Dla mojej niesamowitej rodziny.
Nic nie może się równać z wrażeniem,
jakie mamy wówczas, gdy wracamy do domu i wołamy:
„Jest tam kto, dzieciaki?”,
a w odpowiedzi dwa cienkie głosiki krzyczą
„TATUŚ!”

— Michael

Dla mojego ojca,
który nauczył mnie wartości,
jaką jest ciągłe uczenie się
nowych rzeczy
i podejmowanie nowych wyzwań.

— David

Dla mamy.
Ona rozbudziła
moją intelektualną ciekawość
i zawsze była pod ręką.

— John

Przedmowa

Teoria komputerów opiera się na koncepcji maszyn deterministycznych. Zgodnie z powszechnym oczekiwaniem komputery powinny zachowywać się tak, jak nakazują to dostarczane im instrukcje. W rzeczywistości korzystamy z oprogramowania, które jest przekazicielem naszych zamiarów. Współczesne komputery ogólnego zastosowania i ich programy są tak skomplikowane, że między kliknięciem myszą a jego spodziewanym efektem mogą występować liczne warstwy oprogramowania. Aby skorzystać z mocy komputerów, trzeba zaufać wszystkim warstwom rozpościerającym się między użytkownikiem a maszyną.

W dowolnych miejscach wspomnianych warstw mogą występować błędy, w wyniku których oprogramowanie potrafi robić co innego, niż chciał jego autor, a przynajmniej co innego, niż chciałby jego użytkownik. Błędy te wprowadzają pewną dawkę niedeterminizmu do systemów komputerowych, czego konsekwencją są często poważne problemy z bezpieczeństwem. Problemy te potrafią przyjąć postać włamania, które może zostać wykorzystane do ataku typu „blokada usługi" albo „przepełnienie bufora" (które pozwala uruchomić dowolny kod w miejsce kodu danej aplikacji).

Jeśli niedeterminizm oprogramowania jest wynikiem występujących w nim błędów, to nawet najlepsze pomysły dotyczące ochrony tego oprogramowania mogą być uznane jedynie za domysły. Można stawiać zapory ogniowe, rozwijać na poziomie systemu operacyjnego technologie służące do wykrywania i przeciwdziałania przepełnieniom bufora i ogólnie starać się łatać wszelkie „dziury", ale nie da się w ten sposób zmienić podstawowego modelu zabezpieczeń. Tylko poprawa jakości oprogramowania i zmniejszenie liczby występujących w nim błędów dają jakąś szansę powodzenia wysiłków na rzecz bezpieczeństwa.

Przy dzisiejszych metodach tworzenia oprogramowania wyeliminowanie całego zagrożenia jest nierealne. Pisanie programów obejmuje tyle elementów, które mogą zawierać usterki z punktu widzenia bezpieczeństwa, że samo poznanie tych aspektów wymaga mnóstwa czasu.

Jeśli ma nastąpić postęp w walce z błędami zabezpieczeń, trzeba ułatwić środowiskom programistów rozwiązywanie problemów bezpieczeństwa w tworzonym oprogramowaniu, przy jednoczesnym uwzględnieniu praktycznych ograniczeń. Po-

wstało wiele wspaniałych książek na temat bezpieczeństwa oprogramowania, w tym kilka napisanych przez autorów niniejszej publikacji, ale moim zdaniem należy skończyć z rozwodzeniem się nad złożonością problemu i dostarczyć zespołom programującym niewielki zbiór istotnych kwestii, o których należy pamiętać, aby ulepszyć oprogramowanie małym kosztem. Chodzi o to, aby rozwiązać większość typowych problemów bez specjalnego wysiłku, zamiast szukać nierealnej, idealnej metody poprawy bezpieczeństwa.

Kiedy pracowałem w Departamencie Bezpieczeństwa Wewnętrznego, poprosiłem Johna Viega o zebranie listy 19 „grzechów" programowania. Oryginalna lista była narzędziem uświadamiającym — jej celem było zwrócenie uwagi świata korporacji na najczęściej występujące błędy zabezpieczeń, a nie wskazywanie rozwiązań. Rozwiązania można za to znaleźć w tej książce. Zawiera ona listę najważniejszych problemów bezpieczeństwa, na które muszą uważać producenci oprogramowania, oraz informacje potrzebne do ich uniknięcia. Pokazuje także, jak wykryć te problemy — czy to przeglądając kod, czy też testując program. Prezentowane techniki i metody są rozsądne i rzeczowe, uzupełnione wykazami tego, co należy i czego nie należy robić. Autorzy wykonali ogromną pracę, pisząc przystępną, niezależną książkę, poruszającą większość typowych problemów bezpieczeństwa, nękających dzisiejsze oprogramowanie. Mam nadzieję, że społeczność programistów sięgnie po tę książkę i wykorzysta ją do usunięcia dużej części niedeterminizmu i luk występujących w używanym przez nas na co dzień oprogramowaniu.

Amit Yoran
były dyrektor
Oddziału Krajowego Bezpieczeństwa Komputerowego
Departamentu Bezpieczeństwa Wewnętrznego
Great Falls, Virginia
21 maja 2005 roku

Podziękowania

Książka ta powstała pośrednio jako efekt wizji Amita Yorana. Dziękujemy mu za starania, jakie podejmował w celu podniesienia świadomości bezpieczeństwa oprogramowania w trakcie swojej pracy w Departamencie Bezpieczeństwa Wewnętrznego (oraz później). Chcielibyśmy także podziękować wielu specjalistom od zabezpieczeń za staranne przejrzenie rozdziałów oraz mądre i często brutalnie szczere komentarze. Są to: David Raphael, Mark Curphy, Rudolph Arauj, Alan Krassowski, David Wheeler oraz Bill Hilf. Książka ta nie powstałaby również bez zaciętego uporu kilku osób z wydawnictwa McGraw-Hill. Wielkie dzięki dla J[3]: Jane Brownlow, Jennifer Housh i Jody McKenzie.

O autorach

Michael Howard jest kierownikiem odpowiedzialnym za zarządzanie zabezpieczeniami w firmie Microsoft oraz współautorem nagradzanej książki *Bezpieczny kod*. Jest też współautorem artykułów w kolumnie „Basic Training" pisma „IEEE Security & Privacy Magazine" oraz współautorem dokumentu „Processes to Produce Secure Software" utworzonego dla oddziału Homeland Security przez zespół organizacji National Cyber Security Partnership. Jako architekt projektu „Security Development Lifecycle" firmy Microsoft spędził wiele czasu na definiowaniu i wzmacnianiu dobrych praktyk bezpieczeństwa oraz na poprawianiu procesów tworzenia oprogramowania, aby oferować zwykłym użytkownikom bardziej bezpieczne aplikacje.

Dr David LeBlanc jest głównym architektem oprogramowania w firmie Webroot Software. Wcześniej pracował jako architekt zabezpieczeń w dziale Microsoft Office i był założycielem inicjatywy Trustworthy Computing. Współpracował także jako przyjazny haker z firmą Microsoft. Jest on również współautorem książek *Bezpieczny kod* oraz *Ocena bezpieczeństwa sieciowego* (obie książki wydane po polsku przez APN Promise). Napisał też wiele artykułów. W wolnych chwilach lubi jeździć konno ze swoją żoną Jennifer.

John Viega napisał o 19 śmiertelnych grzechach programowania, a jego praca zyskała duże uznanie. Na jego doświadczeniach oparto tę książkę. Jest założycielem

i dyrektorem Secure Software (*www.securesoftware.com*). Jest współautorem pierwszej książki na temat bezpieczeństwa oprogramowania *Building Secure Software* oraz książek *Network Security and Cryptography with OpenSSL* oraz *Secure Programming Cookbook*, a także twórcą procesu CLASP wprowadzania zabezpieczeń do cyklu tworzenia oprogramowania. Odpowiadał za utworzenie kilku narzędzi bezpieczeństwa dystrybuowanych w ramach licencji Open Source. Pracował też jako adiunkt na uczelni Virginia Tech oraz jako pracownik badawczy w Cyberspace Policy Institute. Jest znany w środowiskach związanych z bezpieczeństwem oprogramowania i z kryptografią; opracowuje nowe standardy bezpiecznych sieci i programów.

Redaktorzy techniczni

Alan Krassowski pracuje jako inżynier zabezpieczeń w Symantec Corporation. Jest szefem zespołu bezpieczeństwa produktów, którego zadanie to pomoc innym zespołom w dostarczaniu bezpiecznych technologii zmniejszających ryzyko i budujących zaufanie klientów. Przez 20 lat pracował przy tworzeniu wielu bezpiecznych programów. Przed przejściem do firmy Symantec był dyrektorem produkcji, inżynierem oprogramowania i konsultantem w wielu czołowych firmach, takich jak Microsoft, IBM, Tektronix, Step Technologies, Screenplay Systems, Quark i Continental Insurance. Ma licencjat z zakresu inżynierii komputerowej w Rochester Institute of Technology w Nowym Jorku.

David A. Wheeler ma wieloletnie doświadczenie w dziedzinie lepszego wytwarzania oprogramowania dla systemów o zwiększonym ryzyku, w tym dużych i bezpiecznych systemów. Jest współwydawcą i współautorem książki *Software inspection: An Industry Best Practise*, autorem książki *Ada 95: The Lovelace Tutorial and Secure Programming for Linux and Unix HOWTO* oraz autorem serii „Secure Programmer", wydawanej przez firmę IBM. Mieszka w Północnej Wirginii.

Wprowadzenie

Na początku 2004 roku Amit Yoran, ówczesny dyrektor Oddziału Krajowego Bezpieczeństwa Komputerowego w Departamencie Bezpieczeństwa Wewnętrznego USA, oznajmił, że 95 procent luk w zabezpieczeniach oprogramowania jest wynikiem 19 „typowych, dobrze znanych" błędów programowania. Nie zamierzamy tu obrażać niczyjej inteligencji i wyjaśniać zapotrzebowania na bezpieczne oprogramowanie w dzisiejszym globalnym świecie — zakładamy, że powody są znane — ale postaramy się pokazać, jak wyszukiwać i usuwać typowe usterki zabezpieczeń w kodzie.

Niepokojącą stroną błędów zabezpieczeń jest to, że naprawdę łatwo jest je popełnić i że efekty bardzo prostego, jednowierszowego błędu mogą być katastrofalne. Wadliwy kod, który był przyczyną robaka Blaster, liczył dwie linie.

Gdybyśmy mogli udzielić tylko jednej mądrej rady, brzmiałaby ona tak: „żaden język programowania ani żadna platforma nie zapewni całkowitego bezpieczeństwa oprogramowania — może to zrobić tylko i wyłącznie jego autor". Powstało mnóstwo literatury na temat tworzenia bezpiecznego oprogramowania i część najważniejszych publikacji została napisana przez autorów tej książki. Istnieje jednak zapotrzebowanie na niewielką, łatwą w czytaniu, praktyczną książkę, omawiającą w skrócie wszystkie podstawowe tematy.

Podczas pisania tej książki, aby zachować jej praktyczny charakter, przestrzegaliśmy czterech reguł:

- Ma być prosta. Nie koncentrowaliśmy się na rzeczach niepotrzebnych. Nie ma tu opowieści o wojnach, zabawnych anegdot — są tylko ważne fakty. Z reguły każdy chce jedynie wykonać swoją pracę i napisać jak najlepszy kod w możliwie najkrótszym czasie. Z tego względu staraliśmy się pisać jak najprościej, aby umożliwić szybkie przeglądanie książki i zdobywanie potrzebnej wiedzy.
- Ma być krótka. Jest to konsekwencją poprzedniego punktu: koncentrując się tylko i wyłącznie na faktach, byliśmy w stanie zachować niewielką objętość książki. W rzeczywistości odnosiło się to także do tego wprowadzenia.
- Ma dotyczyć wielu platform. Internet jest złożonym miejscem z milionami połączonych wzajemnie komputerów, pracujących pod kontrolą różnych systemów operacyjnych i wykorzystujących aplikacje napisane w wielu językach progra-

mowania. Chcieliśmy, aby ta książka nadawała się dla wszystkich programistów, dlatego zawarte w niej przykłady odnoszą się do większości istniejących systemów operacyjnych.

- Ma dotyczyć wielu języków. Jest to konsekwencja poprzedniego punktu — większość przykładów odnosi się do różnych języków. Ponadto w całej książce pokazujemy mnóstwo błędów zabezpieczeń w wielu konkretnych językach programowania.

Układ książki

Każdy rozdział dotyczy jednego grzechu śmiertelnego. Tak naprawdę nie ma ustalonej kolejności grzechów, ale staraliśmy się podać na początku te najbardziej haniebne. Każdy rozdział dzieli się na następujące podrozdziały:

- **Omówienie grzechu** — krótkie przedstawienie grzechu, czyli dlaczego grzech jest tak naprawdę grzechem.
- **Grzeszne języki programowania** — zestawienie języków, w których łatwo ten grzech popełnić.
- **Objaśnienie grzechu** — sedno problemu, czyli główna przyczyna, która sprawia, że grzech jest taki „grzeszny".
- **Przykłady grzesznego kodu** — konkretne przykłady grzechu w różnych językach programowania, dla różnych platform.
- **Grzechy pokrewne** — ewentualne odniesienia do innych grzechów.
- **Wykrywanie grzechu** — wskazówki dotyczące szukania miejsc w kodzie, które mogą zawierać błędy.
- **Wykrywanie grzechu podczas analizy kodu programu** — zasada dość oczywista, czyli czego szukać we własnoręcznie napisanym kodzie, aby wykryć dany grzech. Jak wiadomo programiści są wiecznie zajęci, są to więc podrozdziały bardzo krótkie i treściwe.
- **Techniki testowania wykrywające grzech** — narzędzia i techniki testowania, używane do wykrywania danego grzechu.
- **Przykłady grzechu** — autentyczne przykłady grzechu zaczerpnięte z bazy danych CVE (Common Vulnerabilities and Exposures — *www.cve.mitre.org*), BugTraq (*www.securityfocus.com*) i Open Source Vulnerability Database (*www.osdvb.org*) z komentarzami autorów tej książki. Uwaga! Ze względu na planowane przejście z numerów CVE i CAN wyłącznie na numery CVE, każde odwołanie do numeru CAN trzeba ewentualnie zamienić na odwołanie do numeru CVE. Na przykład, jeśli nie uda się znaleźć numeru CAN-2004-0029 (błąd w Lotus Notes na platformie Linux), trzeba będzie poszukać numeru CVE-2004-0029.

- **Postanowienie poprawy** — jak poprawić kod, aby „zmazać" grzech. Tu również są podawane różne środki zaradcze w różnych językach programowania.
- **Specjalne środki ochronne** — inne metody, które można zastosować nie tyle w celu rozwiązania samego problemu, co raczej w celu utrudnienia napastnikom wykorzystania potencjalnej luki lub w celu zapewnienia ochrony w przypadku faktycznego popełnienia błędu.
- **Materiały dodatkowe** — książka jest krótka, dlatego zawiera odsyłacze do miejsc z dodatkowymi informacjami, takimi jak inne rozdziały, artykuły naukowe i strony WWW.
- **Rachunek sumienia** — jest to naprawdę ważna część każdego rozdziału i należy często do niej zaglądać. Jest to lista tego co wolno, czego nie wolno i na co należy uważać, pisząc nowy kod lub przeglądając stary. Nie wolno lekceważyć tej części! Zalecenia ze wszystkich rozdziałów są zebrane w dodatku B.

Dla kogo jest ta książka

Książka ta, skierowana do wszystkich twórców oprogramowania, przedstawia w skrócie najbardziej typowe i niszczycielskie grzechy programowania oraz sposoby ich eliminowania z kodu przed dostarczeniem oprogramowania użytkownikowi. Z książki mogą korzystać osoby piszące w takich językach, jak: C, C++, Java, C#, ASP, ASP.NET, Visual Basic, PHP, Perl czy JSP. Ponadto uwzględnione są w niej takie platformy, jak Windows, Linux, Apple Mac OS X, OpenBSD i Solaris oraz duzi i mali klienci, a także użytkownicy sieci WWW. Tak naprawdę nie ma znaczenia ani używany język programowania, ani system operacyjny, ani nawet to, jak bezpieczna jest cała platforma. Jeśli aplikacja zawiera luki w zabezpieczeniach, to jej użytkownicy są narażeni na ataki.

Co należy przeczytać

Książka jest krótka, więc nie należy się lenić. Lepiej przeczytać ją w całości, bo nie wiadomo, co trzeba będzie robić następnym razem!

Z drugiej strony niektóre grzechy dotyczą tylko pewnych języków programowania i wpływają tylko na konkretne środowiska. Z tego względu należy przede wszystkim przeczytać o tych grzechach, które mają związek z używanym językiem programowania, docelowym systemem operacyjnym oraz środowiskiem.

Oto krótki wykaz minimum tego, co należy przeczytać w zależności od obowiązującego scenariusza.

- Każdy powinien przeczytać o grzechach VI, XII i XIII.

- Twórcy aplikacji dla systemów Linux, Mac OS X lub Unix powinni przeczytać o grzechu XVI.
- Osoby programujące w C/C++ muszą przeczytać o grzechach I, II i III.
- Osoby programujące dla sieci WWW przy użyciu technologii, takich jak JSP, ASP, ASP.NET, PHP, CGI lub Perl, powinny przeczytać o grzechach VII i IX.
- Osoby tworzące aplikacje komunikujące się z aparatami baz danych, takimi jak Oracle, MySQL, DB2 czy SQL Server, powinny przeczytać o grzechu IV.
- Osoby tworzące rozwiązania sieciowe (WWW, klient-serwer itp.) powinny przeczytać o grzechach V, VIII, X, XIV i XV.
- Twórcy aplikacji stosujących jakiekolwiek szyfrowanie lub wykorzystujących hasła powinni przeczytać o grzechach VIII, X, XI, XVII i XVIII.
- Twórcy aplikacji przeznaczonych dla niewyrobionych użytkowników powinni przeczytać o grzechu XIX.

Wierzymy, że jest to ważna książka, ponieważ zawiera omówienie wszystkich popularnych języków programowania, platform i środowisk wdrażania, autorstwa trzech spośród najbardziej znanych specjalistów z dziedziny zabezpieczeń. Ufamy, że bardzo poszerzy ona wiedzę każdego czytelnika i dostarczy mu mnóstwa cennych wskazówek.

Michael Howard
David LeBlanc
John Viega
lipiec 2005