Od doby červa Morris v roce 1988 se počítačoví červi stali jednou z největších výzev internetového věku. Každý měsíc jsou v široké skupině operačních systémů a aplikací hlášeny kritické zranitelnosti. A rovněž roste v alarmujícím množství počet případů zneužití těchto zranitelností počítačovými červy.

Tato kapitola reprezentuje některé slibné techniky ochrany před průniky (intrusion), které jsou založeny na hostitelích, a které mohou zastavit celou třídu rychle se šířících virů, využívajících útoků přetečením bufferu (například W32/CodeRed[1], Linux/Slapper[2] a W32/Slammer[3]).

Poznámka

Zmínil jsem zde techniky přetečení bufferu, protože je pokládám za nejvýznamnější. Existuje ovšem několik dalších možností, kterým jsem se vyhnul – buď proto, že nejsou natolik důležité, anebo jsou velice specializované a pokrývají pouze malý počet možných exploitací.

13.1 Úvod

Počítačoví červi mohou být klasifikováni na základě způsobu replikace, který používají. Během několika posledních let používala většina úspěšných (také nazývaných "in-the-wild") počítačových červů e-mail, jako svůj infekční nástroj pro rozšíření se do nových hostitelských systémů. Červi tohoto typu se nazývají jako hromadně se rozesílající červi.

I před fakt, že se binární červi typu W32/SKA@m (červ Happy99) silně rozšířili, stali se všeobecně známými až díky e-mailovým virům založeným na makrech a skriptech, jako třeba V97M/Melissa@mm nebo VBS/LoveLetter@mm. Tento trend byl pak následován několika lety úspěšných útoků binárních červů pro Win32, jako například W95/Hybris, W32/ExploreZip, W32/Nimda či W32/Klez.

V nedávné době se začal u autorů virů objevovat nový trend, vedoucí k vytváření agresivních a rychle se šířících červů. Tento trend byl potvrzen představením červa W32/CodeRed, který znamenal velkou bezpečnostní výzvu.

Když se objeví relativně nová a úspěšná strategie psaní virů, dojde vždy k tomu, že se začnou objevovat nové a nové viry, které začnou tuto strategii používat také. Tento proces klonování produkuje stovky rodin virů, které mají stejné základní charakteristiky, obvykle však s některými vylepšeními. Proto se také očekávalo klonování červa W32/CodeRed a vznik nových a ještě více agresivnějších červů.

Objevení červa W32/Slammer, který v 376 bajtech převzal základní koncept červa CodeRed, nebyl proto nijak překvapující. Slammer je jedním z nejrychleji se šířících binárních červů všech dob[4]. Infekce dosáhla vrcholu v několika hodinách a vedla k masivnímu útoku DoS (Denial of Service) na internetu.

Namísto protokolu TCP, který byl použit v červu CodeRed, červ Slammer používá pro útoky protokol UDP. Protože tento protokol (na rozdíl od TCP) není potvrzovaný a protože byl útok realizován pomocí jediného paketu, byl Slammer mnohem rychlejší než CodeRed. Proces, který se pokouší o spojení pomocí TCP, musí pro zjištění neúspěchu počkat, až vyprší timeout. Slammer oproti tomu jednoduše provede útok na možný cíl a bez čekání pokračuje dál. Úspěšný útok trvá stejně dlouho, jako útok neúspěšný – každý z nich spočívá pouze v zaslání paketu a je tedy velmi rychlý.

Abych byl úplně přesný – metody asynchronního spojení TCP mohou být téměř tak efektivní, jako metody UDP, nicméně to vyžaduje výrazně větší schopnosti jak programátora, tak i kódu.

V budoucnosti můžeme očekávat více škodlivých hackerů, kteří budou těžit z "automatizovaných průniků" za pomoci červů. Vzrůstá proto důležitost systémů ochrany proti těmto skupinám červů.

13.1.1 Blokování skriptů a SMTP červů

Skriptovací červi, jako například VBS/Loveletter@mm, se šíří řádově mnohem rychleji, než předchozí "generace" virů. Skriptovací červi donutili vývojáře společnosti Symantec k začlenění technik blokující tyto hrozby do programu Symantec AntiVirus. V důsledku toho byla technologie pro blokování skriptů úspěšně nasazena už v roce 2000[5].

Je pozitivní, že technologie blokování skriptů vedla k velkým změnám v antivirových produktech, které pak dokázaly efektivněji chránit uživatele v domácnostech. Úroveň nebezpečí se pak začala pomalu snižovat. Další techniky blokování skriptů a efektivnější, souborově orientované, heuristiky pak způsobily trvalý pokles hrozby nebezpečných skriptů.

Náhlý vzrůst 32bitových binárních červů, kteří používají vlastní SMTP engine pro šíření sebe sama pomocí e-mailů, byl přirozeným důsledkem rozvoje skriptů a makro virů. Vznik SMTP červů, jako jsou například W32/Nimda@mm a W32/Klez@mm, vedl k vytvoření a začlenění techniky pro blokování červů v produktu Symantec AntiVirus 2002. Blokování červů je sice jednoduchá, nicméně velmi efektivní technika.

Během posledního roku se pomocí takových technik totiž úspěšně podařilo zabránit hromadnému rozšíření červů, jako W32/Bugbear@mm, W32/Yaha@mm, W32/Sobig@mm[6], W32/Brid@mm, W32/HLLW.Lovgate@mm, W32/Holar@mm, W32/Lirva@mm a mnoha dalším variantám.

Oddělení společnosti Symatec, které je zodpovědné za bezpečnost, dostalo po několika měsících od vypuštění nové verze svého produktu s blokujícími technikami, několik tisíc hlášení o tom, kteří červi byli úspěšně zablokování.

Například – v srpnu roku 2003 se stal červ W32/Sobig.F@mm zodpovědným za jeden z nejvýznamnějších e-mailových útoků, který po několik dní paralyzoval e-mailové systémy. Technologie blokování červů zablokovala více než 900 kopií červa – uživatelé antiviru od společnosti Symatec tak byli před červem chráněni ještě předtím, než byli tito červi vůbec definováni. A podle posledních statistik blokování zastavilo více než 12000 kopií červa W32/Mydoom.A@mm. Tabulka 13.1 ukazuje žebříček dvaceti červů seřazených podle počtu zablokování jejich kopií.

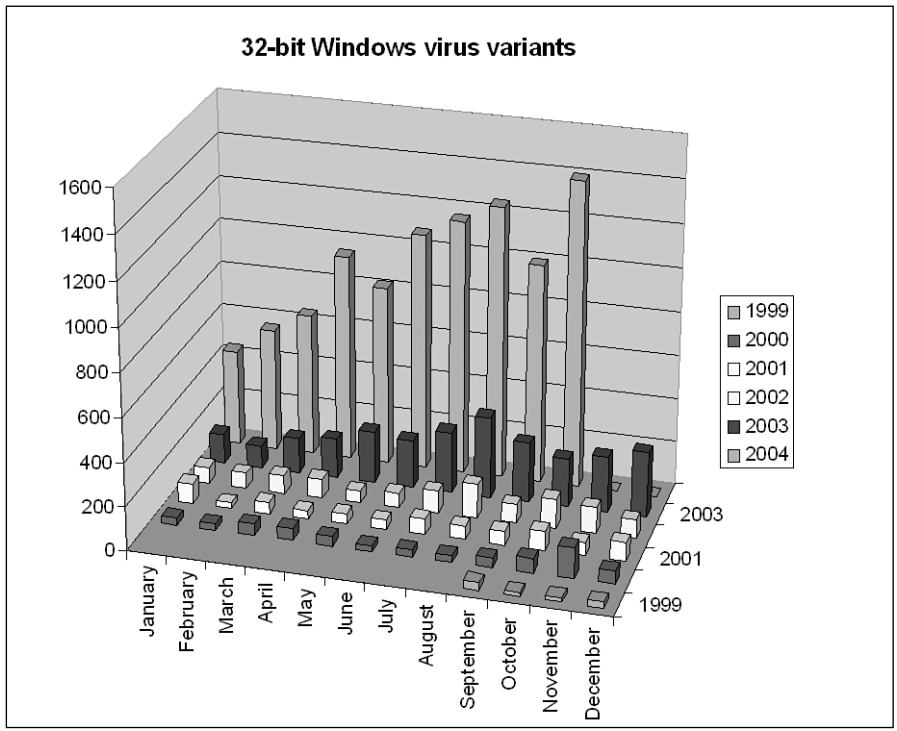
Typicky bývají útoky červů úspěšné až do doby, dokud nejsou v systémech aktualizovány příslušné signatury. Podle tabulky 13.1 je jasně vidět, že bez techniky blokování červů by dalších 12 tisíc počítačových systémů začalo propagovat červa Mydoom.

Tabulka 13.1 – Seznam dvaceti nejvíce zablokovaných červů pro Win32.

Počet hlášení	Jméno červa
12159	W32.Mydoom.A@mm
9709	W32.Netsky.D@mm
5334	W32.Netsky.B@mm
5111	W32.Yaha.K@mm
2598	W32.Netsky.C@mm
2451	W32.Mydoom.F@mm
1275	W32.Netsky.Z@mm
1274	W32.Sobig.E@mm
1210	W32.Mapson.Worm
1048	W32.Netsky.K@mm
1039	W32.Bugbear.B@mm
1021	W32.Sobig.F@mm
971	W32.Netsky.X@mm
888	W32.Dumaru@mm
745	W32.Netsky.Q@mm
673	W32.HLLW.Mankx@mm
652	W32.Sobig.C@mm
629	W32.Sobig.B@mm
390	W32.Mimail.A@mm
372	W32.Netsky.Y@mm

Získávané informace o zablokovaných červech vedou k rychlejšímu vydávání updadů s jejich signaturami. Zákazníci Symantecu tím získávají rychlejší odezvu na vysoce infikující Win32 červy. Tento výsledkem je výborný, zejména s ohledem skutečnost, že autoři virů vytvořili v průběhu každého měsíce roku 2004 cca několik set nových 32bitových virů pro Windows. Mnohé z těch nejvíce úspěšných patří mezi hromadně se rozesílající prostřednictvím e-mailu.

Obrázek 13.1 ukazuje za období od září 1999 do října 2004 celkový počet známých variant 32bitových virů.



Obrázek 13.1 *Celkový počet známých variant 32bitových virů pro Windows měsíčně.*

Z obrázku je zjevné, že tvorba virů se zrychlila v roce 2004, současně s rychle se rozvíjejícím druhem počítačového červa, který pracuje na síťové úrovni a který využívá různé exploitace. Za posledních několik let se objevila asi tisícovka binárních červů založených na e-mailu. Během posledních 12 měsíců se však objevily tisíce nových variant červů, které využívají různých zranitelnosti. Počet těchto červů však může být (v porovnání s e-mailovými červy) ještě podhodnocen, z jednoho hlavního důvodu – je totiž obecně těžší je zaznamenat, nehledě na fakt, že lidé mají každodenní zkušenost s vedlejšími efekty e-mailových červů, stejně jako se spamy. E-mailové schránky jsou jich plné.

Základní myšlenka blokování červů je jednoduchá. Patentovaná technologie funguje tak, že SMTP proxy založená na hostiteli, používá ovladač režimu jádra pro přímé sledování odchozího provozu. Tato komponenta neslouží pouze jako antivirový skener e-mailů, ale slouží také pro blokování červů.

Tato komponenta pro blokování červů ví o všech procesech, které incializují odchozí SMTP provoz, protože všechno prochází přes příslušnou proxy. Příslušná komponenta tak může kontrolovat, zdali je tento proces (nebo jeho nadřazený proces), přítomen v daném e-mailu ve formě přílohy. Škodlivý software, který se sám rozesílá, je tak snadno detekován a následně zablokován. Tento postup funguje i v případě, kdy je přílohou zkomprimovaný soubor, například typu ZIP. Porovnávací algoritmus navíc dokáže rozeznat změny obsahu souboru do té míry, že jsou detekovatelní i červi, kteří během replikace mění strukturu svého těla (například W32/Klez nebo W32/ExploreZip).

Technika blokování červů nemusí být schopna zastavit všechny červy. Má však velké šance na úspěch, zejména v případech, kdy se objeví červi, kteří jsou pouze odvozeni od jiných červů.

13.1.2 Blokování nových útoků – CodeRed a Slammer

Počítačoví červi, jako například W32/CodeRed nebo W32/Slammer, jsou velkou výzvou pro již existující antivirové technologie. Tito vysoce nakažliví červi skáčou z hostitele na hostitele a infikují procesy systémových služeb, přičemž využívají síťových služeb a útoků pomocí přetečení bufferu. Protože není potřeba vytvářet na disku nějaké soubory, a protože je kód injektován přímo do adresovacího prostoru zranitelného procesu, jsou v případě těchto útoků omezeny schopnosti systémů určených pro ověřování integrity souboru.

Tato kapitola se zaměřuje popis aktivních metod, které jsou založeny na hostitelích, a které by měly sloužit jako linie obrany proti neznámým útokům využívající známé metody. Blokování červů na bázi hostitele slibuje detekci neznámých variant červů a jejich útoků, založených na již známých technikách. Techniky blokování červů založené na hostiteli pak nabízejí v takových případech výrazné zvýšení ochrany.

13.2 Techniky blokování útoků využívající přetečení bufferu

"Každý netriviální program obsahuje chyby."

Tato část popisuje některé z nejdůležitějších technik pro detekci a ochranu před útoky využívající přetečení bufferu a související exploity v počítačových systémech. Některé z nich jsou teprve ve fázi výzkumu, jiné se už používají. Tyto procedury pomáhají při ochraně před infekcemi rychle se šířícími počítačovými červy.

Pokud pomineme komplexnost a typ přetečení, v zásadě není rozdíl mezi technikou přetečení internetového červa Morris[7] a dnes používanými pokročilejšími útoky (jako Linux/Slapper[2]). Tito červi jsou totiž založeny na stejných myšlenkách, které souvisejí s přetečením zásobníku nebo struktur heapu. Mohou být rozděleny do několika hlavních kategorií.

Většinu červů na systémech BSD a UNIX, jako jsou třeba Morris, Linux/Slapper, BSD/Scalper nebo Solaris/Sadmind, můžeme označit mezi červy založené na kódu shellu (shellcode).

Kód shellu je krátká posloupnost kódu, která běží v interpretu příkazů (shellu) na vzdáleném systému. Jedná se například o /bin/sh na UNIXu nebo cmd.exe na Windows. Komunita hackerů si mezi sebou vyměňuje tyto kódy, které jsou určeny pro mnoho operačních systémů. Někteří hackeři je pak používají pro útoky pomocí přetečení, eventuálně je upravují pro tento typ útoku. Poté, co shell spustí na vzdáleném stroji, může se do systému nakopírovat červ, který pak nad ním získá úplnou kontrolu. Hackeři používají tuto techniku pro "získání vlastnictví" nad cizím počítačem.

Jiné třídy červů, například W32/CodeRed, tuto techniku nepoužívají. Místo toho ukradnout (hijack) thread některé vadné aplikace a pomocí kódu injektovaného za běhu se spustí jako součást exploitované služby hostitele. Techniky představené v této kapitole poskytují ochranu jak proti shell kódům, tak i proti útokům pomocí injektáže kódu za běhu.

Zde existuje jedna významná skupina útoků, označovaná jako "návrat k LIBC" (return-to-LIBC). V takovém případě se útočník snaží přesměrovat návrat do standardního existujícího kódu v systému (například run-time jazyka C nebo API operačního systému). Útočník se snaží dosáhnout přetečení zásobníku takovým způsobem, aby mohl pomocí instrukce `ret` přesměrovat tok vykonávání do volání zamýšleného API s parametry, které si sám stanoví (zásobník je přepsán zvolenými parametry a stejně tak i "návratovou adresou", která je právě adresou zamýšleného volání API).

Tímto způsobem není prováděn kód v zásobníku, a ani na heapu. To je důležité, protože některé techniky ochrany proti přetečení vyžadují kontrolu kódu, který neběží tam, kde by měl – tedy právě v zásobníku, nebo na hromadě. Z tohoto důvodu je tato třída útoků vůči zmíněným technikám ochrany imunní.

Přestože existující červi tuto techniku nepoužívají, dá se očekávat, že ji v budoucnu využívat začnou. V očekávání vzniku takových červů jsem strávil mnoho času popisováním technik proti útokům typu return-to-LIBC.

13.2.1 Přezkoumání kódu

Nejefektivnější metodou ochrany proti útokům pomocí přetečení bufferu je přezkoumání kódu, které provádějí bezpečnostní experti. Softwarové produkty mnoha společností jsou často vydávány s minimálním, nebo vůbec žádným přezkoumáním kódu, což vede k eventuálním bezpečnostním problémům.

Ale i když je toto přezkoumání prováděno, lidé občas nebývají dostatečně kvalifikováni k tomu, aby dokázali odhalit případné bezpečnostní problémy. Ve všech stádiích vývoje je potřeba si vytrénovat bezpečnostní profesionály. Programátory je potřeba neustále vzdělávat v otázkách bezpečnosti.

Prozkoumávání kódu je důležité především z toho důvodu, že ten, kdo má k dispozici zdrojový kód, má také nejlepší možnosti obrany. Nemůžeme však předpokládat, že vývojář dokáže detekovat všechny bezpečnostní problémy. Většinu takových nedostatků totiž odhalí lidé nezasvěcení do kódu – jako třeba profesionálové v oblasti bezpečnosti a hackeři. Dalším problémem je to, že bezpečnostní přezkoumání kódu se často soustředí na samotný kód, přičemž se opomine samotný návrh aplikace. To samo o sobě vede k závažným problémům se zranitelností.

13.2.1.1 Bezpečnostní aktualizace

Mnoho bezpečnostních profesionálů věří, že publikování zranitelných míst nějakého produktu přinutí danou společnost, aby rychle vytvořila příslušné záplaty. Nicméně – pokud je záplata k dispozici, zákazníci je často zapomínají použít, a to až do té doby, dokud nejsou taková zranitelná místa zneužita proti nim. Důvody pro malé používání bezpečnostních aktualizací jsou následující:

- Lidé neví, že bezpečnostní záplaty existují, nebo je nechtějí používat.
- Ve velkých společnostech je často nákladné je implementovat.
- Záplaty občas neopraví všechny bezpečnostní nedostatky.
- Občas způsobují havárie nebo nekompatibilitu se stávajícími programy.

Používání aktualizací je nejefektivnějším typem ochrany proti některým bezpečnostním rizikům. Opomíjení jejich používání není dobrou praxí, přestože aktualizace na některých systémech občas působí

problémy. Dobrým příkladem je Microsoft Security Bulletin MS03-007, který mnoho lidí nepřesně zná pod označením "zranitelnost WebDav". Jedna z tehdejších chyb přetečení bufferu byla umístěna v okruhu oprávnění 3 (ring 3), v uživatelském režimu. Bylo potřeba opravit funkci run-time knihovny (RTL) nativního API modulu NTDLL.DLL. Kromě toho existovala v jádře zranitelnost ohledně přetečení celých čísla (integer).

Protože původní škodlivý exploit pracoval se součástí WebDav systému IIS, mnoho bezpečnostních profesionálů si myslelo, že k obraně proti možnému útoků postačí, když se WebDav zakáže. Záplata, kterou poskytoval Microsoft, nahrazovala soubor NTDLL.DLL novou verzí, což bylo chápáno jako příliš velký zásah, který by mohl na mnoha systémech způsobit komplikace. Díky tomu se pak spousta lidí příslušnou aktualizaci vůbec nezabývala, přičemž se spokojili s pouhým zakázáním součásti WebDav. Mnoho systémů tak bylo ponecháno bez odpovídající ochrany.

Důležitým poznatkem je tedy to, že mohou být zneužity zranitelnosti v jednotlivých aplikacích. Je-li však chyba v některé sdílené komponentě, například v nějaké součásti operačního systému, jsou napadnutelné i všechny aplikace, které tuto komponentu využívají. Když se objeví zranitelnost některé aplikace, neznamená to ještě, že ostatní aplikace jsou bezpečné. Zakázání aplikace v tomto případě znamená zakrytí skutečného problému. Tato situace je ještě horší, objeví-li se zranitelnosti ve staticky linkovaných knihovnách, jako například zlib nebo openssl. Toto může způsobit zranitelnost celého systému. Mnoho výrobců software si bohužel neuvědomuje, že jsou jejich produkty zranitelné nebo tuto skutečnost rovnou zanedbávají a žádné bezpečnostní aktualizace neposkytují.

Kvůli ochraně zranitelného software proti možným útokům musíme důsledně dotáhnout do konce každou fázi jeho vývoje. K ochraně je potřeba přistupovat na všech úrovních – od zdrojových kódů až po ochranu aplikace za běhu. Přitom je potřeba porozumět všem možnostem i omezením jednotlivých technik ochrany.

13.2.2 Řešení na úrovni kompilátoru

Po nějakou dobu používali programátoři specifický software pro zjišťování bezpečnostních nedostatků aplikace, jako například BoundsChecker. Ten jim pomohl najít mnoho existujících přetečení a dalších problémů s kvalitou programů. Jak se však útoky pomocí přetečení bufferu stávaly stále populárnějšími a úspěšnějšími, začali bezpečnostní profesionálové uvažovat o lepším řešení na úrovni kompilátoru.

Programovací jazyky C a C++ nabízejí velké možnosti všem typům útoků založeným na přetečení bufferu. Protože je kód, zapsaný v těchto jazycích, obzvláště zranitelný, musí programátoři využívat různých technik spojených s kompilátory.

Taková řešení nás samozřejmě nemohou zbavit potřeby provádět bezpečnostní prozkoumání kódu. Techniky na úrovni kompilátorů jsou prvotní ochranou proti většině známých typů útoků pomocí přetečení zásobníku. Většina z nich ovšem proti takovým útokům neposkytuje naprosto stoprocentní ochranu – zpravidla ani neposkytují ochranu proti přetečení, které se týká heapu. Tato kapitola uvádí několik jednoduchých příkladů toho, proč jsou některé systémy zranitelné pomocí útoků na zásobník (stack), i když jsou označeny jako chráněné.

Měli bychom si však uvědomovat, že čím více technik je nasazeno k zajištění ochrany, tím větší množství znalostí a schopností je potřeba k jejich obejití. A to zdaleka všichni potencionální útočníci nemají. Ne-

hledě na fakt, že útočníci, kteří jsou vybaveni dostatečnými znalostmi, potřebují k provedení úspěšného útoku také více času. Na straně útočníka jsou nicméně následující výhody:

- Mají přístup ke zkompilovanému kódu; v případě open-source i ke kódu zdrojovému.
- Mají většinou dostatek času.
- Obtížnost různých exploitů. Některé zranitelnosti jsou útočníky snadno zneužity, zatímco práce na jiných trvá měsíce. Složitost ochrany proti nim se však nemění. Je stejně obtížná a vůbec to nezáleží na tom, jak snadno je možné konkrétní zranitelnost využít. U některých projektů může být obtížná (a také extrémně drahá) změna pouhých dvou řádků kódu.
- I když některé exploity vyžadují naprostou preciznost, exploity na jiných systémech už nemusí po útočnících vyžadovat takovou přesnost.

13.2.2.1 StackGuarg

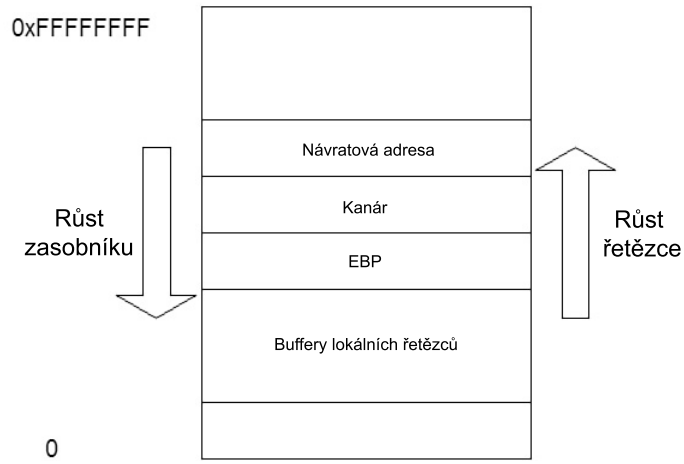
StackGuard byl uveden v roce 19988 jako jedno z prvních rozšíření kompilátorů k ochraně proti některým typům útoků přetečením zásobníku. Byl vytvořen jako rozšíření překladače gcc. Šikovným způsobem používá detekci modifikované návratové adresy pomocí techniky "kanára" ("canary"). Většina přetečení zásobníku spočívá v přetečení bufferu, umístěného poblíž návratové adresy funkce na zásobníku. Chybějící "kontrola ohraničení" (bounds check) umožňuje přetečení bufferu pomocí řetězce velké délky a tudíž i manipulaci funkce návratové hodnoty na zásobníku. Takový útok se nazývá jako *stack smashing*[9].

Když se funkce vrací k volajícímu (caller), vezme si adresu, kterou tam předtím vložil útočník. StackGuard se takovým útokům brání tak, že vedle návratové adresy na zásobník vloží varovnou hodnotu (tzv "kanára" – viz obrázek 13.2).

StackGuard je jednoduchou záplatou pro function_prologue a function_epilogue v gcc. Rozšířením function_prologue o tuto varovnou hodnotu (kanára) a function_epilogue pro její ověřování, může být za běhu programu detekována úprava této hodnoty. Pokud je hodnota změněna, rutina epilogu provede "handler mrtvého kanára" (canary-death-handler) místo návratu funkce. Pokud je tedy útok detekován, kód útočníka nemá možnost se spustit.

Existuje ovšem pár věcí, na které se implementace StackGuardu verze 2.x nezaměřuje – některé z nich bude pokrývat StackGuard 3. Neposkytuje například ochranu proti útokům na ukazatele rámce (frame pointer, EBD). Kanár je totiž umístěn za návratovou adresou a přetečení ukazatele rámce tak nemusí být detekováno. Ke změně ukazatele rámce není potřeba změnit hodnotu kanára.

StackGuard je dále zranitelný vůči útokům, které jsou vedeny proti ukazatelům funkcí mezi lokálními proměnnými. Zůstává však skutečností, že StackGuard dokáže efektivně blokovat mnoho internetových červů (jako například červa Morris). Je to možné ovšem pouze v případě, že aplikace, která obsahuje zranitelný kód (například fingerd), byla se StackGuardem zkompilována.



Obrázek 13.2 *StackGuard umístí "kanára" na zásobník pod "návratovou adresu".*

Červ Morris používal útok založený na kódu shellu, přičemž pro spuštění tohoto kódu modifikoval na zásobníku návratovou adresu funkce main(). Kód shellu zde byl vložen jako "řetězec" prostřednictvím zranitelné služby fingerd[10].

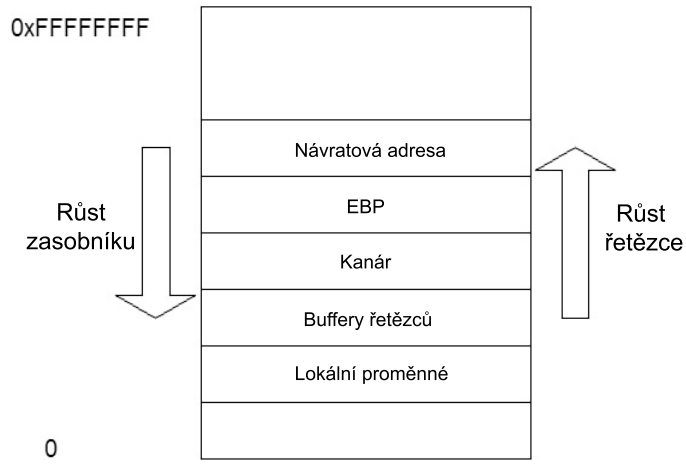
Rekompilace zranitelných služeb se StackGuardem může ochránit před linuxovými červy, kteří používají jednoduché útoky na zásobník. Červi jako Linux/Slapper používají přetečení na heapu, kterým StackGuard nemůže zabránit. Je však důležité poznamenat, že v současných počítačových červech není přetečení hromady obecně používanou technikou – většina červů používá jednoduchou techniku přetečení zásobníku.

Používání StackGuardu je silně doporučeno. Kompilace StackGuardu pro Linuxu jsou dostupné a rekompilace se StackGuardem činí systém mnohem více bezpečnějším.

Pro Microsoft Visual C++ .NET 2003 7.0 byla vyvinuta technika fungující podobným způsobem jako StackGuard. Ve verzi 7.1 došlo ke změnám a nově používaná metoda je spíše podobná technologii ProPolice.

13.2.2.2 ProPolice

ProPolice vyvinul výzkumník IBM Hiroaki Etoh[12]. Uvádí mnoho nových vlastností, založených na StackGuardu. A podobně jako on poskytuje ochranu proti přetečení bufferu na úrovni kompilátoru. Jeho nové metody umožňují přesun hodnoty kanára a optimalizaci umístění bufferů a ukazatele funkcí v zásobníku, takže pokusy o exploitaci ukazatele funkcí jsou mnohem komplikovanější. Ilustrace je uvedena na obrázku 13.3.



Obrázek 13.3 *"Kanár" ProPolice pod ukazatelem rámce a "návratovou adresou".*

ProPolice standardně chrání jak ukazatel rámce, tak i návratovou adresu pomocí metody, kdy je pod ukazatelem rámce umístěna hodnota kanára. Dále také spojí buffery řetězců a umístí je nad lokální proměnné, čímž poskytne lepší ochranu ukazatelům funkcí, které jsou lokálními proměnnými.

ProPolice se také pokouší o vytváření lokálních kopií vložených ukazatelů funkcí, optimalizace kompilátoru zde však může způsobit určité problémy. Další vlastnosti zahrnují ukazatele funkcí ve strukturách, které jsou předány jako parametry, a které obsahují řetězcové buffery.

Podobně jako StackGuard, i ProPolice si našel své místo při tvorbě operačních systémů. V současné době je k dispozici ve verzi 3.3 systému OpenBSD, čímž jej činí mnohem odolnějším proti útokům. ProPolice zajišťuje, že útoky pomocí přetečení zásobníku jsou mnohem komplikovanější a měl by tak nabízet velkou výzvu i velmi schopným útočníkům.

Protože ProPolice zajišťuje integritu zásobníku, neposkytuje žádnou ochranu proti útokům na struktury heapu (hromady)[13], čehož využívá například červ Linux/Slapper[2].

13.2.2.3 Microsoft Visual Studio .NET 2003: 7.0 a 7.1

Microsoft uvedl volbu /GS poprvé ve Visual Studiu .NET 2003. Tato nová volba se nazývá jako "Kontrola bezpečnosti bufferů" (Buffer Security Check), a je přístupná jako volba při generování kódu. Implicitně je zapnutá.

Uvažujme chybný kód v jazyce C ve výpisu 13.1.

Výpis 13.1

Chybný kód v jazyce C.

```
int Bogus(char *mystring)
{
char buf[8];
```

```
strcpy(buf, mystring);      // pozor!
return 0;
}

void main(void)
{
Bogus("Toto je typické přetečení zásobníku!");
}
```

Kompilátor primárně chrání pole, která jsou nejméně pět bajtů dlouhá. Pro kratší buffery není kód pro testování bezpečnosti generován. Je to zřejmě takový bezpečnostní kompromis – předpokládá se, že k přetečení obvykle dochází až v rozsáhlejších bufferech. Jestliže se však útočníkovi podaří vložit svůj kód do chybné funkce, může být funkce exploitována nezávisle na tom, jak je buffer krátký.

Podívejme se nyní na kód, který generuje VC .NET 2003 7.0:

```
00401296 push offset string "Toto je typické přetečení zásobníku!"
0040129B call Bogus (401000h)
```

Prostřednictvím zásobníku jsme vložili do funkce Bogus() ukazatel na dlouhý řetězec. Výpis 13.2 ukazuje, co se děje uvnitř této funkce.

Výpis 13.2

Nastavení "Security Cookie".

```
Bogus:
00401000 sub esp,0Ch
00401003 mov eax,dword ptr [___security_cookie (407030h)]
00401008 xor eax,dword ptr [esp+0Ch]
0040100C lea edx,[esp]
00401010 mov dword ptr [esp+8],eax
```

Funkce Bogus() nejprve zpřístupní hodnotu security_cookie, která je náhodně vygenerovaná pomocí CRT. Speciální rutina CRT tuto hodnotu inicializuje jako náhodné číslo DWORD. Důvod je jednoduchý – může-li útočník uhodnout hodnotu security_cookie, bude schopen způsobit přetečení vložením "falešné" hodnoty security_cookie, což kontrola bezpečnosti bufferu nedetekuje. Takový útok je uskutečnitelný, pokud by útočník obešel tuto kontrolu bezpečnosti, přepsal předchozí rámec nad zásobníkem, pomocí ukazatele funkce spustil svůj kód a nakonec, kvůli utajení, opravil obsah zásobníku.

Hodnota security_cookie je nejprve "XORována" s aktuální návratovou adresou a poté je uložena za návratovou adresou na zásobníku jako cookie. Potom provede chybné (buggy) kopírování, například pomocí funkce strcpy(), viz výpis 13.3.

Výpis 13.3

Podmínka možného přetečení.

```
00401014 mov eax,dword ptr [esp+10h]
00401018 sub edx,eax
0040101A lea ebx,[ebx]
00401020 mov cl,byte ptr [eax]
00401022 mov byte ptr [edx+eax],cl
00401025 inc eax
00401026 test cl,cl
00401028 jne Bogus+20h (401020h)
```

A nakonec epilogová rutina funkce Bogus() vezme uloženou hodnotu cookie a "dekóduje" ji do registru "ecx", což je ukázáno ve výpisu 13.4.

Výpis 13.4

Dekódování "Security cookie".

```
0040102A mov ecx,dword ptr [esp+8]
0040102E xor eax,eax
00401030 xor ecx,dword ptr [esp+0Ch]
00401034 add esp,0Ch
00401037 jmp __security_check_cookie (4013F1h)
```

Další skok epilogu vede do runtime části C, definovaného v setcook.c uvnitř zdrojového kódu CRT. To je ukázáno ve výpisu 13.5.

Výpis 13.5

Standardní "bezpečnostní" handler.

```
void __declspec(naked) __fastcall __security_check_cookie(DWORD_PTR cookie)
{
/* verze x86 zapsaná v assembleru kvůli ochraně všech registrů */
__asm {
cmp ecx, __security_cookie
jne failure
ret
failure:
jmp report_failure
}
}
```

Porovnání se tedy provede vůči původní bezpečnostní hodnotě cookie. Je-li detekován rozdíl, pokračuje kód na adrese report_failure. Pokud předtím nebyl nastaven žádný user_handler, provede se pouze stan-

dardní výpis. Nastavení uživatelského ovladače (user handler) umožňuje zajistit jinou funkcionalitu, než tu, kterou poskytuje výchozí metoda. Adresa uživatelského ovladače (user_handler) je ukazatelem na funkci, který je umístěný v datové sekci, takže v některých případech bude možné jeho přepsání pomocí přetečení. To umožní útočníkovi spustit jeho vlastní kód pomocí tohoto handleru.

Pokud není user_handler nastaven pomocí funkce _set_security_error_handler(), je přetečení zásobníku ohlášeno uživateli a vykonávání programu je ukončeno.

Hodnota cookie je umístěna pod ukazatel rámce, pokud ovšem nějaký existuje. Tímto způsobem může kontrola reagovat na útoky proti tomuto ukazateli.

V kompilátoru verze 7.1 Microsoft zásadním způsobem vylepšil kontrolu bezpečnosti bufferu. Hodnota cookie již není XORována s návratovou adresou, což původně nepřinášelo žádnou viditelnou výhodu. Místo toho je tato cookie uložena a ověřována. Některá problémy zde popsané však vyřešeny nebyly.

Nejdůležitějším rysem edice 7.1 je to, že buffery řetězců jsou spojeny dohromady. Ukazatele funkcí a ostatní lokální proměnné jsou pak umísťovány kompilátorem pod buffery na zásobníku. Implementace Microsoftu ohledně kontroly integrity zásobníku tak následuje nejdůležitější vlastnosti ProPolice.

A podobně jako v případě ProPolice, i bezpečnostní volby Microsoft Visual Studia .NET 2003 způsobují konflikty s možnostmi optimalizace kompilátoru. V optimalizovaném kódu tak mohou vložené ukazatele funkcí sloužit jaké přímé reference do předchozího rámce zásobníku. Takové ukazatele funkcí ovšem mohou být přepsány a exploitovány ještě předtím, než se provede bezpečnostní kontrola, která se neprovádí před návratem funkce. Vnořená volání, která jako parametry používají poškozené ukazatele funkcí (prostřednictvím přímých referencí do volajícího rámce zásobníku), jsou tak kvůli těmto poškozením zranitelná.

Jednou z uvažovaných alternativ je použití direktiv "pragma" vedoucí k vypnutí optimalizace pro některé části kódu (například pro sekce, které předávají ukazatele funkcí). To je však praxe, která dává prostor vzniku dalších problémů, jako je mazání "tajných míst v paměti" (odstraňování dočasných klíčů) na posledním řádku funkce, kterou může chytrá optimalizace eliminovat jako mrtvý kód. Je tomu z toho důvodu, že daná proměnná se po dosažení konce funkce jeví jako nevyužitá.

Zbývající výzvou je pak standardní ošetřování výjimek ve Windows. Když je vyvolána výjimka, prochází se řetěz ovladačů pro nalezení aktivního ovladače výjimky, který bude vyvolán. Mnoho obecných typů exploitů Windows je založeno na tom, že se přepíšou rámce ovladačů výjimek založené na zásobníku, což vede ke spuštění kódu útočníka. Tuto techniku používá několik současných virů, stejně jako červ W32/CodeRed.

Kontrola bezpečnosti bufferu sama o sobě tyto problémy nezmírňuje. Alternativou, která funguje proti těmto útokům, byla vyvinuta společností Symantec. Více informací je uvedeno v části 13.3, která popisuje techniky blokování červů. Je vhodné poznamenat, že pro Visual Studio 2005 Microsoft plánuje provést několik změn v implementaci přepínače /GS. Ty by se měly zaměřit především na nedostatky, popsané v této kapitole.

13.2.3 Řešení na úrovni operačního systému a rozšíření run-time

Kontrola integrity zásobníku pomocí kompilátoru je pouze jednou z mnoha možností ochrany proti přetečení na úrovni operačního systému. I když jsou rekompilované systémové komponenty (ať už samotného operačního systému nebo třetích stran) hůře zranitelné útoky založenými na zásobníku, nechráněné komponenty (ať už systémové nebo jakékoliv jiné) způsobují to, že systém zůstává stále napadnutelný.

Ačkoli většina procesorů Intel neposkytuje ochranu na úrovni stránkování proti operacím vykonávaným na zásobníku, některé procesory ji poskytují, přičemž operační systémy mohou takové ochrany využívat (alternativy k systémům Intel jsou podrobněji popsány později).

K nejzávažnějším problémům ochrany na úrovni překladače patří to, že pro kompilaci je potřebný zdrojový kód. Během několika posledních let se sice objevila nová řešení, která zdrojový kód nevyžadují, nicméně se vztahují ke specifickým procesorům (například Intel) nebo ke specifickým operačním systémům (jako je například Linux). Následující část pak rozebírá některá z nejvýznamnějších rozšíření takových systémů.

13.2.3.1 Solaris na systému SPARC

Mnoho operačních systémů má v sobě zabudované mechanismy ochrany proti některým typům útoků pomocí přetečení bufferu. Například systémy Solaris mohou chráněny pouhou změnou nastavení systému v souboru /etc/system. Systém se tak může, pomocí zásobníku na procesoru SPARC, bránit proti útokům založených na přetečení bufferu. Ilustrace je na obrázku 13.4.

```
set noexec_user_stack=1
set noexec_user_stack_log=1
```

Obrázek 13.4 *Nastavení konfigurace Solarisu na SPARCu.*

Důsledkem této změny nastavení systému je to, že uživatelský prostor zásobníku procesů Solarisu nebude mapován jako vykonatelný (exec), takže pokus o vykonání obsahu zásobníku povede k násilnému ukončení a k výpisu obsahu procesu (core dump). Tento výpis bude také součástí systémového logu (pokud je to nakonfigurováno). Situaci ilustruje obrázek 13.5.

```
#pmap 653
653: /sbin/sh
00010000 272K read/exec            /sbin/sh
00062000  16K read/write/exec      /sbin/sh
00066000  24K read/write/exec      [ heap ]
FFBEE000   8K read/write           [ stack ]
     total 320K
```

Obrázek 13.5 *Uživatelský zásobník procesu "sh" není označen jako vykonatelný ("exec").*

Některé systémy ochrany se pokoušely dosáhnout podobných výsledků použitím vykonatelných a datových segmentů pro procesory od Intelu (viz příklady v části 13.2.5). I toto řešení preventivně zabraňuje vykonání kódu na zásobníku.

Přestože jsou tato řešení atraktivní, je důležité mít na paměti, že existují nebezpečí přetečení, která nevyžadují provádění kódu na zásobníku – například přetečení heapu nebo útoky return-to-LIBC.

Právě v těchto případech mohou pomoci techniky založené na kompilátorech. Tyto technologie – StackGuard, ProPolice, nebo kontrola bezpečnosti bufferů od Microsoftu – se snaží potlačit možné zneužití prostřednictvím změny návratové adresy a ukazatele rámce. Některé z nich také znesnadňují zneužití ukazatelů funkcí. Je tedy vhodné poznamenat, že tyto systémy se navzájem dobře doplňují. Je také zřejmé, že pro zmírnění dalších negativ by měly být použity jiné techniky.

13.2.4 Rozšíření subsystému – Libsafe

Některá řešení přidávají mechanismy ochrany do uživatelského adresovacího prostoru jednotlivých aplikací. Libsafe[14] je run-time ochrana dostupná v systému Linux. Chrání proti napadení návratových adres, stejně jako proti útokům na rámec zásobníku. Není však schopna ochránit procesy, které mezi voláním funkcí nepoužívají ukazatele rámců na zásobníku. V takových případech Libsafe jednoduše nechává aplikace, aby si dělaly, co chtějí.

Libsafe využívá jednu ze standardních vlastností Linuxu, která umožňuje "přetížit" některé funkce v dynamicky linkovaných knihovnách. Nahrává se jako dynamická knihovna a do adresovacího prostoru procesu zavádí jména funkcí, jako jsou memcpy() nebo strcpy(). Pokud je zavedena knihovna GLIBC (standardní run-time knihovna jazyka C v Linuxu), jsou tyto funkce již známy, takže se nepoužijí jejich verze v GLIBC, ale verze obsažené v Libsafe. Když aplikace zavolá strcpy(), bude nejdříve volat Libsafe.

Libsafe trasuje zásobník použitím ukazatelů na rámce ze struktury zásobníku. Poté použije svoji logiku k validaci parametrů a k rozhodnutí, zdali je parametr příliš dlouhý a zdali je schopný přepsat místo uložení ukazatele rámce nebo návratové adresy. Pokud ano, Libsafe okamžitě zastaví běh procesu. Jinak se dynamicky přepne do GLIBC a zavolá původní funkci.

V současné době zajišťuje Libsafe ochranu těmto funkcím – memcpy(), strcpy(), strncpy(), wcscpy(), stpcpy(), wcpcpy(), strcat(), strncat(), wcscat(), [v]sprintf(),[v]snprintf(), vprintf(), vfprintf(), getwd(), gets() a realpath(). Nejedná se sice o vyčerpávající seznam "zranitelných" funkcí, nicméně určitě obsahuje některé z nejobvyklejších zranitelností v kódu C.

Libsafe 2.0 tak chrání nejvíce využívanou skupinu "zranitelných" funkcí před útoky založenými na napadání zásobníku. Chrání také funkce, které mohou být využity pro vykonávání exploitů založených na formátovacích řetězcích[10].

13.2.5 Rozšíření režimu jádra

Spousta rozšíření (extensions) režimu jádra se snaží pracovat s velkým množstvím různých útoků. Taková řešení však vykazují mnoho nedostatků, nehledě na fakt, že nějaké rozšíření v režimu jádra může snadno ovlivnit stabilitu systému, což platí i o technologii, která byla poprvé použita pod názvem PaX[15] pro různé open-source systémy, která zahrnuje přímou manipulaci příznaků stránek v tabulkách strá-

nek. V systémech s uzavřenými zdroji (closed-source), jsou problémy s podobnými rozšířeními ještě více markantnější.

PaX a jeho další implementace, SecureStack[16], nastavují v příznacích stránky "Supervisor" bit takovým způsobem, aby byl způsoben výpadek stránky (page-fault). Ten pak využívá ovladač produktu v případě, že jsou dané stránky zpřístupňovány v uživatelském režimu. Tímto způsobem je možné rozlišit, jestli se instrukční ukazatel odkazuje na zapisovatelnou stránku na zásobníku nebo na heapu.

Implementace využívá důmyslnou techniku, která minimalizuje možné snížení výkonu tak, že k výpadkům stránek dochází při provádění instrukcí, a nikoliv během přístupu k datům. Tato technika udržuje snížení výkonu pod hranicí 5%.

Podstata této techniky spočívá ve využití bufferů TLB (translation look-aside) procesorů Intel. Na 32bitové architektuře Intel popisuje jedna položka stránky tabulky (PTE) každou stránku paměti o velikost i 4kB. PTE popisuje informace o umístění stránky a pomocí několika různých atributů i její dosažitelnost. Jedním z PTE příznaků je také bit "Supervisor". Je-li tento bit nastaven v PTE, která přísluší určité stránce, pak pokus o přístup k ní, v uživatelském režimu, vyvolá výjimku. Dále pak provede ovladač produktu (product's driver), který je nastaven ke zpracování výjimek v režimu jádra, kontrolu bezpečnosti. PaX a SecureStack tento bit nastavují pro některé stránky v uživatelském režimu, jako například zapisovatelné stránky nebo oblasti zásobníku.

Pro podstatu této techniky je klíčová skutečnost, že Pentium a následující procesory mají dva TLB – jeden pro přístup k datům (DTLB), a druhý pro instrukce (ITLB). Výpadky stránek jsou minimalizovány tak, že bit "Supervisor" je nastavován pouze ITLB kopii PTE, nikoli v DTLB[16]. Provádění kódu v zapisovatelných stránkách prostřednictvím ITLB tak může být snadno detekováno a následně znemožněno. Důležitou vlastností této techniky je to, že blokuje provádění kódu na zásobníku a na heapu.

Provádění kódu v zapisovatelných stránkách je však bohužel běžné (typicky na systémech Windows, ale nejenom na nich). Dobrým příkladem této skutečnosti jsou například samorozbalovací archivy. Při legitimizaci pokusů o běh zapisovatelných stránek hlásí systém falešná pozitiva (false positive).

Provádění kódu v zapisovatelných stránkách však naštěstí není běžné na serverových platformách. Pro zmírnění problémů s falešnými pozitivy nabízí PaX nástroj pro označení aplikací jako přátelských. Některé další specifické systémy mohou tento problém dále zmírňovat.

PaX na platformě Intel implementuje ještě jeden mechanismus ochrany před prováděním kódu na zásobníku. Ten je založen na segmentaci adresovacího prostoru procesu a na přístupových právech samotných segmentů. Výhodou této segmentace je to, že nedochází ke ztrátě výkonu. Toto řešení však vyžaduje těsnou spolupráci se samotným operačním systémem, což vede k obtížím při vývoji na platformy, které nemají otevřený kód (open-source).

Významným přínosem je však stabilita. Tento druh řešení je nezávislý nejenom na procesoru, ale také na verzi operačního systému (což zahrnuje i nezávislost na použité verzi service packu). Tyto techniky poskytují prostředky k ochraně proti široké skupině útoků v uživatelském režimu, které jsou nejvíce rozšířené. Nemusí však nezbytně nutně poskytovat ochranu proti přetečení v režimu jádra (na úrovni 0, ring 0), takže takové systémy pak nejsou odolné vůči zranitelnostem v operačním systému nebo v ovla-

dačích třetích stran, u kterých může škodlivý vstup vést k nepříjemným postranním efektům (novější verze PaX mají speciální ochranu pro stránky jádra).

Útočník může ovšem překonat ochranu proti provádění kódu na zásobníku a heapu pomocí útoku typu "return-to-LIBC", který popisován dříve. Tyto problémy však mohou být zmírněny dalšími technikami, které jsou stručně popsány v části 13.3 věnované technikám blokování červů.

13.2.6 Doprovázení programů

Ve výzkumných zprávách MIT[17] byla jednou diskutována zajímavá technika se slibnými výsledky. Tato nová technika se nazývá doprovázení programů (program shepherding).

Doprovázení programů je založeno na použití dynamického optimalizátoru, nazvaného jako DynamoRIO. Cílem RIO je rychlé provádění kódu a jeho optimalizace bez toho, aby byla vyžadována jeho rekompilace. Tento projekt byl založen na spolupráci mezi Hewlett-Packard a Massachusetts Institute of Technology (MIT)[18]. Technika doprovázení programů byla založena na tomto modelu a je výhodná jak z hlediska rychlejšího provádění kódu, tak i z hlediska implementace ověřování toku instrukcí. To je zajištěno implementací vyrovnávací paměti (cache) kódu, do které je kód programu po částech kopírován, a ověřováním tohoto programového kódu před jeho vykonáním. Systém tedy nikdy nevykonává skutečný kód, ale pouze jeho kopii, a to takovým způsobem, že využívá skutečné CPU systému, a nikoliv jeho emulaci. Některé části programu jsou při jeho běhu modifikovány, přičemž je možné tímto způsobem kód ovládat. To umožňuje bezpečné provádění aplikací.

K tomu, aby se rozeznaly některé techniky exploitace, základní systém potřebuje některá rozšíření. Zvláště obtížným problémem je detekce změny toku kódu, která se objeví jako důsledek změny dat v adresovacím prostoru procesu. Například mohou být modifikována místa, jako jsou GOT (globální tabulka offsetů) v Unixu nebo IAT (adresovací tabulka importů) ve Windows – takto lze dosáhnout změn v toku kódu, které je velmi těžké detekovat verifikací toku kódu v cache.

13.3 Techniky blokování červů

Tato část popisuje techniky, které byly vyvinuty a implementovány firmou Symantec jako alternativní řešení k potlačení prvních a druhých generací exploitů používaných červy. Zde předpokládáme, že většina červů raději zaútočí na zranitelné systémy (tedy takové, které nejsou nijak zabezpečeny proti přetečení), protože je jich mnohem více, než systémů dostatečně zajištěných.

Z hlediska útočníka je neefektivní používat při exploitacích náročné techniky, protože celkový výsledek útoku může být označen za úspěšný i bez nich. Tento závěr vzešel z revize několika posledních červů, jako jsou Linux/Slapper a W32/Slammer, které byly zodpovědné za většinu hromadných infekcí v poslední době.

Techniky popsané v této části mohou takové útoky efektivně zastavit; uvedené principy jsou však pouze demonstrační a jejich účelem je ukázat, jak moc efektivní mohou tato řešení být. Nejedná se o ucelenou skupinu myšlenek, ale spíše o ukázku spolehlivých pravidel, která mohou být dostatečně efektivní vůči rychle se šířícím počítačovým červům. Použití takových pravidel může být součástí většího systému pro řízení přístupu, případně mohou být zkombinována s podobnými systémy.