Attacking Antiviruses

Евгений Легеров, admin@gleg.net, skype: admin.gleg.net

Содержание:

1. Введение

'Антивирусная программа (антивирус) — изначально программа для обнаружения и лечения программ, заражённых компьютерным вирусом, а также для предотвращения заражения файла вирусом (например, с помощью вакцинации) ***(wikipedia.ru)***

Устанавливают на: рабочие станции, ноутбуки, сервера - веб, прокси, ftp, и тд.
Наша оценка - более 90 процентов всех компьютеров использует тот или иной антивирус.

Любой антивирус поддерживает определенное количество архивных форматов файлов (RAR, ZIP, LHA .. и тд). Библиотеки для распаковывания таких файлов разработчики антивируса часто пишут сами. Такие библиотеки могут быть подвержены уязвимостям. Из всех возможных типов уязвимостей антивирусов, в данном докладе нас будут интересовать именно распаковщики архивных форматов файлов.

В процессе работы над этим докладом было проведено поверхностное тестирование нескольких различных антивирусов. В каждом из них были обнаружены ранее неопубликованные уязвимости. Файлы, с помощью которых можно воспроизвести найденные уязвимости доступны по адресу - http://www.gleg.net/ruscrypto2008/avtests.tgz

2. Типы уязвимостей

Уязвимости антивирусного ПО которые нас интересуют:

1. Переполнения (buffer overflows)

Типовая программа на C, содержащая эту уязвимисть:
```
<C>
int readFileEntryName(unsigned char *filePtr) {
     char name[256];
```

```
     int namelen;

     namelen = READ_UINT32(filePtr); //<-- мы контролируем 'namelen'
     filePtr += 4;
     memcpy(name, filePtr, namelen); //<--- переполнение!
     return namelen;
}
</C>
```

2. Некорректная обработка целочисленных значений (integer handling vulnerabilities)

```
<C>
int readFileEntryName(unsigned char *filePtr) {
     char name[256];
     int namelen;

     namelen = READ_UINT32(filePtr);
     filePtr += 4;

     if (namelen > 256) {
          printf("Invalid name length\n");
          return -1;
     }

     memcpy(name, filePtr, namelen);
     return namelen;
}
</C>
```

3. Целочисленные переполнения (integer overflows)

```
<C>
char *readFileEntryName(unsigned char *filePtr) {
     char *name;
     int i;
     unsigned int namelen;

     namelen = READ_UINT32(filePtr);
     filePtr += 4;

     name = malloc(namelen * sizeof(unsigned int));

     for (i=0; i<namelen; i++){
          name[i] = READ_UINT32(filePtr);
          filePtr += 4;
     }
```

```
     return name;
}
</C>
```

4. Уязвимости форматной строки

```
<C>
void readFileEntryName(unsigned char *filePtr) {
     char *name;
     name = READ_STRING(filePtr);
     printf(name);
}
</C>
```

А также любые уязвимости отказа сервиса (Denial of Service vulnerabilities) – разыменование нулевого указателя, бесконечные циклы, выделение большого количества памяти и тд.

3. Методы поиска уязвимостей

Применимы не только к антивирусам, но и к любому другому ПО. Нельзя с уверенностью сказать, что какой-то способ поиска уязвимостей лучше чем другие. Использование того или иного способа, зависит от того какими ресурсами мы обладаем.

1. Source code audit (анализ исходых кодов)
1.1 Manual testing (анализ вручную)
Учитывая количество исходного кода современных антивирусов – очень трудоемкий процесс.

1.2 Automatic testing (автоматический анализ кода)
Автоматическое сканирование исходного кода для определения потенциальных уязвимостей. Инструментарий – RATS (C,C++,Perl,PHP,Python), Splint (C), Flawfinder (C,C++), CodeSpy (Java).

2. Binary auditing (анализ исполняемого кода)
2.1 Manual
2.2 Automatic

3. Fuzzing (Фаззинг)

Метод поиска уязвимостей основанный на передаче приложению испорченных (некорректных) данных и наблюдение за процессом выполнение приложения.

Отцом фаззинга считается Бартон Миллер, профессор университета Висконсин, Мэдисон. В 1989 году он совместно со своими студентами разработал примитивную программу для тестирования UNIX приложений. Программа называлась – фазз (fuzz),

http://pages.cs.wisc.edu/~bart/fuzz/fuzz.html

4. Этапы фаззинга

1. Определение цели
Предположим что нас будет интересовать парсер формата RAR какого-то конкретного антивируса.

2. Генерация испорченных данных
В нашем случае это будут испорченные файлы формата RAR. Способы генерации испорченных данных буду рассмотрены далее.

3. Выполнение приложения с использованием испорченных данных
В нашем случае это подразумевает запуск антивируса и сканирование сгенерированных (испорченных) файлов формата RAR.

4. Наблюдение за процессом выполнения приложения
Очень важный этап - нам необходимо отловить все сбои и исключения которые могут возникнуть в процессе работы программы. В нашем случае подойдет бесплатный дебаггер – OllyDbg (http://www.ollydbg.de/).

5. Анализ сбоев программы
Это не обязательный этап. В процессе анализа, который выполняется вручную, выясняется насколько опасны найденные ошибки, в частности можно ли с помощью найденной ошибки выполнить произвольный код на системе пользователя. В данном докладе этот этап затрагиваться не будет.

5. Методы генерации данных

Методы можно разделить на интелектуальные, когда у нас есть спецификация тестируемого формата файлов и неинтелектуальные когда подразумевается, что мы ничего не знаем о тестируемом формате файлов.

Как правило при использовании неинтелектуального метода генерации нужно, чтобы до начала процесса генерации у нас был неиспорченный (исходный) файл.

1. Случайные данные

Пожалуй один из самых примитивных и простых методов. Перезаписываем часть данных исходного файла случайными данными.

2. Перестановка битов

Каждый раз при генерации испорченного файла меняем состояние бита исходного файла на противоположное.

3. Блочный метод

На сегодняшний день один из самых эффективных методов генерации данных.

Одним из первых фаззеров, реализовавших данный метод был SPIKE, выпущенный в 2002 Дэвидом Айтелом. Итак, как работает блочный фаззер (на примере SPIKE):

основа фаззера – блок, это список структур содержащих информацию о размере блока и другие произвольные данные. Рассмотрим простой SPIKE скрипт:

```
s_block_size_ascii_word(“somefiledata”);
s_block_start(“somefiledata”);
s_binary(“41424344”);
s_block_end(“somefiledata”);
```

Сначала скрипт добавляет 4 нулевых байта в выходной буфер. Затем добавляются еще 4 байта (0x41424344). После завершения блока, 4 первых нулевых байта заменяются размером блока в ascii формате. На выходе получаются следующие данные: 4ABCD

После того как мы разработали линейное представление формата файла в виде SPIKE скрипта, мы можем пометить некоторые части скрипта как “переменные”, те:

```
s_block_size_ascii_word_variable(“somefiledata”); // цисловая переменная
s_block_start(“somefiledata”);
s_binary_variable(“41424344”); // строковая переменная
s_block_end(“somefiledata”);
```

SPIKE содержит наборы чисел и строк (так называемые “плохие значения”), которые могут вызвать сбои в тестируемых программах, например, числа: 0, -1,4294967295,0,0x40000000,0x7fffffff и тд., строки: “%n%n%n%n”,”\x00”,”../../../../../”, строки из символов 'A' разной длины.

В процессе генерации данных, SPIKE заменяет очередную “переменную” числом или строкой из набора плохих значений, те вместо числовой переменной SPIKE будет по очереди подставлять 0, -1, потом 4294967295 и тд. После каждой такой замены, SPIKE записывает выходные данные в очередной файл.

6. Определение цели

С помошью блочного фаззера написанного на Python, были написаны тесты

для следующих форматов файлов: RAR.

Протестированы следующие антивирусы:

F-Prot AntiVirus, http://www.f-prot.com
AVG AntiVirus, http://free.grisoft.com
Dr.Web AntiVirus, http://www.drweb.com
CA Anti-Virus, http://www.ca.com
Avira AntiVir Windows Workstations, http://www.avira.com
Kaspersky Anti—Virus, http://www.kaspersky.com

Операционная система, которая использовалась для поиска уязвимостей: Windows XP SP2.

6. Результаты

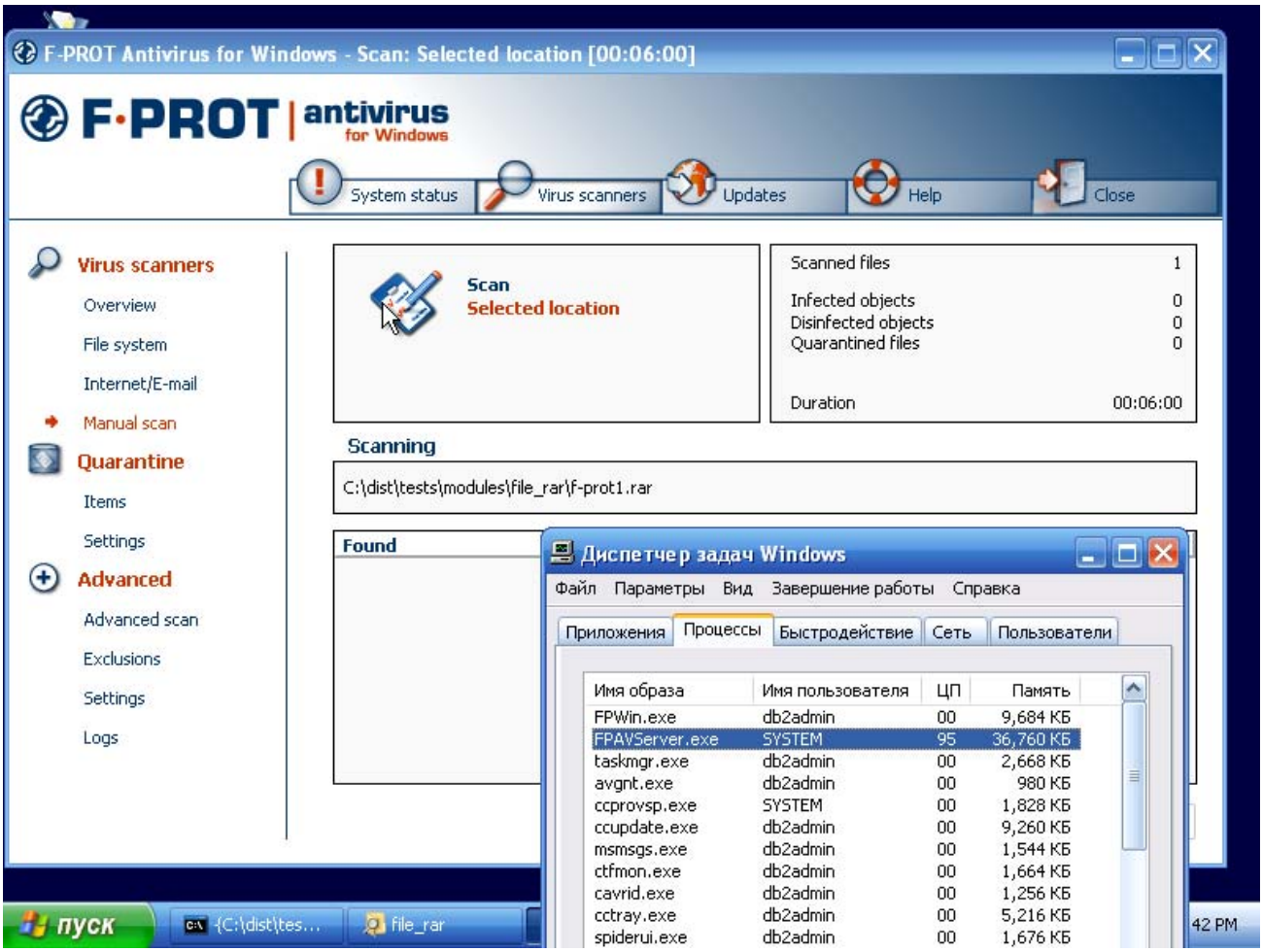
Фаззер RAR формата, количество “испорченных” файлов 2224

Результаты:

Антивирус	**Найденные уязвимости**
F-PROT 6.0	f-prot1.rar- отказ сервиса (бесконечный цикл)
AVG 7.5	avg1.rar - отказ сервиса (бесконечный цикл)
Dr.Web 4.44	drweb1.rar - перезапись памяти, drweb2.rar - отказ сервиса (бесконечный цикл)
CA Anti-Virus 2008	ca1.rar - отказ сервиса (бесконечный цикл)
Avira AntiVir 7.06.00.507	avira1.rar - отказ сервиса (бесконечный цикл)
Kaspersky Anti-Virus 7.0	kasp1.rar – отказ сервиса (бесконечный цикл)

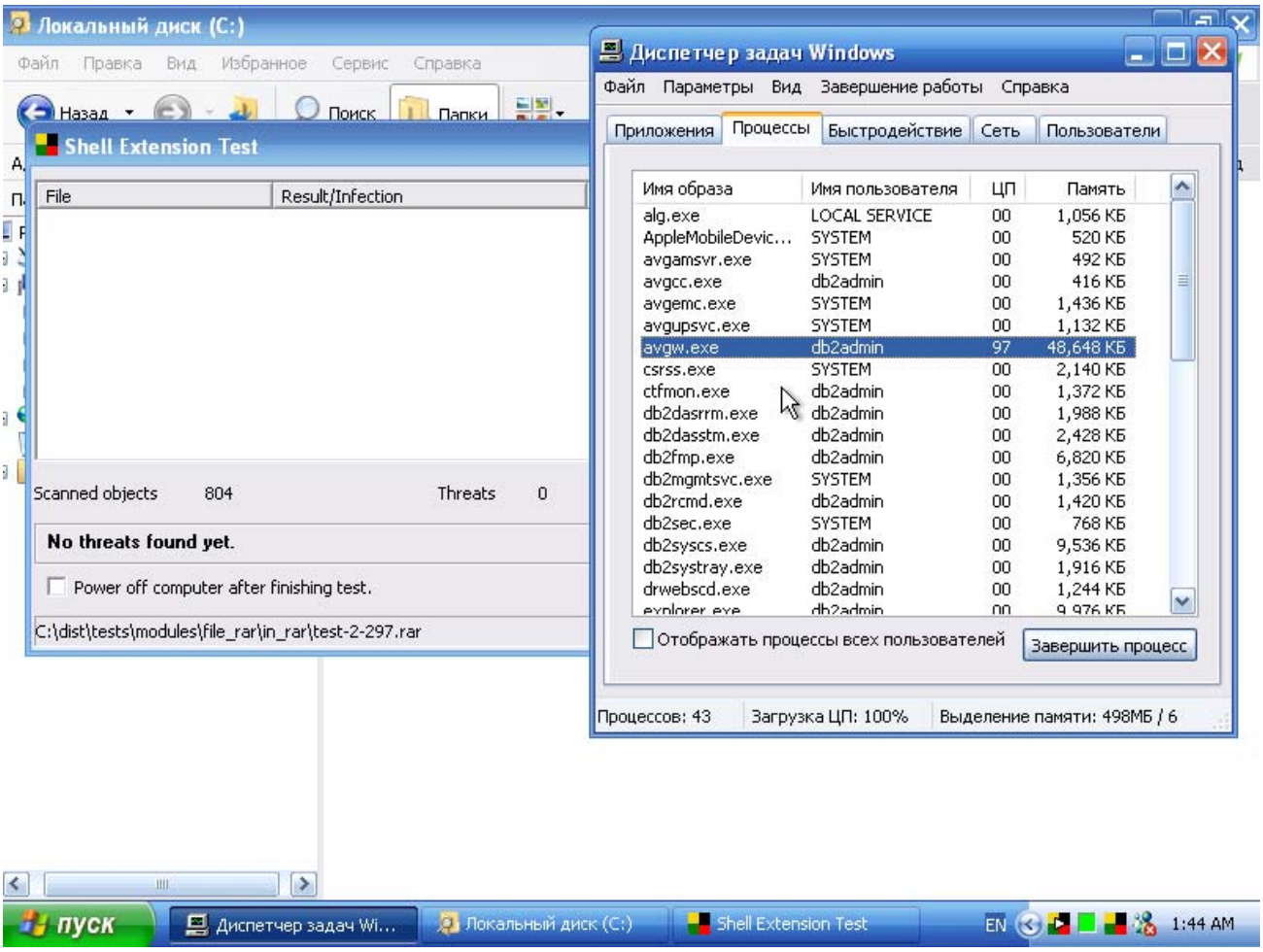
1. F-PROT Antivirus, версия 6.0 с последними обновлениями на момент 11 янв 2008

уязвимость отказа сервиса (бесконечный цикл)



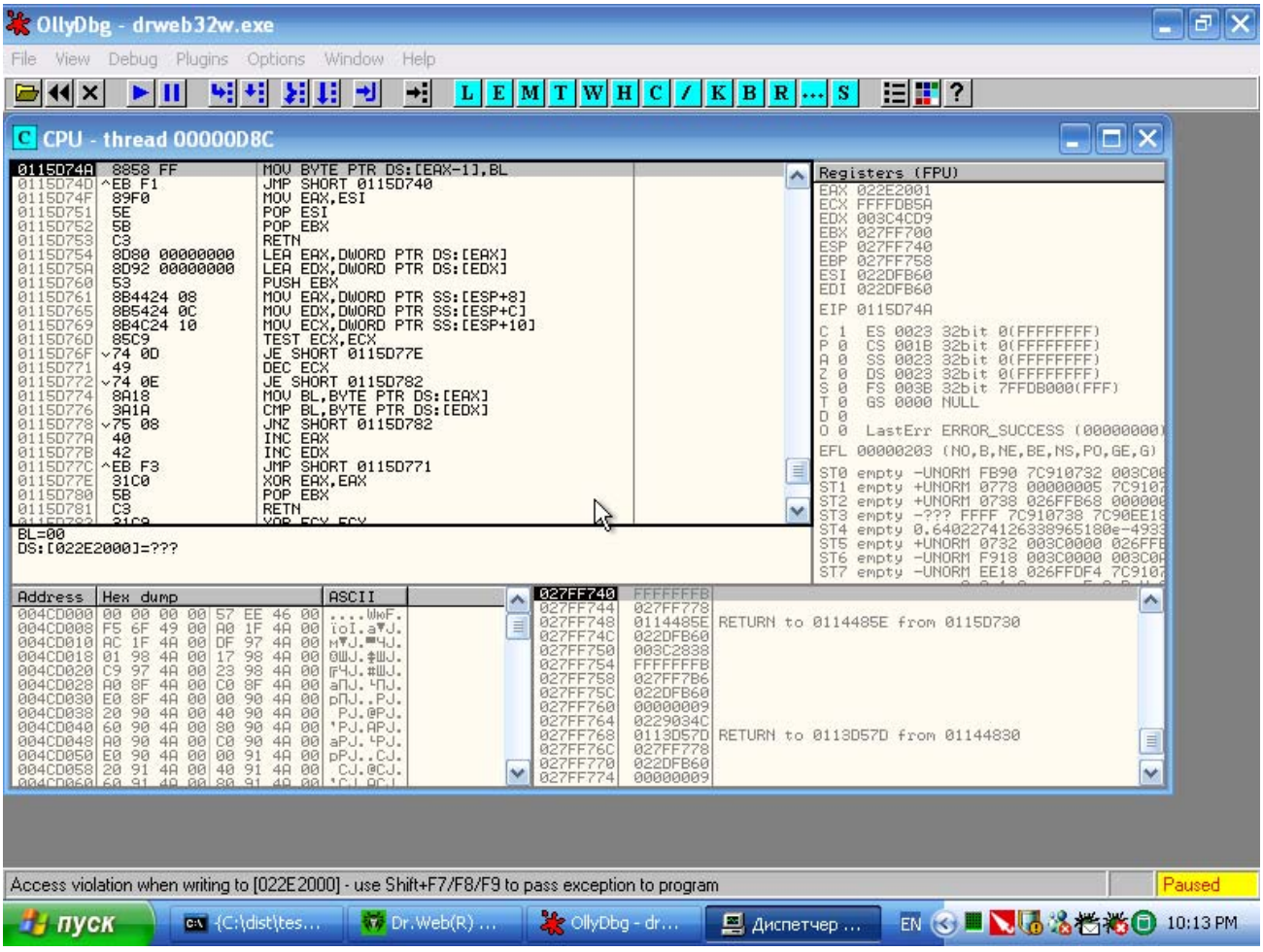
2. AVG 7.5 AntiVirus (free edition)

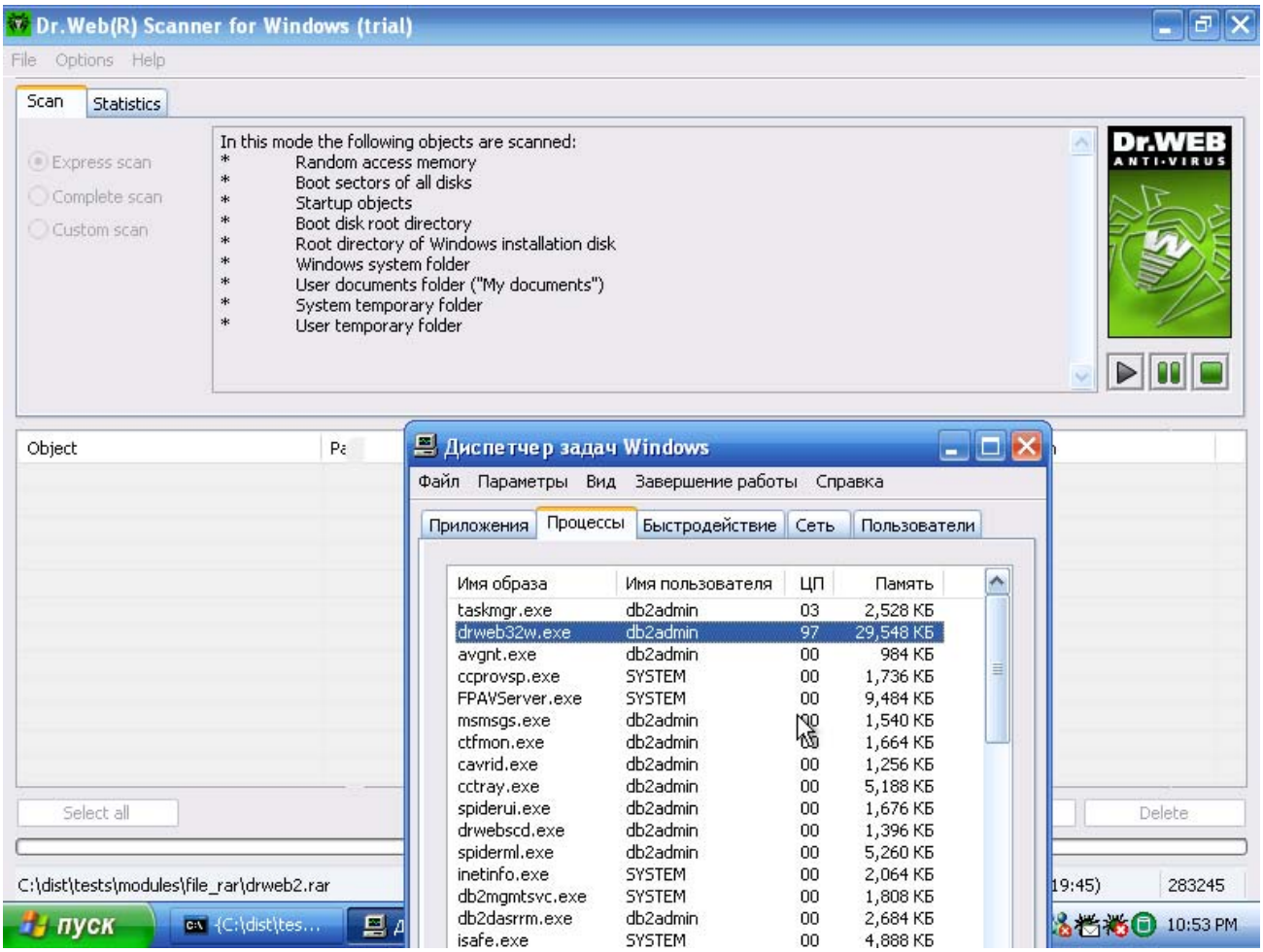
уязвимость отказа сервиса (бесконечный цикл)



3. Dr.Web Anti-Virus, версия 4.44.2, с обновлениями на момент 11 янв 2008.

1) перезапись памяти:

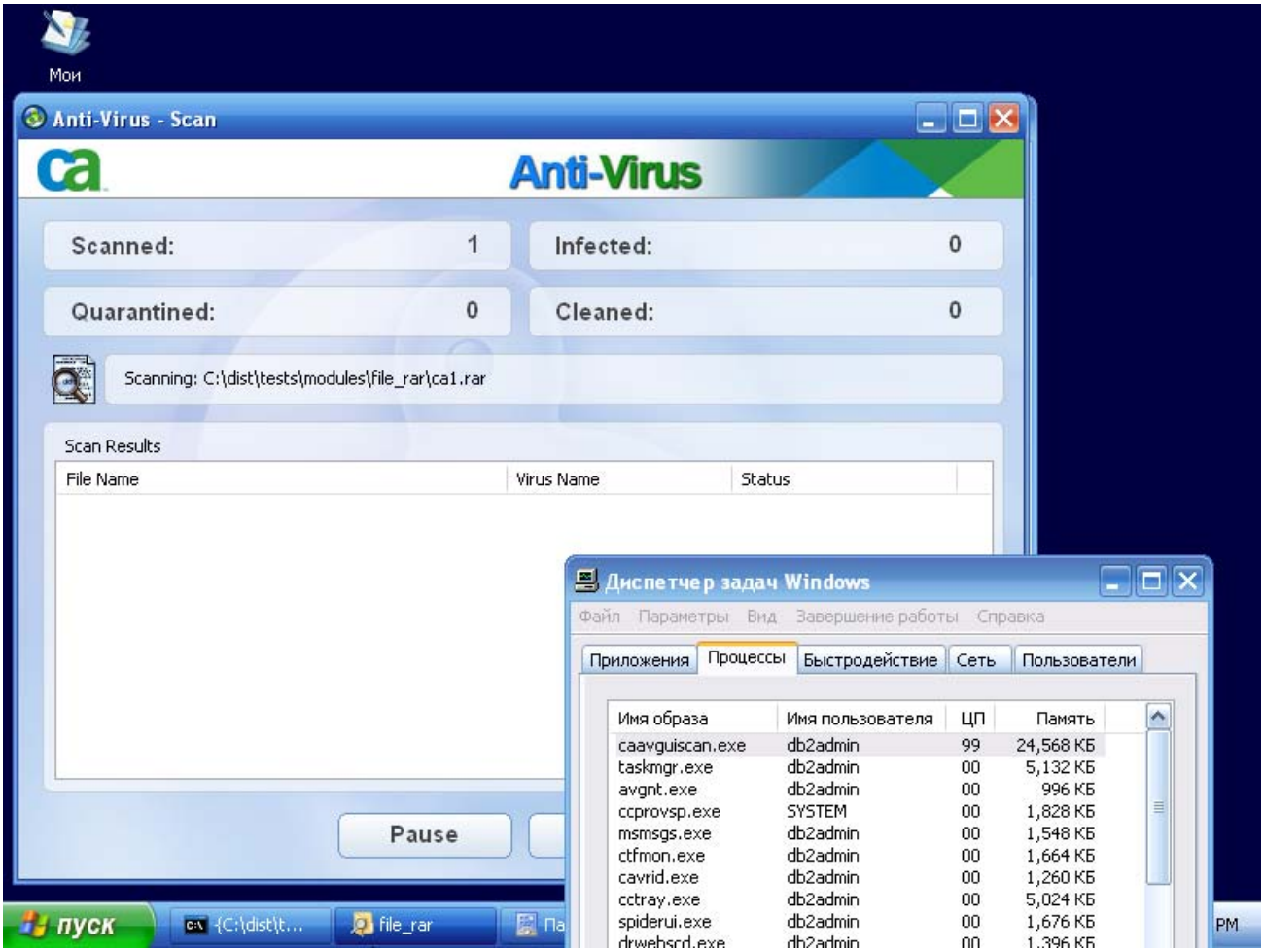




2) отказ сервиса (бесконечный цикл)

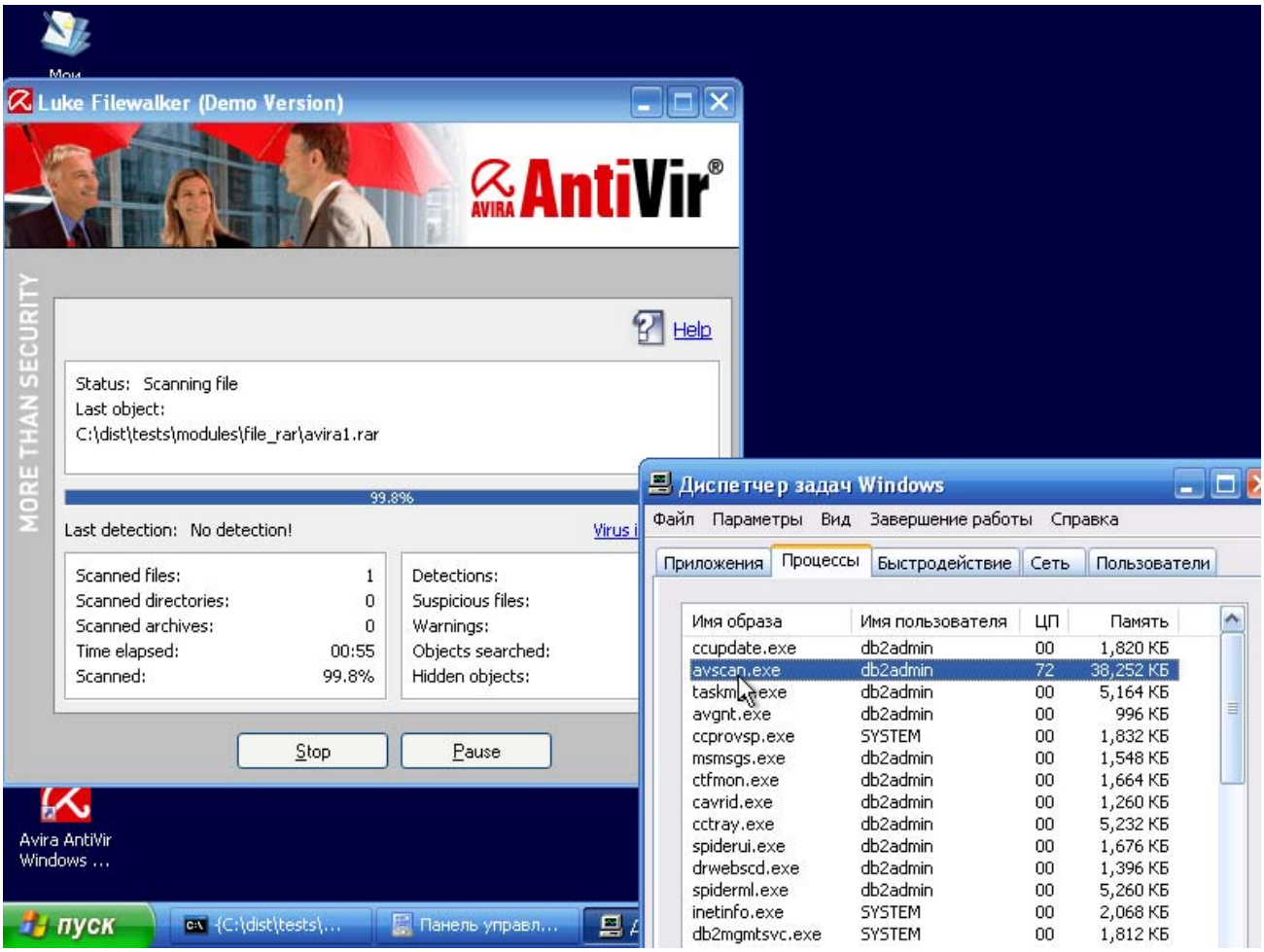
4. CA Anti-Virus 2008

отказ сервиса (бесконечный цикл)



5. Avira AntiVir Windows Workstation, version 7.06.00.507 (demo)

отказ сервиса (бесконечный цикл)



6. Kaspersky Anti-Virus 7.0.0.125, обновления на момент 11 янв 2008

отказ сервиса (бесконечный цикл)