основы взлома мобильных игр

крис касперски ака мыщъх, a.k.a. nezumi, a.k.a souriz, no-email

надоело платить за мобильные игры? заколебала череда бесконечных смертей? хочется переделать спрайты/тексты/заставку под свой вкус? нет ничего проще!!! мобильные игры весьма компактны и потому просты для взлома и анализа. надругаться над ними сможет даже начинающий хакер, а эта статья поможет ему сделать первые шаги, после которых он будет оттачивать хакерское мастерство уже самостоятельно

введение

Популярность мобильных игр стремительно растет. Они прочно оккупировали рынок сотовых телефонов, коммуникаторов, смартфонов, карманных компьютеров и других аналогичных устройств. Большинство игр распространяется на условно-бесплатной основе, то есть требуют деньги, иначе — блокируют часть возможностей и/или ограничивают количество запусков. Но даже полностью бесплатные игры не лишены недостатков. Неудобное управление, быстро кончающиеся жизни… Да мало ли существует причин, побуждающих хакера дорабатывать код в соответствии со своими предпочтениями? Этических проблемы взлома нас не волнуют, поэтому мы немедленно переходим к технической части, благо хвост уже зудит, чешется и рвется в бой. О взломе мобильных игр написано много, но все как-то неконкретно и не в тему. Не так-то просто обобщить свой опыт и передать его другим. Но мыщъх все же попробовал….

мобильные платформы

Основная масса мобильных игр (по некоторым оценкам аж до ~70%) пишется на Java, а точнее — **J2ME**, что расшифровывается как ***Java 2 Micro Edition***. Это урезанная версия языка Java, ориентированная на маломощные системы и поддерживающая огромное множество мобильных устройств. Вместо "живого" машинного кода, сотовому телефону подсовывают так называемый "байт-код", исполняющийся на виртуальной Java-машине (**Java Virtual Machine** или сокращенно **JVM**). Теоретически, игра, написанная для одного сотового телефона, будет работать на любом другом, независимо от особенностей его аппаратного обеспечения, что очень хорошо (хотя на практике переносимость намного хуже). Расплачиваться за это приходится драматическим падением производительности, в условиях и без того маломощных микропроцессоров.

Продвинутые игры (наподобие Fight Hard 3D и RiderX 3D) пишутся на чистом машинном коде, и потому могут исполняться только на микропроцессорах одного семейства (например, ARM 6), что ограничивает сферу их применения. В настоящей статье они не рассматриваются. Поскольку, нельзя объять необъятное, мы сосредоточимся исключительно на взломе Java-приложений, а до Fight Hard'a доберемся не раньше, чем мыщъх купит соответствующий сотовый телефон.

Рисунок 1 телефон Siemens S55 с ИК-адаптером от iRwave

чем мы будет ломать

Выбор хакерского инструментария — дело сугубо личное, индивидуальное и, можно даже сказать, интимное. Поэтому, не следует воспринимать ниже перечисленный список как догму. Это всего лишь один из вариантов. Практически все обозначенные утилиты реализованы в двух-трех вариантах — как консольных, так и графических. Так что каждый может найти программу под свой вкус. Предлагаемая подборка включает в себя только бесплатные программы, игнорируя их коммерческие аналоги, иначе это не хакерство получается, а сплошной рекурсивный спуск (чтобы сломать мобильную игру нужно хакнуть программу, которая ее ломает).

- прежде всего нам потребуется **спецификация на байт-код виртуальной Java-машины**, выложенная на официальном сайте корпорации Sun (на английском языке): http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html, при этом знать сам язык Java совершенно необязательно, хоть и желательно. во всяком случае мыщъх несколько лет успешно хачил Java-приложения непосредственно в JVM, пока, наконец, не купил "Горький вкус Java" Брюса Тейта, разобравшись с основными языковыми концепциями, которые, кстати сказать, ничуть не облегчили ни дизассемблирование байт-кода, ни его анализ;
- лучшим дизассемблером Java-программ была и остается легендарная **IDA Pro**, распространяющаяся на коммерческой основе за нехилые деньги, однако, при желании можно обойтись и без нее, воспользовавшись штатным дизассемблером, входящим в бесплатный Java SDK или любой другой утилитой аналогичного назначения, коих в последнее время развелось как грибов (см. JavaBite, описанный ниже);
- чтобы не корячится над анализом байт-кода, имеет смысл прогнать ломаемое приложение через Java-декомпилятор, выдающий вполне читабельные и структурированные листинги. Java-декомпиляторов существует много. хороших и разных. мыщъх рекомендует бесплатный **avaDec** by wl, которым пользуется сам и который можно скачать с http://www.wasm.ru/baixado.php?mode=tool&id=362. еще стоит попробовать **JDecompiler** с http://java-decompiler.qarchive.org. он так же неплох и вполне себе бесплатен. декомпилированный код можно хачить прямо в исходных текстах с последующей рекомпиляцией, но мыщъх этого делать не рекомендует, поскольку декомпилятор не всегда работает корректно и повторная компиляция

зачастую ведет к краху программы, поэтому, лучше падчить непосредственно сам байт-код;

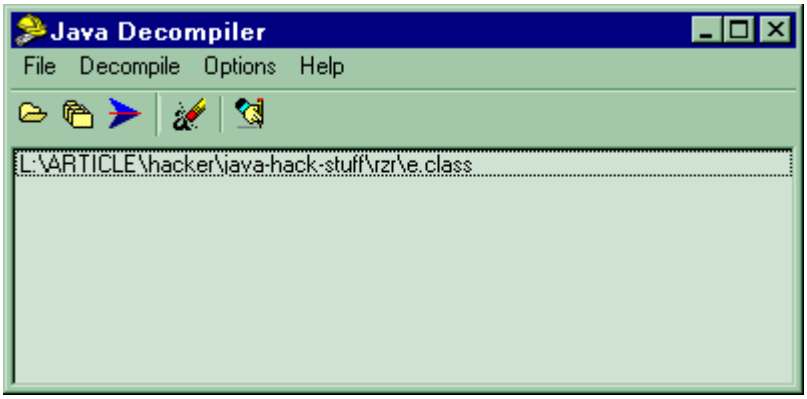

Рисунок 2 внешний вид Java-декомпилятора

- для модификации байт-кода (то бишь "бит-хака") подойдет любой hex-редактор, например, всем известный **hiew**, однако, лучше использовать специализированные инструменты, лучим из которых является бесплатный **JavaBite** by BitArts, наглядно отображающий дерево классов, а так же включающий в себя дизассемблер и ассемблер байт-кода (http://www.wasm.ru/baixado.php?mode=tool&id=284);
- мобильные игры как правило распространяются в виде упакованных jar-файлов, создаваемые одноименной утилитой, входящей в состав Java SDK, однако, это не единственно возможный вариант. архиваторы **7ZIP** (бесплатный) и WinAce (условно-бесплатный) справляются со своей задачей ничуть не хуже. исключение составляют Java-приложения, снабженные цифровыми подписями. ни 7ZIP, ни WinAce их создавать не умеют, да этого и не требуется. любой сотовый телефон загрузит jar-архив и без подписи;
- иногда рядом с jar-архивом лежит jad-файл, без которого некоторые модели телефонов откажутся загружать java-приложение и тут приходится прибегать к помощи бесплатной утилиты **JADgen**, генерирующий jad-файлы на основе jar-архивов (http://softsearch.ru/programs/134-892-jadgen-download.shtml);
- некоторые хакеры рекомендуют использовать **эмулятор сотового телефона** для проверки работоспособности взломанных игр. другие же (к числу которых принадлежит и мыщъх) предпочитают живое "железо", тем более что закачать приложение на телефон — не проблема. правда, при этом существует угроза "завесить" аппарат так, что придется вынимать батарею или даже делать полный rezet, удерживая определенные клавиши при включении (у каждой модели телефона — разные), описание которых можно найти в сервисной документации. впрочем, страхи угробить телефон некорректным взломом очень сильно преувеличены. в 90% случаях некорректный хак пресекается жутко матерящимся Java-верификатором. в 9% случаях игра просто зависает, подвешивая за собой весь телефон. и только в 1% происходит разрушение содержимого энергонезависимой памяти и прочий бэд. так что, неуверенным в себе хакерам все-таки стоит пользоваться эмулятором.

что мы будем ломать

А ломать мы будем милую игрушку **Macroman** (*реинкарнация культовой компьютерной игры выпущенной в 1979 году японской компанией Namco Topy, реализованной практически на всех 8-битных компьютерах типа ZX-Spectrum*), демонстрационная версия которой распространяется бесплатно и валяется практически на любом мобильном сайте: http://www.cec.ru/Files/macroman_demo.jar.

Поставим себе задачу — обессмертить колобка, чтобы игра никогда не кончалась. Главное — разобраться с техникой и стратегией взлома, освоив основные хакерские трюки и приемы. Остальные программы ломаются аналогичным образом и не важно что это — вечная жизнь или снятие ограничения с количества запусков.

Короче, кончай курить мужики! Курить мы будет потом, а сейчас глотнем пива и возьмемся за дело.

Рисунок 3 MacroMan Demo от компании Macrospace

как мы будем ломать

Пускаем мы, значит, Macroman'a и даем ему умереть на зубах зловредных существ (типа приведений), агрессивно бегающих по лабиринту. На экране появляется надпись: "1 Life Left" (осталась одна жизнь). Очевидно, что код, выводящий эту строку, так или иначе связан с кодом, уменьшающим количество жизней при каждом акте поедания колобка. Во всяком случае, во всех императивных языках программирования (к которым принадлежит и Java) ситуация обстоит именно так.

Вот эту строку мы и будем искать. Прямым текстом. Но сначала распакуем jar-архив, пропустив его через 7ZIP (предварительно изменив расширение с .jar на .zip). И вот что мы получим в результате:

```
META-INF        // директория с файлом манифеста в текстовом формате
IMAGES          // директория с изображениями лабиринта и спрайтов в png
MACROMAN        // директория с файлом MacromanMidlet.class в байт-коде
b.class         \
c.class          |
d.class          |
e.class          +- файлы классов в байт-коде
f.class          |
```

- g.class /

Листинг 1 содержимое распакованного jar-архива с ломаемой игрой

Берем FAR (или любой другой файл-менеджер), давим <ALT-F7> (Search), вводим маску файлов "*" (все файлы) и строку для поиска "Life Left", которую и обнаруживаем через секунду поиска в файле "e.class", занимающим всего 19 Кбайт.

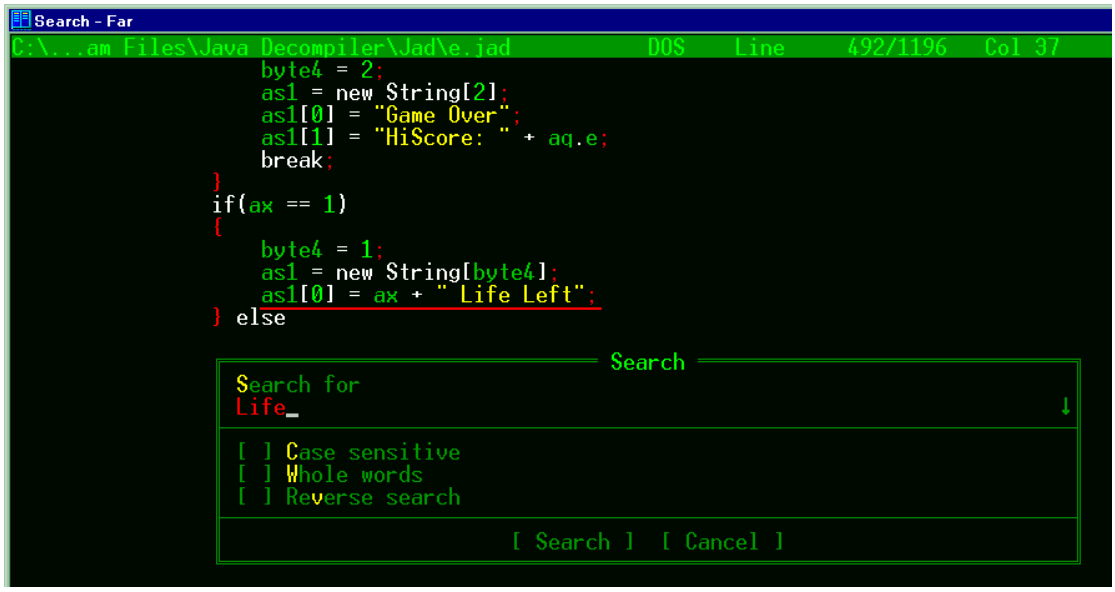

Рисунок 4 поиск строки "Life" в декомпилированном листинге

Прогнав "e.class" через JDec (или любой другой декомпилятор) мы получаем текстовой файл "e.java" размером порядка 36 Килобайт, который тут же открываем в FAR'e по <F4> (Edit), давим <F7> (Search) и вновь ищем строку "Life Left", затаившуюся в окрестностях следующего кода (см. листинг 2):

```
if(ax < 0)              // <- переменная ax, хранящая в себе кол-во жизней
{
        byte4 = 2;
        as1 = new String[2];
        as1[0] = "Game Over";
        as1[1] = "HiScore: " + aq.e;
        break;
}
if(ax == 1)
{
        byte4 = 1;
        as1 = new String[byte4];
        as1[0] = ax + " Life Left";    // искомая строка
}
```

Листинг 2 декомпилированный фрагмент Java-программы, найденный поиском строки "Live Left"

Машинная логика вполне стандартна и особых пояснений не требует. Если переменная "ax" становится меньше нуля — мы получаем "Game Over", в противном случае на экран выводится количество оставшихся жизней.

Следовательно, ***чтобы взломать программу, необходимо найти код, уменьшающий переменную "ax" на единицу при каждом акте смерти***. А как мы его найдем? Да все тем же контекстным поиском! Просто ищем "ax" контекстным поиском, анализируя прилегающий к ней код. Довольно быстро мы выкупим строку инициализации, устанавливающую начальный счетчик жизней равный двум (на самом деле — трем, с учетом того, что смерть наступает только если ax < 0):

```
private byte ax;
…
av = 0;
ax = 2;                 ; // инициализация счетчика жизней
```

```
        f.a(this, a4);
```

Листинг 3 фрагмент кода, отвечающего за начальное количество жизней

Можно, конечно, заменить строку "ax = 2" на "ax =69" (например), но это плохой и порочный путь. Во-первых, вечной жизни мы все равно не обретем, а во-вторых, еще не известно как программа отреагирует на такие издевательства (поскольку, количество оставшихся жизней отображается в виде "колобков" внизу экрана, то при слишком большом их числе поведение программы рискует стать непредсказуемым и крышесрывательным).

Ладно, идем дальше и… Видим заветную команду "`ax--`" в методе "`f()`", уменьшающую значение переменной "`ax`" на единицу (см. листинг 4).

```
public void f()
{
        MacromanMidlet.c();
        as = 3;
        removeCommand(b);
        if(ay <= 10)
        {
                addCommand(a);
                addCommand(e);
        }
                else
        {
                ax--;                   // <- вот здесь уменьшаются наши жизни!!!
                if(ax < 0)
                {
                        addCommand(a);
                        addCommand(e);
                        if(av > aq.e)
                        aq.e = av;
                }
                        else
                {
                        addCommand(f);
                }
        }
        c();
}
```

Листинг 4 декомпилированный фрагмент метода f(), уменьшающего переменную "ax" (счетчик жизней) на единицу

Вот это — то, что нужно!!! Остается найти байт-код, соответствующей данной конструкции языка высокого уровня. Вот тут-то нам и пригодится IDA Pro, ну или утилита JavaBite. Открыв файл "`e.class`" в любой из этих программ, переходим к методу "`f()`" и внимательно исследуем код на предмет обращений к переменной "`ax`".

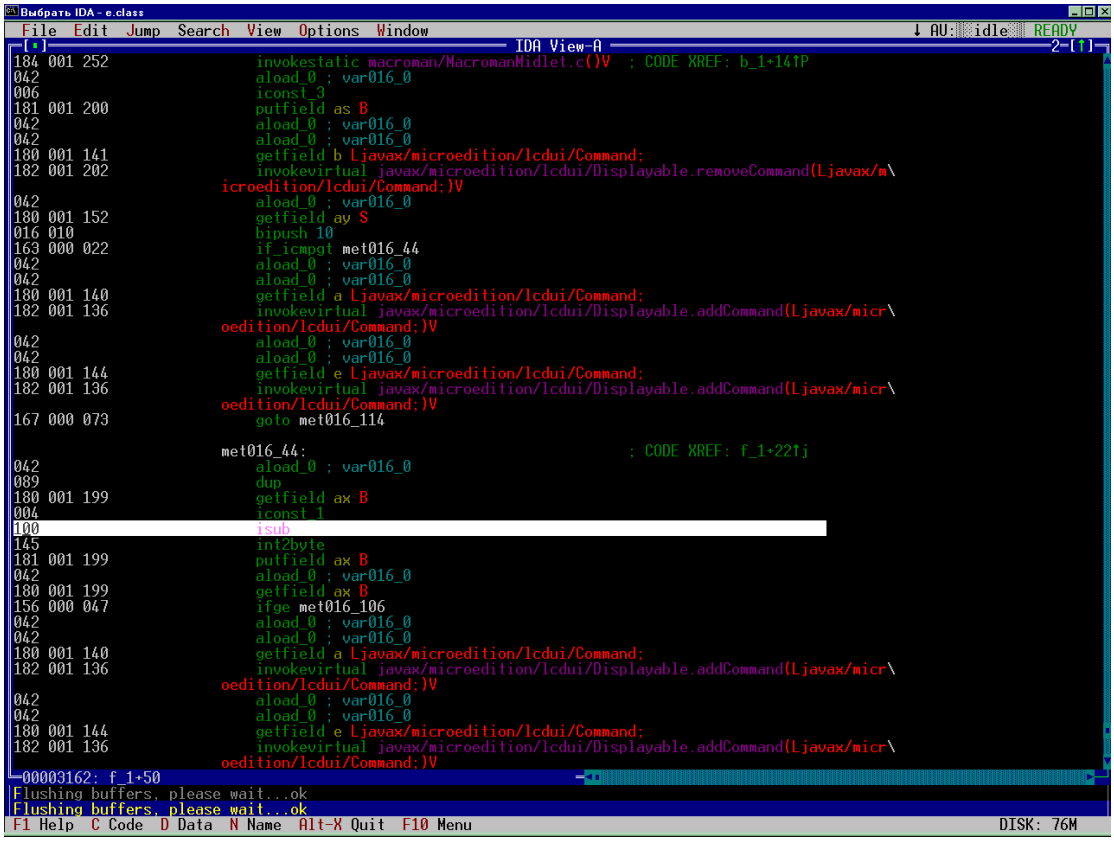

Рисунок 5 дизассемблированный байт-код в IDA Pro

Как легко видеть, в методе "`f()`" обращение к переменной "`ax`" встречается дважды (см. листинг 5):

```
met016_44:                                          ; CODE XREF: f_1+22↑j
042             aload_0 ; var016_0
089             dup
180 001 199     getfield ax B           ; // читаем переменную ax, закидывая ее на стек
004             iconst_1               ; // закидываем на стек константу 1
100             isub                   ; // стягиваем со стека две ячейки и вычитаем их
145             int2byte               ; // преобразуем в int и забрасываем на стек
181 001 199     putfield ax B          ; // обновляем содержимое переменной ax
042             aload_0 ; var016_0
180 001 199     getfield ax B
156 000 047     ifge met016_106
```

Листинг 5 фрагмент дизассемблированного байт-кода метода f(), уменьшающего переменную "ax" (счетчик жизней) на единицу

А что если заменить команду "`isub`" (опкод 64h/100) на "парную" ей команду "`iadd`" (опкод 60h/96)? Эту операцию легко осуществить в любом hex-редакторе, например, в hiew'e. Просто ищем последовательность "042/089/180 001 199/004/**100**/145/181 001 199" (окружающую инструкцию "`isub`") и заменяем 100 на 96. Тогда при каждом столкновении со злобными приведениями количество жизней будет увеличиваться на единицу и… в конце концов мы получим незапланированное переполнение и тогда — трындец. А нам трынденца не надо! Нам надо корректный взлом.

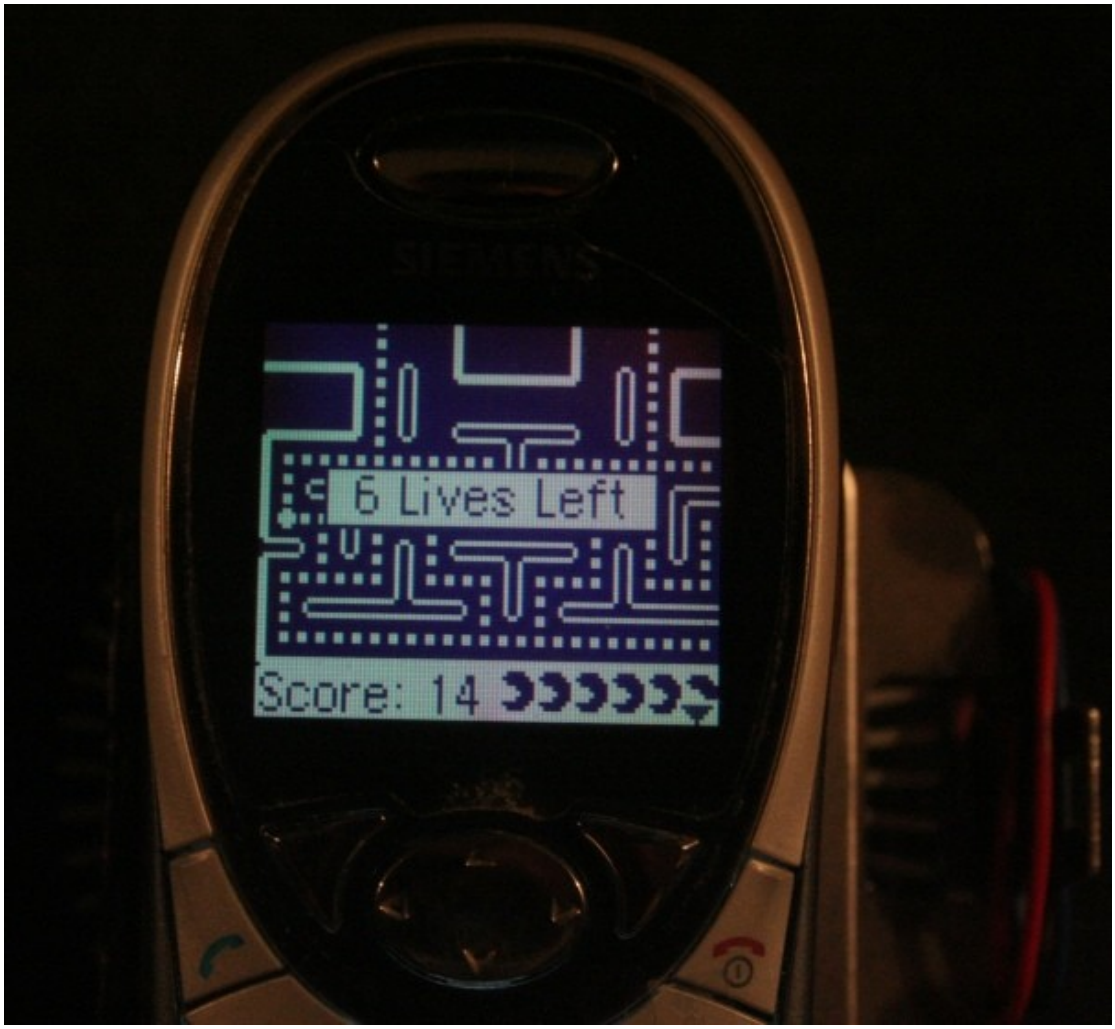

Рисунок 6 с каждой смертью количество жизней увеличивается на единицу

Хорошо! Попробуем заменить инструкцию "`isub`" на команду "`nop`" (опкод 00h). Кстати, говоря, это можно сделать прямо в JaveBite, не прибегая к помощи hiew'a. Достаточно подвести курсор к "`isub`", щелкнуть правой кнопкой мыши и в появившемся контекстном меню выбрать пункт "Edit Instuction". Откроется диалоговое окно со списком всем возможным команд. Находим "`nop`", жмем на "OK" и давим <Ctrl-S> (Save Class), чтобы сохранить результаты правки на диск.

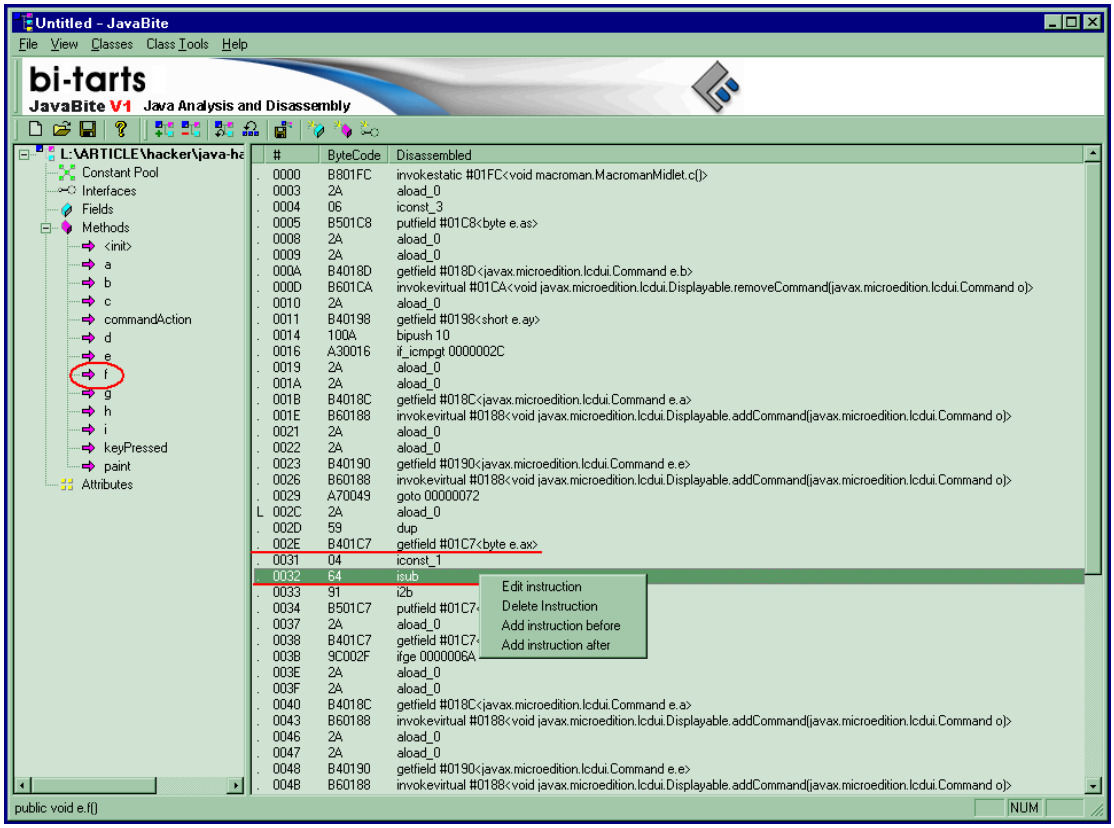

Рисунок 7 модификация байт-кода в JavaBite

Вот только результаты эти… мягко говоря довольно удручающие. И при запуске программы Java-верификатор завершает ее выполнение в принудительном порядке. Это в x86-процессорах с их регистровой архитектурой инструкцию SUB можно безболезненно менять на NOP. Виртуальная машина Java исповедует иной принцип и аргументы команды "`isub`" предварительно забрасываются на вершину стека, в расчете на то, что она стащит их оттуда. Замена "`isub`" на "`nop`" вызывает дисбаланс стека и чтобы восстановить статус-кво необходимо так же "занопить" и команду "`iconst_1`". Инструкцию "`int2byte`" можно не трогать, т. к. она имеет нулевой побочный эффект, сохраняя стек в том состоянии, в каком он был до ее вызова.

Короче говоря, корректно хакнутый байт-код выглядит так:

```
met016_44:                                              ; CODE XREF: f_1+22↑j
042             aload_0 ; var016_0
089             dup
180 001 199     getfield ax B           ; // читаем переменную ax, закидывая ее на стек
000             nop                    ; // ничего не делаем
000             nop                    ; // ничего не делаем
145             int2byte               ; // преобразуем в int и забрасываем на стек
181 001 199     putfield ax B          ; // обновляем содержимое переменной ax
```

Листинг 6 байт-код, получивший "бессмертие" (хакнутые байты выделены полужирным шрифтом)

Сохранив изменения в класс-файле по <Ctrl-S> (или <F9> если мы работаем в hiew'e), нам остается только упаковать все файлы обратно в jar-архив и залить его на сотовый телефон. Для тестирования так сказать.

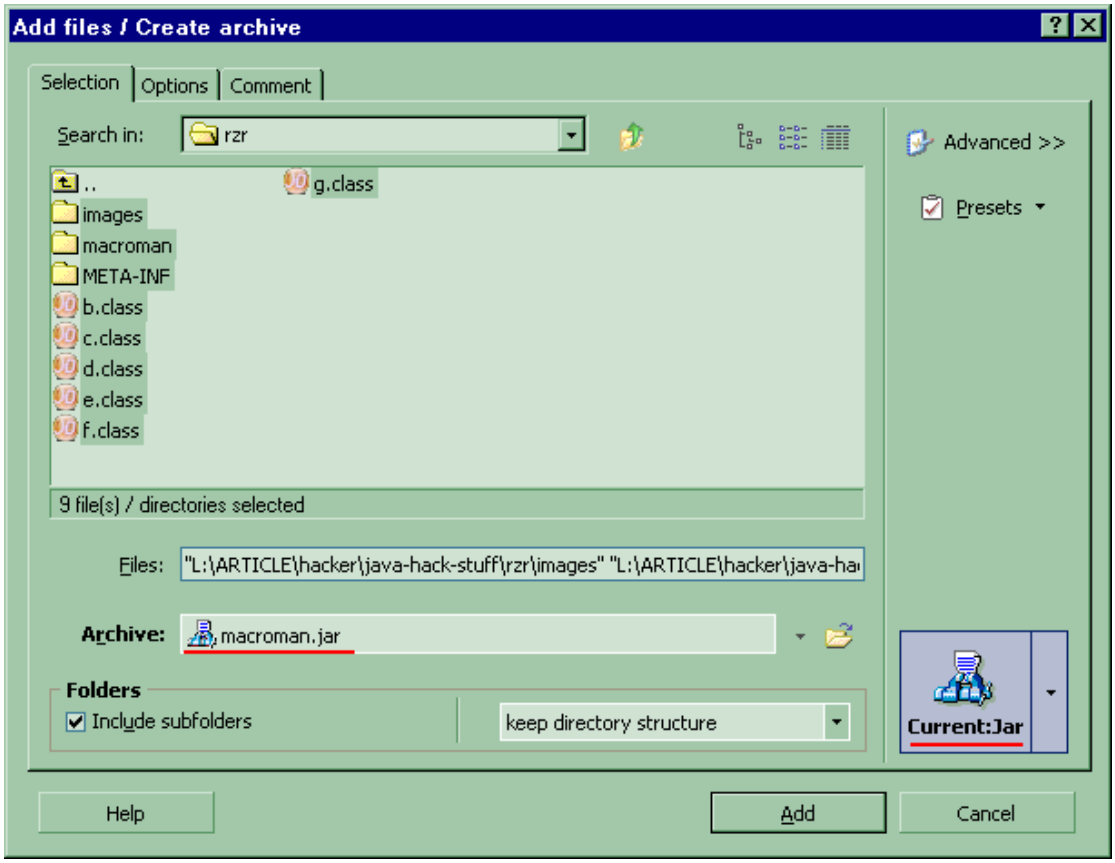

Рисунок 8 создание jar-архива с помощью WinAce

При использовании WinAce достаточно выделить все файлы (включая каталоги), в типе архива указать "JavaSoft-Jar" и плюхнуться на "OK" (см. рис. 8). А вот среди выходных форматов, поддерживаемых архиватором "7ZIP" никакого jar'a нет! То есть, он конечно есть, просто называется Zip'ом. В "Archive format" указываем: "ZIP", в "Compression level" – "Normal", поле "Compression method" выставляем в "Deflate". Остальные параметры оставляем по умолчанию — как есть. Главное, не забыть вместо расширения ".zip" указать ".jar". Ну а имя файла может быть каким угодно.

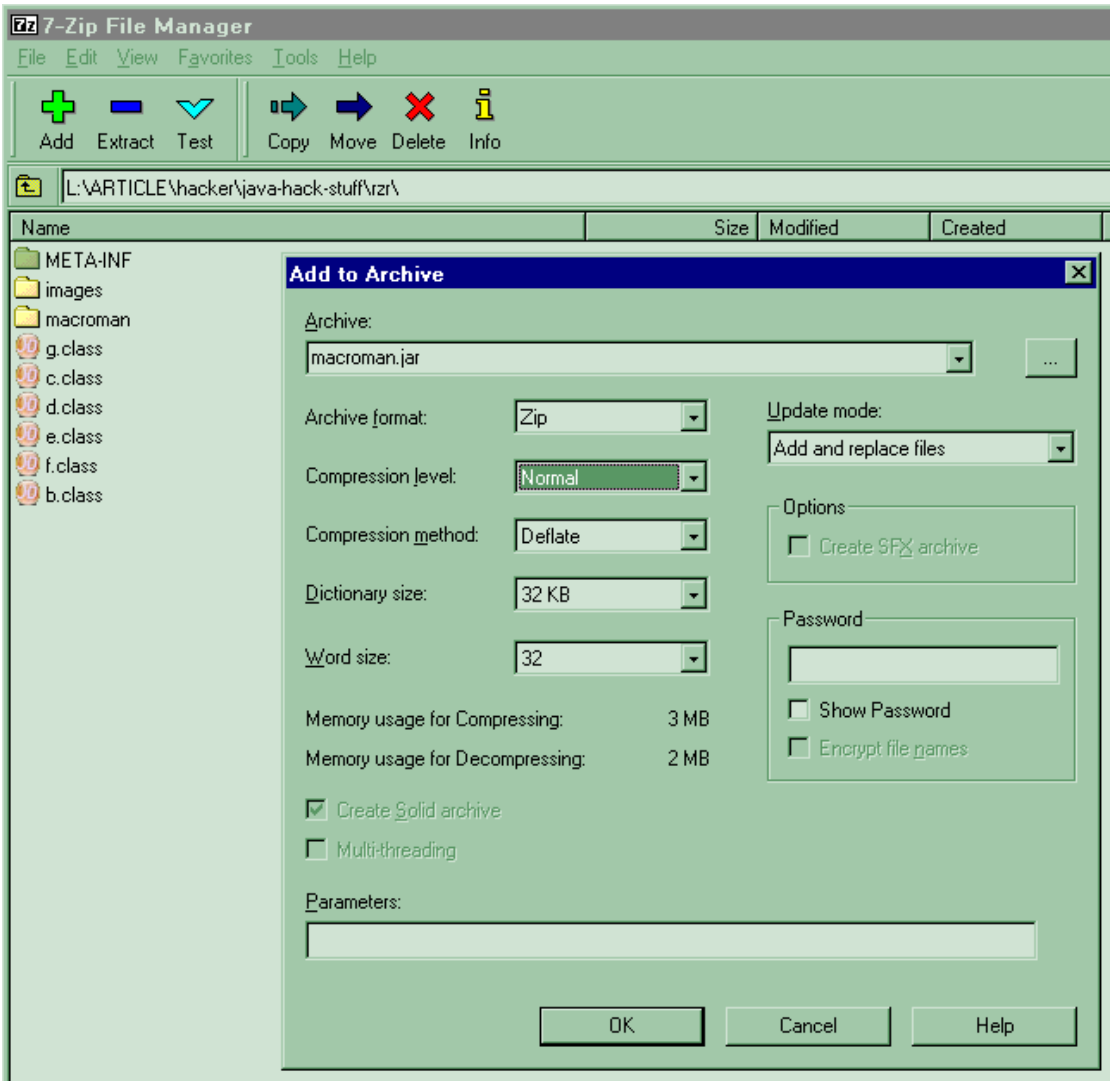

Рисунок 9 создание jar-архива в 7ZIP

заливаем игру на телефон

Вот мы имеем свежеиспеченный хакнутый файл Macroman.jar. Будем заливать его на телефон? А то!!! Сделать это можно разными путями. Например, по инфракрасному порту, Голубому Зубу, прямому кабельному соединению или выложить файл на свой собственный http-сервер, а потом стянуть его оттуда через GPRS. В общем, вариантов множество. Лично мыщъх предпочитает ИК, пример связи с которым продемонстрирован ниже.

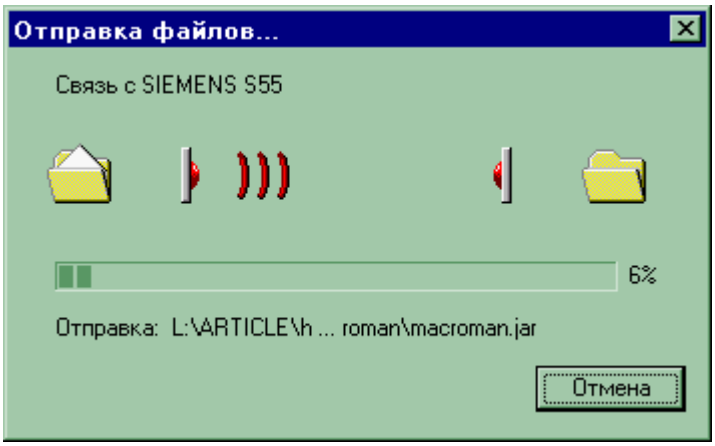

Рисунок 10 заливка игры на телефон по ИК

Итак, игра залита на телефон и… Дрожащими от волнения руками (все-таки наш первый взлом как-никак) мы едва попадаем по клавишам, запускаем игру и… о чудо!!! Она работает! (В смысле что не падет) И самое главное — счетчик жизней навечно застыл на отметке двух. Мы обрели бессмертие, а вместе с ним утратили весь игровой азарт и интерес… Но какой интерес играть в игры? Вот ломать их — настоящий кайф!!!

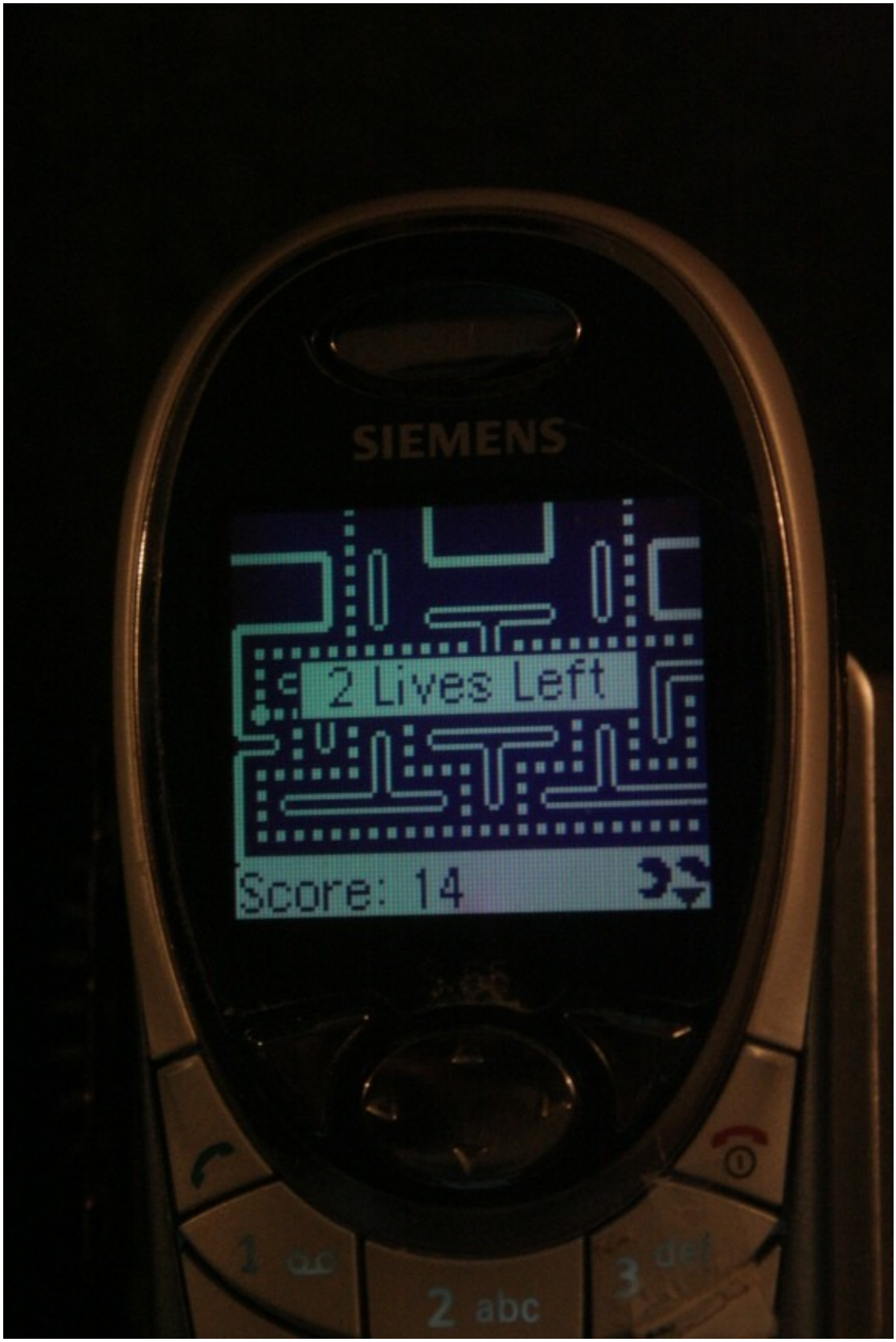

Рисунок 11 счетчик жизней, навечно застывший на отметке двух

заключение или что еще можно сделать

Вот мы и совершили свой первый взлом! Как видно, ничего сложного и сверхъестественного в этом нет. Не маги программы ломают. Это доступно каждому! Главное — сделай свой первый шаг, а уж там… поле деятельности практически безгранично. Можно заменить все текстовые строки, в том числе относящиеся к копирайту компании-создателя. Не то, чтобы это было законно, зато очень приятно и прикольно показывать друзьям мобилу с надписью "hacked by…".

Более творческие настроенные кодокопатели наверняка уже загружают спрайты в графический редактор, коверкая их в готическом хакерском стиле. Ну или меняют логотип на заставке, который так же находится в png файлах, собранных в директории image.

Конечно, мы рассмотрели простейший случай взлома _незащищенной_ программы. Некоторые игры и приложения тем или иным образом проверяют целостность байт-кода, а так же пропускаются через обфускаторы, добавляющие "мусорные" инструкции, отвлекающие внимание и затрудняющие анализ, но _базовая_ техника взлома при этом все равно остается неименной. Найти-и-обезвредить — вот наш девиз!!! Знания приходят с опытом, а опыт — со временем, проведенным за ломанием игр. Так что дерзайте!!! Да… и еще… не слишком-то распространяйтесь о своих хакерских наклонностях. А то ведь и повязать могут, хотя взлом "для себя" закон не запрещает, но это уже по любому тема _совсем_ другого разговора.

>>> врезка описание встретившихся JVM-команд

команда	опкод	операнды	описание
i2b	91h/145	1 операнд на стеке типа int	стягивает с вершины стека значение типа int, усекает до байта, снова переводит в int (с учетом знака) и забрасывает результат обратно на стек
aload_<n>	(2Ah/42)+n	1 операнд на стеке типа objectref	извлекает из objectref переменную по индексу <n> и забрасывает ее на вершину стека
dup	59h/89	1 операнд на стеке	считывает операнд со стека, клонирует его и забрасывает обратно
getfield	B4h/180	2 непосредственных индексных байта, на стеке: objectref и value	забрасывает на вершину стека заданное поле данного класса
iconst_<i>	(2h)+<i>	–	забрасывает на вершину стека константу <i>
isub	64h/100	2 операнда на стеке типа int	стягивает с вершины стека две переменных типа int, вычитает одну из другой и забрасывает результат обратно на стек
iadd	60h/96	2 операнда на стеке типа int	стягивает с вершины стека две переменных типа int, складывает их и забрасывает результат обратно на стек
putfield	B5h/181	2 непосредственных индексных байта, на стеке: objectref и value	стягивает с вершины стека переменную и записывает ее в заданное поле данного класса
nop	00h	–	нет операции

Таблица 1 краткое описание JVM-команд, встретившихся нам в подопытной программе

>>> врезка полезные ссылки

- **Macroman**:
  - java-игрушка, над которой мы будем издеваться на протяжении всей статьи: http://www.cec.ru/Files/macroman_demo.jar;
- **The JavaTM Virtual Machine Specification Second Edition**:
  - официальная спецификация на байт-код Java-машины (на английском языке): http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html;
- **avaDec 0.9b by wl**:
  - хороший Java-декомпилятор, распространяемый на бесплатной основе: http://www.wasm.ru/baixado.php?mode=tool&id=362;
- **JDecompiler (Java Decompiler) 1.0**:
  - еще один хороший и абсолютно бесплатный декомпилятор для Java-программ: http://java-decompiler.qarchive.org;
- **IDA PRO**:
  - лучший дизассемблер всех времен и народов, поддерживающий в том числе и байт-код виртуальной Java-машины, распространяемый за нехилые деньги: http://www.idapro.com;

- **JavaBite**:
  - дизассемблер, ассемблер и модификатор байт-кода JVM в одном флаконе, распространяемый бесплатно с кучей разных плагинов и прочих тулз: http://www.wasm.ru/baixado.php?mode=tool&id=284;
- **JADgen**:
  - бесплатный генератор jad-файлов, требуемых нескорыми моделями телефонов: http://softsearch.ru/programs/134-892-jadgen-download.shtml;
- **7ZIP**:
  - бесплатный архиватор, позволяющий создавать в том числе и jar-архивы: http://www.7-zip.org;
- **Winace 2.6**:
  - условно-бесплатный архиватор, в явном виде поддерживающий формат jar: http://www.tucows.com/software_detail.html?id=194310;
- **Creating a JAR File**:
  - штатная утилита для работы с jar-архивами от компании Sun, с возможностью создания цифровых подписей и прочих прибамбасов, бесплатная: http://java.sun.com/docs/books/tutorial/deployment/jar/build.html;
- **Bluejacking "У вас осталось 10 дней (J2ME)"**:
  - познавательная статья, описывающая основы взлома мобильных игр на конкретных примерах для начинающих (на русском языке): http://netgigg.livejournal.com/8631.html;