Рассмотрим теперь ситуацию при передаче речи в реальном времени. При передаче речи в реальном времени необходимо использовать коды с прямым исправлением ошибок. В качестве исходной ситуации в среде передачи возьмем при

b=7 и ΔРфкс db =0,1db, тогда Рош дк = $1{,}5\cdot10^{-2}$.

Для коррекции ошибок используем БЧХкод, исправляющий четырехкратные ошибки и блок длиной 63 бита. Отсюда непосредственно вытекает, что необходимо использовать код-произведение 4 полномов 6 степени(потому, что это минимальная степень, при которой могут быть сформированы локаторы ошибки в блоке из 63 бит, а 4 полинома – поскольку кратность исправляемой ошибки t= 4). При t= 4 кодовое расстояние

d=2t -1=7. Отсюда по таблицам [3] получим:

$$G(x) = (x^6 + x + 1)(x^6 + x^4 + x^2 + x + 1)(x^6 + x^5 + x^2 + x + 1)(x^6 + x^3 + 1).$$

Определим вероятность правильного приема 63-битового блока данных как:

$$Q = \sum_{t=0}^{4} P(t) \qquad (12),$$

где P(t)- вероятность появления t-кратной ошибки и

$$P(t) = C_{63}^{t} p^{t} q^{63-t} \qquad (13).$$

Тогда Q=0,9973, а Рош бл=$2{,}7\cdot10^{-3}$. Если не применять коррекцию ошибок, то в этих же условиях Ро бл =1-$(1-p)^{63}$=0,614. Таким образом, фактор повышения достоверности оказывается равным F=0,614:0,0027=227, что эквивалентно снижению вероятности ошибок в среде передачи с 0,015 до значения 0,015:227=$6{,}6\cdot10^{-5}$или росту защищенности примерно на 3db. Отсюда следует, что при работе в реальном времени использование кода с прямым исправлением ошибок весьма эффективно, но требует увеличения скорости передачи для компенсации потери пропускной способности, обусловленной введением избыточности. Для рассмотренного примера n=63,а m=63-24=39. тогда $\nu_{\text{пд}}$ =39:63=0,619. При передаче речи на скорости 64Кбит/сек, линейная скорость должна быть равна Вл= Врк: $\nu_{ст}$ =64:0,619=103,4 Кбит/сек.

Выводы

Выполненные расчеты показывают условия реализуемости канала передачи речевой информации в сети электропитания и позволяют, в зависимости от предъявленных требований выбрать оптимальные параметры устройства.

Список литература

1. *Варакин Л.Е.* Системы связи с шумоподобными сигналами. - М.: Радио и связь, 1985. – с. 478.

2. *Швыдкий В.В., Зеленько Ю.В., НочевновД.П.* Оценка обнаруживающей способности БЧХ кодов Сборник «Вісник ЧІТІ» №3 1999р.

3. Передача дискретных сообщений Сборник под редакцией В.П.Шувалова М., «Радио и связь»,1990.

Поступила 10.12.2004г.

УДК 681.3.06

Гальчевський Ю.Л., Гайша О.О.

"ЛОГІЧНІ" ТА "ФІЗИЧНІ" ЗАХИСТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІД НЕСАНКЦІОНОВАНОГО КОПІЮВАННЯ

Проблема піратства є досить важливою і можна сказати наболілою для сучасного світу інформаційних технологій. Економічні аспекти розповсюдження нелегального програмного забезпечення (далі - ПЗ) досить докладно описані в літературі (див., наприклад [1]). Нажаль цивілізованими методами запобігти піратству не уявляється

можливим, адже відповідні нормативно-правові акти залишаються переважно "на папері". Організаційні заходи, такі як реклама, знижки, заохочення також не дають прийнятного результату.

Опускаючи економіку, право та громадську мораль на другий план, спробуємо побудувати систему захисту, що ефективно протидіяла б незаконному розповсюдженню ПЗ.

Одразу відкинемо програми, у яких кожна копія потребує одного і того ж серійного номера незалежно від інших параметрів установки. Одну законно куплену копію такої програми можна просто розтиражувати на дисках разом з її серійним номером, тому такий підхід є надто непрофесійним. Застосування такого методу захисту великими компаніями-виробниками ПЗ (як Microsoft у ОС Windows, або Adobe) пояснюється їх сподіванням на правову протидію піратству. Тут же спробуємо створити систему захисту, яка не дозволятиме простого тиражування разом з серійним номером. (Ситуацію, коли одна ліцензійна копія може бути розтиражована на дисках разом з необхідними реквізитами і нормально працюватиме на інших машинах, назвемо "тиражним" зломом).

Підемо зворотним шляхом і згадаємо як звичайно відбувається злом ПЗ. Поділимо всі методи злому на дві групи: ті що здійснюються з модифікацією коду програми, що підлягає злому, та, відповідно, без його модифікації.

Перший тип підпадає під поняття "фізичного" злому, при якому відбувається фізичне викривлення операторів, що у програмі відповідають наявній системі захисту. Розглянемо приклад. При введенні користувачем пароля, який затверджуватиме законність даної копії, програміст передбачив наступний звичайний код:

```
If (!strcmp(pass,"GoodPassword") AllRight();
else BadPassword();
```

Тобто введений пароль разз порівнюється з еталоном GoodPassword і при збіганні викликається функція AllRight(); якщо ж введено невірний пароль, то викликається функція BadPassword(). Даний код транслюватиметься в приблизно такий набір команд процесора (дуже спрощено):

```
cmp <pass1>,<pass2>
jnz BadPassword
AllRight:
BadPassword:
<exit>
```

Тут порівняння здійснюється командою cmp, яка впливає на флаг нуля процесора і встановлює його, якщо паролі однакові. Команда стрибку на функцію BadPassword відбувається за умови що флаг нуля скинутий (отже цей стрибок відбувається, якщо паролі не співпадають), інакше продовжується виконання програми і відбувається перехід на мітку AllRight. Зломщику достатньо замінити в програмному файлі команду стрибку за умови на логічно протилежну, наприклад jnz на jz або je на jne.

Тоді нормальне виконання програми відбудеться в усіх випадках, окрім введення правильного паролю. Далі зломщик може написати невеличку програмку - так званий патч (від англ. patch - латка), яка буде в автоматичному режимі здійснювати вказану заміну. Таким чином, стає можливим розповсюдження незаконного ПЗ.

Принципово іншим типом злому є написання окремої програмки, що в необхідній мірі імітує роботу системи захисту. Такий вид злому відбувається без змін в оригінальному коді програми, що ламається, і підпадає під поняття "логічного", адже базується на імітації логіки роботи системи захисту (та іноді на логічних помилках програміста).

Наведемо приклад. При реєстрації деяких програм, аби попередити "тиражний" злом, у користувача запитуються якісь персональні дані, а внутрішній серійний номер вираховується по секретному алгоритму на їх основі. Законний користувач отримує зовнішній серійний номер на сервері виробника ПЗ, вказавши свої дані і заплативши гроші. При збіганні двох номерів система захисту вважає програму ліцензійною. Наочно процес реєстрації показаний на рис. 1. Зловмисник вивчає алгоритм роботи процедури

вирахування внутрішнього серійного номеру і пише програмку - генератор паролів (KeyGen, Key Generator, генератор ключів), що імітує ці дії. Відправлення інформації на сервер виробника більше не потрібно, а зовнішній серійний номер вираховується генератором. Таким чином, відбувається "логічний" злом ПЗ.

Також підтипом "логічного" злому є використання логічних помилок програміста, що розглянемо на прикладі уже знайомого коду, написаного для перевірки пароля:

if (!strcmp(pass,"GoodPassword") AURight(); else BadPassword();

Принципіальною слабкістю тут є зберігання еталонного пароля у відкритому вигляді. Така логічна помилка базується на тому, що програміст не передбачив можливості безпосереднього вивчення машинного коду програми. Справді, зломщик може дизасемблювати програму і подивитися, з чим звіряється введений пароль (тобто куди вказує адреса passl у наведеному вище асемблерному лістингу). Якщо навіть в кожному виконуваному файлі прошитий свій унікальний пароль, можна написати просту програмку, що діставатиме його звідти, бо адреса цього паролю все ж буде статичною.

Таким чином, визначили два типи злому: "фізичний" і "логічний", причому методи "логічного" злому можна поділити на дві частини: "тиражний" злом та використання логічних помилок програміста. Наведена класифікація зображена на рис.2. Який саме тип злому обирати, зловмисник вирішує за принципом атаки найслабшої ланки.

Отже, мають існувати два типи захисту, назвемо їх відповідно "фізичний" і "логічний". "Логічний" захист має протидіяти "тиражному" злому (нелегальному розповсюдженню однієї, колись проданої, копії шляхом тиражування її разом з необхідними для установки реквізитами). Логічні помилки не можна викорінити жодним формалізованим методом, адже вони притаманні людському методу мислення і часто з'являються завдяки неправильній логічній базі розробника (тобто із-за неправильного твердження окремих вихідних положень, або повного їх неврахування). Тому єдиним методом боротьби з логічними помилками є загальне підвищення кваліфікації програміста та його добра обізнаність у методах оберненого проектування (reverse engineering). "Фізичний" захист має забезпечувати цілісність коду і, отже, правильне виконання всіх функцій системи захисту, що їх загадав програміст.

Таке розподілення захисних функцій є досить логічним, адже спочатку (на вищому рівні абстракції) програміст реалізує захист від прямого тиражування однієї ліцензованої копії, не турбуючись про можливості його фізичного злому, а потім (на нижчому рівні) забезпечує правильність виконання написаного коду, шляхом впровадження контролю його цілісності.

Перейдемо до безпосереднього розгляду схем, що пропонуються в даній роботі.

Спочатку розглянемо, як може реалізуватися ідеальний "логічний" захист від "тиражного" злому. У схемі, зображеній на рисунку 1 є дві принципові слабкості. По-перше, стійкість захисту спирається на секретність алгоритму, що є непрофесіональним. Саме тому, що цей алгоритм вбудований в кожну копію ПЗ, зломщик може його вивчити і відтворити. По-друге, на сервер передаються персональні дані користувача, що можуть бути легко підроблені (адже достатньо вказати не своє ім'я, а те, що вказане на нелегальному диску).

Для попередження імітації алгоритму пропонуємо застосовувати криптографічні протоколи, а саме доказ при нульовому знанні [2]. При доказі при нульовому знанні один абонент мережі (умовно назвемо його сервером) має якусь секретну інформацію, яку він не хоче розголошувати іншим абонентам (назвемо їх відповідно клієнтами), але може підтвердити будь-якому з них, що вона в нього є. Такий доказ базується на вирахуванні значення деякої функції, аргументами якої є секретна інформація сервера, та деякі параметри, що задаються клієнтом. Причому функція має бути такою, щоб клієнт міг впевнитися шляхом обчислень, що сервер справді має ту секретну інформацію. Така функція може, наприклад, являти собою цифровий підпис переданих клієнтом аргументів. Тепер неважко провести аналогію з системою законного розповсюдження програмного забезпечення. Сервер та клієнт виконують ті ж самі ролі, що й при безпосередній реалізації системи доказу при нульовому знанні. В ролі аргументів, що їх задають клієнти є ідентифікаційні

дані користувача, а доказом законності є істинність електронного підпису, що його присилає сервер. В ролі секретної інформації виступає будь-яка інформація, яку забажає спеціаліст з захисту (наприклад, секретний ключ двохключової системи шифрування). Найголовнішим є те, що цієї інформації немає в клієнтському ПЗ (тобто в програмі, що захищається), а отже її не має і зломщик. Тому стійкість такої системи захисту буде спиратися на стійкість криптографічного алгоритму, що забезпечуватиме доказ при нульовому знанні. Наочно весь процес реєстрації наведений на рис. 3.

Що стосується другої слабкості, то окрім персональних даних користувача, у ідентифікаційному файлі слід передавати на сервер ще й "персональні" дані комп'ютера. Це можуть бути дані про апаратну конфігурацію (наприклад, серійні номери його комплектуючих), та навіть про програмне середовище (наприклад, перелік встановлених програм). Тоді установка програми буде можливою лише на той комп'ютер, що сформував ідентифікаційний файл. Звичайно, що при цьому з'являтимуться невеликі незручності для користувачів, адже при зміні "портрету" ІЖ і виникненні необхідності переустановлення програми, слід буде знов з'єднуватися з сервером виробника. Але цей процес можна автоматизувати, та при повторних установках зробити його навіть непомітним для користувача (все ж потрібне з'єднання з Інтернет). Суттєвою ж перевагою такого "логічного" захисту є повний відхід в минуле генераторів ключів, як класу програм.

Таким чином, злом ПЗ без внесення змін до його коду стає надто малоймовірним і забезпечується стійкістю криптографічного алгоритму, що використовується в протоколі доказу при нульовому знанні, а також відсутністю логічних помилок.

Розглянемо тепер, що можна зробити, аби завадити зловмиснику викривити код програмного забезпечення.

По-перше, зазначимо, що блок перевірки цілісності має бути зашифрованим. В іншому випадку зломщик зможе модифікувати код самого цього блоку таким чином, що успішна перевірка буде зафіксована в усіх випадках крім правильного. Це може бути зроблене шляхом тривіальної, розглянутої вище, заміни асемблерної умовної команди на логічно протилежну (а така кінцева команда має бути обов'язково присутньою перед винесенням вироку - модифікований файл чи ні). Також має бути зашифрованим і файл-еталон, що зберігатиме образ програми.

По-друге, звернемо увагу на те, що все ж мають існувати якісь незашифровані оператори, що здійснюють розпаковку блоку перевірки. Тоді зломщик може вислідити, де починаються такі оператори і куди передається управління після закінчення перевірки цілісності у ліцензійній копії ПЗ. Далі він просто вставляє команду стрибку з того місця, де вперше викликаються оператори причетні до перевірки цілісності, до місця в програмі, де контроль цілісності успішно завершується. Таким чином, програма працює правильно, але блок перевірки ніколи не викликається (він виключений з усіх гілок виконання). Висновок: програма не має нормально виконуватися, якщо не був викликаний блок перевірки (і відповідно не зафіксований її успіх). Цього можна досягти шифруванням окремих невеликих, але критичних ділянок програми якимсь стійким алгоритмом, з подальшим вміщенням ключа до блоку перевірки. Таким чином, без його роботи ці ділянки не розшифровуються і програма нормально не працює.

Критичної до виконання програми інформації може бути багато, тому її краще зашифрувати різними ключами і виділити якийсь блок ключової інформації, який зберігатиметься не у відкритому вигляді, а зашифрований майстер-ключем, який якраз і буде якимсь чином вбудований до блоку перевірки. Наголос на невизначеності алгоритму вбудовування майстер-ключа в блок перевірки ґрунтується на наступних роздумах. Як би не була зашифрована програма, перед виконанням процесором код має бути розшифрованим, а отже має існувати секретний ключ до розшифровки, причому знаходиться він на стороні зловмисника, а, отже за принципом найгірших припущень, відомий йому. Тому можна лише зводити додаткові рубежі у фізичній обороні, що мають сильно збільшити трудомісткість злому. До того ж тут нажаль доводиться спиратися на секретність місцезнаходження майстер-ключа, адже він

знаходитиметься в блоці перевірки, і може бути несанкціоновано розшифрованим.

До речі, зміст в шифруванні блоку перевірки та інших ділянок коду, що здійснює перевірки, взагалі є лише в тому випадку, якщо застосовується двохключовий алгоритм [3]. Справді, при застосуванні симетричного шифру зловмиснику достатньо зробити свій варіант блоку перевірки і зашифрувати його одним відомим ключем (а він має бути відомим, щоб розпакувальник міг його дістати). При застосуванні ж двохключового алгоритму виявляються цікаві можливості: код блоку перевірки розшифровується відкритим ключем, а зашифрований він секретним ключем виробника. Таким чином, розшифровувати та виконувати код може хто завгодно (за допомогою відкритого ключа), а модифікувати його може лише виробник (бо лише в нього є секретний ключ).

Вся розроблена схема "фізичного" захисту для конкретного програмного продукту наочно показана на рис. 4. Розглянемо принцип її роботи.

Програмний файл містить деякі критичні ділянки, без яких неможливе його виконання, зашифровані швидким симетричним алгоритмом за допомогою різних ключів, що утворюють множину S. Множина S, зашифрована майстер-ключем M, який вбудований в блок перевірки цілісності, що міститься в окремій динамічній бібліотеці (DLL). Там же знаходиться розпакувальник блоку перевірки цілісності. Сам блок зашифрований двохключовим алгоритмом.

Програмний файл викликає розпакувальник. Той, за допомогою відкритого ключа, розшифровує в пам'ять код блоку перевірки. Той розшифровує також відкритим ключем файл-еталон, що містить образ оригінального програмного файлу (наприклад, його контрольну суму) і порівнює еталон з наявним значенням. При збіганні відтворюється ключ M, і за допомогою нього розшифровуються ключові дані S. По мірі виникненні необхідності, функції, вбудовані в динамічну бібліотеку, за допомогою відповідного ключа з множини S, розшифровують критичну інформацію, що міститься в програмному файлі, і відбувається нормальне виконання програми.

Вузловим моментом, завдяки якому взагалі став можливим даний підхід до "фізичного" захисту, є застосування двохключових алгоритмів шифрування, які дозволяють створювати інформацію (код або дані), яка може бути прочитана ким завгодно (з використанням відкритого ключа), але не може бути модифікована жодним суб'єктом, окрім власника секретного ключа. В такому випадку стійкість запропонованого захисту базується на стійкості алгоритму двохключового шифрування. Таким чином, маємо використання загальновживаних криптопротоколів, але в зворотному напрямку, навпаки. При традиційному їх використанні інформація шифрується клієнтом за допомогою відкритого ключа, а розшифровується на сервері секретним ключем. В пропонованому ж випадку інформація шифрується сервером за допомогою секретного ключа, а розшифровується у клієнта відкритим (ключі рівноправні).

Застосування наведеного "логічного" захисту і надійної технології контролю цілісності коду має повністю вирішити проблему комп'ютерного піратства, адже воно стане просто неможливим. Цікаві перспективи могла б дати технологія контролю цілісності коду, якби вона була запроваджена на рівні операційної системи і, відповідно, мала привілеї, недоступні для звичайних програм. Тоді побудова системи захисту, стійкість якої рівна стійкості застосованих криптографічних алгоритмів стала б можливою.

Застосування ж запропонованого "логічного" захисту повністю робить неможливим такий розповсюджений вид злому, як написання генераторів ключів. Розроблений метод використання двохключових алгоритмів "навпаки", дозволяє створити ділянки коду та даних, що можуть бути прочитані, але не можуть бути змінені. Така методика може бути корисною при побудові інших систем контролю цілісності, особливо для зберігання еталонних образів інформації.

Список літератури

1. Матеріали незалежного дослідження "Expanding Global Economies:The Benefits of Reducing Software Piracy" //TOC&BSA:http://www.idc.com, http://www.bsa.org, 2004.

2. *Володимир Жельніков*, Криптографія від папірусу до комп'ютера.-М: ABF, 1996.

3. У. Диффи. Первые десять лет криптографии с открытым ключом. //ТИИЭР, т.76, №5, Май 88 - М, Мир, 1988.

Надійшла 10.11.2004р.

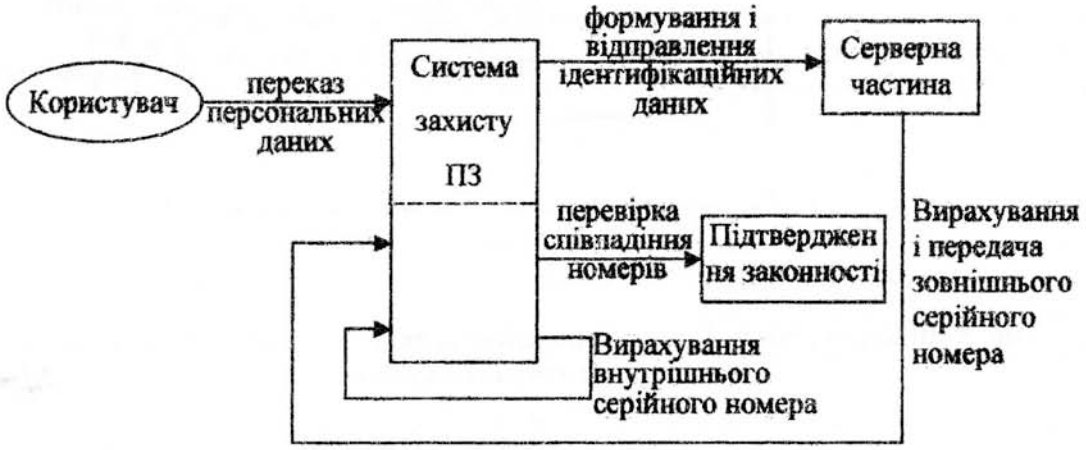

Рис. 1 - Принцип роботи "логічного" захисту, при якому серійний номер вираховується на основі ідентифікаційних ознак користувача.

Рис. 2 - Класифікація методів злому ПЗ.

Рис. 3 Принцип роботи "логічного" захисту, який не може бути зламаний шляхом написання генератора ключів.

Рис. 4 Можлива схема примусового контролю цілісності коду для конкретного програмного продукту