. The Internet Operating System Counter. <http://www.leb.net/hzo/ioscount>.

Appendix A. History

Here are a few key events in the development of this book, starting from most recent events:

2002-10-29 David A. Wheeler

Version 3.000 released, adding a new section on determining security requirements and a discussion of the Common Criteria, broadening the document. Many smaller improvements were incorporated as well.

2001-01-01 David A. Wheeler

Version 2.70 released, adding a significant amount of additional material, such as a significant expansion of the discussion of cross-site malicious content, HTML/URI filtering, and handling temporary files.

2000-05-24 David A. Wheeler

Switched to GNU's GFDL license, added more content.

2000-04-21 David A. Wheeler

Version 2.00 released, dated 21 April 2000, which switched the document's internal format from the Linuxdoc DTD to the DocBook DTD. Thanks to Jorge Godoy for helping me perform the transition.

2000-04-04 David A. Wheeler

Version 1.60 released; changed so that it now covers *both* Linux and Unix. Since most of the guidelines covered both, and many/most app developers want their apps to run on both, it made sense to cover both.

2000-02-09 David A. Wheeler

Noted that the document is now part of the Linux Documentation Project (LDP).

1999-11-29 David A. Wheeler

Initial version (1.0) completed and released to the public.

Note that a more detailed description of changes is available on-line in the "ChangeLog" file.

Appendix B. Acknowledgements

*As iron sharpens iron, so one man
sharpens another.*

Proverbs 27:17 (NIV)

My thanks to the following people who kept me honest by sending me emails noting errors, suggesting areas to cover, asking questions, and so on. Where email addresses are included, they've been shrouded by prepending my "thanks." so bulk emailers won't easily get these addresses; inclusion of people in this list is *not* an authorization to send unsolicited bulk email to them.

- Neil Brown (thanks.neilb@cse.unsw.edu.au)
- Martin Douda (thanks.mad@students.zcu.cz)
- Jorge Godoy
- Scott Ingram (thanks.scott@silver.jhuapl.edu)
- Michael Kerrisk
- Doug Kilpatrick
- John Levon (levon@movementarian.org)
- Ryan McCabe (thanks.odin@numb.org)
- Paul Millar (thanks.paulm@astro.gla.ac.uk)
- Chuck Phillips (thanks.cdp@peakpeak.com)
- Martin Pool (thanks.mbp@humblebug.org.au)
- Eric S. Raymond (thanks.esr@snark.thyrsus.com)
- Marc Welz
- Eric Werme (thanks.werme@alpha.zk3.dec.com)

If you want to be on this list, please send me a constructive suggestion at dwheeler@dwheeler.com. If you send me a constructive suggestion, but do *not* want credit, please let me know that when you send your suggestion, comment, or criticism; normally I expect that people want credit, and I want to give them that credit. My current process is to add contributor names to this list in the document, with more detailed explanation of their comment in the ChangeLog for this document (available on-line). Note that although these people have sent in ideas, the actual text is my own, so don't blame them for any errors that may remain. Instead, please send me another constructive suggestion.

Appendix C. About the Documentation License

A copy of the text of the edict was to be issued as law in every province and made known to the people of every nationality so they would be ready for that day.

Esther 3:14 (NIV)

This document is Copyright (C) 1999-2000 David A. Wheeler. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License (FDL), Version 1.1 or any later version published by the Free Software Foundation; with the invariant sections being “About the Author”, with no Front-Cover Texts, and no Back-Cover texts. A copy of the license is included below in Appendix D.

These terms do permit mirroring by other web sites, but be *sure* to do the following:

- make sure your mirrors automatically get upgrades from the master site,
- clearly show the location of the master site (<http://www.dwheeler.com/secure-programs>), with a hypertext link to the master site, and
- give me (David A. Wheeler) credit as the author.

The first two points primarily protect me from repeatedly hearing about obsolete bugs. I do not want to hear about bugs I fixed a year ago, just because you are not properly mirroring the document. By linking to the master site, users can check and see if your mirror is up-to-date. I’m sensitive to the problems of sites which have very strong security requirements and therefore cannot risk normal connections to the Internet; if that describes your situation, at least try to meet the other points and try to occasionally sneakernet updates into your environment.

By this license, you may modify the document, but you can’t claim that what you didn’t write is yours (i.e., plagiarism) nor can you pretend that a modified version is identical to the original work. Modifying the work does not transfer copyright of the entire work to you; this is not a “public domain” work in terms of copyright law. See the license in Appendix D for details. If you have questions about what the license allows, please contact me. In most cases, it’s better if you send your changes to the master integrator (currently David A. Wheeler), so that your changes will be integrated with everyone else’s changes into the master copy.

I am not a lawyer, nevertheless, it’s my position as an author and software developer that any code fragments not explicitly marked otherwise are so small that their use fits under the “fair use” doctrine in copyright law. In other words, unless marked otherwise, you can use the code fragments without any restriction at all. Copyright law does not permit copyrighting absurdly small components of a work (e.g., “I own all rights to B-flat and B-flat minor chords”), and the fragments not marked otherwise are of the same kind of minuscule size when compared to real programs. I’ve done my best to give credit for specific pieces of code written by others. Some of you may still be concerned about the legal status of this code, and I want make sure that it’s clear that you can use this code in your software. Therefore, code fragments included directly in this document not otherwise marked have also been released by me under the terms of the “MIT license”, to ensure you that there’s no serious legal encumbrance:

Source code in this book not otherwise identified is
Copyright (c) 1999-2001 David A. Wheeler.

Permission is hereby granted, free of charge, to any person
obtaining a copy of the source code in this book not
otherwise identified (the "Software"), to deal in the
Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute,
sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE AND NONINFRINGEMENT.

IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Appendix D. GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000

Free Software Foundation, Inc.
59 Temple Place, Suite 330,
Boston,
MA
02111-1307
USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title Page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors,

and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version .

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique

number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Addendum

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix E. Endorsements

This version of the document is endorsed by the original author, David A. Wheeler, as a document that should improve the security of programs, when applied correctly. Note that no book, including this one, can guarantee that a developer who follows its guidelines will produce perfectly secure software. Modifications (including translations) must remove this appendix per the license agreement included above.

David A. Wheeler

David A. Wheeler is an expert in computer security and has long specialized in development techniques for large and high-risk software systems. He has been involved in software development since the mid-1970s, and been involved with Unix and computer security since the early 1980s. His areas of knowledge include computer security, software safety, vulnerability analysis, inspections, Internet technologies, software-related standards (including POSIX), real-time software development techniques, and numerous computer languages (including Ada, C, C++, Perl, Python, and Java).

Mr. Wheeler is co-author and lead editor of the IEEE book *Software Inspection: An Industry Best Practice*, author of the book *Ada95: The Lovelace Tutorial*, and co-author of the *GNOME User's Guide*. He is also the author of many smaller papers and articles, including the *Linux Program Library HOWTO*.

Mr. Wheeler hopes that, by making this document available, other developers will make their software more secure. You can reach him by email at dwheeler@dwheeler.com (no spam please), and you can also see his web site at <http://www.dwheeler.com>.

