Kto je zodpovedný za chyby a bezpečnostné slabiny v softvéri?

PETER OROSI

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava

Abstrakt. Či už chceme alebo nie, chyby sú neoddeliteľnou súčasťou softvéru. Preto sa zamyslíme nad príčinou vzniku týchto chýb a nad možnosťami ako im predchádzať. Spomenieme niektoré metódy manažmentu kvality softvéru, napr. inšpekciu alebo normy rodiny ISO 9000. Zamyslíme sa aj nad mierou, do akej sú softvérové spoločnosti zodpovedné za chyby vo svojich softvérových produktoch a do akej miery je im možné pripisovať právnu zodpovednosť za prípadné škody. Pritom vezmeme do úvahy aj druhú stranu mince, t.j. zlomyseľné kódy, ktoré sa úmyselne snažia o spôsobovanie škôd, najmä využívaním bezpečnostných slabín v softvéri. Na záver si stručne predstavíme základné koncepty bezpečného operačného systému EROS a zamyslíme sa nad jeho nasadením do reálnej prevádzky.

Úvod

Určite každý z nás pozná tzv. "modrú smrť" operačného systému Windows 98. Preslávená modrá obrazovka sa stala námetom už nejedného vtipu. V dobe najväčšej slávy tohto systému nejednému používateľovi privodzovala občas až infarktové stavy a vyvolávala lavínu expresívnych výrazov na stranu Microsoftu. Bol však Microsoft naozaj až taký zlý ? Je jasné, že chybám v softvéri sa nedá vyhnúť ani pri maximálnej opatrnosti. Pokým budú programy písať ľudia, budú tieto naďalej plné chýb. Rôzne zdroje udávajú rôzny počet chýb na 1000 riadkov zdrojového kódu. Ak by sa jednalo len o jednu chybu na 1000 riadkov, je jasné, že také zložité programové systémy, akými sú aj operačné systémy musia obsahovať stovky chýb. Nemenej zlé sú chyby v bežných aplikáciách. Určite nikoho neteší práca s nestabilnou aplikáciou, ktorá každú chvíľu „padá“. Často sa vyskytujú aj chyby, ktoré síce zásadným spôsobom neohrozujú vykonávanie aplikácie, ale používateľovi dokážu znepríjemniť prácu. Jedná sa o chyby ktoré neboli odhalené pri testovaní produktu a musia byť odstránené dodatočne v rámci údržby, na základe podnetov zákazníkov. Osobitnou kapitolou sú škodlivé kódy ako vírusy, trójske kone, červy a podobne. Tieto sa väčšinou snažia zneužiť bezpečnostné

Manažment v softvérovom inžinierstve, máj 2005, s. 1-10.

chyby v operačnom systéme na prienik a spôsobenie škody. Bez ich pričinenia by sme sa o týchto chybách nedozvedeli, pretože na prvý pohľad sa nám zdá, že systém funguje normálne. V nasledujúcich častiach preberieme problematiku chýb vo vývoji softvéru podrobnejšie. Z dôvodu obmedzeného rozsahu sa zameriam iba na niektoré časti tejto širokej problematiky.

Zdroje a charakterizácia chýb

Napriek niekoľko desaťročnej histórii tvorby softvéru, proces jeho vývoja stále nie je imúnny voči chybám. Zlepšili sa ale metódy na odhaľovanie chýb, prípadne na prevenciu voči chybám. Napr. existencia typových jazykov (Java), testovacie nástroje (xUNIT), statická analýza kódu atď. Zaujímavé však je, že počet chýb skúseného programátora a programátora začiatočníka je približne na rovnakej úrovni [2]. Samozrejme tu zrejme treba zohľadniť, že kód, na ktorom pracuje skúsený programátor je väčšinou oveľa komplikovanejší ako u začiatočníka, preto sú aj chyby ktoré urobí sofistikovanejšie a oveľa ťažšie rozpoznateľné. Takisto pri existencii testovacích nástrojov si musíme dať pozor, či tie samé v sebe neobsahujú chyby, pretože inak je výsledok našej námahy nepoužiteľný. Jedným z najčastejších problémov býva oprava chýb, kedy odstránením problému sa často zavedú nové chyby. Ak neotestujeme všetky funkcie v systéme po oprave chyby (testovanie vykonaných zmien na všetkých moduloch, tzv. regresné testovanie) riskujeme zavedenie nových chýb. Ak hovoríme o kvalite softvéru máme na mysli najmä mieru do akej softvér vyhovuje špecifikácia, ale aj s ohľadom na počet chýb. Manažment kvality má za cieľ dosiahnutie kvalitného produktu.

Tak ako poznamenal Mark Paulk z univerzity Carnegie Mellon [2]: „Základný problém s kvalitou softvéru je, že programátori robia chyby“. Táto ľudská omylnosť sa výraznejšie prejavuje s rastúcou zložitosťou systému. Čím zložitejší systém, tým väčší počet chýb možno predpokladať. Užitočné, ale postrádateľné funkcie systému zapríčiňujú väčšinu z tejto zložitosti. Nie všetky chyby sa navonok prejavujú (tzv. latentné chyby). Chyby sa môžu vyskytnúť aj v skorších etapách životného cyklu ako pri implementácii, napr. nesprávnym pochopením požiadaviek zákazníka, alebo v dôsledku návrhu, ktorý je v rozpore so špecifikáciou a pod. Ak má byť ale projekt úspešne ukončený, tieto chyby musia byť vo výslednom projekte odstránené. Väčšina chýb vo výslednom produkte je dôsledok omylov programátorov (preklepy a iné chyby z nepozornosti).

Niektoré vlastnosti softvéru, aj keď nemôžu byť považované priamo za chyby, obsahujú predpoklady na vznik bezpečnostných slabín. Jedná sa napr. o populárny systém pluginov, podporovaný množstvom aplikácií ako prostriedku na rozšírenie funkcionality, prípadne používanie skriptovania v rámci aplikácie. V prípade ak je plugin alebo skript napísaný s úmyslom spôsobiť škodu, aplikácia ho nevie jednoducho rozlíšiť od užitočného obsahu a preto sa stáva zraniteľnou. Možné spôsoby ochrany sú napr.:

- Analýza – zamietnutie kódu ak tu je potenciál, že kód môže spôsobiť škodu.

- Prepísanie – modifikácia kódu pred spustením tak, aby nemohol byť škodlivý.
- Monitorovanie – monitorovanie kódu počas jeho vykonávania a následné zastavenie pri pokuse spáchať škodu.
- Auditovanie – kontrola program a ak spáchal nejakú škodu, vykonanie príslušnej akcie.

Otázka je, do akej miery sú chyby v softvéri škodlivé, resp. akceptovateľné. Napr. Richard Clarke, poradca prezidenta Busha sa vyjadril k otázke chýb nasledovne: „Už nemôže byť dlhšie akceptovateľné, aby sme si kúpili softvér, ktorý obsahuje defekty“ [4]. V praxi je však význam slova akceptovateľný softvér úplne iný. Zväčša nie je v silách softvérových spoločností dodať svoje produkty v bezchybnom stave už od prvej verzie, skôr sa stalo dobrým zvykom zverejňovanie rôznych patchov a vychytanie najzávažnejších chýb až od verzie 2.0. Ekonomický prínos tohto zaužívaného postupu je výrazný. Na otázku, či je aj správny je však odpoveď skôr nie ako áno. Správny prístup je vývoj softvéru už od počiatku s cieľom systematicky predchádzať vzniku chýb. Nie je správne iba podľa potreby opravovať chyby, ale dôraz by sa mal klásť na pochopenie príčin týchto chýb a ich odstránenie (napr. pretečenie zásobníka)

Takisto si treba uvedomiť rozdiel medzi robustným softvérom a korektným softvérom. Softvér korektný vzhľadom na špecifikáciu vždy môže skončiť chybou, stačí ak napríklad používateľ zadá chybný vstup (taký, ktorý nezodpovedá špecifikácii).

Možnosti prevencie chýb a manažment kvality

Inšpekcia

Jedným z prostriedkov na zabezpečovanie kvality softvéru je aj inšpekcia. Hlavným cieľom inšpekcie je otestovať program do detailu, nezaujímame sa pri tom o samotný proces tvorby softvéru. Podľa skúseností odborníkov, aj keď formálna verifikácia produktu používa matematické požiadavky, ani tieto nemusia byť vždy schopné správne zachytiť návrhárov alebo zákazníkov úmysel. Inšpekcia dokumentácie požiadaviek zaručuje, že zachytené požiadavky zodpovedajú skutočnosti.

Takisto platí, že systémy, ktoré reagujú na množstvo okolných udalostí sú vo všeobecnosti menej dôveryhodné ako čisto sekvenčné systémy, pretože paralelizmus spracovania prináša so sebou určitú mieru nedeterminizmu. Typickým príkladom je operačný systém, kde by sme potrebovali otestovať všetky možné stavy, do ktorých sa možno dostať (čo v praxi predstavuje problém). Nedeterminizmus môže spôsobovať komplikácie aj z dôvodu, že rovnaký test môže v jednom prípade prejsť, a v inom prípade za zdanlivo rovnakých okolností (iba miernej zmeny stavu) rovnaký test zlyhá. Jedným z možných riešení je prispôsobenie návrhu požiadavkám testovania, napr. preto, aby bolo možné zabezpečiť časovanie podľa našich predstáv.

Osobitné problémy nás čakajú pri inšpekcii objektovo orientovaného kódu z dôvodu, že metódy v rámci rôznych objektov sa navzájom môžu volať komplikovaným spôsobom. Prieskumy ukázali, že zdroj veľkého počtu ťažko odhaliteľných chýb v OO kóde má svoj pôvod v chybách šírených v systéme, t.j. jedno volanie metódy na určitom mieste obsahuje chybu a táto chyba sa prešíri cez reťaz volaní ďalších pomocných metód, až do miesta kde spôsobí problémy. Autori v [5] túto vlastnosť nazvali delokalizácia. Jedným zo spôsobov ako sa pri inšpekcii vyhnúť delokalizácii je systematické štúdium kódu a testovanie, ktoré je však časovo náročne a pri rozsiahlych projektoch nereálne. Preto býva používaný skôr spôsob „podľa potreby“, programátor sa zaoberá iba štúdiom kódu, ktorý považuje za opodstatnený, čo je však spojené s určitým rizikom. Metódy inšpekcie, ktoré môžeme použiť sú napr.:

- Technika riadená abstrakciou – vyžaduje od inšpektorov reverzným spôsobom vytvoriť abstraktnú špecifikáciu každej triedy a metódy, jedná sa vlastne o efektívne riešenie problému delokalizácie.
- Technika prípadov použitia – zaoberá sa kódom z dynamického hľadiska, zabezpečuje aby každý objekt reagoval korektne na všetky možné prípady svojho použitia. Dôraz sa kladie najmä na volanie správnych metód používateľom triedy a na to aby zmeny stavu objektu pri volaní každej metódy boli konzistentné
- Metóda kontrolného zoznamu – je založená na sérii špecifických otázok zameraných na typické zdroje chýb. Môže sa napríklad jednať o otázky: Ak sa vyžaduje volanie metódy rodiča v konštruktore, je toto volanie prítomne ? Sú všetky inštancie premenných inicializované na rozumnú hodnotu ?

Samotná inšpekcia zvyčajne nemá trvať dlhšie ako 2 hodiny, a počet preverovaných riadkov kódu by sa mal pohybovať okolo 200. Výsledným počtom odhalených a neodhalených chýb sa metóda zaraďuje medzi klasické testovacie postupy (viacej neodhalených chýb) a metódy formálnej verifikácie (minimum chýb).

ISO 9000

Ďalším z prostriedkov na zabezpečenie kvality je skupina noriem rodiny ISO 9000. Vychádza sa z predpokladu, že kvalitný proces zabezpečí kvalitný produkt. Jedná sa o všeobecne použiteľnú normu, v zásade nezávislú od oblasti nasadenia. Ako preukázateľný dôkaz, že spoločnosť spĺňa požiadavky na proces, slúži certifikát udelený certifikačnou inštitúciou. Tá je zase akreditovaná pomocou akreditačnej inštitúcie. Udelené certifikáty sú medzinárodne uznávané. Normy, ktoré si spoločnosť môže dať certifikovať sú ISO 9001 (ak organizácia vykonáva návrh, vývoj, inštaláciu a údržbu produktu), ISO 9002 (ako ISO 9001 ale bez etapy návrhu a vývoja), ISO 9003 (testovanie), ISO 9004 (zlepšovanie výkonnosti). Proces získania certifikátu je však veľmi zdĺhavý a vhodný iba pre dostatočne veľké spoločnosti. Základnou požiadavkou týchto noriem je pravidelné auditovanie, rozoznávame dva typy auditov:

- vykonaný externou certifikačnou inštitúciou

- vykonávaný interne v rámci spoločnosti vyškoleným pracovníkom

Cieľom auditov je nepretržitý proces zlepšovania kvality, overuje sa, či činnosti vzťahujúce sa na kvalitu a v súvislosti s ňou dosahované výsledky zodpovedajú ustanoveniam ako aj to, či sú tieto ustanovenia dostatočne vhodné. Stručne by sa dali činnosti auditora charakterizovať takto:

- Povedz mi čo robíš (popíš firemný proces)
- Povedz mi kde sa to spomína (odvolanie na manuál procesu)
- Dokáž, že je to to, čo sa vykonalo (predlož dôkaz v podobe dokumentácie, na kvalitu dokumentácie sa kladú vysoké nároky)

Napriek svojej komplikovanosti má norma svoje nesporné výhody, ak ku jej dodržiavaniu pristupujeme zodpovedne, je vysoká pravdepodobnosť, že výsledný produkt bude dobre zdokumentovaný, v súlade so špecifikáciou a vysoko kvalitný. Držiteľ certifikátu ISO 900x má takisto konkurenčnú výhodu v porovnaní so spoločnosťou bez certifikátu.

XP – extrémne programovanie

Zaujímavé sú aj metódy, ktoré zavádza extrémne programovanie (XP). Pri písaní kódu pomocou XP programátori najskôr venujú niekoľko minút príprave testov (napr. pomocou automatizovaných xUNIT testov) pre zatiaľ prázdne implementácie tried a vzápätí implementujú tieto metódy tak, aby prešli testami. Takýto postup, ktorý sa nazýva TDD (test driven development) zachováva návrh systému na najjednoduchšej možnej miere, pretože sa nezaťažujeme funkciami, ktoré možno bude treba v budúcnosti realizovať. Ku kvalite produktu prispievajú aj niektoré ďalšie zásady, napr. jeden programátor píše kód, druhý ho kontroluje, dodržiavanie pracovného režimu max. 40 hodín za týždeň (únava je tiež zdrojom chýb). Takisto proces vývoja býva nepretržite konzultovaný so zákazníkom, čím sa znižuje možnosť objavenia chýb až vo fáze implementácie (v porovnaní s klasickými softvérovými procesmi). Aj keď postupy XP nie sú vhodné pre každý typ projektov, v praxi ukazujú svoju opodstatnenosť. Výsledná kvalita a potrebný čas vývoja pre isté typy projektov môže byť vhodnejšia pri použití XP ako klasických metód vývoja softvéru.

Súvis medzi kvalitou, zložitosťou, spoľahlivosťou a prostriedkami na vývoj softvéru

Exitujú dve hlavné metódy zabezpečenia softvéru

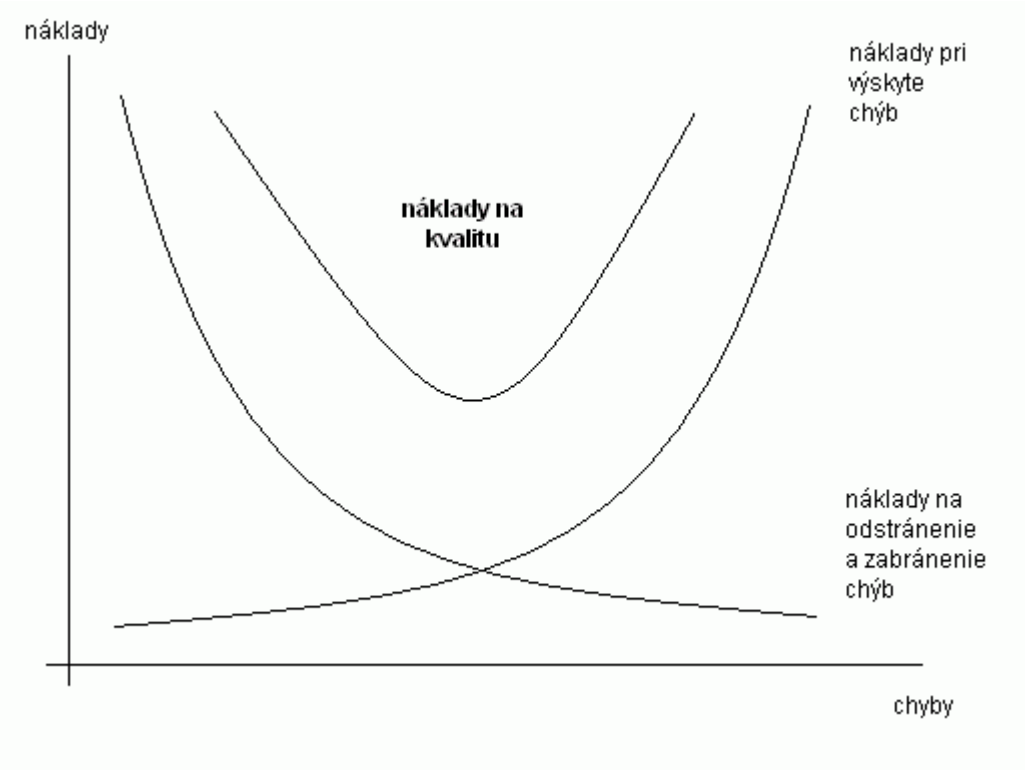
1. zabránenie chýb prostredníctvom formálnych prostriedkov špecifikácie a verifikácie
2. odolnosť voči chybám prostredníctvom diverzity, pod odolnosťou voči chybám rozumieme schopnosť počítačového systému pokračovať v činnosti aj po výskyte chýb

Prvý spôsob formálnych metód sa zatiaľ podarilo úspešne použiť iba na jednoduchšie systémy a jeho nevýhodou je aj veľká náročnosť.

Druhý spôsob pomocou diverzity ráta s viacerými implementáciami rovnakej časti systému. Jedná sa predovšetkým o metódu N-verzií a zotavovacích blokov. Vo všeobecnosti je metóda zotavovacích blokov účinnejšia, pretože stačí, aby bola jedna verzia rovnakej časti správna, kým pri N-verziách sa vyžaduje správnosť väčšiny častí.

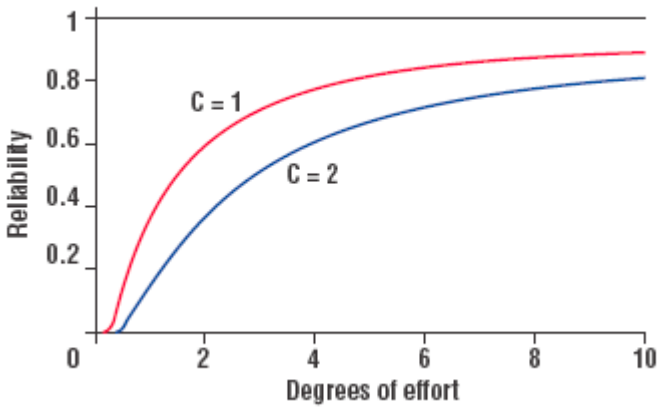
Pri moderných systémoch spôsobuje problémy znovupoužitie súčiastok, pokiaľ nemáme istotu, že súčiastky neobsahujú chyby. Rozdelenie zdrojov pre diverzitu môže viesť k zvýšenej ako aj zníženej spoľahlivosti, v závislosti od architektúry systému. Kľúč k zvýšeniu spoľahlivosti nie je v zvyšovaní stupňa diverzity (počet rôznych implementácií rovnakej veci), ale v existencii jednoduchých základných komponentov, ktoré zabezpečujú funkčnosť kritických častí aj v prípade zlyhania komponentov mimo jadra.

Očividne čím je systém zložitejší, tým ťažšie sa dá zaručiť jeho spoľahlivosť a tým viac je náchylný na chyby. Náklady na softvér sú obmedzené, preto s nimi treba narábať rozumne. Kvalitou softvéru rozumieme mieru splnenia požiadaviek a zároveň absencie chýb v produkte. Súvis medzi nákladmi, chybami, a kvalitou zachytáva obrázok 1.



Obr. 1 Súvis medzi nákladmi, chybami a kvalitou

Predpokladajme, že spoľahlivosť softvéru predstavuje exponenciálnu funkciu času *R(t)*, a ďalej predpokladajme, že početnosť zlyhaní je priamoúmerná zložitosti softvéru (C) a nepriamo úmerná úsiliu na vývoj (E). Vo výsledku sa spoľahlivosť R javí ako funkcia C a E, R = f(C, E). Pre najjednoduchší prípad bez použitia diverzity je situácia zobrazená na obrázku 2.



Obr. 2. Súvis medzi spoľahlivosťou (R) systému, vynaloženým úsilím (E) a zložitosťou softvéru (C).

Zodpovednosť za chyby

Softvéru sa často vyčíta jeho problematická „bezchybnosť“. Vzhľadom na vysokú zložitosť softvéru býva prakticky nemožné objaviť všetky chyby ešte pred nasadením. Softvér, ktorý sa bežne využíva obsahuje chyby a používateľov tieto chyby niekedy môžu vyjsť draho. Odhaduje sa, že len v USA sa tieto chyby pohybujú každoročne na úrovni 50 miliárd dolárov. Vyvstáva preto otázka, či by softvérové spoločnosti nemali niesť zodpovednosť za svoje chybné produkty. Nedá mi nespomenúť známy vtip o autách, keby ich vyvíjal Microsoft. Autá, ktoré fungujú bezchybne len na 90% diaľnic, občas sa bezdôvodne zastavujú a na svojho majiteľa kladú rôzne nezmyselné požiadavky. Takéto autá vykazujú väčšinu nepekných vlastností, s ktorými sa môžme u (nielen) Microsoftu stretnúť a ich nasadenie do praxe si zrejme nevieme predstaviť. Jednoducho by si nikto nezobral na zodpovednosť prípadne dôsledky zlyhania.

Z pohľadu softvérových spoločností je však všetko v najlepšom poriadku. Stačí, keď si prečítame licenciu väčšiny programov. Takmer vždy objavíme magickú vetu o tom, že výrobca nezodpovedá za prípadné škody. V praxi častokrát najväčšie problémy spôsobujú latentné bezpečnostné chyby, ktoré sa na prvý pohľad nejavia ako chyby, len napr. umožňujú zlomyseľným kódom prienik do systému. Je snáď Microsoft aj iné firmy preto zodpovedný za svoje chyby ?

Ako ukazuje už obrázok 1, odstránenie všetkých chýb je možné iba za teoretického predpokladu (takmer) nekonečných nákladov. Preto sú v súčasnosti požiadavky na bezchybný softvér ďaleko od toho, čo možno reálne dosiahnuť. Bezchybnosť softvéru treba požadovať, ale iba pri absolútne kritických podmienkach nasadenia, napr. v prípade, ak softvér zodpovedá za riadenie kritických častí jadrovej elektrárne, prípadne riadenie lietadla. V podobných prípadoch je na mieste aj formálna verifikácia. Vo všetkých ostatných prípadoch sa jedná o kompromis medzi kvalitou a cenou.

Vzhľadom na veľkú zložitosť softvéru môžu spoločnosti niesť zodpovednosť iba do určitej miery, v niektorých prípadoch je to však dostatok na zaplatenie pokuty poškodenému zákazníkovi. Takéto prípady sa už reálne odohrali. Microsoft zatiaľ podobným súdnym sporom odoláva. A to aj napriek tomu, že práve chyby v jeho operačnom systéme umožňujú vykonávať autorom zlomyseľných kódov veľké škody. Situáciu možno prirovnať k zabezpečenému domu (bezpečný operačný systém) a jeho obyvateľovi (používateľ operačného systému). Autor zlomyseľného kódu (zlodej), by sa chcel vlámať do domu používateľa a spôsobiť mu škodu. Ak je dom dostatočne zabezpečený (napr. robustný plot s ostnatým drôtom a pod vysokým napätím okolo celého domu) bude zdolanie takejto prekážky zväčša nad zlodejove sily. Ak naopak obyvateľ nechá otvorené ešte aj vchodové dvere, s najväčšou pravdepodobnosťou bude vykradnutý. Analogicky platí, že operačný systém plný dier láka k zneužitiu. Aj keď hlavná zodpovednosť leží na zlodejovi, ponechávať systém nezabezpečený je nerozvážne. Preto aj keď pisateľ vírusov nesie priamu zodpovednosť za spôsobené škody, výrobca operačného systému takisto nesie určitý stupeň nepriamej zodpovednosti. Keď sa teraz pokúsim odpovedať, či by mal byť za túto nepriamu zodpovednosť výrobca operačného systému právne postihnuteľný, moja odpoveď je nie, pravdepodobne by nemal, pretože je v súčasnosti nereálne očakávať úplne bezchybný softvér a hlavné bremeno zodpovednosti je aj tak na autoroch zlomyseľných kódov.

Aj keď sa často zdôrazňuje výhodnosť iných, alternatívnych operačných systémov na báze Linux, ani tieto systémy nie sú voči chybám imúnne. Jedným z možných dôvodov menšej chybovosti týchto systémov je zrejme aj ich nižšia rozšírenosť a potom aj iná filozofia bezpečnosti. To, že je softvér dodávaný aj s voľnými zdrojovými kódmi, však ešte neznamená, že musí byť aj kvalitnejší. Napr. v zdroji [6] sa uvádza: „Je naozaj otázne, aj z pohľadu spoľahlivosti, či mať väčší počet inšpektorov iba s bežnými znalosťami, dáva lepší výsledok“. To na čo sa tu naráža, je aj kvalita inšpektorov ako aj fakt, že nie všetci, ktorí pracujú s takýmito voľnými zdrojovými kódmi, nutne hľadajú v nich chyby. Pravdepodobne výsledný kód bude mať menej očividných chýb, ale vo výsledku aj tak veľa chýb ostáva neodhalených. Hlavná výhoda open-source je však istota, že do kódu nebol primiešaný žiadny trójsky kôň prípadne iné škodlivé a postranné činnosti. A nemenej významný je aj spôsob opravovania chýb. Ak sa nejaká chyba objaví, zväčša odstránenie chyby je otázkou hodín, alebo dní. Oproti klasickému prístupu softvérových spoločností je tento spôsob pružnejší.

Pri hľadaní zdrojov som našiel zaujímavý článok [7], ktorý sa zaoberá tvorbou bezpečného operačného systému s názvom EROS (Extremely Reliable Operating System). Zatiaľ sa jedná o projekt čisto na akademickej pôde, avšak svojimi výsledkami predstavuje do budúcnosti minimálne zaujímavú alternatívu k existujúcim operačným systémom (Windows, Linux). Jeho hlavná črta je jednoduchá, vnútorne konzistentná architektúra, kedykoľvek je nejaká aktivita v rozpore s prísnymi pravidlami, je požiadavka na túto aktivitu zamietnutá. Takisto bola vykonaná formálna verifikácia jeho kritických bezpečnostných častí. Medzi jeho ďalšie vlastnosti patria:

- bezpečný reštart, nemožnosť vzniku inkonzistencií v súborovom systéme
- bezstavový kernel , stav vykonávania sa cache-uje z priestoru alokovaného používateľom, zabezpečuje sa konzistencia cache so stavom v používateľskom priestore
- štruktúra aplikácií, aplikácie sú stavané z ako skupina spolupracujúcich jednoduchých komponentov, pričom každý z komponentov (aj v rámci jednej aplikácie) beží pod inými prístupovými oprávneniami (napr. právo zápisu na disk) - schopnosťami. Tieto schopnosti sú chránené v rámci kernelu. Ak napríklad vírus dosiahne kontrolu nad jedným z týchto komponentov, škody ktoré môže spôsobiť sú minimálne, pretože nemajú dostatok oprávnení na ovplyvnenie systému ako celku (na rozdiel od klasických operačných systémov). Návrh aplikácii ako malých jednoduchých komponentov uľahčuje testovanie. Pomocou dobre zvolených testov je zväčša možné otestovať všetky stavy komponentu, takisto kompilátor sa stará predchádzanie rôznych pretečení zásobníkov.
- adaptácia aplikácií, v súčasnosti prebiehajú snahy prepísať bezpečným spôsobom na túto platformu podporu pre najdôležitejšie protokoly HTTP, SMTP, FTP atď. (jedná sa o rozhrania vykonávané s vysokým stupňom právomocí), neskôr sú naplánované aplikácie, ktoré spúšťajú rôzne skriptovacie jazyky (prehliadače, word procesory atď.). Plánuje sa zviesť aj emulácia Linuxu

Ak by sa tento systém niekedy v budúcnosti dočkal reálneho nasadenia aj širokej programátorskej podpory pri tvorbe aplikácií, bol by to významný krok v pred na poli softvérovej bezpečnosti.

Záver

Chyby sú neoddeliteľnou súčasťou softvéru, bolo tomu tak v minulosti a určite to tak bude aj v blízkej budúcnosti. Určitú nádej v boji proti chybám predstavujú rôzne metódy manažmentu kvality, ako aj precízne metódy tvorby softvéru, ani tieto však nepredstavujú úplne riešenie problému. Vždy sa nájdu chyby, ktoré zostanú neodhalené. Určitú nádej na zmenu tohto stavu predstavujú systémy ako EROS, operačný systém novej generácie, kde je bezpečnosť kladená nekompromisne na prvé miesto. To či sa niekedy stanú podobné operačné systémy a rovnako aj aplikácie pre

tieto operačné systémy bežnou praxou alebo sa jedná iba o naivnú víziu sa ukáže až v budúcnosti. V každom prípade by to bol dobrý začiatok.

Použitá literatúra

1. Michael A. Cusumano: Who is liable for bugs and security flaws in software? *Communications of the ACM*, Vol. 47, No. 3 (March 2004), 25-27
2. Gerard J. Holzmann: The Logic of Bugs. *ACM SIGSOFT Software Engineering Notes*, Vol. 27, No. 6 (November 2002), 81-87
3. Bieliková, M.: Softvérové inžinierstvo – Princípy a manažment, 2000
4. Tish Keefe: Software Insecurity**.** *Computerworld* (August 2002), No. 5
5. Alastair Dunsmore, Marc Roper, Murray Wood: Practical Code Inspection Techniques for Object-Oriented Systems: An Experimental Comparison. *IEEE Software*, Vol. 20, No. 4 (August 2003), 21-29
6. Ross Anderson, Terry Bollinger, Dowg Brown, Erique Draler, Philip Machankk, Gary McGraw, Art Pyster, Howard Schidt, Tim Shimeall, Nancy Mead: Information Security Policy. *IEEE Software*, Vol. 17, No.5 (September 2000), 26-32
7. Jonathan S. Sharpio, Norm Hardy: EROS: A Principle-Driven Operating System from the Ground Up. *IEEE Software*, Vol. 19, No. 1 (January 2002), 26 - 33

Annotation

Who is liable for bugs and security flaws in software ?

Regardless we want or not, erros are inherent part of sotware. That's why we discuss causes of these errors and the possibilities how to prevent them. We remark few methods of software quality management, e.g. inspection or ISO 9000 family standards. We consider extent of responsibilty by software companies by which they should be held liable for bugs in their own software. We also take into our consideration the reverse site of the coin, especially malicious codes which are intentionally trying to cause damage by using security flaws in software. Finally we notice basic concepts of secure operating system EROS and we contemplate its application in real operation.