11. *Павличенко И.П., Щеглов А.Ю.* Новые технологии защиты вычислительных систем. Механизмы разграничения прав доступа к файловой системе и обеспечения замкнутости программной среды//Информационные технологии, 2002. — №12.

12. *Щеглов А.О.* Защита компьютерной информации от несанкционированного доступа. Изд. Наука и Техника, С.-Петербург, 2004. -С. 384.

13. *Мельников В.* Защита информации в компьютерных системах — М.: Финансы и статистика; Электроинформ, 1997.

14. *Шорошев В.В.* Перспективный метод защиты информационных ресурсов корпоративных сетей Интранет. Бизнес и безопасность, 2003. -№ 6. -С. 38-46.

Поступила 14.11.2006 г.

УДК 004.681.3 **Гайша О.О.**

РОЗРОБКА МЕТОДИКИ МОДЕЛЮВАННЯ АЛГОРИТМІВ РОБОТИ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

Постановка задачі

Усі системи захисту інформації (далі по тексту – СЗІ) в основі своєї роботи мають якісь алгоритми, що виконують певні дії над зовнішніми вхідними даними та компонентами системи. Звичайно при розробці або аналізі СЗІ виникає потреба у наочному представленні послідовності дій алгоритму. Схематичне зображення надає чимало переваг порівняно з текстовим описом та математичною моделлю. Використовуючи різні рівні деталізації, можна проводити поступовий синтез/аналіз алгоритму роботи як завгодно складної системи.

Таким чином, при проектуванні СЗІ зручно використовувати графічні схеми її роботи. Однак можливі різні види і методи представлення такої інформації. Проаналізуємо наявні методики і запропонуємо авторську методику візуалізації алгоритмів СЗІ.

Аналіз посилань

Звичайним підходом для відображення будь-яких алгоритмів (в т.ч. і алгоритмів роботи СЗІ) є побудова блок-схем, згідно [1]. Рівень варіювання деталізації такого відображення не є дуже високим, і, звичайно одному графічному блокові може відповідати один чи кілька послідовних однотипних операторів програми. Таким чином, адекватне зображення більш укрупнених блоків, що реалізують певні логічні операції стає утрудненим і звичайно процес унаочнення спрощується до побудови прямокутників із вписаним текстом. Цей текст звичайно відображує назву якого-небудь процесу, що є черговою стадією послідовності дій алгоритму. Головним недоліком такого підходу є відсутність інформації про те, над чим виконується та чи інша операція, і куди передаються її результати. Таким чином, відсутня інформація про потоки даних. Можна сказати, що такий підхід добре підходить для відображення простих за змістом алгоритмів високого рівня абстракції. Для описання якихось складних (заплутаних) дій, як, наприклад, криптографічних протоколів або схем, цей метод унаочнення непридатний.

Ще одним близьким способом унаочнення є відображення схеми процесу за допомогою графу [2]. Принципово такі схеми можуть як завгодно детально відображувати логіку роботи програми (замінюючи вузли графу на більш деталізовані підграфи), але мають суттєвий недолік. У графі кожна вершина зображується у вигляді точки з порядковим номером, і для розуміння роботи програми слід весь час звертатися до таблиці відповідності вершин графа процесам або станам СЗІ. Оскільки існує можливість створення схем різного рівня деталізації, то така особливість на думку автора є значним недоліком, що перешкоджає швидкому розумінню структури графу. Крім того, кожному ребру орієнтованого графа відповідає певна операція над даними, однак не має механізму відображення, що саме це за

операція (окрім підписування і ребер графу порядковими номерами, що ще на порядок ускладнює сприйняття графу за рахунок необхідності утримування в пам'яті людини ще й співвідношень між номерами дуг та операціями).

Відомим методом опису процесів різної природи (в т.ч. і алгоритмів роботи комп'ютерних програм, отже, і СЗІ) є створення діаграм потоків даних [3] (DFD – Data Flow Definition). При такій методиці пропонується відображувати кілька типів сутностей: сховища даних, активні суб'єкти, безпосередньо потоки даних. Тут недоліком навпаки є недостатнє відображення суті процесу. Добре унаочнюються сукупності використовуваних даних, але процеси, що над ними виконуються, потребують більшої конкретизації.

Мета роботи

Проаналізувавши відомі методики зображення алгоритмів, можна сформувати наступну мету даної роботи: розробити зручну та легку до сприйняття методику відображення наочних схем алгоритмів, що дозволятиме швидко встановлювати послідовність дій і логіку роботи програмних СЗІ; в якості прикладу використання створеної методики, зобразити який-небудь реальний алгоритм роботи системи захисту програмного забезпечення (далі по тексту - ПЗ).

Основна частина

Розроблювана методика буде оперувати двома видами понять: сутностями та процесами.

Сутності, які будуть ініціювати нові процеси в роботі СЗІ будемо називати суб'єктами (або активними сутностями). Сутності, які будуть використовуватися для роботи якого-небудь процесу, наприклад, міститимуть певну інформацію, називатимемо об'єктами (пасивними сутностями). Схеми, на яких відображатимуться алгоритми роботи СЗІ, і які оперуватимуть трьома основними поняттями: процесами, суб'єктами та об'єктами називатимемо відповідно процесно-суб'єктно-об'єктними (ПСО-схемами)

При створенні програмних СЗІ можна провести чітку межу між суб'єктом та об'єктом: суб'єкт є кодом, що виконує процесор, тобто він сам може виконувати певні дії, в той час, як об'єкт навпаки являє собою дані, що їх використовує код ПЗ. Конкретним прикладом суб'єкту може бути певний блок коду, функція, окрема програма, бібліотека, тощо. Об'єктом може бути якийсь масив даних, наприклад, сукупність байт що відповідає ключеві шифрування, якимось необхідним відомостям, тощо.

Особливим випадком сутності є користувач системи, що може, в залежності від контексту задачі грати роль як активної, так і пасивної сутності. Якщо, користувач за вимогою СЗІ виконує якісь активні дії над запропонованими йому даними, то він є суб'єктом. Наприклад, СЗІ може запросити користувача зателефонувати постачальнику ліцензій на ПЗ та передати йому необхідні дані, щоб отримати активаційний код, який дозволяє роботу цього захищеного ПЗ.

В другому випадку користувач використовується лише як джерело певних даних (тобто для зняття з нього необхідної інформації) без виконання ним якихось активних змістовних для роботи СЗІ дій. Наприклад, він може вводити пароль, або прикладати палець до пристрою зчитування відбитків пальців. Тут користувач виступає в ролі пасивного джерела інформації і може розглядатися в ролі об'єкта.

Тож, відповідно до розглянутої класифікації сутностей пропонується на ПСО-схемах активні сутності відображати прямокутниками, а пасивні – еліпсами. У ці геометричні фігури вписуватимемо текст, що відображатиме назву сутності, наприклад, на рис.1 наведено три сутності: суб'єкт – динамічна бібліотека DLL, об'єкт – файл з ключем шифрування, і користувач ПЕОМ, що в даному випадку виступає об'єктом, який лише надає інформацію, але не виконує яких-небудь дій над нею. Розміри фігур треба робити мінімально можливими для вписування тексту, щоб не загромаджувати рисунок.

На рисунку 1 бачимо, що у прямокутнику, який відповідає суб'єкту динамічної бібліотеки, зображено ще два прямокутники. Таким способом пропонується зображувати

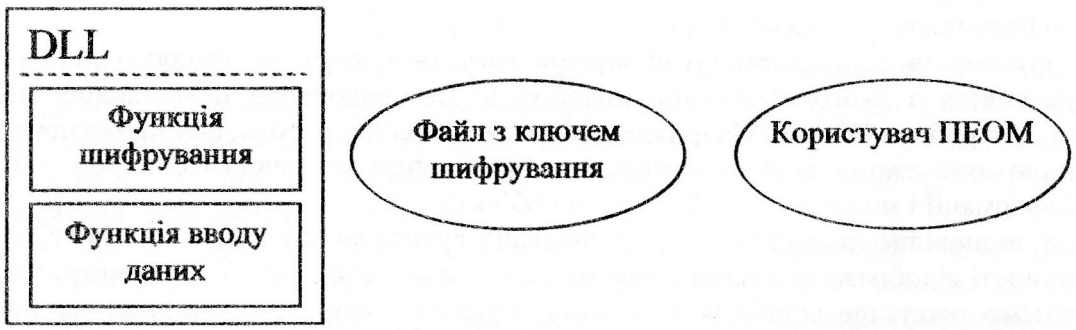
суттєві елементи суб'єктів (при необхідності і об'єктів), які проясняють зміст даної схеми. У розглядуваному прикладі треба було підкреслити, що у складі суб'єкта DLL міститься два важливих елементи (функція шифрування та функція вводу даних), які виконуватимуть дії на даній схемі. Назву материнської сутності-контейнеру пропонується зображувати більшим шрифтом угорі відповідного прямокутника (слово DLL), під ним проводити пунктирну риску, а потім розташовувати вкладені дочірні сутності.

Можливий такий випадок, коли у активній сутності будуть включені деякі відомості, що не виконуватимуть певних дій (тобто в цілому є об'єктом), але при зображенні схеми необхідно підкреслити, що ці відомості знаходяться саме в даному суб'єкті. В такому випадку в складі суб'єкту можуть вміщуватися об'єкти (еліпс у прямокутнику). Наприклад, ми реалізували СЗІ, що забороняє незаконну роботу, в одній підпрограмі. Якщо там не буде якихось необхідних для іншої функціональної частини програми відомостей (якогось необхідного об'єкту), то зломщик може просто затерти усі оператори нашої СЗІ порожніми командами асемблеру NOP. Це реальний приклад необхідності використання об'єкту (необхідних для нормального функціонування системи відомостей) в суб'єкті (тобто в коді СЗІ).

Спрощеним варіантом є зображення назв складових сутності під штриховою лінією взагалі без обмежуючих фігур (тобто назви складових сутності не брати в прямокутні або еліптичні рамки), відділяючи їх одна від одної порожнім рядком, або малою штриховою лінією. В такому разі увага не розосереджується на велику кількість символічної інформації. Цей варіант видається найкращим, а вводити рамки для складових сутності рекомендується лише якщо необхідно підкреслити їх відокремленість або тип: активною є складова (суб'єкт) чи пасивною (об'єкт) (приклад такого зображення наведено на рис. 3, де у сутності ПФЗ є дві окремі складові).

Отже, проаналізувавши спосіб зображення сутностей, перейдемо до зображення процесів. Процеси пропонується зображувати стрілками, що виходять із однієї сутності, та входять до іншої. Існує обмеження на тип сполучуваних сутностей: хоча б одна із них має бути суб'єктом. Твердження пояснюється тим, що кожен процес має породжувати активна сутність – суб'єкт. Приклади процесів на рис. 2.

Можливий також варіант, коли процес розпочинається однією складовою суб'єкта, а результат передається іншій складовій цього ж суб'єкта. Тоді стрілку необхідно розпочинати із тієї області батьківського суб'єкта, що знаходиться найближче всього до ініціюючої складової, і закінчувати на цьому ж батьківському суб'єкті, але в тій його області, що найближче всього лежить до тієї дочірньої сутності, що отримує результат (такою є дія 3 на рис.3, алгоритм якого розглядається як приклад нижче).



а) суб'єкт DLL; *б*) об'єкт файл; *в*) об'єкт

Рис. 1. Приклади активних та пасивних сутностей

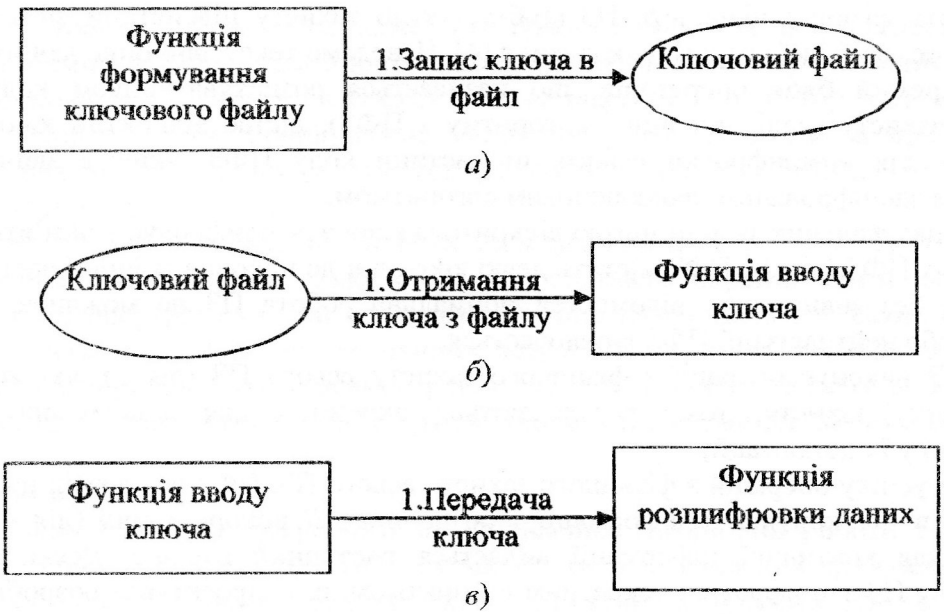


Рис. 2. Приклади процесів різного типу
(*а* – процес суб'єкт-об'єкт, або запис даних;
***б* – процес об'єкт-суб'єкт, або зчитування даних;**
***в* – процес суб'єкт-суб'єкт або передача даних)**

Суть кожного процесу відображується іменником в називному відмінку із додатковим поясненням. Назва процесу підписується до кожної стрілки, і розташовується таким чином, щоб можна було легко ідентифікувати стрілку, до якої він належить (підписи до стрілок не повинні суміщатися, чи тим більше накладатися). Для задоволення останньої умови текст підписів назв процесів має зображуватися меншим розміром шрифту, ніж назви сутностей, і мати варійоване вирівнювання, встановлюване в залежності від того, з якого боку від стрілки розташований підпис.

Для уникнення використання багатозначного слова «процес» кожну активну стадію роботи алгоритму, що уособлюється однією стрілкою, будемо називати дією. Тоді в тексті буде зручно давати пояснення схеми, посилаючись на певний процес наступним чином: дія 1 на рис. 2.

В реальних схемах кількість виконуваних елементарних процесів може бути значною, тому для швидкого встановлення послідовності їх виконання, дії нумеруються відповідно до алгоритму роботи СЗІ.

Ще одним аспектом, який потрібно зображувати на схемах алгоритмів, є виконання програмою якихось проміжних дій, несуттєвих для роботи СЗІ (можливо несуттєвих на цьому рівні деталізації). Наприклад, система захисту виконує якісь дії до завантаження функціональної частини ПЗ, потім виконується прикладна частина програми, а потім знову потрібно працювати системі захисту. При цьому робота СЗІ являє собою одне ціле – процес, який треба розглядати (і, отже, зображувати) комплексно. Тому при відображенні роботи СЗІ проміжні несуттєві дії іноді потрібно показувати, для чого пропонується використовувати малий квадрат, в якому зображатимуться лише три крапки «…» (див. Рис. 3).

Іноді при зображенні схем високого рівня абстракції постає необхідність у зображенні дій, що можуть бути виконані необов'язково. В такому випадку стрілку відповідної дії пропонується зображувати пунктиром (такою є дія 6 на рис. 3).

Для оцінювання ефективності пропонованої методики розглянемо реальний приклад авторської розробки, що являє собою спосіб захисту від модифікацій блоку операторів, які

контролюють фізичну цілісність ПЗ (тобто спосіб захисту підсистеми фізичного захисту програми), основні ідеї котрого викладені в [4]. Наведемо текстовий опис даного алгоритму:

а) Окремий блок операторів, що називається розпакувальником коду підсистеми фізичного захисту (далі при описі алгоритму - ПФЗ), дістає відкритий ключ (дія 1); він необхідний для розшифровки всього чи частини коду ПФЗ, який в звичайному стані зберігається зашифрованим двохключовим алгоритмом;

б) Розпакувальник за допомогою відкритого ключа розшифровує у пам'яті зашифровані ділянки коду ПФЗ (дія 2); ПФЗ містить деякі критичні до виконання іншої частини програми відомості і без знання цих відомостей нормальна робота ПЗ не можлива, тому варіант затирання або нейтралізації ПФЗ виключається;

в) ПФЗ виконує операції з фізичного захисту всього ПЗ (дія 3), які являють собою окрему ланку захисту, тому розглядаються окремо і для даного алгоритму немає необхідності у їх деталізації;

г) При успіху операцій з фізичного захисту всього ПЗ, ПФЗ дає дозвіл на використання вбудованої в неї критичної інформації і розпочинає її використання (дія 4); найкращим випадком для критичної інформації видається наступний варіант. Деякі функціональні ділянки коду ПЗ зашифровані секретним алгоритмом, що спроектував розробник захисту, а сам алгоритм вбудовано в ПФЗ, і так забезпечується залежність всієї функціональної частини ПЗ від ПФЗ. Тут в якості критичних до виконання функціональної частини програми відомостей виступає алгоритм розшифровки зашифрованих ділянок функціонального коду ПЗ;

д) ПЗ починає своє нормальне функціонування (дія 5), і переходить у байдужий для системи захисту стан; але в разі, якщо ПФЗ веде поступову динамічну розшифровку по запиту (а не розшифрувала одразу усі ділянки коду), то будуть відбуватися періодичні виклики ПФЗ для розшифровки необхідних ділянок коду (дія 6). Для нас тут головним є не стійкість пропонованого алгоритму, а достатньо велика складність сприйняття його текстового опису. Відповідна ПСО-схема наведена на рис. 3.

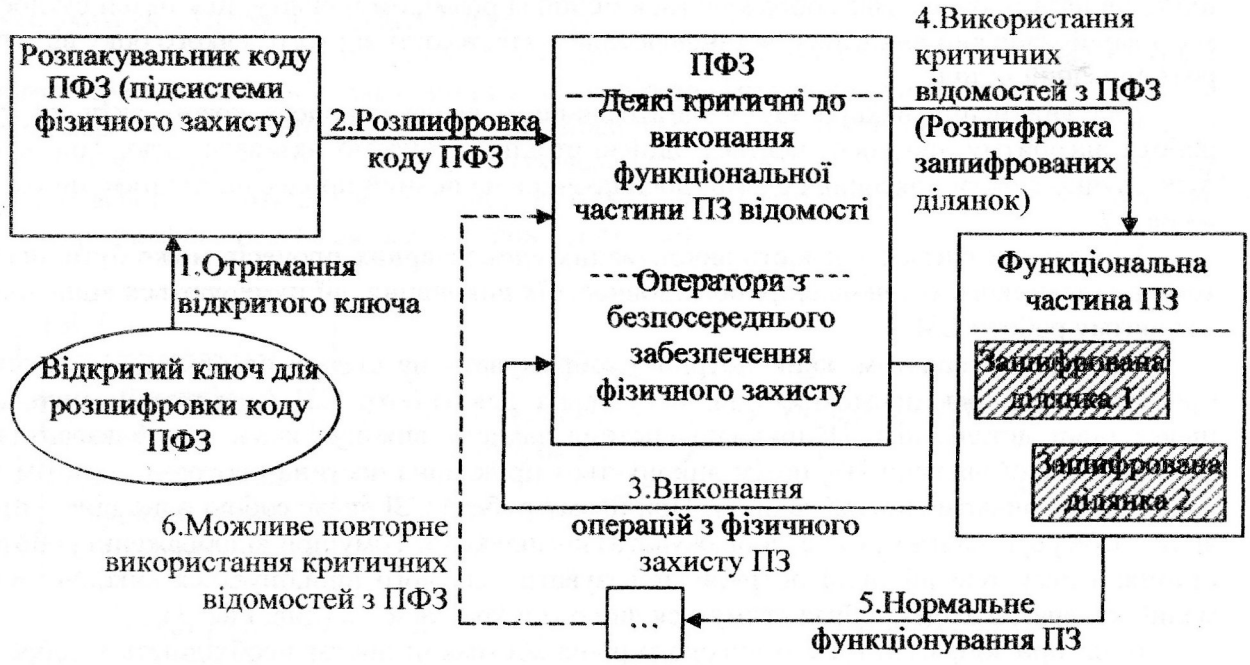


Рис. 3. ПСО-схема алгоритму забезпечення цілісності коду підсистеми фізичного захисту

Сама по собі схема рис. 3 не може дати повного уявлення про наведений алгоритм (звичайно на відміну від його текстового опису). Однак, якщо схему доповнити текстовим описом (за основу все ж взяти ПСО-схему, а не навпаки), то розбір алгоритму значно спрощується. Основні його стадії можна легко зрозуміти зі схеми, а при виникненні уточнюючих питань, звернутися до текстового опису, з яким у ПСО-схеми встановлено взаємно однозначну відповідність.

Аналізуючи рис. 3 можна встановити ще одну принципову особливість ПСО-схем. Для цього звернемо увагу на дію 3, яка в цілому представляє окремий сам по собі складний процес, що може оперувати своїми об'єктами за допомогою своїх суб'єктів. Отже, ПСО-схеми можуть легко каскадуватися за рівнями деталізації, тобто можемо розглядати схему алгоритму високого рівня, у якого деякі стрілки (дії) потім можуть деталізуватися у вигляді своїх окремих ПСО-схем. Таким чином, зручно реалізується принцип проектування зверху-вниз.

Висновки

Розроблено методику графічного представлення алгоритмів роботи СЗІ у вигляді процесно-суб'єктно-об'єктних схем. ПСО-схеми дають чітке уявлення про послідовність дій алгоритму, реалізують наочне представлення суб'єктів та об'єктів, що є складовими частинами СЗІ, а також потоків інформації між ними. За допомогою ПСО-схем можна легко встановлювати логіку роботи програмних СЗІ, аналізувати або синтезувати необхідні алгоритми, переходячи між різними рівнями деталізації. В якості реального прикладу використання створеної методики, зображено алгоритм роботи авторської СЗІ, що здійснює контроль цілісності підсистеми фізичного захисту ПЗ.

Список літератури

1. ГОСТ 19.701-90. Схемы алгоритмов и программ. Обозначения условные графические.

2. *Евстигнеев В.А.* Применение теории графов в программировании. / Под ред. А.П. Ершова. – М.: Наука, 2000. – 352 с.

3. *Калянов Г.Н.* Структурный системный анализ (автоматизация и применение). – М.: Издательство "ЛОРИ", 1996.

4. *Гальчевський Ю.Л., Гайша О.О.* «Логічні» та «фізичні» захисти програмного забезпечення від несанкціонованого копіювання // Захист інформації: Науково-технічний журнал. – 2005. – №2(23). – С. 34-40.

Надійшла 20.12.2006 р.

УДК 65.012.8: 004.492

Гроздов Г.Г.

МЕТОДИКА ОПРЕДЕЛЕНИЯ ПОКАЗАТЕЛЯ ЭФФЕКТИВНОСТИ МЕХАНИЗМОВ ЗАЩИТЫ ИНФОРМАЦИИ

В настоящее время актуальна задача построения и оценки эффективности механизмов защиты информации в различных комплексных системах защиты информации (КСЗИ) автоматизированных систем (АС).

Можно выделить такие классы информации, защита которых должна обеспечиваться механизмами защиты информации (ЗИ), входящими в состав КСЗИ: информация, составляющая коммерческую и военную тайну. В АС указанных классов могут иметь приоритетное значение различные требования к механизмам ЗИ [1, 2].

В научно-технической литературе рассматриваются два аспекта эффективности системы ЗИ. С одной стороны, система защиты информации должна эффективно