ОЖҚ. Дәріс №11. Программалық жасақтаманы бұзу. Буфердің толып кетуін пайдаланатын шабуылдар

Тұтынушы компьютерін бұзудың негізгі әдістерінің бірі жүйеде іске қосылған программалық жасақтамадағы осалдықтарды программашы көздеген мақсаттан өзгеше жасап, пайдалануға негізделген. Мысалы, тұтынушы браузеріне жасырын – **drive-by-download** арқылы жұқтыруға бағытталған шабуыл жиі жасалады. Осы шабуылды жүргізу арқылы киберқылмыскер зиянды мазмұнды веб-серверге қою арқылы Тұтынушы браузерін зиянды кодпен толтырады. Тұтынушы веб-сайтқа кіргеннен кейін браузер жұқтырады. Кейде веб-серверлер хакерлер үшін толығымен жұмыс істейді және олар адамдарды өз веб-сайттарына тартудың жолын табуы керек (мүмкін ақысыз программалар немесе фильмдер туралы уәделері бар спам жіберу арқылы). Бірақ бұзушылар зиянды мазмұнды заңды веб-сайтқа орналастыруы мүмкін (мысалы, жарнамада немесе форумда). Жақында футбол Майами Дельфиндер командасының веб-сайты осылай бұзылды, Дельфиндер суперкубокты жеңіп алды, бұл жылдың көптен күткен оқиғаларының бірі болды. Осы оқиғадан бірнеше күн бұрын веб-сайт өте танымал болды және оған кірген көптеген тұтынушылардың компьютерлері зиянды кодты жұқтырды. Drive-by-download көмегімен бастапқы инфекциядан кейін бұзушының коды нағыз **зомби-программаны** (***зиянды кодты***) жүктейтін браузерде іске қосылғанда, осы программаны орындады және жүйені әрбір жүктеу кезінде оны тұрақты іске қосуды қамтамасыз етті.

Пәндік аймаққа байланысты, біздің назарымыз операциялық жүйеге зиян келтіру тәсілдеріне бағытталған. Веб-сайттар мен дерекқорларды бұзу программалық жасақтамасында осалдықтарды қолданудың көптеген жолдары қарастырылмайды. Drive-by-downloads шабуылы үлкен суреттің бөлігі емес, өйткені біз көптеген осалдықтар мен зиянды кодты қолданушы қосымшаларына енгізу ядроға да қатысты екенін көреміз.

Льюис Карроллдың әйгілі “Алиса в Зазеркалье” кітабында Қара патшайым Алисаны қашып кетуге мәжбүр етті. Олар жан дәрменімен жүгірді, бірақ әрқашан бір жерде қалып кетті. Элис бұл біртүрлі деп ойлап, ол туралы патшайымға айтты. Ал патшайым: "біздің елде өте тез және өте ұзақ жүгірсеңіз ғана басқа жерде бола аласыз, оны біз іс жүзінде жасадық. Сіз білесіз бе, тек орнында қалу үшін жан дәрменімен жүгіруге

тура келеді. Егер сіз басқа жерге баруыңыз керек болса, сіз кем дегенде екі есе жылдам жүгіруіңіз керек!». Қара патшайымның әсері түрлердің эволюциялық күресіне тән. Миллиондаған жылдар бойы ата-бабалар мен зебралар мен арыстандар дамыды. Зебралар тезірек жүгіре бастады, көру, есту және иіс сезімі күшейе түсті, бұл оларға арыстандардан қашуға мүмкіндік берді. Бірақ уақыт өте келе арыстандар да тезірек жүгіре бастады, олар үлкен, түсініксіз және камуфляж түсіне ие болды, бұл оларға зебраларды сәтті аулауға мүмкіндік берді.Олар орнында қалу үшін жүгіреді. Қара патшайымның әсері программаларды қолдануға да қатысты. Бұзушылар үнемі жетілдіріліп отыратын қауіпсіздік шараларымен күресу үшін күрделене түсуде.

Әрбір зиянды код белгілі бір программада белгілі бір осалдықты пайдаланса да, үнемі қайталанатын осалдықтардың жалпы санаттары бар, олар бұзу тетіктерін түсіну үшін зерттеуге тұрарлық. Келесі бөлімдерде осындай тетіктердің бірнешеуі ғана емес, сонымен қатар оларды қолдануға кедергі келтіретін қарсы шаралар, тіпті мұндай трюктерге қарсы кейбір қарсы шаралар және т.б. сіз үшін қорғаныс құралдары мен шабуылдардың бәсекелестігі туралы жеткілікті түсінік аласыз және мұның бәрі қара патшаймен бірге жүгіру сияқты.

Компьютерлік қауіпсіздік тарихындағы ең танымал технологиялардың бірі-буфердің толып кетуінен бастайық. Ол 1988 жылы кіші-Роберт Моррис жазған алғашқы желілік құртта қолданылған және қазіргі уақытта кеңінен қолданылады. Барлық қарсы шараларға қарамастан, зерттеушілер буфердің толып кетуі бізбен біраз уақыт қалады деп болжайды (Ван дер Вейн,2012). Буферлік толып кетулер қазіргі заманғы жүйелердің көпшілігінде қол жетімді үш маңызды қорғаныс механизмін ұсынуға өте ыңғайлы: **стек индикаторы немесе "канарейки" (*stack canaries*), деректерді орындаудан қорғау (*data execution protection*)** және **адрестік кеңістікті үлестіруді рандомизациялау (*address-space layout randomization*)**. Осыдан кейін форматтаушы жолды (format string) пайдалану сияқты зиянды кодты енгізудің басқа технологиялары қарастырылады, **бүтін мәндердің толып кетуі (*integer overflows*)** және **жоқ объектілерге сілтемелер (*dangling pointer exploits*)**.

Буфердің толып кетуін қолданатын шабуылдар

Көптеген шабуылдардың негізінде барлық операциялық жүйелер C немесе C++ программалау тілінде жазылған (программашылар бұл тілдерді жақсы көретіндіктен, сонымен қатар олардағы программалар өте тиімді объект кодына қосылғандықтан). Өкінішке орай, C немесе C++ тілінің бірде-бір компиляторы массивтің шекараларын тексермейді. Мысалы, жолды (белгісіз өлшемді) белгіленген өлшемді буферге оқитын, бірақ толып кетуді тексермейтін C тілі кітапханасының ***gets*** функциясы мұндай шабуылдың құрбаны болып табылады (кейбір компиляторлар тіпті кодта ***gets*** функциясының болуы туралы ескертеді).

Тиісінше, программалық кодтың келесі бөлігі компилятормен тексерілмейді:

```
01. void A() {
02. char B[128]; /* резервирование в стеке буфера объемом 128 байт */
03. printf ("Type log message:");
04. gets (B); /* чтение сообщения со стандартного ввода в буфер */
05. writeLog (B); /* вывод строки в файл журнала */
06. }
```

A-функциясы – бұл не болып жатқанын жеңілдетілген түрде тіркеу процедурасы. Әр орындалған сайын Тұтынушы тіркеу хабарламасын енгізуге шақыру алады, содан кейін Тұтынушы терген мәтінді B-буферінде C тілінің кітапханасынан gets функциясын қолдана отырып оқиды және ақырында *writeLog* үй функциясы шақырылады, ол Тіркеу жазбасын тартымды форматта шығарады деп болжанады (жазбаны кейінірек іздеуді жеңілдету үшін тіркеу хабарламасына күн мен уақытты қосқанда). A функциясы артықшылықты үдерістің бөлігі болып табылады делік, мысалы, Тұтынушы идентификаторы ең жоғары деңгейлі құқықтармен (SETUID root) орнатылған программа. Бақылауды алуға қабілетті бұзушы негізінен, түбірлік артықшылықтарға ие.

Жоғарыда көрсетілген кодта бірнеше ақаулар бар, бірақ оларды бірден анықталынбайды. Мәселе ***gets*** функциясы жаңа жолдың символына сәйкес келмейінше стандартты енгізуден таңбаларды оқиды. Ол B буферінде тек 128 байт бар екенін білмейді. Тұтынушы 256 таңбадан тұратын жолды терді делік. Қалған 128 байтпен не болады? ***gets*** буфердің

шекараларын бұзу фактісін тексермегендіктен, қалған байттар да сақталады стекте буфердің ұзындығы 256 байт болатын сияқты. Бұл жад орындарында сақталғандардың бәрі жай ғана қайта жазылады. Мұның салдары әдетте апатты.

19-суретте жергілікті айнымалыларды стекте сақтайтын жұмыс істейтін негізгі программа көрсетілген. Бір сәтте ол суретте көрсетілгендей қоңыраудың стандартты реттілігі қайтару мекен-жайы стекіне орналастырудан басталады (нұсқаулықтан кейінгі нұсқауды көрсететін қоңырау). Содан кейін Басқару А процедурасына беріледі, ол жергілікті айнымалы үшін сақтау орнын бөлу үшін стек көрсеткішін 128-ге азайтады (В буфері).

Сонымен, егер тұтынушы 128 таңбадан көп енгізсе не болады? Бұл жағдай суретте көрсетілген.(9.19) В жоғарыда айтылғандай, gets функциясы барлық байттарды буферге және одан жоғары деңгейге көшіреді, мүмкін стек ішіндегі көп нәрсені қайта жазады, бірақ, атап айтқанда, стекке салынған қайтару мекенжайын қайта жазады.

19-сурет. Қалыптасқан жағдайлар: а – негізгі программа жұмыс істеген кезде; б – А процедурасын шақырғаннан кейін; в – сұр түспен көрсетілген буфер толып кеткен кезде

Басқаша айтқанда, тіркеу жазбасының бір бөлігі енді жадтағы орынды толтырады,онда жүйенің болжамына сәйкес нұсқаулықтың мекен-жайы болуы керек функциядан оралған кезде сөзсіз ауысу жүзеге асырылады.

Тұтынушы әдеттегі тіркеу хабарламасын тергендіктен, оның таңбалары дұрыс код мекенжайын орындай алмайды. Басқару А функциясынан оралғаннан кейін, программа дұрыс емес жерге сөзсіз ауысуды жүзеге асыруға тырысады, ал жүйе оны мүлдем ұнатпайды. Көп жағдайда программа дереу төтенше жағдайға көшеді.

Енді бұл өте ұзақ хабарламаны қате терген қарапайым қолданушы емес, программаның орындалу барысын бұзуға бағытталған "құйрықты" хабарлама жіберген бұзушы болсын.Айталық, оларға в буферінің басталу мекен-жайы қайтару мекен-жайының орнына жазылуы үшін мұқият тексерілген кіріс берілген. Нәтижесінде *А* функциясынан қайтарылған кезде программа В буферінің басына сөзсіз ауысуды жүзеге асырады және буферде байттарды код ретінде орындайды. Бұзушы буфердің мазмұнын басқаратындықтан, ол бастапқы программаның контекстінде бұзу кодын орындау мақсатында оның машина нұсқауларын толтыра алады. Бұзушы жадты өз кодымен қайта жазды және оның орындалуына қол жеткізді. Енді программа толығымен бұзушының бақылауында және ол оны қалаған нәрсені жасауға мәжбүр етуі мүмкін. Бұзушы коды көбінесе қабықты іске қосу үшін қолданылады (мысалы, exec жүйелік қоңырауы арқылы), оған машинаға оңай қол жеткізуге мүмкіндік береді. Сондықтан,бұл код көбінесе қабықтың көшірмесін бастамаса да, оны іске қосу коды немесе **шелл-код** (***shellcode***) деп аталады.

Бұл әдіс ***gets*** функциясын қолданатын программаларға қатысты ғана емес (оны қолданудан аулақ болу керек), сонымен қатар Тұтынушы берген деректерді оның шекараларын бұзбай буферге көшіретін кез келген код болған кезде де жұмыс істейді. Бұл ұтынушы деректерінде пәрмен жолының параметрлері, қоршаған орта жолының деректері, желі арқылы жіберілген деректер немесе тұтынушы файлынан оқылған деректер болуы мүмкін.

Мұндай деректерді көшіретін немесе жылжытатын көптеген функциялар бар: strcpy,memcpy, strcat және басқалары.Әрине, сіз жазған және байттарды буферге жылжытатын ескі цикл де осал болуы мүмкін. Ал егер хакер нақты қайтару мекенжайын білмесе ше? Көбінесе ол шамамен қай жерде екенін болжай алады, бірақ шелл коды дәл емес. Бұл жағдайда оның алдына **nop-бағыттаушылар** (***nop sled***) – ештеңе жасамайтын NO OPERATION (операция жоқ) бір байттық нұсқауларының тізбегін

орналастыру әдеттегі шешім болады. Қашан бұзушы nop-бағыттаушыларға "қона" алады, кодты орындау түптеп келгенде нағыз шелл-кодқа дейін жетеді. Nop-бағыттағыштар стекте жұмыс істейді, бірақ олар үйінділерде де жұмыс істейді, онда бұзушылар көбінесе өздерінің Nop-бағыттағыштары мен шелл-кодтарын үймеге қою арқылы өз мүмкіндіктерін арттыруға тырысады. Мысалы, браузерде зиянды JavaScript коды мүмкіндігінше көп жадыны таратуға және оны толтыруға тырысуы мүмкін ұзын nop-бағыттаушы және көлемі шағын шелл-код. Содан кейін, егер бұзушы программаның орындалу барысын ауыстырып, оны үйіндідегі еркін мекен-жайға бағыттай алса, онда оның nop-бағыттағыштарға түсу мүмкіндігі өте жоғары болады. Бұл технология **үйіндіге бүрку** (***heap spraying***) деп аталады.

Стек индикаторы ("канарейка")

Қысқаша айтылған бұзылудан қорғаудың жиі қолданылатын әдісі – стек индикаторын немесе "канарейканы" (stack canaries) пайдалану. Бұл атау тау-кен тәжірибесінен шыққан. Шахтадағы жұмыс қауіп-қатерге толы. Улы газдардың шығарылуы мүмкін, мысалы, улы газ, бұл кеншілерді өлтіреді. Барлық өзге де улы газдың иісі болмайды, сондықтан, кеншілер мүмкін тіпті оны байқамайды. Бұрын кеншілер мұндай шығарылудан қорқып, ертерек ескерту жүйесі ретінде канарейканы (құс) шахтасына алып баратын. Улы газдардың кез-келген шығарылуы канарды өлтіреді. Егер құс өлсе, онда жер бетіне көтерілу керек.

Қазіргі компьютерлік жүйелер сонымен қатар сандық болса да, канарийлерді ерте ескерту жүйесі ретінде қолданылады. Идея қарапайым. программа функцияны шақыратын жерде компилятор қайтару мекен-жайы алдында "канарейка" ерікті мәнін сақтау үшін кодты енгізеді.Функциядан оралмас бұрын, компилятор "канарей"мәнін тексеру үшін кодты енгізеді. Егер мән өзгерсе,онда бір нәрсе дұрыс болмады. Бұл жағдайда программаның орындалуын жалғастырудан гөрі дабыл түймесін және апаттық тоқтатуды басқан дұрыс.

Стектік "канарейкаларды" айналып өту

"Канарейкалар" бұрын қарастырылған шабуылдармен жақсы жұмыс істейді, бірақ буфердің толып кету мүмкіндігі әлі де сақталады. Мысалы, кодтың келесі үзіндісін қарастырыңыз. Ол екі жаңа функцияны

қолданады: жолды буферге көшіруге арналған C тіл кітапханасының құрамынан ***strcpy*** функциясы және жолдың ұзындығын анықтайтын ***strlen*** функциясы.

"Канарейка" дестесін айналып өту ***len*** ұзындығын өзгерту және кейіннен қайтару мекенжайын тікелей өзгерту арқылы жүзеге асырылады:

```
01. void A (char *date) {
02. int len;
03. char B [128];
04. char logMsg [256];
05.
06. strcpy (logMsg, date); /* алдымен хабарлама жолына деректер жолы көшіріледі */
07. len = str len (date); /* Деректер жолындағы таңбалар санын анықтау */
08. gets (B); /* енді нақты хабарлама алу */
09. strcpy (logMsg+len, B); /*және оны деректерден кейін журнал хабарламасына көшіру */
10. writeLog (logMsg); /*соңында, дискіге тіркеу хабарын жазу*/
11. }
```

Алдыңғы мысалдағыдай, A-функциясы тіркеу хабарламасын стандартты енгізуден оқиды, бірақ қазір оған ағымдағы деректер нақты қосылады (A-функциясының жол аргументі ретінде берілген). Алдымен ол деректерді тіркеу хабарламасына көшіреді (6-жол). Деректер жолының ұзындығы аптаның күніне, айға және т. б. байланысты әр түрлі болуы мүмкін, мысалы, "сәрсенбі" сөзінде 8 әріп, ал "сенбі" сөзінде – 5. Бұл ай атауларына да қатысты. Сондықтан екінші әрекет деректер жолындағы таңбалар санын анықтайды (7-жол). Содан кейін функция Тұтынушының кірісін алады (8-жол) және оны деректер жолынан кейін бірден бастап тіркеу хабарламасына көшіреді. Бұл деректер алынған көшірме тіркеу хабарламасынан және жолдың ұзындығынан басталуы керек екенін

көрсету арқылы жасалады (9-жол). Соңында, функция бұрынғыдай тіркеу хабарламасын дискіге жазады.

Жүйе стектік "канареяларды" қолданады делік. Қайтару мекенжайын қалай өзгертуге болады? Құпия – B буферінің толып кетуін ұйымдастырған кезде крекер дереу қайтару мекен-жайына кіруге тырыспайды. Оның орнына, ол Len айнымалысының мәнін тікелей оның үстінде орналасқан стекке өзгертеді. 9-жолда len B буферінің мазмұны қай жерде жазылатынын анықтайтын орын ауыстыру ретінде қызмет етеді, программашының Жоспары тек деректер жолын өткізіп жіберуден тұрады, өйткені крекер Len айнымалысының мәнін басқарады, ол осы айнымалы мәнді "канарды" айналып өтіп, қайтару мекенжайын қайта жазу үшін қолдана алады.

Сонымен қатар, буфердің толып кетуі қайтару мекен-жайымен шектелмейді. Толып кетуді ұйымдастыру арқылы қол жетімді кез-келген функция көрсеткіші өте қолайлы.Функция көрсеткіші қалыпты көрсеткішке ұқсас, бірақ деректерді емес, функцияны көрсетеді. Мысалы, CIS++ программасында программашы a айнымалысын сапа ретінде жариялай алады жол аргументін алатын және нәтижені берместен басқаруды қайтаратын функция көрсеткіші:

```
void (*f)(char*);
```

Синтаксис, бәлкім, біршама жұмбақ, бірақ бұл шын мәнінде айнымалының тағы бір декларациясы. Алдыңғы мысалдағы a функциясы жоғарыда келтірілген қолтаңбаға сәйкес келетіндіктен, қазір "f = A" деп жазуға болады және A-ның орнына біздің программада F-ті қолдануға болады. Кітаптың міндеті функция көрсеткіштерінің егжей-тегжейін терең зерттеуді қамтымайды, бірақ бұл көрсеткіштер операциялық жүйелерде жиі кездеседі деп сендіремін.

Енді крекер функция көрсеткішін қайта жаза алды делік. программа меңзерді пайдаланып функцияны шақырғаннан кейін, ол хакер енгізген кодты іске қосады. Зиянды код жұмыс істеуі үшін функция индексі стекте болмауы мүмкін.Сгодятся және сол көрсеткіштер функциялар, олар куче. Хакер функция көрсеткішінің мәнін немесе басқаруды қайтару мекенжайын коды бар буфер мекен-жайына өзгерте алатындықтан, программа анықтаған басқару ағынын өзгерте алады.

Деректердің орындалуын болдырмау

Мүмкін сіз қазір: "күте тұрыңыз! Мәселе мынада, крекер функциялардың көрсеткіштерін немесе Басқару мекен-жайын қайта жаза алады, бірақ ол кодты енгізіп, жүйені орындай алады. Неге байттың үйіндіде немесе стекте орындалуының алдын алмасқа? деп жар салуыңыз мүмкін. "Егер сіз осылай жасасаңыз, онда Сізге жақсы ой келді. Бірақ көп ұзамай біз мұндай сілкіндіретін ойлар буфердің толып кету шабуылын әрдайым тоқтата алмайтындығын көреміз. Бірақ идеяның өзі өте ақылды. **Кодты енгізу шабуылдары** (***code injection attacks***) хакер ұсынған байттарды заңды код ретінде орындау мүмкін болмаса, жұмыс істемейді. Қазіргі заманғы орталық үдерісорлардың **NX-бит** (***NX bit***) деп аталатын қасиеті бар, ол "no-eXecute" дегенді білдіреді, яғни орындалмайды. Бұл бит деректер сегменттерін (қадалар, стектер және ғаламдық айнымалылар) және мәтіндік сегменттерді (кодты қамтитын) ажырату үшін әсіресе пайдалы. Шын мәнінде, көптеген заманауи операциялық жүйелер деректер сегменттерін олардың мазмұнын орындай алмай-ақ, жазу мүмкіндігін және мәтіндік сегменттерді оларды жазбай-ақ орындау мүмкіндігін қамтамасыз етуге тырысты. Бұл саясат **OpenBSD-де W^X ретінде белгілі ("WExclusive-OR X" немесе "W XOR x"** деп оқылады). Бұл жадты жазуға немесе орындауға болатындығын білдіреді, бірақ бір уақытта екеуі де емес. Mac OS X, Linux және Windows-та ұқсас қорғаныс схемалары бар. Жалпы алғанда, бұл қауіпсіздік шарасы деп аталады **DEP** (**Data Execution Prevention**), яғни деректердің орындалуын болдырмау. Кейбір жабдықтар NX-битті қолдамайды. Бұл жағдайда DEP жұмыс істейді, бірақ мәжбүрлеу программалық жасақтама болып табылады. DEP бұрын қарастырылған барлық шабуылдардың алдын алады. Крекер үдеріске қалағаныңызша шелл кодын енгізе алады. Жад орындалмайынша, бұл кодты іске қосу әдісі пайда болмайды.

Кодты қайталап пайдалану шабуылдары

DEP деректер аймағында кодты орындауға мүмкіндік бермейді. "Канарлар" стектері қайтару мекен-жайлары мен функция көрсеткіштерін қайта жазуды қиындатады (бірақ жоққа шығармайды). Өкінішке орай, бұл оқиғаның соңы емес, өйткені бір күні біреуге түсінік те төмендеді. Ол мынаны түсінді: "неліктен кодты екілік түрде жеткілікті болған кезде енгізу керек?». Басқаша айтқанда, жаңа кодты енгізудің орнына бұзушы

екілік код пен кітапханалардағы бар функциялар мен нұсқаулардан қажетті функцияны жасайды. Алдымен біз осындай шабуылдардан ең қарапайымын – **кітапханаға қайтару шабуылын** (***return to libe***), содан кейін күрделі, бірақ өте танымал **өзара бағытталған программалау технологиясын** (***return-oriented programming***) қарастырамыз .

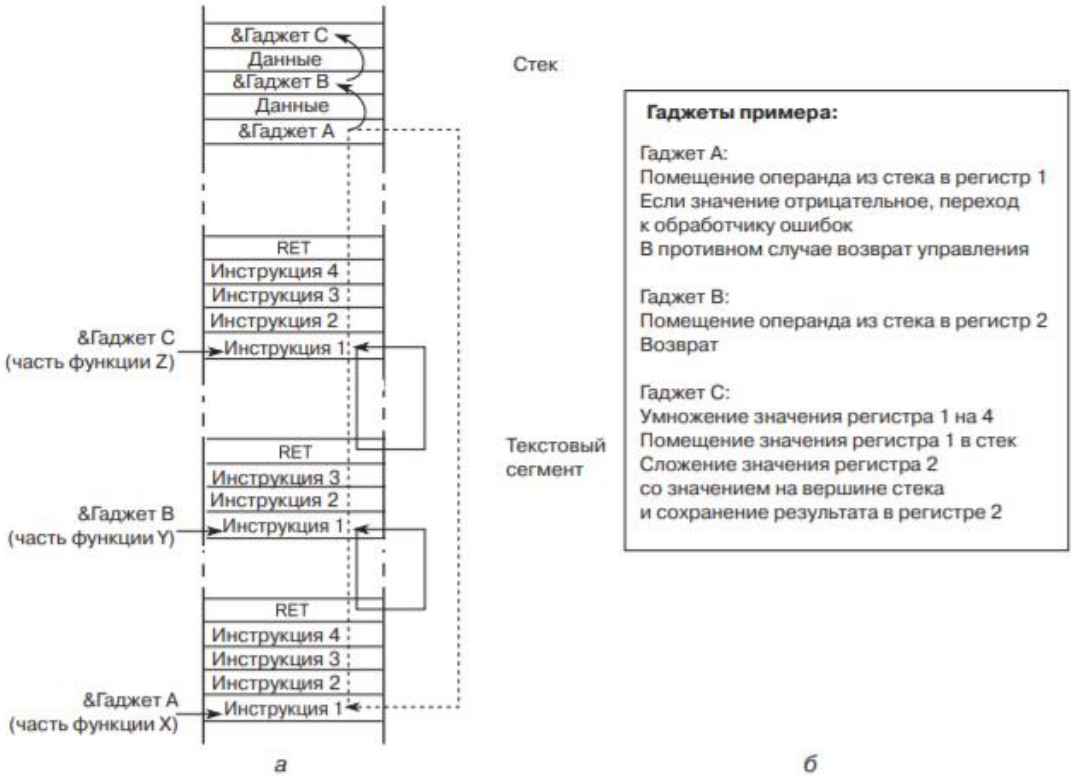
Буфердің толып кетуі (19-суретті қараңыз) ағымдағы функциядан қайтару мекенжайын қайта жазуға әкелді делік, бірақ крекер (бұзушы) ұсынған стек коды орындалмайды. Басқаруды басқа бір орынға беруге бола ма ? деген сұрақ туындайды және ол мүмкін екен. С тіліндегі барлық дерлік программалар ***libc*** кітапханасымен байланысты (оны әдетте бірнеше программалар бөліседі), көптеген с программаларына қажет негізгі функцияларды қамтиды. Осындай функциялардың бірісбұл аргумент ретінде жолды алып, оны орындау үшін қабыққа беретін жүйе. Осылайша, system функциясын қолдана отырып, крекер өзіне қажет кез-келген программаны орындай алады. Сондықтан, Шелл-кодты орындаудың орнына, крекер жай ғана орындалатын пәрменді қамтитын жолды стекке орналастырады және қайтару мекен-жайы арқылы басқаруды жүйелік функцияға жібереді. Бұл **кітапханаға оралу шабуылы** (***return to libe***) деп аталады және бірнеше нұсқалары бар. Жүйе крекерді қызықтыратын жалғыз функция емес. Мысалы, крекер mprotect функциясын да қолдана алады деректер сегментінің бөлігі орындалады. Сонымен қатар, libc кітапхана функциясына сөзсіз ауысудың орнына, крекер қайта бағыттау деңгейін қолдана алады. Мысалы, Linux-та крекер басқаруды процедуралық орналасу кестесіне қайтара алады (**procedure Linkage Table (PLT)**). PLT динамикалық байланыстыруды жеңілдететін құрылым болып табылады және кодтың үзінділерін қамтиды,олардың орындалуы кезінде, өз кезегінде, библиялық функциялар динамикалық түрде шақырылады. Басқаруды осы кодқа қайтару жанама нәтижеге әкеледі орындау кітапханалық функциялары. Өзара бағытталған программалау тұжырымдамасы (***Return-Oriented Programming (ROP***)) программа кодын қайта пайдалану ниетін шектен шығарады. Басқаруды кітапхана функцияларына кіру нүктелеріне қайтарудың орнына, крекер мәтіндік сегменттегі кез-келген нұсқауларды басқара алады.Мысалы, ол кодты басқаруды функцияның басында емес, ортасында бере алады. Орындау осы сәттен бастап жалғасады және нұсқаулар бірінен соң бірі орындалады. Бірнеше нұсқауларды орындағаннан кейін кезек басқа

қайтару нұсқауларына жетеді делік. Енді біз бірдей сұрақ қоямыз: басқаруды қайда беруге болады? Крекердің стек бақылауы болғандықтан, ол қайтадан кодты басқаруды қалаған жерге жіберуге мәжбүр етуі мүмкін. Мұны екі рет жасағаннан кейін, ол мұны үшінші, төртінші, оныншы және т. б. жасай алады. Сондықтан, өзара бағытталған программалауды қабылдау кодтың кішкене тізбегін іздеуден тұрады, ол біріншіден, қолайлы нәрсені жасайды, екіншіден, қайтару нұсқауымен аяқталады. Хакер мұндай кодтық тізбектерді стекке орналастырылған қайтару мекенжайлары арқылы бір жолға сала алады. Жеке фрагменттер **гаджеттер** (***gadgets***) деп аталады. Әдетте олар өте шектеулі функционалдылыққа ие, мысалы, Олар екі регистрді қосуды, жадты регистрге жүктеуді немесе мәнді стекке қоюды орындай алады. Басқаша айтқанда, гаджеттер жиынтығы крекер қолдана алатын өте таңқаларлық Нұсқаулық сияқты көрінуі мүмкін стекті шебер басқару арқылы еркін функционалдылықты құру.Сонымен қатар, стек көрсеткіші сәл таңқаларлық әртүрлілік ретінде қызмет етеді команда санауышы.

9.20-а, суреті гаджеттердің стекке қайтару мекен-жайлары арқылы қалай байланысатынын көрсетеді. Гаджеттер-қайтару нұсқаулығымен аяқталатын кодтың кішкене бөліктері. Бұл нұсқаулық мекен-жайды беру үшін стектен алады осы мекен-жайы бойынша басқару және онымен жұмысты жалғастыру. Бұл жағдайда крекер алдымен гаджетті басқаруды белгілі бір х функциясына, содан кейін в гаджетін Y функциясына жібереді және т.б. крекердің міндеті-мұндай гаджеттерді орындалатын екілік кодқа жинау.Ол гаджеттерді өзі жасамайтындықтан, кейде гаджеттермен күресуге тура келеді,олар идеалдан алыс болуы мүмкін, бірақ ол жоспарлаған жұмыс үшін өте қолайлы. Мысалы, 20-б, суретте А гаджеті нұсқаулар тізбегіне кіреді және онда тексеру бар деп болжанады. Крекерге бұл тексеру мүлдем қажет болмауы мүмкін, бірақ ол бар болғандықтан, оған келісуге тура келеді. Көптеген мақсаттар үшін кез-келген теріс емес санды 1-регистрге қою жақсы болар еді. Келесі гаджет кез-келген стек мәнін 2-регистрге орналастырады, ал үшінші гаджет 1-регистрдің мәнін 4-ке көбейтеді, оның мәнін стекке орналастырады және оны 2-регистрдің мәніне қосады. Осы үш гаджетті біріктіру крекерді бүтін сандар жиымындағы элементтің мекенжайын есептеу үшін қолдануға болатын нәрсеге әкеледі. Жиымның индексі беріледі бірінші мәнді

деректер дестесін, ал базалық адресі массив деректердің екінші мағынасында болуы керек.

20, а-суретте гаджеттердің стекке қайтару мекен-жайлары арқылы қалай байланысатынының мысалын көрсетеді. Гаджеттер – қайтару нұсқауымен аяқталатын кодтың кішкене бөліктері. Бұл нұсқаулық басқаруды осы мекен-жайға беру және одан жұмысты жалғастыру үшін стектен мекенжайды алады. Бұл жағдайда крекер алдымен гаджетті басқаруды белгілі бір x функциясына, содан кейін в гаджетін Y функциясына және т.б. береді. Ол гаджеттерді өзі жасамайтындықтан, кейде ол гаджеттермен күресуге мәжбүр болады, олар идеалдан алыс болуы мүмкін, бірақ ол жоспарлаған жұмыс үшін өте қолайлы.



20-сурет. Өзара бағытталған программалау: гаджеттерді байланыстыру

Мысалы, *20, б-суретте* А гаджеті нұсқаулық тізбегіне кіреді және тексеруден өтеді деп болжанады. Крекерге бұл тексеру мүлдем қажет болмауы мүмкін, бірақ ол бар болғандықтан, оған келісуге тура келеді. Көптеген мақсаттар үшін кез-келген теріс емес санды 1-регистрге қою жақсы болар еді. Келесі гаджет кез-келген стек мәнін 2-регистрге орналастырады, ал үшінші гаджет 1 регистрінің мәнін 4-ке көбейтеді,

оның мәнін стекке орналастырады және оны 2 регистрінің мәнімен қосады. Осы үш гаджетті біріктіру крекерді бүтін сандар массивіндегі элементтің мекенжайын есептеу үшін қолдануға болатын нәрсеге әкеледі. Массив индексі стек ішіндегі бірінші деректер мәнімен қамтамасыз етіледі, ал массивтің негізгі Мекен-жайы екінші деректер мәнінде болуы керек.

Өзара бағытталған программалау өте күрделі болып көрінуі мүмкін, және, мүмкін, солай. Бірақ, әдеттегідей, адамдар жұмыстың максималды көлемін автоматтандыру үшін құралдарды ойлап тапты. Мысал ретінде гаджет құрастырушылар және тіпті ROP компиляторлары бар. Қазіргі уақытта ROP қылмыстық әлемде қолданылатын зиянды кодты құрудың маңызды технологияларының бірі болып табылады.

Адрестік кеңістікті үлестірудің рандомизациялары

Мұндай шабуылдардың алдын алу туралы тағы бір идея бар. Қайтару адресін өзгертуден және қайсыбір (ROP) программаны іске асырудан басқа, крекер басқаруды мүлдем дәл мекен-жайларға жібере алуы керек, ROP технологиясымен ***nop***-бағыттаушылары бар нөмір жұмыс істемейді. Адрестер тіркелген кезде мәселені шешу қиын емес, ал егер олар бекітілмеген болса ше? **Адрестік кеңістікті бөлуді рандомизациялау** (***Address Space layout Randomization (ASLR)***) программа іске қосылған сайын функциялар мен деректер мекенжайларын ерікті түрде тағайындауға бағытталған. Нәтижесінде бұзушының жүйеге зиян келтіру міндеті айтарлықтай қиындайды. Атап айтқанда, ASLR көбінесе бастапқы стек, қадалар мен кітапханалардың позицияларын кездейсоқ таңдаумен айналысады. Көптеген заманауи операциялық жүйелер стек "канарлармен" және DEP-мен бірге ASLR-ді де қолдайды. Олардың көпшілігі бұл технологияны қолданушы қосымшалары үшін ұсынады, бірақ аз ғана бөлігі оны үнемі және операциялық жүйенің өзегіне қолданады (Giuffrida et al., 2012). Осы үш қорғаныс механизмінің бірлескен күш-жігері бұзушылар жолындағы кедергілерді едәуір арттырды. Енгізілген кодқа немесе тіпті жадта бар кейбір функцияға қарапайым көшу өте қиын жұмыс болды. Бұл механизмдер қазіргі операциялық жүйелерде маңызды қорғаныс сызығын құрайды. Мұның бәрінде әсіресе тартымды-бұл өнімділік тұрғысынан өте қолайлы бағамен қорғауды қамтамасыз ету.

ASLR-ді айналып өту

Барлық үш қорғаныс құралы болса да, бұзушылар әлі де жүйеге зиян келтіре алады. ASLR-де зиянкестерге айналып өту жолдарын табуға мүмкіндік беретін бірқатар әлсіз жақтар бар. Біріншісі-ASLR әрқашан жеткілікті таңдау жасай бермейді. Көптеген ASLR енгізулерінде әлі де белгіленген жерлерде белгілі бір код бар. Сонымен қатар, сегменттің мекен-жайын кездейсоқ таңдаған кезде де, рандомизация әлсіз болуы мүмкін және крекер оны ерекше трюктерсіз жеңе алады. Мысалы, 32 биттік жүйелерде энтропияны шектеуге болады, өйткені стектің барлық биттерін рандомизациялау мүмкін емес. Стек әдеттегі стек сияқты жұмыс істеуі үшін, ең аз маңызды биттерді рандомизациялау қолайлы емес. ASLR-ге ең маңызды шабуыл жад туралы ақпаратты ашу арқылы жасалады. Бұл жағдайда крекер осалдықты программаны тікелей бақылау үшін емес, жадыны бөлу туралы ақпараттың ағып кетуін ұйымдастыру үшін пайдаланады, содан кейін оны басқа осалдықты пайдалану үшін пайдалануға болады. Қарапайым мысал ретінде келесі кодты қарастырайық:

```
01. void C() {
02. int index;
03. int prime [16] = { 1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47 };
04. printf ("Какое простое число из показанных здесь вы хотели бы видеть?");
05. index = read user input ();
06. printf ("На позиции %d находится простое число %d\n", index, prime[index]);
07. }
```

Кодта Тұтынушының кіруін оқуға шақыру бар, онда бұл оқу C тілінің стандартты кітапханасының бөлігі емес, біз бұл оқу мүмкіндігі бар деп санаймыз және Тұтынушы пәрмен жолында терген бүтін санды қайтарады. Біз сондай-ақ қателіктер жоқ деп болжаймыз. Бірақ сонымен бірге, осы кодтан ақпараттың ағып кетуін ұйымдастыру өте қарапайым. Сізге тек 15-тен көп немесе 0-ден аз индекс беру керек. Индекс программасы тексерілмегендіктен, ол кез-келген бүтін санның мәнін жадтан оңай қайтарады. Сәтті шабуыл жасау үшін көбінесе бір функцияның мекен-жайы жеткілікті. Себебі, әр кітапхана жүктелетін

орынды ерікті түрде таңдауға болатынына қарамастан, осы позициядан әрбір жеке функция үшін салыстырмалы түрде ауыстыру әдетте белгіленген мәнге ие болады. Басқаша айтқанда, бір функцияның қайда екенін біліп, басқалардың қайда екенін білесіз. Олай болмаса да, егер сізде бір ғана код мекен-жайы болса, Snow et al.(2013) жұмысында көрсетілген көптеген басқа мекен-жайларды табу өте оңай.

Басқару ағынын қайта бағыттауға байланысты емес шабуылдар

Осы уақытқа дейін программаны басқару ағынына шабуылдар қарастырылды: функция көрсеткіштері мен қайтару мекен-жайларын өзгерту. Мақсат әрқашан программаны жаңа функцияларды орындауға мәжбүрлеу болды, тіпті егер бұл функциялар екілік түрінде болған кодтан алынған болса да. Бірақ бұл жалғыз мүмкіндік емес. Псевдокодтың келесі үзіндісінде көрсетілгендей, крекер үшін қызықты мақсат деректердің өзі болуы мүмкін:

```
01. void A() {
02. int author ized;
03. char name [128];
04. authorized = check credentials (...); /* атакующий не авторизован,
поэтому возвращается 0 */
05. printf ("Как вас зовут?\n");
06. gets (name);
07. if (authorized != 0) {
08. printf ("Добро пожаловать %s, вот все ваши секреты \n", name)
09. /* ... демонстрация секретных данных ... */
10. } else
11. printf ("Извините, %s, но вы не прошли авторизацию.\n");
12. }
13. }
```

Бұл код авторизацияны тексеруге арналған. Құпия деректерді тек тиісті өкілеттіктері бар Тұтынушыларға қарауға рұқсат етіледі. Өкілеттікті тексеретін Функция С тілі кітапханасының құрамына кірмейді, бірақ ол программада бір жерде бар және қателіктері жоқ деп болжанады. Енді крекер 129 таңбаны енгізді делік. Алдыңғы жағдайдағыдай, буфер толып кетті, бірақ қайтару мекен-жайы қайта жазылмады. Оның орнына крекер авторизация айнымалысының мәнін нөлден басқа мән беру арқылы өзгертті. программа апаттық жағдайға кірмеді және бұзушының кодын орындамады, бірақ ол рұқсат етілмеген Тұтынушының атына құпия ақпараттың ағып кетуіне жол берді.

Буфердің толып кетуі: нүкте әлі орнатылмаған

Буфердің толып кетуі-бұзушылар қолданатын ең көне және маңызды жадты бұрмалау технологиясының бірі. Ширек ғасырдан астам уақытқа байланысты оқиғалардың болуына және қорғаныс құралдарының көптігіне қарамастан (біз олардың ең маңыздыларын ғана қарастырдық), бұл проблемадан құтылу мүмкін емес сияқты (Ван дер Вейн, 2012). Осы уақыт ішінде қауіпсіздік проблемаларының едәуір бөлігі осы кемшіліктен туындады, оны жою қиын, өйткені айналасында буфердің толып кетуін тексермейтін көптеген с программалары бар. Қару жарысы әлі аяқталған жоқ. Бүкіл әлемдегі зерттеушілер барлық жаңа қорғаныс құралдарын зерттеп жатыр. Бұл құралдардың кейбіреулері екілік кодтарға бағытталған, кейбіреулері С және С++ компиляторлары үшін қауіпсіздік кеңейтілімдерінен тұрады. Хакерлер өздерінің зиянды технологияларын жетілдіретінін ескерген жөн. Бұл бөлімде біз сізге кейбір маңызды технологияларға шолу жасауға тырыстық, бірақ бұл идеяның көптеген нұсқалары бар. Осы кітаптың келесі басылымында бұл бөлім өзектілігін жоғалтпайтынына сенімдіміз (және кеңейтілуі мүмкін).

ОЖҚ_Дәріс №12. Вирустар. Вирустар қалай жұмыс істейді. Вирус-компаньон. Орындалатын файлдарды зақымдайтын вирустар. Резиденттк вирустар. Жүктеу секторын зақымдайтын вирустар. Құрылғылар драйверлерінің вирустары. Макровирустар. Программалардың бастапқы мәтіндерін зақымдайтын вирустар. Вирустардың таралуы

В этом разделе мы рассмотрим вирусы, после чего перейдем к рассмотрению червей. В Интернете полно информации о вирусах, поэтому джинн уже выпущен из бутылки. Кроме того, людям трудно защититься от вирусов, не зная, как они работают. И наконец, вокруг вирусов ходит столько легенд, что их давно следует развеять. Так что же такое вирус? Коротко говоря, **вирус** (virus) — это программа, способная размножаться, присоединяя свой код к другим программам аналогично тому, как размножаются биологические вирусы. Кроме этого, вирус способен и на другие действия. Черви похожи на вирусы, но они занимаются копированием самих себя. В данный момент эти различия нас не интересуют, поэтому термин «вирус» будет употребляться для обоих случаев. А черви будут рассмотрены в следующем разделе.

Как работают вирусы

Теперь посмотрим, какие бывают вирусы и как они работают. Создатель вирусов, назовем его Вирджил, по всей вероятности, работает на ассемблере (или, может быть, на C), чтобы получить компактный эффективный результат. После создания вируса он вставляет его в программу на собственной машине. Затем эта зараженная программа распространяется, возможно, путем выкладывания ее в коллекцию бесплатного программного обеспечения в Интернете. Этой программой может быть захватывающая новая игра, пиратская версия коммерческого программного продукта или что-нибудь еще не менее привлекательное. Затем люди начинают скачивать зараженную программу.

После установки на машину жертвы вирус бездействует до тех пор, пока не будет выполнена зараженная программа. Как только она будет запущена, начинается, как правило, заражение других имеющихся на машине программ, а затем выполнение **действий по предназначению** (payload). Во многих случаях эти действия могут откладываться до наступления конкретной даты, чтобы дать вирусу распространиться до того, как он будет замечен. Выбранная дата может иметь политическую подоплеку (например, если он запускается в сотую или пятисотую годовщину каких-нибудь гонений или оскорблений той этнической группы, к которой принадлежит автор).

Далее будут рассмотрены семь разновидностей вирусов, отличающихся друг от друга объектами заражения. Это «компанейские» вирусы, а также вирусы, заражающие исполняемые программы, память, загрузочный сектор, драйвер устройства, макросы и исходные тексты программ. Несомненно, в будущем появятся и новые типы вирусов.

Вирус-компаньон

Сам по себе вирус-компаньон (companion virus) программу не заражает, он запускается, когда предполагается запуск какой-нибудь программы. Эта разновидность очень стара и относится к тем давним временам, когда на планете царила MS-DOS, но она по-прежнему жива. Ее концепцию поясним на примере. Когда в MS-DOS пользователь набирает

```
Prog
```

MS-DOS сначала ищет программу по имени prog.com. Если она не может найти такую программу, то ищет программу по имени prog.exe. В Windows, когда пользователь щелкает на кнопке Пуск (Start), а затем на пункте меню Выполнить (Run) или нажимает клавишу с логотипом Windows, а затем, удерживая ее, нажимает клавишу R, он попадает в среду, где возможно то же самое. В наше время большинство программ содержится в файлах с расширениями .exe, а файлы с расширением .com встречаются крайне редко.

Предположим, что Вирджилу известны многие, кто запускает prog.exe из приглашения MS-DOS или из командной строки Windows, вызываемой щелчком на пункте меню Выполнить (Run). Тогда он может выпустить вирус с именем файла prog.com, который будет выполнен в тот момент, когда кто-нибудь попытается запустить prog (если только не будет введено полное имя prog.exe). Когда prog.com завершит свою работу, он просто выполнит prog.exe и пользователь останется в неведении.

Это чем-то похоже на атаку, осуществляемую на рабочем столе Windows, в которой используются ярлыки программ (символьные ссылки на программы). Вирус может изменить объект ярлыка, заменив его указателем на вирус. После того как пользователь дважды щелкнет на значке, вирус будет выполнен. Когда вирус отработает, он просто запускает исходный объект ярлыка.

Вирусы, заражающие исполняемые файлы

На следующей по сложности ступеньке располагаются вирусы, заражающие исполняемые файлы. Самые простые из них просто переписывают исполняемую программу своим кодом. Они называются **перезаписывающими вирусами** (overwriting viruses). Логика заражения, закладываемая в такие вирусы, показана в листинге 2.

Листинг 9.2. Рекурсивная процедура, отыскивающая исполняемые файлы в системе UNIX

```
#include <sys/types.h> /* стандартные заголовки POSIX */
#include <sys/stat.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
struct stat sbuf; /* для вызова lstat, чтобы определить, */
/* не являтся ли файл символьной ссылкой */
search(char *dir_name)
```

```
{ /* рекурсивный поиск исполняемых файлов */
DIR *dirp; /* указатель на открытый каталог */
struct dirent *dp; /* указатель на запись каталога */
dirp = opendir(dir_name); /* открытие указанного каталога */
if (dirp == NULL) return; /* каталог не может быть открыт, */
/* и о нем следует забыть */
while (TRUE) {
dp = readdir(dirp); /* чтение следующей записи каталога */
if (dp == NULL) { /* NULL означает конец каталога */
```

```
chdir (".."); /* возвращение в родительский каталог */
break; /* выход из цикла */
}
}
if (dp->d_name[0] == '.') continue; /* пропуск каталогов . и .. */
lstat(dp->d_name, &sbuf); /* является ли запись символьной
ссылкой? */
if (S_ISLNK(sbuf.st_mode)) continue; /* пропуск символьных ссылок */
if (chdir(dp->d_name) == 0) { /* если chdir сработала, значит,
это каталог */
search("."); /* да, вход и проведение в нем поиска */
} else { /* нет (это файл), заражение файла */
if (access(dp->d_name,X_OK) == 0) /* если файл исполняемый, */
/* заражение файла */
infect(dp->d_name);
}
closedir(dirp); /* каталог обработан, закрытие каталога
и выход */
}
```

Основная программа этого вируса сначала копирует его двоичную программу в массив, открывая *argv[0]* и считывая его содержимое для большей сохранности. Затем она сканирует всю файловую систему, начиная с корневого каталога, сделав его текущим и вызвав процедуру *search* с каталогом root в качестве параметра.

Рекурсивная процедура *search* обрабатывает каталог, открывая его, а затем считывая записи по одной с помощью функции *readdir* до тех пор, пока эта функция не вернет значение *NULL*, свидетельствующее о том, что записей больше нет. Если запись относится к каталогу, то этот каталог обрабатывается за счет превращения его в текущий и рекурсивного вызова процедуры *search*. Если запись относится к файлу, этот файл заражается за счет вызова процедуры *infect* с именем инфицируемого файла в качестве параметра. Файлы, начинающиеся с символа «.», пропускаются во избежание проблем с каталогами «.» и «..». Пропускаются также символьные ссылки, поскольку программа предполагает, что она может войти в каталог, используя системный вызов *chdir*, а потом вернуться на прежнее место, переходя по ссылкам «..», которые считаются жесткими связями, а не символьными ссылками. Более изощренная программа может также обработать и символьные ссылки.

Процедура заражения *infect* (в листинге отсутствует) должна просто открыть указанный в ее параметре файл, скопировать вирус, хранящийся в массиве поверх файла, а затем закрыть файл.

Этот вирус может быть «усовершенствован» разными способами. Во-первых, в процедуру *infect* может быть включен тест, генерирующий случайное число и в большинстве случаев просто возвращающий управление без каких-либо действий. Скажем, в одном случае из 128 происходит заражение, так что вероятность раннего обнаружения будет понижена и вирус получит неплохие шансы на распространение. У биологических вирусов есть такое же свойство: вирусы, быстро убивающие свою жертву, не распространяются так же быстро, как те вирусы, которые вызывают постепенную гибель организма, давая жертве неплохие шансы на распространение вируса. Другая конструкция предусматривает более высокую степень заражения (скажем, 25 %), но сокращает количество одновременно заражаемых файлов, чтобы снизить активность использования диска и вызвать меньше подозрений.

Во-вторых, процедура *infect* может проверить файл на зараженность, чтобы не тратить зря время на повторное заражение одного и того же файла. В-третьих, можно принять меры к сохранению прежнего времени последней модификации файла и его размера, чтобы скрыть его зараженность. Для программ, размер которых превышает размер вируса, будет оставлен прежний размер, но программы, чей размер был меньше вируса, теперь станут больше. Поскольку большинство вирусов меньше большинства программ, эта проблема слишком остро не стоит.

Хотя эта программа не такая уж большая (вся программа на C, умещающаяся на одной страничке, и текстовый сегмент после компиляции занимают меньше 2 Кбайт), ассемблерная версия может быть еще короче. Людвиг привел пример программы на ассемблере для MS-DOS, которая заражала все файлы в своем каталоге и занимала после ассемблирования всего 44 байта (Ludwig, 1998).

Далее в этой главе будут рассмотрены антивирусные программы, выслеживающие и удаляющие вирусы. Здесь интересно отметить, что логика, приведенная в листинге 2, которую вирус мог использовать для поиска и заражения всех исполняемых файлов, может быть применена также в антивирусной программе для отслеживания всех зараженных программ с целью удаления вируса. Технологии заражения и лечения идут рука об руку, поэтому чтобы эффективно бороться с вирусами, необходимо детально разбираться в их работе.

С точки зрения Вирджила, недостатком перезаписывающего вируса является легкость его обнаружения. В конечном итоге, при выполнении зараженной программы она может распространять вирус дальше, но не может работать по своему прежнему предназначению, что тут же заметит пользователь. Поэтому многие вирусы самостоятельно присоединяются к программе и делают свою черную работу, но после этого дают программе возможность нормально работать. Это так называемые **паразитические вирусы** (parasitic viruses).

Эти вирусы могут прикрепляться к началу или концу программы, или встраиваться в нее. Если вирус прикрепляется к началу, то сначала он должен скопировать программу в оперативную память, поместить себя в ее начало, а затем скопировать программу, следующую прямо за ним, из оперативной памяти (рис. 24, *б*). К сожалению, программа не будет запускаться по своему новому виртуальному

адресу, поэтому вирусу приходится перемещать программу либо на размер ее смещения, либо на виртуальный адрес 0 после завершения своего собственного выполнения.

Чтобы избежать сложностей, связанных с загрузкой вируса в начало программы, большинство вирусов являются вирусами с «задней» загрузкой. Они прикрепляют себя к концу, а не к началу выполняемой программы, изменяя в заголовке содержимое поля со стартовым адресом, чтобы оно указывало на начало вируса (рис. 24, *в*). Теперь вирус будет выполняться с разных виртуальных адресов (в зависимости от того, на какой зараженной программе он запущен), но все это означает, что Вирджилу нужно обеспечить свой вирус позиционной независимостью, используя относительные, а не абсолютные адреса. Для опытного программиста это не составит труда, а некоторые компиляторы могут делать это по запросу.

Сложные форматы исполняемых программ, такие как файлы с расширением .exe в Windows и практически все современные двоичные форматы в UNIX, позволяют программе иметь несколько сегментов текста и данных. Загрузчик собирает их в памяти и совершает перемещение. В некоторых системах (например, в Windows) размер всех сегментов (секций) кратен 512 байтам.

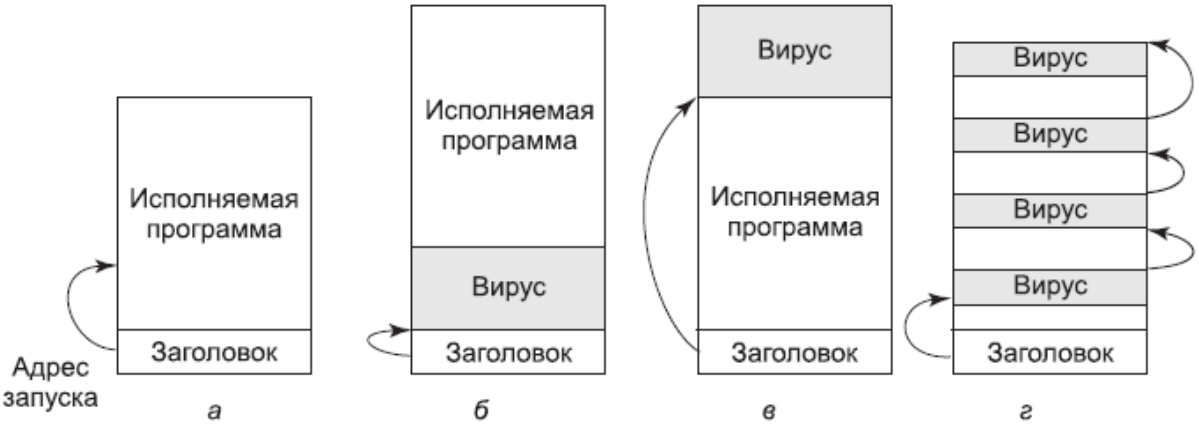


Рис. 9.24. *а* — исполняемая программа; *б* — с вирусом в начале; *в* — с вирусом в конце; *г* — с вирусом, распределенным по свободным местам программы

Если сегменты не заполнены, компоновщик заполняет их нулями. Разбирающиеся в этом вирусы могут попытаться спрятаться в этих пустотах. Если вирус помещается в них целиком (рис. 9.24, *г*), размер файла остается прежним, что будет несомненным плюсом, поскольку счастлив тот вирус, которому удалось спрятаться. Вирусы, использующие этот принцип, называются **пустотными**, или **заполняющими, вирусами** (cavity viruses). Конечно, если загрузчик не загружает пустующие области в память, вирусу потребуется иной способ запуска.

Резидентные вирусы

До сих пор мы считали, что при выполнении зараженной программы вирус запускается, передает управление настоящей программе, а затем завершает свою

работу. В отличие от этого **резидентные вирусы** (memory-resident virus) остаются в оперативной памяти на все время работы машины, либо спрятавшись в самых верхних адресах памяти, либо, возможно, «прячась в траве», среди векторов прерываний в нескольких сотнях байтов, остающихся, как правило, незадействованными. Самые изощренные вирусы могут даже изменить карту памяти операционной системы, заставив ее считать память, куда загрузился вирус, занятой, чтобы устранить угрозу ее перезаписи.

Типичный резидентный вирус перехватывает один из векторов системного или обычного прерывания за счет копирования содержимого в рабочую переменную и помещения в вектор собственного адреса, направляя это прерывание на себя. Лучше всего перехватить системное прерывание. В таком случае вирус получает возможность запускаться в режиме ядра при каждом системном вызове. Когда он завершает свою работу, он просто осуществляет настоящий системный вызов, передавая управление по сохраненному адресу системного прерывания.

А зачем вирусу запускаться при каждом системном вызове? Естественно, для того чтобы заражать программы. Вирус может просто выжидать, пока не поступит системный вызов *exec*, а затем, зная, что файл, имеющийся у него в распоряжении, является исполняемым двоичным (и, наверное, вполне для этого подходящим) файлом, заражает его. Для этого процесса не требуется такой большой активности диска, как при использовании кода, представленного в листинге 2, поэтому он меньше бросается в глаза. Перехват всех системных прерываний также дает вирусу огромные возможности для шпионского сбора информации и нанесения различного вреда.

Вирусы, поражающие загрузочный сектор

В главе 5 говорилось о том, что при включении большинства компьютеров программа BIOS считывает главную загрузочную запись с начала загрузочного диска в память и выполняет имеющуюся в ней программу. Эта программа определяет, какой из разделов активен, считывает первый загрузочный сектор из этого раздела и выполняет имеющуюся на нем программу. Эта программа затем либо загружает операционную систему, либо активизирует загрузчик операционной системы. К сожалению, много лет назад одного из соратников Вирджила осенила идея создать вирус, способный переписать главную загрузочную запись или загрузочный сектор. Такие вирусы, называющиеся **вирусами загрузочного сектора** (boot sector viruses), встречаются еще довольно часто.

Как правило, вирус загрузочного сектора (а в это понятие включены и вирусы главной загрузочной записи) сначала копирует настоящий загрузочный сектор в безопасное место на диске, чтобы иметь возможность загрузить операционную систему по окончании своей работы. Разработанная Microsoft программа форматирования диска fdisk пропускает первую дорожку, поэтому она является неплохим укромным местечком на Windows-машинах. Еще можно выбрать любой свободный сектор на диске, а затем обновить список сбойных секторов, чтобы пометить убежище как сбойный сектор. Вообще-то если вирус большой, то он может всего себя замаскировать под сбойные сектора. По-настоящему агрессивный вирус может даже выделить себе обычное дисковое пространство под настоящий

загрузочный сектор и себя самого и соответствующим образом обновить дисковый битовый массив или список свободных блоков. Такие действия требуют глубоких знаний структур внутренних данных операционной системы, но у Вирджила был хороший преподаватель по курсу операционных систем, и он сам прилежно учился.
При загрузке компьютера вирус копирует самого себя в оперативную память — либо в верхнюю ее часть, либо в нижнюю, среди неиспользуемых векторов прерываний. В этот момент машина находится в режиме ядра с выключенным блоком управления памятью, незагруженной операционной системой и незапущенными антивирусными программами. И вирусам здесь сплошное раздолье. Когда все готово, вирус загружает операционную систему, оставаясь обычно в памяти в качестве резидента и наблюдая за всем происходящим.
Есть все же одна проблема, заключающаяся в последующем получении управления. Обычно для этого используются специфические сведения о том, как операционная система управляет векторами прерываний. К примеру, Windows не переписывает разом все векторы прерываний. Вместо этого она по одному загружает драйверы устройств, и каждый из них захватывает необходимый ему вектор прерывания. Этот процесс занимает около минуты.
Это дает вирусу необходимые ему управляющие возможности. Он начинает с перехвата всех векторов прерываний (рис. 25, *а*). По мере загрузки драйверов некоторые из векторов переписываются, но если только драйвер часов не загрузится первым, то позже будет вполне достаточно прерываний от часов, которые запустят вирус. Потеря вирусом принтерного прерывания показана на рис. 25, *б*. Как только вирус увидит, что один из контролируемых им векторов прерываний переписан, он может переписать этот вектор опять, зная, что теперь это безопасно (на самом деле некоторые векторы прерываний в процессе запуска системы переписываются по нескольку раз, но все идет по вполне определенной схеме, и Вирджил это твердо усвоил). Перезахват принтера показан на рис. 25, *в*. Когда все будет уже загружено, вирус восстановит все векторы прерываний, а себе оставит только вектор прерываний, используемый системными вызовами. Теперь мы имеем дело с резидентным вирусом, контролирующим системные вызовы. Вообще-то именно так большинство резидентных вирусов обретают свое существование.

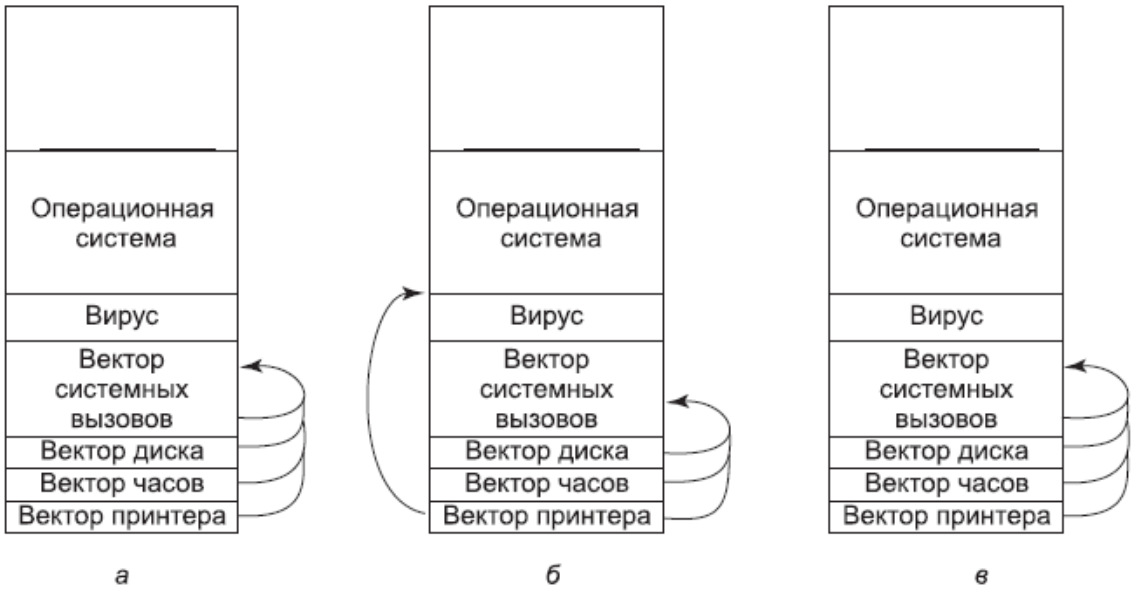


Рис. 25. Ситуация: *а* — после захвата вирусом всех векторов обычных и системных прерываний; *б* — после того, как операционная система отобрала вектор прерывания принтера; *в* — после того, как вирус заметил потерю вектора прерывания принтера и захватил его повторно

Вирусы драйверов устройств

Подобное проникновение в память немного напоминает спелеологию (изучение пещер) — вам нужно пройти по извилистому маршруту, опасаясь, что что-то упадет вам на голову. Было бы намного проще, если бы операционная система оказала любезность и загрузила вирус вполне официально. Приложив немного усилий, можно добиться желаемого результата. Фокус в том, что нужно заразить драйвер устройства, что наталкивает на мысль о **вирусе драйвера устройства** (device driver virus). В Windows и некоторых UNIX-системах драйверы устройств представляют собой обычные исполняемые программы, которые находятся на диске и загружаются в процессе запуска системы. Если один из них может быть заражен, то вирус всегда будет загружаться вполне официально во время запуска системы. Что еще лучше, драйверы работают в режиме ядра, и после того, как драйвер загружен, он будет вызван, давая вирусу шанс захватить вектор прерываний, используемый системными вызовами. Только один этот факт уже является сильным аргументом в пользу запуска драйверов устройств в виде программ, работающих в пользовательском режиме (как это делается в MINIX 3),
если они будут заражены, то они не смогут нанести такого же ущерба, как зараженные драйверы, работающие в режиме ядра.

Макровирусы

Многие программы, например Word и Excel, позволяют пользователям создавать макросы для объединения нескольких команд, которые позже можно выполнить одним нажатием клавиши. Макросы могут быть также подключены к пунктам меню и выполняться при выборе одного из таких пунктов. В Microsoft Office макросы могут содержать целые программы на полноценном языке программирования Visual Basic. Макросы не компилируются, а интерпретируются, но это влияет только на скорость выполнения, а не на их функциональность. Поскольку макросы могут

иметь отношение к определенному документу, Office хранит макросы для каждого документа вместе с документом.
А теперь рассмотрим суть проблемы. Вирджил создал документ в программе Word и макрос, подключенный к функции *OPEN FILE* (открыть файл). В этом макросе содержится **макровирус** (macro virus). Затем он посылает документ по электронной почте своей жертве, которая, естественно, его открывает (если предположить, что почтовая программа не сделала это за нее). Открытие документа приводит к выполнению макроса OPEN FILE. Поскольку макрос может содержать любую программу, он может делать что угодно: например, заражать другие документы Word, удалять файлы и делать многое другое. Следует честно признать, что Microsoft ввела в Word предупреждение, появляющееся, когда открывается файл с макросом, но большинство пользователей не понимают, что это означает, и все равно продолжают открытие файла. Кроме того, вполне законные документы тоже могут содержать макросы. Существуют программы, не выдающие даже такого предупреждения, что еще более затрудняет обнаружение вируса.
С ростом количества почтовых вложений отправка документов с вирусами, встроенными в макросы, упростилась. Намного проще создавать такие вирусы, чем скрывать настоящий загрузочный сектор где-нибудь в списке сбойных блоков, прятать вирусы среди векторов прерываний и захватывать вектор прерываний, используемый системными вызовами. Это означает, что теперь создание вирусов становится по силам куда менее образованным людям, чем прежде, снижающим общее качество продукта и приносящим создателям вирусов дурную славу.

Вирусы, заражающие исходные тексты программ
Паразитические вирусы и вирусы загрузочного сектора слишком зависимы от применяемой платформы; у вирусов, скрывающихся в документах, эта зависимость выражена несколько меньше (Word запускается на Windows и на Macintosh, но не на UNIX). Самыми переносимыми из всех существующих являются **вирусы исходного кода** (source code viruses**)**. Представьте себе вирус из листинга 9.2, но с модификацией, заставляющей искать не двоичные исполняемые файлы, а программы на языке С, для чего нужно изменить всего одну строчку (вызов процедуры *access*).
Процедура *infect* должна быть изменена для вставки строки

```
#include <virus.h>
```

в верхнюю часть каждого исходного текста программы на языке С. Для активации вируса нужна еще одна вставка — это строка

```
run virus( );
```

Для того чтобы решить, куда поместить эту строку, нужно провести синтаксический разбор кода на языке С, поскольку это должно быть место, которое позволяет осуществить вызов процедуры. Не будет работать строка, помещенная в середину комментария. Не лучшим выбором будет и помещение строки внутрь цикла. Если

предположить, что вызов может быть помещен в приемлемое место (например, перед самым концом процедуры *main* или перед оператором *return*, если таковой присутствует), то после компиляции программы в ней будет содержаться вирус, извлеченный из virus.h (хотя файл proj.h может привлечь меньше внимания, если кто-нибудь его увидит).

При работе программы будет вызван вирус, который может делать все что угодно, например, выискивать другие программы на языке С с целью их заражения. Если такая программа будет найдена, вирус может включить в нее всего лишь две строки, рассмотренные ранее, но это будет работать только на локальной машине, на которой, предположительно, уже установлен файл virus.h. Чтобы вирус работал на удаленной машине, должен быть включен полный исходный код вируса. Это может быть сделано за счет включения исходного кода вируса в виде инициализированной символьной строки, предпочтительно в форме списка 32-разрядных шестнадцатеричных целых чисел, чтобы никто не догадался, что это такое. Эта строка, наверное, будет слишком длинной, но с сегодняшними программными кодами, занимающими по несколько миллионов строк, ее легко не заметить.

Для неискушенного читателя все эти способы могут показаться слишком сложными. Кто-то может вполне резонно сомневаться в том, что все они могут работать на практике. Не сомневайтесь, могут. Вирджил — прекрасный программист, и у него уйма свободного времени. Хотите убедиться — откройте местную газету и найдите там подтверждение.

Как распространяются вирусы

Распространение может вестись по нескольким сценариям. Начнем с классического. Вирджил создает вирус, вставляет его в какую-нибудь написанную (или похищенную) им программу и приступает к ее распространению, поместив ее, к примеру, на веб-сайт с условно-бесплатным программным обеспечением. Со временем кто-нибудь скачает эту программу и запустит ее на выполнение. С этого момента возможны несколько вариантов. Для начала вирус, наверное, заразит несколько файлов на жестком диске на тот случай, если жертва чуть позже решит поделиться некоторыми из них со своими друзьями. Он также может попытаться заразить загрузочный сектор на жестком диске. После заражения загрузочного сектора откроется путь к формированию при последующих загрузках резидентного вируса, работающего в режиме ядра.

В настоящее время Вирджилу доступны и другие варианты. Вирус может быть создан для проверки подключения зараженной машины к локальной (возможно, беспроводной) сети, что весьма вероятно. Тогда вирус может приступить к заражению незащищенных файлов на всех подключенных к сети машинах.

Это заражение не будет распространяться на защищенные файлы, но с этим можно справиться, заставив зараженную программу работать необычным образом. Пользователь, запустивший такую программу, скорее всего, обратится за помощью к системному администратору. Затем администратор попытается сам поработать со странно ведущей себя программой, чтобы понаблюдать за результатами ее работы. Если администратор работает с этой программой, войдя в систему как привилегированный пользователь, то вирус получает возможность заразить

двоичные файлы системы, драйверы устройств, файлы операционной системы и загрузочные секторы. Нужна лишь небольшая ошибка, и будут скомпрометированы все машины в локальной сети.
Нередко машины, находящиеся в корпоративной сети, могут входить на удаленные машины через Интернет или по закрытой сети или даже удаленно выполнять команды без входа в систему. Такая возможность предоставляет вирусу новые пути распространения. Вот так одна нечаянная ошибка может привести к заражению всех машин компании. Чтобы предотвратить такое развитие событий, у всех компаний должна быть строгая установка, предписывающая администраторам никогда не допускать подобных ошибок.
Другой способ распространения вируса заключается в публикации зараженной программы в конференции сети USENET (например, Google) или на веб-сайте, на который постоянно выкладываются программы. Также можно создать веб-страницу, требующую для просмотра установку в браузере специального плагина, а затем позаботиться о заражении этого плагина.
Еще одна разновидность атаки заключается в инфицировании документа с последующей его пересылкой по электронной почте множеству людей или помещением в список рассылки в конференции сети USENET, как правило, в виде приложения к сообщению. Даже люди, совершенно не склонные к запуску программы, отправленной им каким-то незнакомцем, могут не понимать, что щелчок на ссылке для открытия приложения может открыть путь вирусу на их машину. Чтобы еще больше усугубить ситуацию, вирус может заглянуть в адресную книгу пользователя, а затем разослать себя по найденным там адресам, используя в качестве содержимого поля Тема (Subject) что-нибудь вполне логичное или интересное, например:

- Тема: Корректировка планов;
- Тема: Re: О последнем сообщении;
- Тема: Собака вчера вечером околела;
- Тема: Я серьезно болен;
- Тема: Я тебя люблю.

Когда приходит такое письмо, получатель думает, что отправитель письма — его друг или коллега по работе, и ни о чем не подозревает. А когда письмо открыто, уже слишком поздно. Вирус «I LOVE YOU», распространившийся по всему миру в июне 2000 года, действовал именно таким образом и нанес ущерб в несколько миллиардов долларов.
К распространению вирусов имеет отношение и распространение технологий создания вирусов. Существуют группы создателей вирусов, которые активно общаются через Интернет, помогая друг другу разрабатывать новые технологии, инструменты и вирусы. Большинство из них, скорее всего, любители, а не закоренелые преступники, но результаты их деятельности могут иметь поразительный эффект. Еще одна категория создателей вирусов — это военные, которые рассматривают вирусы как оружие, потенциально способное к выводу из строя компьютеров противника.

Еще один вопрос, имеющий отношение к распространению вирусов, заключается в том, как избежать обнаружения этого распространения. Общеизвестно, что в тюрьмах неважное компьютерное оснащение, поэтому Вирджил предпочтет не попадать в их стены. Выгрузка только что созданного вируса со своей домашней машины является весьма опрометчивым шагом. Если атака пройдет успешно, полиция может выследить его в результате поиска завирусованного сообщения с самой ранней меткой времени, поскольку оно, возможно, ближе всех находится к источнику атаки.

Чтобы меньше высовываться, Вирджил может пойти в интернет-кафе в отдаленном городе и там войти в систему. Он может принести вирус на флеш-накопителе USB и самостоятельно его считать или, если машина не оборудована портами USB, попросить молодую симпатичную девушку за стойкой считать файл book.doc, чтобы он мог его распечатать. После того как файл попадет на жесткий диск, он его переименует в virus.exe и запустит на выполнение, заражая всю локальную сеть вирусом, который активизируется месяц спустя на тот случай, если полиция вздумает сделать запрос в авиакомпании на получение списка всех пассажиров, прилетавших на прошлой неделе.

Еще один вариант — забыть о флеш-накопителе USB и извлечь вирус с удаленного веб-сайта или FTP-сайта. Или принести с собой ноутбук и подключить его к порту Ethernet, любезно предоставляемому администрацией интернет-кафе туристам, путешествующим с ноутбуком, которые хотят ежедневно читать свою электронную почту. Подключившись к сети, Вирджил может приступить к заражению всех подключенных к ней машин.

О вирусах можно еще долго рассказывать. В частности, о том, как они стараются спрятаться и как антивирусы стараются от них избавиться. Они могут даже прятаться внутри вполне здоровых животных. Не верите — посмотрите работу Rieback et al. (2006). Мы еще вернемся к этим темам, когда перейдем к рассмотрению вопросов защиты от вредоносных программ.

ОЖҚ. Дәріс №13. Құрттар. Тыңшы (шпиондық) программалар. Тыңшы программаларды тарату әдістері. Тыңшы программа жасайтын әрекеттер. Руткиттер

Интернетке қосылған компьютерлердің қауіпсіздігіне алғашқы қол сұғушылық 1988 жылы 2 қарашада кешке Корнелл университетінің аспиранты Роберт Таппан Моррис Интернетте жазған құртты іске қосқан кезде болды. Бұл университеттерде, корпорацияларда бірнеше мың компьютердің істен шығуына әкелді және бүкіл әлем бойынша үкіметтік зертханаларда құрт бақыланбады және жойылды. Бұл оқиға сонымен бірге жалғасып жатқан дау тудырды өңделуде. Әрі қарай, біз осы оқиғалардың егжей-тегжейін қарастырамыз. Қосымша техникалық ақпаратты Spafford (1989) мақаласынан алуға болады. Жанрдағы сол оқиға полиция триллері Hafner and Markoff (1991) кітабында баяндалған.

Мұның бәрі 1988 жылы Моррис операциялық бөлмеде екі қатені анықтаудан басталды Berkeley UNIX жүйесі бүкіл Интернет арқылы машиналарға рұқсатсыз кіруге мүмкіндік береді. Олардың бірі буфердің толып кету қатесі екенін тағы да көреміз. Жалғыз өзі ол өзін-өзі көбейтетін программа жасады бұл қателіктерді және сөзбе-сөз қолданатын құрт (құрт) бірнеше секунд ішінде ол қол жеткізе алатын барлық машиналарда көбейеді.

Ол бірнеше ай бойы программада жұмыс істеп, оны мұқият жөндеді және оған іздерін байқауға мүмкіндік береді.

1988 жылғы 2 қарашадағы нұсқа сынақ немесе қорытынды болды ма, жоқ па белгісіз, бірақ қалай болғанда да, ол шыққаннан кейін бірнеше сағат ішінде интернетке қосылған Sun және VAX жүйелерінің көпшілігін тізе бүктірді. Мотивация әрекеті Моррис те белгісіз, мүмкін ол бұл идеяны программалау қатесінің салдарынан пайда болған жоғары технологиялық ұтыс ойыны ретінде қарастырған шығар астында бақылау. Техникалық тұрғыдан құрт екі программадан тұрды: өзін-өзі жүктеу программасы және іс жүзінде құрт. Жүктеуші l1 деп аталатын файлға орналастырылған с тіліндегі 99 жол болды.с. бұл жүктеуші құрастырылып, шабуылдаушы машинада орындалды.

Іске қосылған кезде ол жүктелген машинамен байланысып, негізгі машинаны жүктеді құрт және оны іске қосты. Өзінің тіршілігін жасыру үшін бірнеше шаралар қабылдағаннан кейін, құрт өзінің жаңа қожайынының бағыттау кестелеріне қарады оның қандай машиналарға қосылғанын көріңіз және программаны таратуға тырысыңыз бұл машиналарға өздігінен тиеу.

Жаңа машиналарды жұқтыру үшін үш әдіс қолданылды. 1-әдіс rsh пәрменін пайдаланып жойылған қабықты іске қосу үшін. Кейбір компьютерлер басқа компьютерлерге сенеді және rsh-ді қажет етпестен іске қосуға мүмкіндік береді аутентификация. Егер бұл жұмыс істесе, жойылған қабық құртты қотарып, жаңа машиналарды жұқтыруды жалғастырды.

2-әдіс finger деп аталатын барлық UNIX жүйелерінде бар программаны қолданды. Бұл Интернетке қосылған Тұтынушыға пәрменді енгізуге мүмкіндік береді finger name@site адам туралы ақпаратты нақты берілген параметрлер бойынша көрсету. Бұл ақпаратқа әдетте оның нақты аты, тіркеу аты, Үй атауы кіреді және жұмыс мекен-жайы мен телефон нөмірлері, оның хатшысының аты және

телефон нөмірі, нөмірі факс және басқа ұқсас ақпарат. Бұл телефон кітапшасының электронды баламасы. Finger программасы келесідей жұмыс істейді. Әр UNIX машинасында finger daemon деп аталатын фондық үдеріс үнемі жұмыс істейді, ол жауап береді бүкіл интернеттен келіп түскен сұраулар. Құрт finger программасына жүгінді параметр ретінде арнайы жасалған 536 байттық жол. Бұл жол жын буферінің толып кетуіне әкеліп соқты және оның стек мазмұнын қайта жазды болды-суретте көрсетілген. 9.19, в. бұл жағдайда буфердің толып кетуін тексерудің болмауынан тұратын программа ақаулығы қолданылды. Демон программасы процедурадан оралған кезде, ол сол кезде сұраныс алды басқару негізгі программаға емес, процедураға қайтарылды ішінде 536 байт жол бар. Бұл процедура sh программасын іске қосуға тырысты. Егер ол сәтті болса, құрт жұмыс істейтін қабықты өз қолына алды шабуылға ұшыраған машинада.

3-әдіс sendmail пошта жүйесінде қате болған кезде қолданылды, бұл құртқа бастапқы жүктеушінің көшірмесін жіберіп, оны іске қосуға мүмкіндік берді.

Жүйеге кіргеннен кейін құрт пароль жүйесін бұзуға тырысты. Ол үшін Морриске өз зерттеулерін жүргізудің қажеті жоқ еді. Ол тек хабарласуы керек еді әкесіне, АҚШ Ұлттық қауіпсіздік департаментінің қауіпсіздік сарапшысы, қайта басып шығару үшін кодтарды бұзу бөлімінде жұмыс істеген осыдан он жыл бұрын Моррис ср және кен Томпсон (Моррис және Томпсон, 1979) Bell Labs зертханасында жазған осы тақырыптағы классикалық мақала. Әрқайсысы бұзылған пароль құртқа кез-келген машинаның кіруіне мүмкіндік берді пароль иесі тіркелді. Жаңа машинаға қол жеткізе отырып, құрт оның басқа іске қосылған көшірмесінің болуын тексерді. Егер құрт қазірдің өзінде болса, Жаңа көшірме жүйеден шықты ол өз жұмысын жалғастырған кезде жетіден бір жағдайды қоспағанда, мүмкін тіпті тарату қабілетін сақтауға тырысу, жүйелік әкімші осы машинада құрттың өз нұсқасын іске қосқан кезде нағыз құртты алдау мақсатында.

1/7 қатынасы тым көп құрттардың пайда болуына әкелді, бұл барлық жұқтырған машиналардың тоқтап қалуына себеп болды: олардың бәрі құрттармен бітелген. Егер Моррис бұл кәсіпті тастап кетсе және құрт жай болса мен басқа құртты тауып аламын (немесе қатынасы 1/50), содан кейін бұл құрт, мүмкін, қалды еді асырылды. Моррис достарының бірі ғылыми журналистпен сөйлескен кезде ұсталды "Нью-Йорк Таймс" редакциясы, Джон Марков және репортерды сендіруге тырысты, мұның бәрі тек апат, құрт зиянсыз және автор өте өкінеді болған оқиға туралы. Дос абайсызда RTM шабуылдаушысының тіркеу аты екенін айтты. Марков үшін RTM-ді физикалық атауға айналдыру қиын болған жоқ, тек finger программасын іске қосу керек болды. Келесі күні бұл оқиға барлық газеттердің алғашқы жолақтарында болды, тіпті алдағы туралы ақпаратты сол жерден шығарды президенттік сайлаудың үш күнінен кейін.

Моррис кінәлі деп танылып, федералды сотпен сотталды. Ол сотталды 10 000 доллар айыппұлға, 3 жылға бас бостандығынан айыруға (шартты түрде) және 400 сағаттық қоғамдық пайдалы жұмыс. Оның сот шығындары 150 000 доллардан асып кеткен шығар. Бұл айыптау көптеген дау тудырды. Компьютер қауымдастығындағы көптеген адамдар қиямет күні ол тамаша аспирантом, кімнің безобидная тентектік шыққан бақылау. Құрттағы ештеңе Моррис бірдеңе тырысты деп ойлаған жоқ ұрлау немесе зақымдау. Сондай-ақ, ол қауіпті қылмыскер және түрмеде отыруы керек

деген пікірлер болды. Кейін Моррис Гарвардта магистр дәрежесін алды және қазір Массачусетс технологиялық институтының профессоры.

Осы оқиғаның нәтижесінде компьютерлік жедел жәрдем тобы құрылды

Негізгі назар аударатын CERT (computer Төтенше жауап тобы) бұзу әрекеттері туралы хабарламаларға, сондай-ақ оның құрамында мамандар тобы бар қауіпсіздік мәселелерін талдау және оларды шешу әдістерін әзірлеу. Бұл болса да сөзсіз алға қадам, бірақ топтың қараңғы жағы болды. CERT шабуыл кезінде қолдануға болатын жүйенің ақаулары туралы ақпарат жинады және бұл ақауларды жою жолдары. Қажет болса, бұл ақпарат интернетте бірнеше мың жүйелік әкімшілерге кеңінен таратылды. Өкінішке орай, қаскүнемдер (жүйелік әкімшілер ретінде көрінуі мүмкін) ақаулар туралы есептерді ала алады және келесі сағаттарда (немесе тіпті күндер) олар жабылғанға дейін.

Со времен червя Морриса было выпущено множество других разнообразных червей.

Они работают по тому же сценарию, что и червь Морриса, только используют другие ошибки в других программах. Они распространяются намного быстрее вирусов, поскольку перемещают сами себя.

Тыңшы (шпиондық) программалар

Зиянды программалардың мұндай түрі барған сайын таныс болып келеді

шпиондық программалар (spyware). Қарапайым сөзбен айтқанда, тыңшылар-бұл программалар құпия түрде, компьютер иесінің білместен, олар оның машинасына жүктеледі және іске қосылады оның артында өзінің қара істерін жасайтын фонда. Бірақ таңқаларлық, беріңіз олар үшін дәл анықтау өте қиын. Мысалы, Windows Update программасы Windows машиналарына түзетулерді Тұтынушыға ескертусіз автоматты түрде жүктейді. Сол сияқты, көптеген антивирустық программалар автоматты түрде жұмыс істейдіфондық режимде жаңарту. Бұл программалардың ешқайсысы тыңшы болып саналмайды.

Басқалары нақты анықтама беруге тырысты (шпиондық программа, порнография емес). Барвински және басқалар (2006) шпиондық программаның төрт сипаттамасы бар екенін айтты. Біріншіден, ол жасырынып жатыр, сондықтан жәбірленушіге оны табу оңай емес. Екіншіден, ол Тұтынушы туралы деректерді жинайды (кірілген веб -сайттар, құпия сөздер және тіпті несие карталарының нөмірлері). Үшіншіден, ол жиналған ақпаратты өзінің алыстағы хостына жібереді. Төртіншіден, ол оны автокөліктен алып тастау әрекетінен аман қалуға тырысады. Сонымен қатар, кейбір шпиондық программалар төменде сипатталған параметрлерді өзгертеді және басқа қауіпті және тітіркендіргіш әрекеттерді орындайды.

Барвински (Barwinski et al., 2006) шпиондық программаларды үш негізгі категорияға бөледі. Тыңшылардың бірінші категориясы - маркетинг: тыңшылар ақпаратты жинайды және оны иесіне жібереді, әдетте белгілі бір машиналарға мақсатты жарнаманы жібереді. Шпиондық программалардың екінші санаты - бұл қадағалау, олар компаниялар не істеп жатқанын және қандай веб -сайттарға кіретінін бақылау үшін қызметкерлерінің машиналарына әдейі орнатады. Үшінші санатты классикалық зиянды программа ретінде жіктеу ықтималдығы жоғары, соның

арқасында жұқтырған компьютер өз қожайынынан тапсырыс күтетін зомби машиналарының армиясына қосылады.

Қандай веб-сайттарда шпиондық программа бар екенін анықтау үшін эксперимент жүргізілді және 5000 веб-сайтқа кірді. Ересектерге арналған ойын-сауыққа, ұрланған программалық қамтамасыз етуді таратуға, интернет-турагенттіктер мен жылжымайтын мүлікке қатысты веб-сайттар шпиондық программалық қамтамасыз етудің негізгі жеткізушілері болып табылды.

Неғұрлым терең зерттеу Вашингтон университетінде жүргізілді (Мощук және т.б., 2006). Сонымен бірге 18 миллион URL тексерілді, олардың 6 пайызында шпиондық программа табылды. AOL / NCSA келтірген зерттеуде үй компьютерлерінің 80% -ы шпиондық программамен жұқтырғаны таңқаларлық емес, әр компьютерге 93 шпиондық программа. Вашингтон университетінің зерттеуі ересектерге арналған сайттар, атақты сайттар мен жұмыс үстеліндегі тұсқағаздарға арналған сайттардың шпиондық көрсеткіштері жоғары екенін көрсетті, бірақ олар туристік агенттіктер мен жылжымайтын мүлік сайттарына қарамады.

Тыңшы программаларды тарату әдістері

Табиғи сұрақ туындайды: компьютерлер шпиондық программаны қалай жұқтырады? Инфекция жолдарының бірі барлық зиянды программалармен бірдей: трояндық жылқылар арқылы. Тыңшылар ақысыз программалық қамтамасыз етудің едәуір бөлігінде бар, олардың авторлары шпиондық программадан ақша табады. программалармен еркін алмасуға арналған біртұтас программалық жасақтама

1 Поттер Стюарт 1958-1981 жылдары АҚШ Жоғарғы сотының судьясы болды. Ол енді порнография туралы көпшіліктің пікірімен сәйкес келетін өз пікірін жазып, оны анықтай алмайтынын мойындады, бірақ сонымен бірге: «Мен оны көрген бойда бірден білемін».

9.9. 747

(мысалы, Қазаа), барлаушылармен қапталған. Сонымен қатар, көптеген веб-сайттарда шпиондық программа жұқтырған веб-беттерге апаратын баннерлік жарнамалар көрсетіледі.

Инфекцияның басқа жолы жиі жүктеу деп аталады. Шпиондық программаны (немесе кез келген зиянды программаны) вирус жұққан веб-бетке үнемі кіргеннен кейін алуға болады. Инфекцияның үш түрі бар. Біріншіден, веб -браузер браузерді орындалатын (.exe) файлға қайта бағыттайды. Браузер файлмен кездескенде, ол Тұтынушыны программаны іске қосқысы немесе сақтағысы келетінін сұрайтын диалогтық терезені шығарады. Нақты жүктеулер бір механизмді қолданатындықтан, көптеген Тұтынушылар іске қосу түймесін басып, шолғышты программаны жүктеуге және орындауға мәжбүрлейді. Бұл машинаны жұқтырады және шпиондық программа қалаған нәрсені жасай алады.

Екінші нұсқа вирус жұққан құралдар тақтасына қатысты. Internet Explorer де, Firefox та үшінші тараптың құралдар тақтасын қолдайды. Кейбір шпиондық программалар жасаушылар көптеген пайдалы қасиеттері бар тартымды бақылау тақтасын жасайды, содан кейін оны керемет қосымша ретінде кеңінен жарнамалайды. Бақылау тақтасын орнатқан кез келген адам тыңшы алады.

Мысалы, тыңшы әйгілі Alexa құралдар тақтасында бар. Шын мәнінде, бұл схема - бұл трояндық жылқы, тек басқа пакетте.

Инфекцияның үшінші жолы жанама. Көптеген веб -беттерде Microsoft жасаған ActiveX басқару деп аталатын технология қолданылады. Бұл басқару элементтері x86 үдерісорының екілік файлдары болып табылады, олар Internet Explorer-ге кіреді және оның функцияларын кеңейтеді, мысалы, суреттердің, дыбыстардың немесе бейнеклиптердің белгілі бір түрлерін веб-беттерде көрсету. Негізінде бұл технология мүлдем заңды. Бірақ іс жүзінде бұл өте қауіпті және, бәлкім, шпиондық программалық қамтамасыз етудің енуінің негізгі әдісі болып табылады. IE (Internet Explorer) әрқашан осы тәсілдің мақсаты болып табылады, бірақ Firefox, Chrome, Safari немесе басқа браузерлер емес.

ActiveX басқару элементтері бар бетке кіргенде, барлық жағдай IE қауіпсіздік параметрлеріне байланысты болады. Егер параметрлерде қауіпсіздік деңгейі тым төмен орнатылса, шпиондық программа автоматты түрде жүктеледі және орнатылады. Қауіпсіздіктің төмен деңгейін орнатудың себебі, жоғары деңгейді орнату кезінде көптеген веб -сайттар дұрыс көрсетілмегендігінде (егер олар, әрине, көрсетілмесе) немесе ЖК үнемі Тұтынушы қолданатын белгілі бір әрекеттерге рұқсат сұрауында. түсінбейді.

Енді Тұтынушы қауіпсіздік деңгейін тым жоғары етіп қойды делік. Вирус жұқтырған веб-бетке кіргенде, IE activeX бақылауын анықтайды және веб-бетте берілген хабарды қамтитын диалогтық терезені көрсетеді. Ол былай деуі мүмкін: Интернетке кіруді жылдамдататын программаны орнатқыңыз және іске қосқыңыз келе ме? Көбісі бұл пайдалы идея деп ойлайды, Иә түймесін басыңыз және машина жұқтырылады. Тәжірибелі қолданушылар диалогтық терезеде қалған екі ақпаратты тексере алады, оларда тағы екі жазба бар. Олардың бірі - веб-беттің сертификатына сілтеме (ЭЦҚ бөлімінде талқыланған) олар естімеген кейбір сертификаттау органы (CA) қойған. Бұл жазбада пайдалы ақпарат жоқ, тек ОА сертификатты алуға жеткілікті қаражаты бар компанияның бар екенін растайды. Тағы бір жазба - кірілген веб-бетте берілген басқа веб-бетке гиперсілтеме. Бұл ActiveX басқару элементтері не істейтінін түсіндіруге арналған, бірақ шын мәнінде бұл ActiveX басқару элементтерінің қаншалықты жақсы екендігі және олар сіздің Интернет-серфинг тәжірибеңізді қаншалықты жақсартатыны туралы жалпы ақпарат болады. Осындай жаңылыстыратын ақпаратты ескере отырып, тіпті күрделі қолданушылар да Иә түймесін жиі басады.

Егер олар Жоқ түймесін басса, веб-беттегі сценарий шпиондық программаны бәрібір жүктеп алу үшін IE ақауын пайдаланады. Егер ол сәйкес ақаулық таппаса, ол ActiveX басқару элементін қайта -қайта жүктеуге тырысуы мүмкін, бұл IE-дің сол диалогтық терезені әр уақытта көрсетуіне әкеледі. Көптеген тұтынушылар не істеу керектігін білмейді (олар Тапсырмалар менеджеріне барып, IE жұмысын тоқтатуы керек), сондықтан олар ақырында бас тартады және Иә түймесін басады және машина жұқтырады.

Көбінесе осыдан кейін шпиондық программа 20-30 ғасырлық лицензиялық келісімнің ағылшын тіліндегі ортағасырлық ақын Джеффри Чосерге ғана таныс тілде жазылғанын көрсетеді, бірақ оның заңгерлік кәсіпке қатысы жоқ ұрпақтарына емес. Тұтынушы лицензиялық келісімді қабылдағаннан кейін, ол программаны

бақылаусыз жүргізуге келіскендіктен, шпиондық программалық жасақтама сатушысына шағымдану құқығынан айырылуы мүмкін (кейде жергілікті заңдар мұндай лицензияларды қайтарып алады). Егер лицензияда: «Лицензиат осылайша, лицензияны қайтарып алу мүмкіндігінсіз лицензия берушіге лицензиаттың анасын өлтіруге және мұрагерлік құқығын ұсынуға құқық береді» делінген болса, лицензиат сотта дәлелдемелермен қиналады. ол лицензия алушы оның мәтінімен келіскеніне қарамастан мұрагерлікке келеді.

Тыңшы программа жасайтын әрекеттер

Енді шпиондық программа әдетте не істейтінін қарастырайық. Төмендегі тізімдегі барлық элементтер жалпы сипатта:

1. Браузердің бастапқы бетін өзгерту;
2. Браузердегі сүйікті беттер тізімін (бетбелгілер жасалатын) өзгерту;
3. Браузерге жаңа құралдар тақтасын қосу;
4. Тұтынушының әдепкі медиа ойнатқышын өзгерту;
5. Тұтынушының әдепкі іздеу жүйесін өзгерту;
6. Windows жұмыс үстеліне жаңа белгішелерді қосу;
7. Веб-беттердегі жарнамалық баннерді шпиондық программа таңдалғанмен ауыстыру;
8. Windows стандартты диалогтық терезелерінде жарнамаларды орналастыру;
9. Тоқтатуға болмайтын қалқымалы жарнамалық хабарлардың тұрақты ағынын генерациялау.

Первые три пункта изменяют поведение браузера, как правило, таким образом, что даже перезапуск системы не в состоянии восстановить прежние значения. Такая атака известна как **мягкое ограбление браузера** (browser hijacking). Мягкое – потому что есть ограбления и похуже. Действия из двух следующих пунктов этого списка изменяют установки реестра, переключая ничего не подозревающего пользователя на использование другого медиаплеера (который отображает ту рекламу, которая нужна программе-шпиону) и другой поисковой машины (возвращающей ссылки на те веб-сайты, которые нужны программе-шпиону). Добавление значков на рабочий стол является явной попыткой заставить пользователя запустить только что установленную программу. Замена рекламного баннера (изображения в GIF-формате размером 468 × 60) на последующих веб-страницах создает впечатление, что посещаемые веб-сайты рекламируют сайты, выбранные программой-шпионом. Но самое раздражающее действие представлено в последнем пункте: всплывающая реклама, которая может быть закрыта, но которая тут же генерирует другую рекламу *ad infinitum* (до бесконечности), и остановить этот процесс невозможно. Кроме этого, программы-шпионы иногда отключают брандмауэр, удаляют конкурирующую программу-шпиона и наносят другой вред.

Многие программы-шпионы поставляются с деинсталляторами, но они редко работают, поэтому неопытные пользователи не могут их удалить. К счастью, освоена новая индустрия, занимающаяся производством антишпионского программного обеспечения, в которое вовлечены и

существующие антивирусные фирмы. Но граница между законными программами и программами-шпионами по-прежнему размыта.

Программы-шпионы не следует путать с **бесплатным программным продуктом с размещенной в нем рекламой** (adware), когда вполне законные (но небольшие) поставщики программного обеспечения предлагают две версии своего продукта: с рекламой, но бесплатную, и без рекламы, но платную. Эти компании не скрывают существования двух версий и всегда предлагают пользователям выбрать платное обновление, позволяющее избавиться от рекламы.

Руткиттер

Руткит (rootkit) – это программа или набор программ и файлов, пытающихся скрывать свое присутствие, даже если владелец зараженной машины прикладывает определенные усилия к их розыску и удалению. Как правило, руткиты содержат вредоносные программы, которые также скрыты. Руткиты могут быть установлены с использованием одного из рассмотренных ранее методов, включая методы установки вирусов, червей и программ-шпионов, а также другими способами, один из которых будет рассмотрен далее.

Разновидности руткитов

Рассмотрим пять разновидностей существующих в настоящее время руткитов. В каждом случае нас будет интересовать вопрос: «Где прячется руткит?».

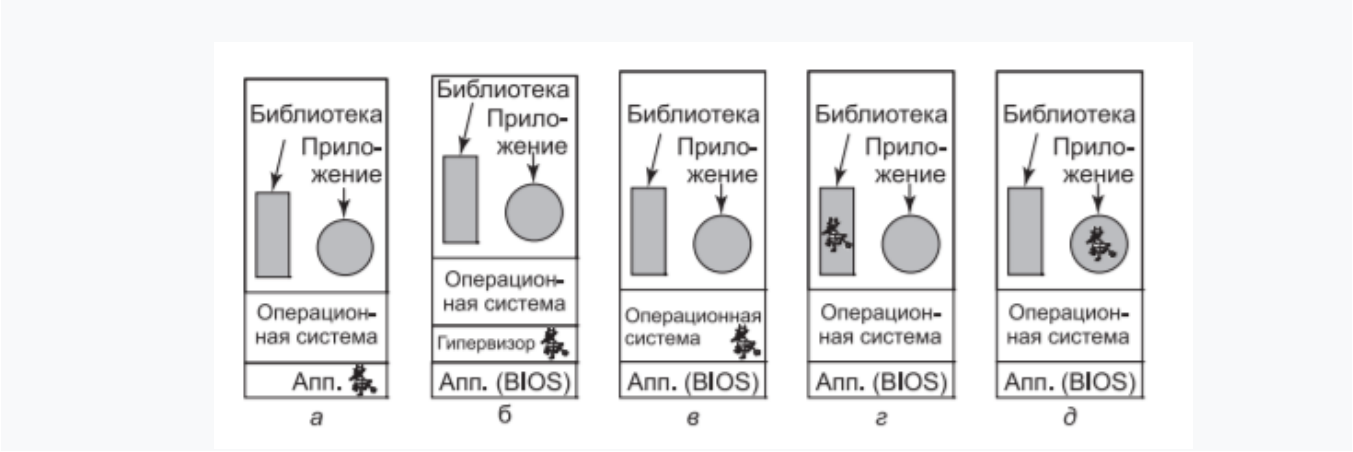
1. **Руткиты во встроенном программном обеспечении**. По крайней мере теоретически руткиты могут прятаться за счет перезаписи BIOS той копией, в которой они находятся. Такие руткиты будут получать управление при каждом запуске машины, а также при каждом запуске функции BIOS. Если после каждого применения руткит сам себя шифрует, а перед каждым применением дешифрует, то его будет крайне сложно обнаружить.
2. **Руткиты-гипервизоры**. Очень коварная разновидность руткитов, способная запускать всю операционную систему и все приложения на виртуальной машине под своим управлением. Первое доказательство существования этой концепции, ее синяя таблетка (как в фильме «Матрица»), было продемонстрировано польским хакером Джоанной Рутковской (Joanna Rutkowska) в 2006 году. Эта разновидность руткитов, как правило, модифицирует загрузочную последовательность, чтобы при включении питания в роли надстройки над оборудованием выполнялся гипервизор, который затем запустит операционную систему и ее приложения на виртуальной машине. Сильной стороной этого, как и предыдущего, метода встраивания руткита является то, что ничего не скрывается ни в операционной системе, ни в библиотеках или программах, поэтому детекторы руткитов, ведущие в них поиск, останутся ни с чем.
3. **Руткиты в ядре**. В настоящее время к самой распространенной

разновидности руткитов относятся те из них, которые заражают операционную систему и прячутся в ней в виде драйверов устройств или загружаемых модулей ядра. Руткит может запросто заменить большой, сложный и часто изменяемый драйвер на новый, в котором содержится старый драйвер плюс сам руткит.

4.**Руткиты в библиотеках**. Другим местом, где может прятаться руткит, является системная библиотека, например libc в Linux. Такое размещение дает вредоносной программе возможность проверять аргументы и возвращать значения системных вызовов, изменяя их по своему усмотрению и оставаясь незамеченной.

5.**Руткиты в приложениях**. Еще одним местом, где прячутся руткиты, являются большие прикладные программы, особенно те, которые в процессе своей работы создают множество новых файлов (профили пользователей, изображения предпросмотра и т. д.). Эти новые файлы являются неплохим местом, где можно укрыться, не вызвав ни у кого удивления самим фактом существования этих файлов.

Пять мест, где могут прятаться руткиты, показаны на рис. 26.



26-сурет. Руткит жасыра алатын бес орын

Еще один класс методов обнаружения основан на хронометрировании, особенно виртуализированных устройств ввода-вывода. Предположим, что считывание какого-нибудь регистра устройства PCI на настоящей машине занимает 100 тактов и это время имеет высокую степень повторяемости. В виртуальной среде значение этих регистров поступает из памяти, и время их считывания зависит от того, откуда оно производится: из принадлежащей центральному процессору кэш-памяти первого уровня, из кэш-памяти второго уровня или из самой оперативной памяти. Программа обнаружения может без особого труда заставить это значение перемещаться вперед и назад между этими состояниями и замерять разницу между показателями времени считывания. Учтите, что эта изменчивость вызвана определенными причинами, а не изменением самого времени считывания.

Еще одна область, в которой можно провести замеры времени, – это выполнение привилегированных команд, особенно тех, которые на реальном

оборудовании требуют всего лишь нескольких тактов и сотни или тысячи тактов, когда их требуется эмулировать. Например, если считывание значений некоторых защищенных регистров центрального процессора на настоящем оборудовании занимает 1 нс, то не существует способа осуществления миллиарда системных прерываний и эмуляций за 1 с. Конечно, гипервизор может обмануть программу обнаружения, сообщая об эмулированном, а не о реальном времени на всех критичных по времени системных вызовах. Программа обнаружения может проигнорировать эмулированное время, подключившись к удаленной машине или веб-сайту, предоставляющему точный масштаб времени. Поскольку программе обнаружения нужно измерять только интервалы времени (например, сколько времени займет выполнение миллиарда считываний значений защищенных регистров), разница во времени между локальными и удаленными часами не имеет значения.

Если между аппаратурой и операционной системой не проскользнул какой-нибудь гипервизор, руткит может прятаться внутри операционной системы. Его трудно обнаружить, загружая компьютер, поскольку операционной системе нельзя доверять. Например, руткит может установить большое количество файлов, все имена которых начинаются на «$$$», и, читая содержимое каталогов от имени пользовательских программ, никогда не сообщать о существовании таких файлов.

Один из способов обнаружения руткита при таких обстоятельствах заключается в загрузке компьютера с надежного внешнего носителя, такого как подлинный DVD или флеш-накопитель USB. После этого диск может быть просканирован программой, разработанной для борьбы с руткитами, без опасений, что руткит вмешается в сканирование. В качестве альтернативы можно сделать криптографический хэш каждого файла операционной системы и сравнить его с тем списком, который был сделан при установке системы и сохранен вне ее, в недоступном месте. Если такие хэши с оригинала не делались, они могут быть вычислены с установочного флеш-накопителя USB, компакт-диска или DVD или можно сравнить сами файлы.

Руткиты в библиотеках и прикладных программах спрятать труднее, но если операционная система была загружена с внешнего носителя и ей можно доверять, их хэши также можно сравнить с заведомо хорошими и сохраненными на флеш-накопителе USB или компакт-диске.

До сих пор речь шла о пассивных руткитах, которые не вмешиваются в работу программы их обнаружения. Но есть еще и активные руткиты, которые отыскивают и уничтожают эти программы или по крайней мере модифицируют их, чтобы они всегда объявляли: «Руткиты не найдены!». Для этого требуется предпринять более сложные меры воздействия, но, к счастью, активные руткиты на нашем горизонте еще не появились.

Обнаружение руткитов

Если нет доверия к аппаратному обеспечению, операционной системе,

библиотекам и приложениям, то руткиты обнаружить довольно трудно. Например, вполне очевидным способом поиска руткитов является создание листингов всех файлов, находящихся на диске. Но системные вызовы, читающие каталог, библиотечные процедуры, осуществляющие системные вызовы, и программы, осуществляющие листинг, — все могут быть частью вредоносного программного обеспечения и проверять результат, опуская все файлы, имеющие отношение к руткитам. И все же ситуация не безнадежна.

Обнаружить руткиты, запускающие собственный гипервизор, а затем операционную систему и все приложения на виртуальной машине под своим управлением, довольно сложно, но все же возможно. Для этого нужно внимательно присмотреться к малейшим различиям в производительности и функциональности между виртуальной и реальной машинами. Гарфинкель (Garfinkel et al., 2007) предложил ряд таких способов, рассмотренных далее. Карпентер (Carpenter et al., 2007) также рассматривал эту тему.

Целый класс методов обнаружения основан на том факте, что гипервизор сам по себе потребляет физические ресурсы и потери этих ресурсов могут быть обнаружены. Например, гипервизор нуждается в использовании некоторых записей TLB, конкурируя с виртуальной машиной за обладание этими дефицитными ресурсами. Программа обнаружения может оказать давление на TLB, понаблюдать за производительностью и сравнить ее с ранее замеренной производительностью на «голом» оборудовании.

Существует два мнения о том, что нужно делать с обнаруженными руткитами. Согласно одному из них, системный администратор должен уподобиться хирургу, удаляющему раковую опухоль: вырезать все с максимальной осторожностью. Приверженцы другого мнения говорят, что пытаться удалять руткиты слишком опасно. У них могут остаться скрытые составляющие. С этой точки зрения единственным решением может быть возвращение к последней резервной копии, о которой заведомо известно, что она не заражена. Если резервной копии нет, потребуется переустановка системы.

Руткит компании Sony

В 2005 году компания Sony BMG выпустила определенное количество аудиоком- пакт-дисков, содержащих руткит. Он был обнаружен Марком Руссиновичем (Mark Russinovich) (соучредителем веб-сайта, посвященного инструментарию системных администраторов Windows, www.sysinternals.com), который работал над разработкой программы по обнаружению руткитов и был очень удивлен, обнаружив руткит в собственной системе. Он сообщил об этом в своем блоге, и вскоре эта история распространилась по Интернету и средствам массовой информации. Об этом были написаны научные статьи (Arnab and Hutchison, 2006; Bishop and Frincke, 2006; Felten and Halderman, 2006; Halderman and Felten, 2006; Levine et al., 2006). На то, чтобы поднявшаяся волна улеглась, понадобились годы.

Далее будет дано краткое описание случившегося.

Когда пользователь вставляет компакт-диск в привод компьютера с операционной системой Windows, эта система ищет файл под названием autorun.inf, который содержит перечень предпринимаемых действий, как правило, запуск имеющейся на компакт-диске программы (например, мастера установки программы). Обычно на компакт-дисках нет таких файлов, поскольку автономные проигрыватели компакт-дисков игнорируют их присутствие. Вероятно, некий гений в компании Sony подумал, что можно будет по-умному остановить музыкальное пиратство, поместив файл autorun.inf на выпускаемые ими компакт-диски, которые, будучи вставленными в компьютер, немедленно и молча устанавливают на него руткит размером 12 Мбайт. Затем отображалось лицензионное соглашение, в котором об установленном программном обеспечении ничего не говорилось. Во время отображения лицензионного соглашения программа Sony проверяла компьютер на наличие какой-нибудь из 200 известных программ копирования, и если проверка была удачной, программа требовала их остановки. Если пользователь соглашался с лицензионными условиями и останавливал все программы копирования, музыка могла воспроизводиться, а если нет, музыка не воспроизводилась. Даже если пользователь отказывался соглашаться с условиями лицензии, руткит оставался установленным.

Руткит работал следующим образом. Он вставлял в ядро Windows определенное количество файлов, чьи имена начинались с sys. Одним из файлов был фильтр, перехватывающий все системные вызовы к приводу компакт-дисков и запрещающий всем программам, кроме музыкального плеера Sony, читать компакт-диск. Это делало невозможным вполне законное копирование компакт-диска на жесткий диск. Еще один фильтр перехватывал все вызовы, предназначенные для чтения файла, процессы и листинги реестра и удалял все записи, начинающиеся с sys (даже из программ, совершенно не связанных с Sony и музыкой), чтобы скрыть руткит. Это весьма стандартный прием у начинающих разработчиков руткитов.

До того как Руссинович обнаружил руткит, он уже был установлен на многих машинах, что неудивительно, поскольку он присутствовал на более чем 20 млн компакт-дисков. Дэн Каминский (Dan Kaminsky, 2006) изучил границы его распространения и обнаружил, что руткитом были заражены компьютеры в более чем 500 000 сетей по всему миру.

Когда новость распространилась по миру, первой реакцией компании Sony стало заявление, что каждый имеет право на защиту интеллектуальной собственности. В интервью радиостанции National Public Radio Томас Хессе (Thomas Hesse), президент глобального цифрового подразделения Sony BMG, сказал: «Большинство людей, я думаю, даже не знают, что такое руткит, так стоит ли им переживать?» Когда сам этот ответ вызвал целый шквал возмущений, Sony отступила и выпустила «заплатку», которая удаляла замаскировавшиеся sys-файлы, но не трогала сам руткит. Под растущим

давлением Sony в конце концов выложила на своем веб-сайте деинсталлятор, но чтобы получить его, пользователи должны были предоставить адрес своей электронной почты и согласиться с тем, что Sony сможет посылать им в будущем рекламные материалы (которые многие называют спамом).

Когда история стала уже забываться, выяснилось, что предоставленный Sony деинсталлятор содержал технические дефекты, делающие зараженный компьютер очень уязвимым к атакам через Интернет. Также обнаружилось, что руткит содержал код из проектов с открытым кодом в нарушение их авторских прав (которые разрешали свободное использование программ *при условии свободного распространения их исходного кода*).

Вдобавок к беспрецедентному общественному скандалу компания Sony понесла юридическую ответственность. Штат Техас предъявил ей иск за нарушение своего закона против программ-шпионов, а также законов о честном ведении торговли (поскольку руткит устанавливался даже при несогласии с условиями лицензии). Позже были поданы коллективные иски в 39 штатах. В декабре 2006 года эти иски были улажены, когда компания Sony согласилась заплатить 4,25 млн долларов, прекратить включать руткит в выпускаемые в будущем компакт-диски и предоставить каждому пострадавшему право скачать три альбома из ограниченного числа музыкальных каталогов. В январе 2007 года Sony признала, что в нарушение законов США ее программное обеспечение также тайно отслеживало пользовательские музыкальные пристрастия и отправляло отчеты компании. По условиям сделки с Федеральной торговой комиссией компания Sony согласилась выплатить тем людям, чьи компьютеры были повреждены ее программным обеспечением, компенсацию в размере 150 долларов.

История с руткитом компании Sony была приведена для тех читателей, у которых могло сложиться мнение, что руткиты не более чем любопытный учебный материал, не имеющий отношения к реальному миру. Большой объем дополнительной информации об этом рутките можно получить в Интернете, введя в поисковую строку «Sony rootkit».

Дәріс №14. Инсайдерлік шабуылдар. Логикалық (қисындық) бомбалар. Зиянкестер кіретін саңылау (тесік – лазейки). Жүйеге кіруді жалған ету. Зиянды программалар. Троян аттары.

Инсайдерлік шабуылдар

Мәселелердің мүлдем басқа санатын бағдарламашылар мен компанияның компьютерде жұмыс істейтін немесе бағдарламалық жасақтаманың маңызды компоненттерін құратын басқа да қызметкерлері орындайтын ішкі диверсиялық жұмыс деп атауға болады. Бұл шабуылдар сыртқы шабуылдардан ерекшеленеді, өйткені олардың мамандары (инсайдерлер — insiders) арнайы білімі мен рұқсаты бар, олар бөтен адамдарда жоқ (сырттан келгендер — outsiders). Төменде бірнеше мысалдар келтірілген, олардың әрқайсысы бұрын бірнеше рет қайталанған. Олардың әрқайсысының өз ерекшеліктері бар, олар шабуыл жасайтындарға, осы шабуыл кімге бағытталған және шабуылдаушы қол жеткізуге тырысады.

Логикалық бомбалар

Қазіргі уақытта сыртқы мамандарды жұмысқа кеңінен тарту кезінде бағдарламашылар өз орындары үшін жиі алаңдайды. Кейде олар өздерінің ықтимал (мәжбүрлі) зейнетке шығуын аз ауырлату үшін қадамдар жасайды. Бопсалауға бейім адамдар логикалық бомбаларды (логикалық бомба) жазудан шығудың жолын табады. Мұндай бомба-бұл компанияда жұмыс істейтін бағдарламашылардың бірі жасаған және өндіріс жүйесіне жасырын енгізілген бағдарламалық кодтың үзіндісі. Бағдарламашы күн сайын оған парольді енгізіп жатқанда, оған бұл жеткілікті және ол ештеңе істемейді. Бірақ егер бағдарламашы кенеттен жұмыстан босатылып, өндірістік бөлмеден шығарылса, келесі күні (немесе келесі аптада) логикалық бомба күнделікті енгізілген паролінсіз қалады және "жарылып кетеді". Сол тақырыпта басқа да өзгерістер болуы мүмкін. Белгілі жағдайлардың бірінде логикалық бомба төлемді тексерді. Егер бағдарламашының жеке нөмірі екі төлем кезеңінде пайда болмаса, бомба "жарылды" (Spafford et al., 1989).

"Жарылыс" деп аталатын нәрсе дискіні тазарту, файлдарды кездейсоқ өшіру, негізгі бағдарламаларға түсініксіз өзгерістер енгізу немесе маңызды файлдарды шифрлау болуы мүмкін. Соңғы жағдайда, компанияға таңдау жасау қиын: полицияға қоңырау шалу (ол бірнеше айдан кейін кінәліні соттауға әкелуі мүмкін немесе болмауы мүмкін, бірақ содан кейін жоғалған файлдарды қалпына келтіру мүмкін болмайды) немесе бопсалау және проблеманы жою үшін астрономиялық сомаға кеңесші ретінде бағдарламашыны қайта жалдау (оны жою кезінде ол жаңа логикалық бомба қоймайды деп үміттенеді).

Вирус жұқтырған компьютерге логикалық бомба салған жағдайлар тіркелді. Әдетте, компьютерлер белгілі бір күн мен сағатта бәрін бірге "жару" үшін бағдарламаланған. Бірақ бағдарламашы кімнің компьютері ұрылатындығын алдын-ала

білмейтіндіктен, логикалық бомбаларды жұмыс орнын сақтау немесе бопсалау үшін пайдалануға болмайды. Көбінесе мұндай бомбалар саяси маңызды күнде "жарылысқа" бейімделеді. Кейде оларды сағат бомбалары деп атайды (time bombs).

Қалмауы

Инсайдерлер ұйымдастыратын басқа қауіпсіздік саңылаулары-бұл Бос орындар (backdoor) . Оларды жүйелік бағдарламашылар жүйеге әдеттегі тексеруді айналып өтуге мүмкіндік беретін кодты енгізу арқылы жасайды. Мысалы, бағдарламашы пароль файлында не бар екеніне қарамастан, кез-келген адамға zzzzz тіркеу атауын қолдана отырып кіруге мүмкіндік беру үшін кіру бағдарламасына код қоса алады. Кіру бағдарламасының әдеттегі коды суретте көрсетілгендей көрінуі мүмкін. 22, а. саңылауды енгізу үшін код суретте көрсетілгенге өзгереді. 9.22, б.

```
while (TRUE) {
        printf("login: ");
        getstring(name);
        disableechoing( );
        printf("password: ");
        getstring(password);
        enableechoing( );
        v = checkvalidity(name, password);
        if (v) break;
}
executeshell(name);
```

а

```
while (TRUE) {
        printf("login: ");
        getstring(name);
        disableechoing( );
        printf("password: ");
        getstring(password);
        enableechoing( );
        v = checkvalidity(name, password);
        if (v || strcmp(name, "zzzzz") == 0) break;
}
executeshell(name);
```

б

22-сурет. Код: *а* — обычный; *б* — с внедренной лазейкой

Strcmp процедурасы шақырылған кезде, тіркеу атауы zzzzz енгізілмегені тексеріледі. Егер бұл атау енгізілсе, кіру қай пароль енгізілгеніне қарамастан жүзеге асырылады. Егер бұл бос кодты компьютер өндірушісінде жұмыс істейтін бағдарламашы енгізсе, содан кейін ол шығарған компьютерлермен бірге орнатылса, бағдарламашы оның иесі кім немесе пароль файлының мазмұны қандай болмасын, осы компания жасаған кез-келген компьютерге кіре алады. Бұл операциялық жүйенің өндірушісінде жұмыс істейтін бағдарламашыға қатысты. Бос орын аутентификация процесін айналып өтуге мүмкіндік береді. Компаниялар өздерінің жұмыс тәжірибесіне кодты шолуды (code reviews) енгізу арқылы саңылаулардың енгізілуіне жол бермейді. Осы технологияға сәйкес, бір рет бағдарламалаушы модульді құруды және тестілеуді аяқтады, ол код дерекқорына енгізіледі. Мерзімді түрде команданың барлық бағдарламашылары жиналады және олардың әрқайсысы бүкіл топқа оның кодын не істейтінін түсіндіреді. Бұл біреудің бос орынға ие болу ықтималдығын едәуір арттырып қана қоймайды, сонымен қатар бағдарламашы үшін ойын ставкаларын арттырады, өйткені егер ол қылмыс орнында ұсталса, бұл оның мансабын дамытудың оң факторы болмайды. Егер бағдарламашылар бұл ұсынысқа қатты қарсылық білдірсе, сіз екі қызметкерді бір-бірінің кодын тексеруге мәжбүр ете аласыз.

Жүйеге кіруді бұрмалау

Бұл инсайдерлік шабуылда шабуылдаушы-бұл жүйеге кіруді бұрмалау (login spoofing) деп аталатын технологияны қолдана отырып, басқа адамдарға тиесілі парольдерді жинайтын заңды пайдаланушы. Әдетте ол көптеген қызметкерлер пайдаланатын жергілікті желіге қосылған көптеген қоғамдық компьютерлері бар ұйымдарда қолданылады. Мысалы, көптеген университеттерде көптеген компьютерлерге толы залдар бар, олардың әрқайсысынан студенттер кез-келген ОЖ-ге кіре алады. Бұл шабуыл келесідей жұмыс істейді. Әдетте, ешкім кірмеген кезде, Unix компьютер экранында суретте көрсетілгендей сурет пайда болады. 9.23, а. пайдаланушы компьютерде отырғанда және тіркеу атын тергенде, жүйе парольді сұрайды. Егер дұрыс пароль енгізілсе, пайдаланушы жүйеге кіріп, қабық іске қосылады (және, мүмкін, пайдаланушының графикалық интерфейсі).

а *б*

Рис. **23**. Экран входа в систему: *а* — настоящий; *б* — фальсифицированный

Енді келесі сценарийді қарастырыңыз. Mal атты шабуылдаушы қолданушы экрандағы кескін суретте көрсетілгендей көрінуі үшін бағдарлама жазды. 9.23, ә. бұл суретте көрсетілген экрандағы сурет сияқты көрінеді. 9.23, А, оны енгізу бағдарламасы емес, оны Мэлл жазған бұрмалау шығаратындығынан басқа. Енді Мэл өзінің жалған кіру бағдарламасын іске қосады және қауіпсіз қашықтықтан не болып жатқанын бақылау үшін жағына шығады. Пайдаланушы компьютерге кіріп, тіркеу атын терген кезде, бағдарлама пароль сұрауымен жауап береді және экрандағы таңбалардың шығуын өшіреді. Тіркеу аты мен құпия сөз жиналғаннан кейін олар файлға жазылады және жалған кіру бағдарламасы оның қабығын жою үшін сигнал жібереді. Осы әрекеттің нәтижесінде Мэлдің жүйеден шығуы, осы кіру бағдарламасын іске қосуға көшу және суретте көрсетілген шақыруды экранға шығару жүреді. 9.23, А. пайдаланушы теру кезінде қате жіберді деп болжайды және жүйеге қайта кіреді. Бұл жолы бәрі дұрыс жұмыс істейді. Сонымен қатар, Mal тіркеу аты мен паролінен басқа жұп алады. Көптеген компьютерлерге кіру және жалған кіру бағдарламасын іске қосу арқылы ол парольдердің үлкен жинағын жинай алады.

Жалғыз тәсілімен болдырмау мұндай оқиғалардың дамуы болып табылады начало ретпен кіру шлюз, ол перехватывается пользовательскими бағдарламалары. Бұл үшін Windows-та Ctrl+ALT+DEL тіркесімі қолданылады. Егер пайдаланушы компьютерде отырса және Ctrl+-ALT+DEL пернелер тіркесімін басудан бастаса, ағымдағы пайдаланушы жүйеден шығады және кіру бағдарламасы іске қосылады. Бұл механизмді айналып өту мүмкін емес.

Зиянды бағдарламалар

Бұрынғы уақытта (айталық, 2000 жылға дейін) скучно (бірақ ақылды) жасөспірімдер кейде зиянды бағдарламалық жасақтаманы жазу үшін бос уақыттарын өткізіп, содан кейін бүкіл әлем бойынша бәрін толтыру үшін еркін жүзуге жібермек болды.

Трояндық жылқылар, вирустар мен құрттар кіретін бұл бағдарламалар жалпы зиянды бағдарламалар (зиянды бағдарламалар) деп аталады және бүкіл әлемге тез таралады. Олардың авторлары зиянды бағдарламалардан туындаған миллиондаған шығындар туралы және олардың жұмысының нәтижесінде қанша адам құнды ақпаратты жоғалтқаны туралы жарияланымдарды оқып, өздерінің бағдарламашылығының деңгейіне таң қалады. Олар үшін бұл жай ғана күлкілі әзілдер, өйткені олар одан ешқандай материалдық пайда көрмейді.

Бірақ бұл уақыттар өтті. Зиянды бағдарламалар қазір газет басылымдарында өз жұмысының нәтижелерін көрмеуді жөн көретін ұйымдасқан қылмыстық топтардың өтініші бойынша жазылады. Олар Интернет арқылы ең жоғары жылдамдықпен тарату және олардың құрбандарының мүмкіндігінше көп машиналарын жұқтыру үшін жасалынған. Машина жұқтырған кезде, түсірілген машиналардың мекен-жайлары туралы хабарламаларды тиісті компьютерлерге жіберетін бағдарламалық жасақтама орнатылады. Сондай-ақ, зиянды бағдарламаларды таратын қылмыскерлерге машинаға өзіне тән емес әрекеттерді орындау үшін оңай команда беруге мүмкіндік беретін бос орын (backdoor) машинаға енгізіледі. Осылайша түсірілген машиналар зомби (зомби) деп аталады, ал олардың жиынтығы ботнет (ботнет) деп аталады, қысқартылған robot network, яғни роботтардан тұратын желі.

Ботнеттерді басқаратын қылмыскерлер оларды әр түрлі (бірақ әрқашан коммерциялық) мақсаттар үшін жалға бере алады. Кең таралған қосымшалардың бірі-коммерциялық спам жіберу. Үлкен спам-шабуылда полиция оның көзін іздеуге тырысады, бірақ спам бүкіл әлемдегі мыңдаған машиналардан келетінін көреді. Егер олар осы машиналардың иелеріне жетсе, онда бұл балалар, шағын кәсіпкерлер, үй шаруасындағы әйелдер, қарт әйелдер және басқа да көптеген адамдар спам жіберуге қатысы барын жоққа шығарады. Лас жұмыс үшін басқа адамдарға тиесілі машиналарды пайдалану барлық қылмыскерлерді бақылауды қиындатады.

Орнатқаннан кейін зиянды бағдарламаны басқа қылмыстық мақсаттарда да қолдануға болады. Мүмкін бопсалау үшін. . Зиянды фрагментті елестетіп көріңіз құрбанның қатты дискісіндегі барлық файлдарды шифрлайтын, содан кейін келесі хабарламаны көрсететін бағдарламалар:

СІЗДІ GENERAL ENCRYPTION ҚАРСЫ АЛАДЫ!

Қатты дискіні шифрлау кілтін сатып алу үшін, 100 доллар тұратын аяқталмаған шағын вексельдерді мына мекен-жайға жіберіңіз: BOX 2154, PANAMA CITY, PANAMA. Рахмет. БІЗ СІЗДІҢ АЛАҢДАУШЫЛЫҒЫҢЫЗ ҮШІН РИЗАМЫЗ.

Зиянды бағдарламалардың тағы бір кең таралған әрекеті — вирус жұққан машинада пернетақта логгері (keylogger) пернесін басу тіркеушісін орнату. Бұл бағдарлама барлық пернелерді тіркейді және тіркеу жазбаларын белгілі бір машиналарға немесе бірқатар машиналарға (соның ішінде зомбилерге) жібереді, осылайша мұның бәрі қылмыскерлерге түседі. Жеткізуші машиналарға қызмет көрсететін Интернет-провайдерлерді ынтымақтастыққа тарту көбінесе қиын, өйткені олардың көпшілігі қылмыскерлермен (және кейде қылмыстық бизнеске жатады), әсіресе сыбайлас жемқорлық деңгейі жоғары елдерде.

Пернетақтада терілген жолдардан алынған құнды ақпарат несие карталарының нөмірлерінен тұрады, оларды заңды сатушылардан тауарларды сатып алу үшін пайдалануға болады. Жәбірленушілер белгілі бір цикл аяқталғаннан кейін шоттарынан үзінді көшірме алғанға дейін несие карталарының нөмірлерін ұрлады деп күдіктенбегендіктен, қылмыскерлер өз қаражаттарын бірнеше күн немесе тіпті апта бойы жұмсай алады.

Мұндай шабуылдарға қарсы тұру үшін барлық несие карталарын шығаратын компаниялар Ерекше шығындар схемаларын анықтау үшін жасанды интеллект бағдарламаларын қолданады. Мысалы, егер несие картасын әдетте жергілікті дүкендерде қолданатын адам күтпеген жерден Тәжікстанға несие картасын ұсынатын компания үшін ондаған қымбат ноутбуктерге тапсырыс берсе, дабыл қоңырауы соғылады, ал оның қызметкері әдетте карта ұстаушысын осы транзакция туралы сыпайы түрде хабардар ету үшін шақырады. Әрине, қылмыскерлер мұндай бағдарламалық жасақтама туралы біледі, сондықтан олар радардың назарына түспеу үшін өз шығындарының стилін өзгертуге тырысады.

Пернетақта журналшысы жинаған деректерді зомби машинасында орнатылған бағдарламалар жинаған басқа деректермен біріктіруге болады, бұл қылмыскерлерге жеке басын ұрлауға мүмкіндік береді (identity theft) . Бұл қылмысты жасаған кезде қылмыскер адам туралы айтарлықтай ақпарат жинайды, мысалы, оның туған күні, анасының аты, әлеуметтік қамсыздандыру картасының нөмірі, банктік шот нөмірлері, парольдер және т. б. д.жәбірленушінің рөлін сәтті ойнауға және жүргізуші куәлігінің телнұсқасы, банктік дебеттік карта, туу туралы куәлік және тағы басқалар сияқты жаңа физикалық құжаттарды алуға мүмкіндік береді. Мұның бәрі өз кезегінде басқа қылмыскерлерге одан әрі пайдалану үшін сатылуы мүмкін.

Зиянды бағдарлама арқылы жасалған қылмыстың тағы бір түрі-Пайдаланушы интернет-банкингтің жеке кабинетіне сәтті кіргенге дейін жасырын болу. Осыдан кейін бағдарлама бірден шотта бар соманы анықтау үшін транзакцияны бастайды және бүкіл соманы қылмыскердің шотына аударады, ол бірден басқа шотқа, содан кейін тағы бір рет басқа шоттарға (сыбайлас жемқорлыққа ұшыраған түрлі елдерге) аударылады. Сондықтан полицияға қаражаттың өтуін бақылау үшін барлық іздеу тапсырыстарын жинау үшін бірнеше күн немесе апта қажет, егер ол осы қаражатқа жетсе де, оның талаптары қанағаттандырылмай қалуы мүмкін. Мұндай қылмыстарға

үлкен капитал қатысады және олар енді жасөспірімдердің бос әурешіліктеріне ешқандай қатысы жоқ.

Қылмыстық ортада пайдаланудан басқа, зиянды бағдарламалар өнеркәсіпте қолданылады. Компания жүйеде жүйелік әкімші болмаған кезде ғана іске қосылатын зиянды бағдарламаны шығара алады. Егер жағдай қолайлы болса, ол өндіріс процесіне кедергі келтіреді, өнімнің сапасын төмендетеді және сол арқылы бәсекелеске проблемалар туғызады. Барлық басқа жағдайларда ол ұйқысыз болады, бұл оны табуды қиындатады.

Мақсатты зиянды бағдарламаның тағы бір мысалы-белгілі бір компанияның өршіл вице-президенті жасаған және жергілікті желіге қосылған вирус. Егер тексеру нәтижесінде вирус компания президентінің машинасында іске қосылғанын анықтаса, ол электрондық кестені іздей бастайды және ондағы екі кездейсоқ таңдалған ұяшықты қайта орналастыра бастайды. Ерте ме, кеш пе, президент бұзылған электрондық кесте негізінде дұрыс емес шешім қабылдайды, нәтижесінде жұмыстан босатылып, сіз болжаған адамға орын береді.

Кейбір адамдар күні бойы "синустың артындағы чиппен" жүреді (радиожиілікті сәйкестендірудің RFID чипін киетін адамдармен шатастырмау керек). Олардың басқаларға нақты немесе жалған талаптары бар және кек алғысы келеді. Бұл зиянды бағдарлама оларға көмектесе алады. Көптеген заманауи компьютерлер негізгі Енгізу-шығару жүйесін (BIOS) флэш-жадта сақтайды, оның мазмұнын белгілі бір бағдарламаның басқаруымен қайта жазуға болады (өндірушіге бағдарламалық кодтың түзетулерін электронды түрде таратуға мүмкіндік беру үшін). Зиянды бағдарлама кез-келген Қоқысты флэш-жадқа жаза алады, ал компьютер жүктеуді тоқтатады. Егер флэш-жад чипі ұяға орнатылса, онда мәселені шешу үшін компьютерді ашып, чипті ауыстыру керек. Ал егер чип флэш-жады впаян да, ана ақылы, тура, бәлкім, лақтыруға төлем және сатып алу жаңа.

Сіз басқа да көптеген мысалдар келтіре аласыз, бірақ менің ойымша, сіз оның мәнін таптыңыз. Егер сізге қосымша сұмдықтар қажет болса, кез-келген іздеу жүйесінде malware сөзін теріңіз, сонда сіз ондаған миллион сілтемелер аласыз.

Сұрақ туындайды: неліктен зиянды бағдарламалар соншалықты оңай таралады? Себептері бірнеше. Біріншіден, әлемдегі компьютерлердің 90% - ында тек бір Операциялық жүйе, Windows (немесе оның нұсқалары) жұмыс істейді, бұл оны оңай мақсатқа айналдырады. Егер әрқайсысы нарықтың 10% - ын алатын он Операциялық жүйе туралы айтатын болсақ, онда зиянды бағдарламаларды тарату әлдеқайда қиын болар еді. Биология әлеміндегі сияқты, әртүрлілік жақсы қорғаныс болып табылады.

Екіншіден, ерте кезден бастап Microsoft Windows-ты техникалық тәжірибесіз адамдар үшін қол жетімді ету үшін көп күш жұмсады. Мысалы, бұрын Windows жүйелері, әдетте, UNIX жүйелерінен айырмашылығы, парольсіз кіру үшін конфигурацияланған болатын, мұнда пароль енгізу талабы тарихи түрде

қалыптасқан (дегенмен, Linux Windows-қа көбірек ұқсауға тырысқан сайын, бұл керемет әдет-ғұрып өз позициясын жоғалтады). Көптеген басқа жағдайларда, жоғары қауіпсіздік пен пайдаланудың қарапайымдылығы арасында ымыраға келу кезінде Microsoft үнемі өзінің нарықтық стратегиясы ретінде екіншісін таңдайды. Егер сіз қауіпсіздікті қолданудың қарапайымдылығынан гөрі маңызды нәрсе деп санасаңыз, кітапты біраз уақытқа бөліп, ұялы телефонды әр қоңырау алдында PIN-кодты енгізуге орнатыңыз — кез-келген құрылғыда осындай параметрлер бар. Егер мұны қалай жасау керектігін білмесеңіз, пайдаланушы нұсқаулығын өндірушінің веб-сайтынан жүктеп алыңыз. Бұл не туралы екені түсінікті ме?

Келесі бірнеше бөлімдерде зиянды бағдарламалардың кейбір жалпы формалары, олардың құрылғысы және тарату әдістері қарастырылады. Әрі қарай тарауда олармен күресудің кейбір жолдары қарастырылады.

Трояндық жылқылар

Зиянды бағдарламаны жасау — бұл сіздің жатын бөлмеңізде де жасалуы мүмкін. Бірақ миллиондаған адамдарды оны компьютерлеріне орнату - бұл басқа мәселе. Зиянды бағдарламаны жасаушы Мэл бұл мәселені қалай шешеді? Ең көп таралған әдіс - бұл шынымен пайдалы бағдарламаны құру және оған зиянды модульді енгізу. Бұл ойындар, музыка ойнатқыштары, "ересектерге арналған" мазмұнды көруге арналған бағдарламалар және тартымды графикасы бар кез-келген нәрсе болуы мүмкін. Адамдар мұндай қосымшаны өздері жүктеп, орнатады. Тегін бонус ретінде олар орнатылған зиянды бағдарламаны алады. Бұл тәсіл Гомердің "Одиссеясынан"грек жауынгерлері бар ағаш атты еске алу үшін трояндық ат (трояндық жылқы шабуылы) шабуылы деп аталады. Компьютерлік қауіпсіздік әлемінде бұл ұғым адамдар өз еркімен жүктейтін бағдарламада немесе веб-бетте жасырылған кез-келген зиянды кодты білдіреді.

Тегін бағдарлама іске қосылған кезде ол зиянды бағдарламаны дискіге жазып, оны іске қосатын функцияны шақырады. Содан кейін зиянды бағдарлама, мысалы, файлдарды жою, өзгерту немесе шифрлау үшін жасалған қара бизнеске кірісе алады. Сондай-ақ, ол несие карталарының нөмірлерін, парольдерді және басқа да пайдалы ақпаратты іздеп, оларды Интернет арқылы Мэлге жібере алады. Сірә, ол қандай да бір IP портына қосылып, машинаны спам жіберуге немесе қашықтағы иесінің өтініші бойынша кез-келген әрекетті жасауға дайын зомбиге айналдырып, командаларды күтеді. Әдетте зиянды бағдарлама машинаны қайта іске қосқан сайын оны қайта іске қосуға кепілдік беретін командаларды да қамтиды. Бұл үшін қаражат барлық операциялық жүйелерде бар.

Трояндық жылқылардың барлық "сүйкімділігі" — олардың авторынан жәбірленушінің компьютерін бұзудың қажеті жоқ-ол өзі үшін бұл жұмысты жасайды.

Жәбірленушіні трояндық бағдарламаны орындауға мәжбүрлеудің тағы бір әдісі бар. Мысалы, UNIX-тің көптеген пайдаланушылары команданы іздеу кезінде каталогтарды қарауды басқаратын $PATH қоршаған орта айнымалысын пайдаланады. Оның мазмұнын қабыққа келесі пәрменді теру арқылы көруге болады: echo $PATH

Белгілі бір жүйеде ast пайдаланушысы үшін мүмкін параметрлер келесі каталогтарды қамтуы мүмкін:

:/usr/ast/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/ucb:/usr/man\

:/usr/java/bin:/usr/java/lib:/usr/local/man:/usr/openwin/man

Басқа пайдаланушыларда басқа іздеу жолдары болуы мүмкін. Пайдаланушы қабыққа prog пәрменін терген кезде

қабық алдымен /usr/ast/bin/prog бағдарламасында мұндай бағдарламаның бар-жоғын тексереді. Егер ол бар болса, онда ол орындалады. Егер ол жоқ болса, қабық оны /usr/local/bin/ prog, /usr/bin/prog, /bin/prog және т.б. табуға тырысады, осы кәсіпорыннан бас тартпас бұрын барлық 10 каталогты кезекпен қарап шығады. Егер осы каталогтардың бірі қорғалмаған болса, крекер оған бағдарламаны орналастырады. Егер бұл Іздеу тізімінде осы атаумен бағдарламаның алғашқы пайда болуы болса, ол орындалады және трояндық ат өз жұмысын бастайды.

Көбінесе бағдарламалар /bin немесе /usr/bin-де болады, сондықтан трояндық жылқыны /usr/bin/X11/ls файлына орналастыру жұмыс істемейді, өйткені біріншісі осы атаумен нақты бағдарламаны табады. Бірақ крекер /usr/bin/X11 каталогына la файлын орналастырды делік. Егер пайдаланушы теріп, терсе la орнына LS (каталогта бар файлдардың аттарын көрсететін бағдарламалар), трояндық ат өзінің қара жұмысын орындау арқылы іске қосылады, содан кейін la командалары жоқ екендігі туралы дұрыс хабарлама шығады. Трояндық жылқыларды каталогтарға ешкім қарамайтын күрделі жолдармен енгізу және оларға жиі рұқсат етілген типографияларға сәйкес келетін атауларды тағайындау кез-келген адамның ерте ме, кеш пе осы бағдарламалардың біреуін шақыруына әкеледі және бұл біреудің артықшылықты пайдаланушы құқығына ие болады (тіпті мұндай пайдаланушылар да типографияға рұқсат береді) және бұл жағдайда трояндық ат /bin/ls файлын трояндық аттың нұсқасымен ауыстыруға мүмкіндік алады, осылайша дәл осы нұсқа әрдайым іске қосылады.

Сондай-ақ, заңды пайдаланушы болып табылатын біздің зиянкес Мэл артықшылықты пайдаланушының құқықтарына келесі жолмен жол сала алады. Ол трояндық жылқысы бар ls нұсқасын өзінің каталогына орналастырады, содан кейін артықшылықты пайдаланушының назарын аудара алатын күдікті әрекеттерді жасайды: мысалы, жүйенің барлық есептеу ресурстарын қолданатын 100 процесті

бір уақытта іске қосады. Артықшылықты қолданушы командалар тізбегін теру арқылы Mala үй каталогының мазмұнына қызығушылық танытуы мүмкін

```
cd /home/mal ls –l
```

Кейбір снарядтар $PATH мазмұнымен жұмыс жасамас бұрын алдымен жергілікті каталогқа жүгінетіндіктен, артықшылықты пайдаланушы мэлл орналастырған трояндық атты артықшылықты пайдаланушының құқығымен іске қоса алады, оны шабуылдаушы іздеді. Содан кейін трояндық жылқы /home/mal/bin/sh құра алады. Ол үшін екі жүйелік қоңырау шалынады: /home/mal/bin/sh файлының иесін тамырға өзгерту үшін chown және chmod — оның SETUID битін орнату үшін. Енді Мэл осы қабықты іске қосып, артықшылықты пайдаланушы бола алады.

Егер Мэл жиі тоқтап қалса, ол трояндық атпен алаяқтықтың келесі әдістерінің бірін қолдана алады. Бірінші әдіс-трояндық жылқы жәбірленушіде Quicken сияқты электрондық банкинг бағдарламасы орнатылғанын тексереді. Егер бағдарлама орнатылған болса, трояндық ат одан қолма-қол ақшаны алу үшін оны жалған шотқа (жақсырақ алыс елде) аударуды бұйырады.

Осыған ұқсас, егер трояндық ат ұялы телефонда (немесе смартфонда) жасалса, ол шынымен қымбат ақылы нөмірлерге мәтіндік хабарлама жібере алады, мысалы, Молдовада (бұрынғы Кеңес Одағының бір бөлігінде).

ОЖҚ. Дәріс №15. НЕГІЗГІ ОПЕРАЦИЯЛЫҚ ЖҮЙЕЛЕРДІҢ ҚАУІПСІЗДІК МОДЕЛДЕРІ. Модели безопасности основных операционных систем. Механизмы защиты операционных систем. Анализ защищенности современных операционных систем. Основные защитные механизмы ОС семейства WINDOWS (NT/2000/XP)

Операционная система есть специально организованная совокупность программ, которая управляет ресурсами системы (ЭВМ, вычислительной системы, других компонентов ИВС) с целью наиболее эффективного их использования и обеспечивает интерфейс пользователя с ресурсами.
Операционные системы, подобно аппаратуре ЭВМ, на пути своего развития прошли несколько поколений.
ОС первого поколения были направлены на ускорение и упрощение перехода с одной задачи пользователя на другую задачу (другого пользователя), что поставило проблему обеспечения безопасности данных, принадлежащих разным задачам.
Второе поколение ОС характеризовалось наращиванием программных средств обеспечения операций ввода-вывода и стандартизацией обработки прерываний. Надежное обеспечение безопасности данных в целом осталось нерешенной проблемой.
К концу 60-х гг. XX в. начал осуществляться переход к мультипроцессорной организации средств ВТ, поэтому проблемы распределения ресурсов и их защиты стали более острыми и трудноразрешимыми. Решение этих проблем привело к соответствующей организации ОС и широкому применению аппаратных средств защиты (защита памяти, аппаратный контроль, диагностика и т.п.).
Основной тенденцией развития вычислительной техники была и остается идея максимальной доступности ее для пользователей, что входит в противоречие с требованием обеспечения безопасности данных.
Под механизмами защиты ОС будем понимать все средства и механизмы защиты данных, функционирующие в составе ОС. Операционные системы, в составе которых функционируют средства и механизмы защиты данных, часто называют защищенными системами.
Под безопасностью ОС будем понимать такое состояние ОС, при котором невозможно случайное или преднамеренное нарушение функционирования ОС, а также нарушение безопасности находящихся под управлением ОС ресурсов системы. Укажем следующие особенности ОС, которые позволяют выделить вопросы обеспечения безопасности ОС в особую категорию:
- управление всеми ресурсами системы;
- наличие встроенных механизмов, которые прямо или косвенно влияют на безопасность программ и данных, работающих в среде ОС;
- обеспечение интерфейса пользователя с ресурсами системы;
- размеры и сложность ОС.

Большинство ОС обладают дефектами с точки зрения обеспечения безопасности данных в системе, что обусловлено выполнением задачи обеспечения максимальной доступности системы для пользователя.
Рассмотрим типовые функциональные дефекты ОС, которые могут привести к созданию каналов утечки данных.

1. Идентификация. Каждому ресурсу в системе должно быть присвоено уникальное имя – идентификатор. Во многих системах пользователи не имеют возможности удостовериться в том, что используемые ими ресурсы действительно принадлежат системе.
2. Пароли. Большинство пользователей выбирают простейшие пароли, которые легко подобрать или угадать.
3. Список паролей. Хранение списка паролей в незашифрованном виде дает возможность его компрометации с последующим НСД к данным.
4. Пороговые значения. Для предотвращения попыток несанкционированного входа в систему с помощью подбора пароля необходимо ограничить число таких попыток, что в некоторых ОС не предусмотрено.
5. Подразумеваемое доверие. Во многих случаях программы ОС считают, что другие программы работают правильно.
6. Общая память. При использовании общей памяти не всегда после выполнения программ очищаются участки оперативной памяти (ОП).
7. Разрыв связи. В случае разрыва связи ОС должна немедленно закончить сеанс работы с пользователем или повторно установить подлинность субъекта.
8. Передача параметров по ссылке, а не по значению (при передаче параметров по ссылке возможно сохранение параметров в ОП после проверки их корректности, нарушитель может изменить эти данные до их использования).
9. Система может содержать много элементов (например, программ), имеющих различные привилегии.

Основной проблемой обеспечения безопасности ОС является проблема создания механизмов контроля доступа к ресурсам системы. Процедура контроля доступа заключается в проверке соответствия запроса субъекта предоставленным ему правам доступа к ресурсам. Кроме того, ОС содержит вспомогательные средства защиты, такие как средства мониторинга, профилактического контроля и аудита. В совокупности механизмы контроля доступа и вспомогательные средства защиты образуют механизмы управления доступом.

Средства профилактического контроля необходимы для отстранения пользователя от непосредственного выполнения критичных с точки зрения безопасности данных операций и передачи этих операций под контроль ОС. Для обеспечения безопасности данных работа с ресурсами системы осуществляется с помощью специальных программ ОС, доступ к которым ограничен.

Средства мониторинга осуществляют постоянное ведение регистрационного журнала, в который заносятся записи о всех событиях в системе. В ОС могут использоваться средства сигнализации о НСД, которые используются при обнаружении нарушения безопасности данных или попыток нарушения.

Контроль доступа к данным. При создании механизмов контроля доступа необходимо, прежде всего, определить множества субъектов и объектов доступа. Субъектами могут быть, например, пользователи, задания, процессы и процедуры. Объектами – файлы, программы, семафоры, директории, терминалы, каналы связи, устройства, блоки ОП и т.д. Субъекты могут одновременно рассматриваться и как объекты, поэтому у субъекта могут быть права на доступ к другому субъекту. В

конкретном процессе в данный момент времени субъекты являются активными элементами, а объекты – пассивными.

Для осуществления доступа к объекту субъект должен обладать соответствующими полномочиями. Полномочие есть некий символ, обладание которым дает субъекту определенные права доступа по отношению к объекту, область защиты определяет права доступа некоторого субъекта ко множеству защищаемых объектов и представляет собой совокупность всех полномочий данного субъекта.

При функционировании системы необходимо иметь возможность создавать новые субъекты и объекты. При создании объекта одновременно создается и полномочие субъектов по использованию этого объекта. Субъект, создавший такое полномочие, может воспользоваться им для осуществления доступа к объекту или же может создать несколько копий полномочия для передачи их другим субъектам.

С традиционной точки зрения средства управления доступом позволяют специфицировать и контролировать действия, которые субъекты (пользователи и процессы) могут выполнять над объектами (информацией и другими компьютерными ресурсами). В данном разделе речь пойдет о логическом управлении доступом, которое, в отличие от физического, реализуется программными средствами. Логическое управление доступом – это основной механизм многопользовательских систем, призванный обеспечить конфиденциальность и целостность объектов и, до некоторой степени, их доступность (путем запрещения обслуживания неавторизованных пользователей).

Рассмотрим формальную постановку задачи в традиционной трактовке. Имеется совокупность субъектов и набор объектов. Задача логического управления доступом состоит в том, чтобы для каждой пары "субъект-объект" определить множество допустимых операций и контролировать выполнение установленного порядка.

Отношение "субъекты-объекты" можно представить в виде матрицы доступа, в строках которой перечислены субъекты, в столбцах – объекты, а в клетках, расположенных на пересечении строк и столбцов, записаны дополнительные условия (например, время и место действия) и разрешенные виды доступа.

Фрагмент матрицы может выглядеть, например, как показано в табл. 15.1.

Тема логического управления доступом – одна из сложнейших в области информационной безопасности. Дело в том, что само понятие объекта (а тем более видов доступа) меняется от сервиса к сервису. Для операционной системы к объектам относятся файлы, устройства и процессы. Применительно к файлам и устройствам обычно рассматриваются права на чтение, запись, выполнение (для программных файлов), иногда на удаление и добавление. Отдельным правом может быть возможность передачи полномочий доступа другим субъектам (так называемое право владения). Процессы можно создавать и уничтожать. Современные операционные системы могут поддерживать и другие объекты.

15.1. Фрагмент матрицы доступа

	Файл	Программа	Линия связи	Реляционная таблица
Пользователь 1	o r w с системной консоли	e	rw с 8:00 до 18:00	
Пользователь 2				a

Обозначение: "o" – разрешение на передачу прав доступа другим пользователям, "r" – чтение, "w" – запись, "e" – выполнение, "a" – добавление информации.

Для систем управления реляционными базами данных объект – это база данных, таблица, представление, хранимая процедура. К таблицам применимы операции поиска, добавления, модификации и удаления данных, у других объектов. В результате при задании матрицы доступа нужно принимать во внимание не только принцип распределения привилегий для каждого сервиса, но и существующие связи между сервисами (приходится заботиться о согласованности разных частей матрицы). Аналогичная трудность возникает при экспорте/импорте данных, когда информация о правах доступа, как правило, теряется (поскольку на новом сервисе она не имеет смысла). Следовательно, обмен данными между различными сервисами представляет особую опасность с точки зрения управления доступом, а при проектировании и реализации разнородной конфигурации необходимо позаботиться о согласованном распределении прав доступа субъектов к объектам и о минимизации числа способов экспорта/импорта данных.

Матрицу доступа, ввиду ее разреженности (большинство клеток – пустые), неразумно хранить в виде двухмерного массива. Обычно ее хранят по столбцам, т.е. для каждого объекта поддерживается список "допущенных" субъектов вместе с их правами. Элементами списков могут быть имена групп и шаблоны субъектов, что служит большим подспорьем администратору. Некоторые проблемы возникают только при удалении субъекта, когда приходится удалять его имя из всех списков доступа; впрочем, эта операция производится нечасто.

Списки доступа – исключительно гибкое средство. С их помощью легко выполнить требование о гранулярности прав с точностью до пользователя. Посредством списков несложно добавить права или явным образом запретить доступ (например, чтобы наказать нескольких членов группы пользователей). Безусловно, списки являются лучшим средством произвольного управления доступом.

Подавляющее большинство операционных систем и систем управления базами данных реализуют именно произвольное управление доступом. Основное достоинство произвольного управления – гибкость. К сожалению, у "произвольного" подхода есть ряд недостатков. Рассредоточенность управления доступом ведет к тому, что доверенными должны быть многие пользователи, а не только системные операторы или администраторы. Из-за рассеянности или

некомпетентности сотрудника, владеющего секретной информацией, эту информацию могут узнать и все остальные пользователи. Следовательно, произвольность управления должна быть дополнена жестким контролем за реализацией избранной политики безопасности.

Второй недостаток, который представляется основным, состоит в том, что права доступа существуют отдельно от данных. Ничто не мешает пользователю, имеющему доступ к секретной информации, записать ее в доступный всем файл или заменить полезную утилиту ее "троянским" аналогом. Подобная "разделенность" прав и данных существенно осложняет проведение несколькими системами согласованной политики безопасности и, главное, делает практически невозможным эффективный контроль согласованности.

Возвращаясь к вопросу представления матрицы доступа, укажем, что для этого можно использовать также функциональный способ, когда матрицу не хранят в явном виде, а каждый раз вычисляют содержимое соответствующих клеток. Например, при принудительном управлении доступом применяется сравнение меток безопасности субъекта и объекта.

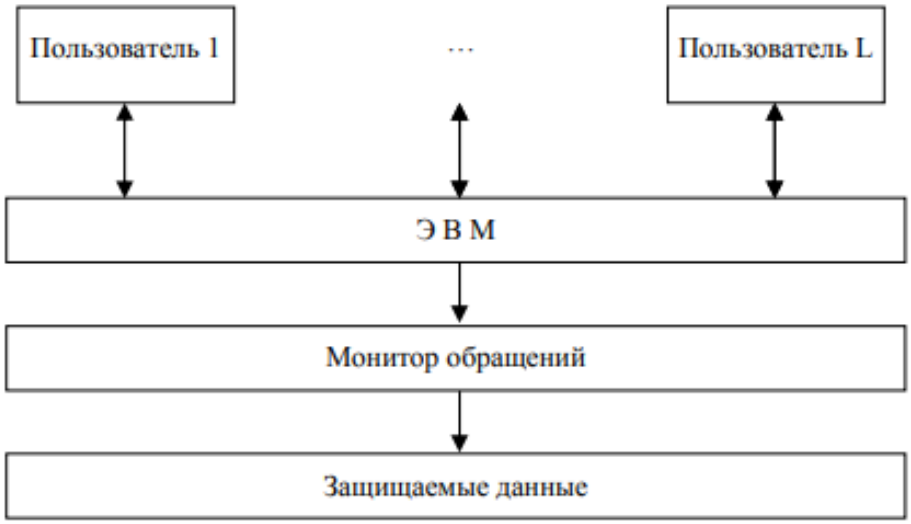


Рис. 15.1. Схема модели Харрисона, Руззо и Ульмана

Удобной надстройкой над средствами логического управления доступом является ограничивающий интерфейс, когда пользователя лишают самой возможности попытаться совершить несанкционированные действия, включив в число видимых ему объектов только те, к которым он имеет доступ. Подобный подход обычно реализуют в рамках системы меню (пользователю показывают лишь допустимые варианты выбора) или посредством ограничивающих оболочек, таких как restricted shell в ОС Unix.

При принятии решения о предоставлении доступа обычно анализируется следующая информация:

1) идентификатор субъекта (идентификатор пользователя, сетевой адрес компьютера и т.п.). Подобные идентификаторы являются основой произвольного (или дискреционного) управления доступом;

2) атрибуты субъекта (метка безопасности, группа пользователя и т.п.). Метки безопасности – основа мандатного управления доступом.

Непосредственное управление правами доступа осуществляется на основе одной из моделей доступа:

- матричной модели доступа (модель Харрисона-Руззо-Ульмана);
- многоуровневой модели доступа (модель Белла-Лападулы).

Разработка и практическая реализация различных защищенных ОС привела Харрисона, Руззо и Ульмана к построению формальной модели защищенных систем. Схема модели Харрисона, Руззо и Ульмана (HRU-модели) приведена на рис. 15.1.

15.2. АНАЛИЗ ЗАЩИЩЕННОСТИ СОВРЕМЕННЫХ ОПЕРАЦИОННЫХ СИСТЕМ

15.2.1. АНАЛИЗ ВЫПОЛНЕНИЯ СОВРЕМЕННЫМИ ОС ФОРМАЛИЗОВАННЫХ ТРЕБОВАНИЙ К ЗАЩИТЕ ИНФОРМАЦИИ ОТ НСД

Анализировать выполнение современными универсальными ОС требований, задаваемых для класса защищенности АС 1В, не имеет смысла в принципе. Для большинства ОС либо полностью не реализуется основной для данных приложений мандатный механизм управления доступом к ресурсам, либо не выполняется его важнейшее требование "Должно осуществляться управление потоками информации с помощью меток конфиденциальности. При этом уровень конфиденциальности накопителя должен быть не ниже уровня конфиденциальности записываемой на него информации". В связи с этим далее будем говорить лишь о возможном соответствии средств защиты современных ОС классу АС 1Г (защита конфиденциальной информации).

В качестве альтернативных реализаций ОС рассмотрим семейства Unix и Windows (естественно, Windows NT/2000, так как о встроенных механизмах защиты ОС Windows 9x/Me говорить вообще не приходится).

Сначала остановимся на принципиальном, даже, можно сказать, концептуальном противоречии между реализованными в ОС механизмами защиты и принятыми формализованными требованиями. Концептуальном в том смысле, что это противоречие характеризует не какой-либо один механизм защиты, а общий подход к построению системы защиты.

Противоречие состоит в принципиальном различии подходов (соответственно требований) к построению схемы администрирования механизмов защиты и, как следствие, это коренным образом сказывается на формировании общих принципов задания и реализации политики безопасности в организации, распределения ответственности за защиту информации, а также на определении того, кого относить к потенциальным злоумышленникам (от кого защищать информацию).

Для иллюстрации из совокупности формализованных требований к системе защиты конфиденциальной информации рассмотрим следующие два требования:

1) право изменять правила разграничения доступа (ПРД) должно предоставляться выделенным субъектам (администрации, службе безопасности и т.д.);

2) должны быть предусмотрены средства управления, ограничивающие распространения прав на доступ.

Данные требования жестко регламентируют схему (или модель) администрирования механизмов защиты. Это должна быть централизованная схема, единственным элементом которой выступает выделенный субъект, в частности, администратор (администратор безопасности). При этом конечный пользователь исключен в принципе из схемы администрирования механизмов защиты.

При реализации концепции построения системы защиты, регламентируемой рассматриваемыми требованиями, пользователь не наделяется элементом доверия, так как он может считаться потенциальным злоумышленником, что и имеет место на практике.

Теперь в общих чертах рассмотрим концепцию, реализуемую в современных универсальных ОС. Здесь "владельцем" файлового объекта, т.е. лицом, получающим право на задание атрибутов (или ПРД) доступа к файловому объекту, является лицо, создающее файловый объект. Так как файловые объекты создают конечные пользователи, то именно они и назначают ПРД к создаваемым им файловым объектам. Другими словами, в ОС реализуется распределенная схема назначения ПРД, где элементами схемы администрирования являются собственно конечные пользователи.

В данной схеме пользователь должен наделяться практически таким же доверием, как и администратор безопасности, при этом нести наряду с ним ответственность за обеспечение компьютерной безопасности. Отметим, что данная концепция реализуется и большинством современных приложений, в частности СУБД, где пользователь может распространять свои права на доступ к защищаемым ресурсам. Кроме того, не имея в полном объеме механизмов защиты компьютерной информации от конечного пользователя, в рамках данной концепции невозможно рассматривать пользователя в качестве потенциального злоумышленника. А как мы увидим далее, именно с несанкционированными действиями пользователя на защищаемом компьютере (причем как сознательными, так и нет) связана большая часть угроз компьютерной безопасности.

Отметим, что централизованная и распределенная схемы администрирования – это две диаметрально противоположные точки зрения на защиту, требующие совершенно различных подходов к построению моделей и механизмов защиты. При этом сколько-нибудь гарантированную защиту информации можно реализовать только при принятии концепции полностью

централизованной схемы администрирования, что подтверждается известными угрозами ОС.

Возможности моделей, методов и средств защиты будем рассматривать применительно к реализации именно концепции централизованного администрирования. Одним из элементов данной концепции является рассмотрение

пользователя в качестве потенциального злоумышленника, способного осуществить НСД к защищаемой информации.

15.2.2. ОСНОВНЫЕ ВСТРОЕННЫЕ МЕХАНИЗМЫ ЗАЩИТЫ ОС И ИХ НЕДОСТАТКИ

Кратко остановимся на основных механизмах защиты, встроенных в современные универсальные ОС. Сделаем это применительно к возможности реализации ими принятой нами для рассмотрения концепции защиты конфиденциальной информации.

15.2.2.1. ОСНОВНЫЕ ЗАЩИТНЫЕ МЕХАНИЗМЫ ОС СЕМЕЙСТВА UNIX

Защита ОС семейства Unix в общем случае базируется на трех основных механизмах:

1) идентификации и аутентификация пользователя при входе в систему;
2) разграничении прав доступа к файловой системе, в основе которого лежит реализация дискреционной модели доступа;
3) аудит, т.е. регистрация событий.

При этом отметим, что для различных клонов ОС семейства Unix возможности механизмов защиты могут незначительно различаться, однако будем рассматривать ОС Unix в общем случае, без учета некоторых незначительных особенностей отдельных ОС этого семейства.

Построение файловой системы и разграничение доступа к файловым объектам имеет особенности, присущие данному семейству ОС. Рассмотрим кратко эти особенности. Все дисковые накопители (тома) объединяются в единую виртуальную файловую систему путем операции монтирования тома. При этом содержимое тома проецируется на выбранный каталог файловой системы. Элементами файловой системы являются также все устройства, подключаемые к защищаемому компьютеру (монтируемые к файловой системе). Поэтому разграничение доступа к ним осуществляется через файловою систему.

Каждый файловый объект имеет индексный дескриптор, в котором среди прочего хранится информация о разграничении доступа к данному файловому объекту. Права доступа делятся на три категории: доступ для владельца, доступ для группы и доступ для остальных пользователей. В каждой категории определяются права на чтение, запись и исполнение (в случае каталога – просмотр).

Пользователь имеет уникальные символьный идентификатор (имя) и числовой идентификатор (UID). Символьный идентификатор предъявляется пользователем при входе в систему, числовой используется операционной системой для определения прав пользователя в системе (доступ к файлам и т.д.).

Принципиальные недостатки защитных механизмов ОС семейства Unix. Рассмотрим в общем случае недостатки реализации системы защиты ОС семейства Unix в части невыполнения требований к защите конфиденциальной информации, напрямую связанные с возможностью НСД к информации.

Для начала отметим, что в ОС семейства Unix, вследствие реализуемой ею концепции администрирования (не централизованная), невозможно обеспечить замкнутость (или целостность) программной среды. Это связано с невозможностью установки атрибута "исполнение" на каталог (для каталога данный атрибут ограничивает возможность "обзора" содержимого каталога). Поэтому при

разграничении администратором доступа пользователей к каталогам, пользователь, как "владелец" создаваемого им файла, может занести в свой каталог исполняемый файл и, как его "владелец", установить на файл атрибут "исполнение", после чего запустить записанную им программу. Эта проблема непосредственно связана с реализуемой в ОС концепцией защиты информации.

Не в полном объеме реализуется дискреционная модель доступа, в частности, не могут разграничиваться права доступа

для пользователя "root" (UID = 0), т.е. данный субъект доступа исключается из схемы управления доступом к ресурсам. Соответственно все запускаемые им процессы имеют неограниченный доступ к защищаемым ресурсам. С этим недостатком системы защиты связано множество атак, в частности:

- несанкционированное получение прав root;
- запуск с правами root собственного исполняемого файла (локально либо удаленно внедренного), при этом несанкционированная программа получает полный доступ к защищаемым ресурсам и т.д.

Кроме того, в ОС семейства Unix невозможно встроенными средствами гарантированно удалять остаточную информацию. Для этого в системе абсолютно отсутствуют соответствующие механизмы.

Необходимо также отметить, что большинство ОС данного семейства не обладают возможностью контроля целостности файловой системы, т.е. не содержат соответствующих встроенных средств. В лучшем случае дополнительными утилитами может быть реализован контроль конфигурационных файлов ОС по расписанию в то время, как важнейшей возможностью данного механизма можно считать контроль целостности программ (приложений) перед их запуском, контроль файлов данных пользователя и т.д.

Что касается регистрации (аудита), то в ОС семейства Unix не обеспечивается регистрация выдачи документов на

"твердую копию", а также некоторые другие требования к регистрации событий.

Если же трактовать требования к управлению доступом в общем случае, то при защите компьютера в составе ЛВС необходимо управление доступом к узлам сети. Однако встроенными средствами зашиты некоторых ОС семейства Unix управление доступом к узлам не реализуется.

Из приведенного анализа видно, что многие механизмы, необходимые с точки зрения выполнения формализованных требований, большинством ОС семейства Unix не реализуется в принципе, либо реализуется лишь частично.

15.2.2.2. ОСНОВНЫЕ ЗАЩИТНЫЕ МЕХАНИЗМЫ ОС СЕМЕЙСТВА WINDOWS (NT/2000/XP)

Теперь кратко остановимся на основных механизмах защиты, реализованных в ОС семейства Windows, и проведем анализ защищенности ОС семейства Windows (NT/2000). Отметим, что здесь ряд объектов доступа (в частности, устройства, реестр ОС и т.д.) не являются объектами файловой системы. Поэтому возникает вопрос, как следует трактовать требование "Система защиты должна контролировать доступ наименованных субъектов (пользователей) к наименованным объектам (файлам, программам, томам и т.д.)". Не ясно, являются

ли объектами доступа, к которым, следуя формальным требованиям, необходимо разграничивать доступ пользователей, например, реестр ОС и т.д.

В отличие от семейства ОС Unix, где все задачи разграничительной политики доступа к ресурсам решаются средствами управления доступом к объектам файловой системы, доступ в данных ОС разграничивается собственным механизмом для каждого ресурса. Другими словами, при рассмотрении механизмов защиты ОС Windows встает задача определения и задания требований к полноте разграничений (это определяется тем, что считать объектом доступа).

Также, как и для семейства ОС Unix, здесь основными механизмами защиты являются:

1) идентификация и аутентификация пользователя при входе в систему;
2) разграничение прав доступа к ресурсам, в основе которого лежит реализация дискреционной модели доступа (отдельно к объектам файловой системы, к устройствам, к реестру ОС, к принтерам и др.);
3) аудит, т.е. регистрация событий.

Здесь явно выделяются (в лучшую сторону) возможности разграничений прав доступа к файловым объектам (для NTFS) – существенно расширены атрибуты доступа, устанавливаемые на различные иерархические объекты файловой системы (логические диски, каталоги, файлы). В частности, атрибут "исполнение" может устанавливаться и на каталог, тогда он наследуется соответствующими файлами.

При этом существенно ограничены возможности управления доступом к другим защищаемым ресурсам, в частности, к устройствам ввода. Например, здесь отсутствует атрибут "исполнение", т.е. невозможно запретить запуск несанкционированной программы с устройств ввода.

Принципиальные недостатки защитных механизмов ОС семейства Windows (NT/2000/XP). Прежде всего рассмотрим принципиальные недостатки защиты ОС семейства Windows, напрямую связанные с возможностью НСД к информации. При этом в отличие от ОС семейства Unix в ОС Windows невозможна в общем случае реализация централизованной схемы администрирования механизмов защиты или соответствующих формализованных требований. Вспомним, что в ОС Unix это распространялось лишь на запуск процессов. Связано это с тем, что в ОС Windows принята иная концепция реализации разграничительной политики доступа к ресурсам (для NTFS).

В рамках этой концепции разграничения для файла приоритетнее, чем для каталога, а в общем случае – разграничения для включаемого файлового объекта приоритетнее, чем для включающего. Это приводит к тому, что пользователь, создавая файл и являясь его "владельцем", может назначить любые атрибуты доступа к такому файлу (т.е. разрешить к нему доступ любому иному пользователю). Обратиться к этому файлу может пользователь (которому назначил права доступа "владелец") вне зависимости от установленных администратором атрибутов доступа на каталог, в котором пользователь создает файл. Данная проблема непосредственно связана с реализуемой в ОС Windows концепцией защиты информации.

Далее, в ОС семейства Windows (NT/2000/XP) не в полном объеме реализуется дискреционная модель доступа, в частности, не могут разграничиваться права доступа для пользователя "Система". В ОС присутствуют не только

пользовательские, но и системные процессы, которые запускаются непосредственно системой. При этом доступ системных процессов не может быть разграничен. Соответственно, все запускаемые системные процессы имеют неограниченный доступ к защищаемым ресурсам. С этим недостатком системы защиты связано множество атак, в частности, несанкционированный запуск собственного процесса с правами системного. Кстати, это возможно и вследствие некорректной реализации механизма обеспечения замкнутости программной среды.

В ОС семейства Windows (NT/2000/XP) невозможно в общем случае обеспечить замкнутость (или целостность) программной среды. Это связано совершено с иными проблемами, чем в ОС семейства Unix, в которых невозможно установить атрибут "исполнение" на каталог. Для выяснения сложности данного вопроса рассмотрим два способа, которыми в общем

случае можно реализовать данный механизм, причем оба способа несостоятельны. Итак, механизм замкнутости программной среды в общем случае может быть обеспечен:

− заданием списка разрешенных к запуску процессов с предоставлением возможности пользователям запускать процессы только из этого списка. При этом процессы задаются полнопутевыми именами, причем средствами разграничения доступа обеспечивается невозможность их модернизации пользователем. Данный подход просто не реализуется встроенными в ОС механизмами;

− разрешением запуска пользователями программ только из заданных каталогов при невозможности модернизации этих каталогов. Одним из условий корректной реализации данного подхода является запрет пользователям запуска программ иначе, чем из соответствующих каталогов. Некорректность реализации ОС Windows данного подхода связана с невозможностью установки атрибута "исполнение" на устройства ввода (дисковод или CD-ROM). В связи с этим при разграничении доступа пользователь может запустить несанкционированную программу с дискеты, либо с диска CD-ROM (очень распространенная атака на ОС данного семейства).

Здесь же стоит отметить, что с точки зрения обеспечения замкнутости программной среды [т.е. реализации механизма,

обеспечивающего возможность пользователям запускать только санкционированные процессы (программы)] действия пользователя по запуску процесса могут быть как явными, так и скрытыми.

Явные действия предполагают запуск процессов (исполняемых файлов), которые однозначно идентифицируются своим именем. Скрытые действия позволяют осуществлять встроенные в приложения интерпретаторы команд. Примером таковых могут служить офисные приложения. При этом скрытыми действиями пользователя будет запуск макроса.

В данном случае идентификации подлежит лишь собственно приложение, например, процесс winword.exe. При этом он

может помимо своих регламентированных действий выполнять те скрытые действия, которые задаются макросом (соответственно, те, которые допускаются интерпретатором), хранящимся в открываемом документе. То же относится и к любой виртуальной машине, содержащей встроенный интерпретатор команд. При

этом отметим, что при использовании приложений, имеющих встроенные интерпретаторы команд (в том числе офисных приложений), не в полном объеме обеспечивается выполнение требования по идентификации программ.

Возвращаясь к обсуждению недостатков, отметим, что в ОС семейства Windows (NT/2000/XP) невозможно встроенными средствами гарантированно удалять остаточную информацию. В системе просто отсутствуют соответствующие механизмы.

Кроме того, ОС семейства Windows (NT/2000/XP) не обладают в полном объеме возможностью контроля целостности файловой системы. Встроенные механизмы системы позволяют контролировать только собственные системные файлы, не обеспечивая контроль целостности файлов пользователя. Кроме того, они не решают важнейшую задачу данных механизмов

– контроль целостности программ (приложений) перед их запуском, контроль файлов данных пользователя и др.

Что касается регистрации (аудита), то в ОС семейства Windows (NT/2000/XP) не обеспечивается регистрация выдачи документов на "твердую копию", а также некоторые другие требования к регистрации событий.

Опять же, если трактовать требования к управлению доступом в общем случае, то при защите компьютера в составе ЛВС необходимо управление доступом к узлам сети (распределенный пакетный фильтр). В ОС семейства Windows (NT/2000/XP) механизм управления доступа к узлам в полном объеме не реализуется.

Что касается разделяемых сетевых ресурсов, то фильтрации подвергается только входящий доступ к разделяемому ресурсу, а запрос доступа на компьютере, с которого он осуществляется, фильтрации не подлежит. Это принципиально, так как не могут подлежать фильтрации приложения, которыми пользователь осуществляет доступ к разделяемым ресурсам. Благодаря этому, очень распространенными являются атаки на протокол NETBIOS.

Кроме того, в полном объеме управлять доступом к разделяемым ресурсам возможно только при установленной на всех компьютерах ЛВС файловой системы NTFS. В противном случае невозможно запретить запуск несанкционированной программы с удаленного компьютера, т.е. обеспечить замкнутость программной среды в этой части.

Из приведенного анализа можно видеть, что многие механизмы, необходимые с точки зрения выполнения формализованных требований, ОС семейства Windows не реализуют в принципе, либо реализуют лишь частично.

С учетом сказанного можем сделать важный вывод относительно того, что большинством современных универсальных ОС не выполняются в полном объеме требования к защите АС по классу 1Г. Это значит, что, учитывая требования нормативных документов, они не могут без использования добавочных средств защиты применяться для защиты даже конфиденциальной информации. При этом следует отметить, что основные проблемы защиты здесь вызваны не невыполнимостью ОС требований к отдельным механизмам защиты, а принципиальными причинами, обусловленными реализуемой в ОС концепцией защиты. Концепция эта основана на реализации распределенной схемы

администрирования механизмов защиты, что само по себе является невыполнением формализованных требований к основным механизмам защиты.

15.2.3. АНАЛИЗ СУЩЕСТВУЮЩЕЙ СТАТИСТИКИ УГРОЗ ДЛЯ СОВРЕМЕННЫХ УНИВЕРСАЛЬНЫХ ОС.

СЕМЕЙСТВА ОС И ОБЩАЯ СТАТИСТИКА УГРОЗ

На сегодняшний день существует достаточно большая статистика угроз ОС, направленных на преодоление встроенных в ОС механизмов защиты, позволяющих изменить настройки механизмов безопасности, обойти разграничения доступа и т.д. Таким образом, статистика фактов НСД к информации показывает, что большинство распространенных систем (универсального назначения) довольно уязвимы с точки зрения безопасности. И это несмотря на отчетливую тенденцию к повышению уровня защищенности этих систем.

Здесь необходимо отметить, что на практике современные информационные системы, предназначенные для обработки конфиденциальной информации, строятся уже с учетом дополнительных мер безопасности, что также косвенно подтверждает изначальную уязвимость современных ОС.

Рассмотрим операционные системы, фигурирующие в публикуемых списках системных и прикладных ошибок, позволяющих получить несанкционированный доступ к системе, понизить степень ее защищенности или добиться отказа в обслуживании (системного сбоя):

MS Windows 9X	BSD	AIX	BSD	Novell Netware
MS Windows NT	Solaris	SCO	HPUX	IOS (Cisco)
MS Windows 2000	Sun OS	Linux	IRIX	Digital Unix

Общее количество известных успешных атак для различных ОС, представлено в табл. 15.2, а их процентное соотношение – на диаграмме рис. 15.12.

Вследствие того, что большинство атак для операционных систем, построенных на базе Unix (BSD или AT&T), достаточно похожи, целесообразно объединить их в одну группу. Тоже самое можно сказать и об ОС семейства Windows. Таким образом, далее будем рассматривать только семейства ОС: Unix, MS Windows, Novell NetWare.

15.2. Количество известных успешных атак для различных ОС

9.1. Количество известных успешных атак для различных ОС

Тип ОС	Количество атак	Тип ОС	Количество атак
MS Windows NT/2000	130	Linux	167
MS Windows 9X/ME	120	IRIX	84
BSD	64	HPUX	65
BSDI	10	AIX	42
Solaris	125	SCO	40
Sun OS	40	Novell NetWare	10
Digital Unix	25	IOS (Cisco)	7

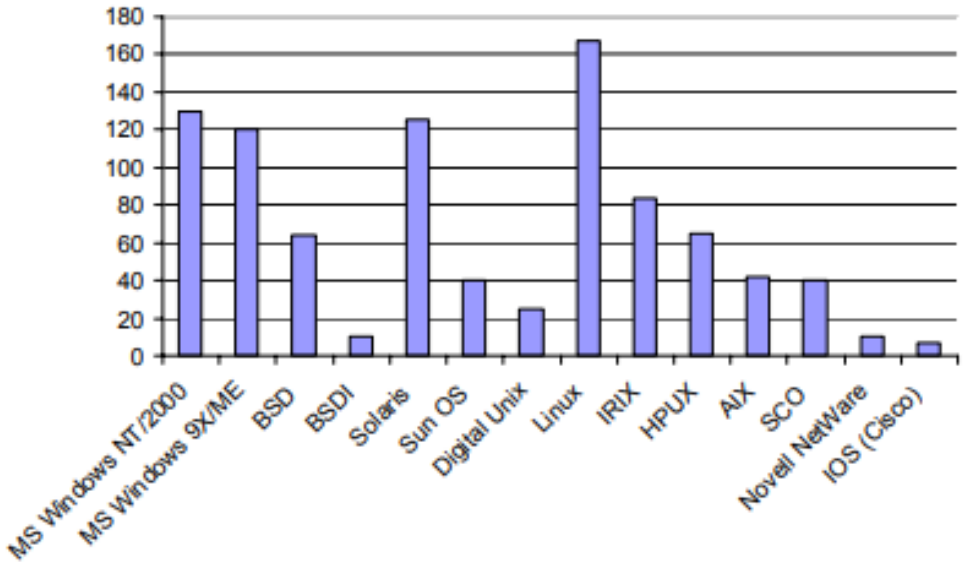
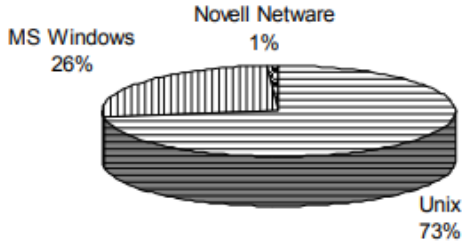


Рис. 9.1. Статистика соотношения угроз для различных ОС

Общее количество известных успешных атак для различных групп ОС представлено в табл. 9.2, а их процентное соотношение – на диаграмме рис. 9.2.

9.2. Общее количество успешных атак для различных групп ОС

Тип ОС	Количество атак
MS Windows	230
Unix	660
Novell Netware	10



Относительно ОС Novell следует заметить, что данная ОС изначально создавалась как защищенная (не универсального назначения) ОС, основной функцией которой был защищенный файловый сервис. Это, с одной стороны, должно было обеспечить

ее более высокий уровень защищенности, с другой стороны, налагало определенные ограничения по использованию. Однако, начиная с пятой версии, данная ОС начала приобретать свойства универсальности (с точки зрения применяемых протоколов и приложений), что в какой-то мере сказалось и на уровне ее защищенности.