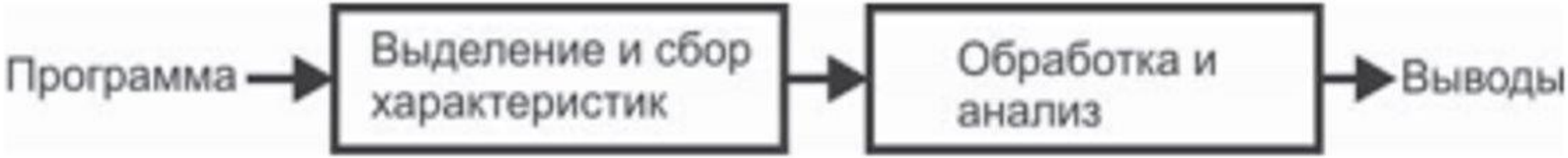
Обнаружение вирусов

Процедуры детектирования вирусов

Существуют различные типы антивирусных программ: сканеры, фаги, инспекторы, мониторы, вакцинаторы и прочие. Принцип действия большинства из них основан на детектировании (обнаружении) вирусов. Целью детектирования является разбиение всех программ, попавших в поле зрения антивируса, на два класса:

- **«здоровые»**.
- **«больные»**, то есть либо зараженные вирусом, либо представляющие собой вирус per se.

В общем случае процедуру детектирования вирусов можно разделить на следующие этапы:



Методы сбора характеристик о программе

Методы этапа, предназначенного для выделения и сбора характеристик «подозрительной» программы, целесообразно разделить на:

1)**статические** – оперирующие с двоичным образом программы на носителе информации или в памяти;

2)**динамические** – рассматривающие программу как процесс выполнения алгоритма, то есть как последовательную смену состояний.

В свою очередь, методы этапа, посвященного обработке данных и анализу характеристик, принято разделять на:

1)**формальные** - оперирующие фиксированной моделью вируса и использующие заранее определенные алгоритмы.

2)**эвристические** - пытающиеся воспроизвести процесс познания, присущий человеку.

Анализ косвенных признаков

Наиболее примитивные методы обнаружения вирусов основаны на изучении **косвенных признаков** («слабых сигнатур»), характеризующих зараженность. Как правило, проверка наличия или отсутствия этих признаков возможна либо «на глазок», либо с применением штатных утилит операционной системы.

Примерами подобных признаков являются отличительные метки, проставляемые внутри зараженных программ самими вирусами: строчка **«MsDos»** в конце файла (вирусы семейства Jerusalem); значение **«62 секунды»** в атрибуте времени последнего доступа к файлу (вирусы семейства Vienna); байт со значением **55h** перед PE заголовком (вирус Win9X.CIH) и прочие. Другими косвенными признаками могут служить нетрадиционная структура программы (например, наличие нескольких кодовых сегментов в PE-модуле), бит разрешения записи в кодовый сегмент, «свежая» дата создания у заведомо «несвежего» файла и т. п.

Анализ косвенных признаков

Кроме того, большое количество косвенных признаков можно обнаружить, сканируя код программы на характерные фрагменты. Например:

1)байт **E9h** в начале COM-файлов соответствует команде «JMP» и может свидетельствовать о зараженности ее «стандартным» методом;

2)цепочка байтов **E80000h** или **E8000000h** в начале программы может означать попытку вычислить «дельта-смещение»;

3)команда **«MOV AH, 4Ch»** (константа 4CB4h) помогает вирусу искать файлы в каталоге, а **«CMP AX, 'MZ'»** (цепочка байтов 3Dh 4D4 5Ah) – различать COM- и EXE-программы;

4)константа **0EDB88320h** часто используется при расчете CRC-32, а константы типа **553B5C78h** или **0AE17EBEFh** встречаются в вирусах, ищущих в таблицах экспорта адреса функций CreateFileA и FindFirstFileA по контрольным суммам их имен;

5)и т. п

Анализ косвенных признаков

Разумеется, однозначным признаком зараженности «слабые сигнатуры» служить никоим образом не могут и не должны.

В качестве примера, подтверждающего этот тезис, можно вспомнить историю, произошедшую в конце 1980-х годов. Один из популярных «импортных» антивирусов – программа TNTVIRUS – «вакцинировал» все программы на диске, записывая в конец строчку «MsDos», чтобы вирус Jerusalem считал их уже зараженными и не трогал.

Другой же «антивирус», отечественный ANTIKOT, «лечил» файлы, распознавая наличие в них вируса исключительно по строчке «MsDos». Нетрудно догадаться, к чему приводило массовое «лечение» ранее вакцинированных программ.

Простые сигнатуры

Наиболее популярным методом детектирования вредоносных программ является проверка сигнатур. Под **сигнатурой** понимается: фрагмент (или набор фрагментов), который всегда встречается в конкретном вирусе и никогда – в иных программах (в том числе и в других вирусах).

Сигнатуры используются для детектирования вредоносных программ, сохраняющих свой код постоянным от копии к копии. Кроме того, сигнатурный поиск можно применить и для поиска некоторых разновидностей полиморфных вирусов. Единственная разновидность вирусов, к которой совсем не применимо сигнатурное детектирование, – это **«метаморфы»**, то есть вирусы, использующие идеи замены и пермутации (перемешивания) команд для всего своего тела.

Простые сигнатуры

В идеале сигнатура должна включать в себя всю постоянную часть вируса, но на практике – для минимизации требуемой памяти и ускорения поиска в файле – используются сравнительно короткие цепочки байтов, состоящие из нескольких отрезков. В общем случае сигнатура S может быть представлена в виде множества троек: **S = {Ci , Pi , Ti }**, где **i** – номер отрезка сигнатуры (если она состоит из нескольких частей); **Ci** – значение отрезка; **Pi** – позиция отрезка; **Ti** – шаг трассировки, на котором необходимо контролировать отрезок. То есть три различных компонента сигнатуры отвечают на вопросы «что», «где» и «когда». Разумеется, в составе сигнатуры могут встречаться и другие компоненты вспомогательного назначения. В частности, это может быть признак: откуда отсчитывать смещение фрагмента – от начала файла, конца файла или точки входа в программу. Впрочем, этот признак тоже отвечает на вопрос «где».

Простые сигнатуры

Дабы проиллюстрировать сказанное, составим сигнатуру для вируса **Seat.2389**, упомянутого ранее в разделе про полиморфные MS-DOS-вирусы. (Там же можно найти и листинги двух «мутаций»). Напомним, это был «олигоморфный» вирус, шифровавший тело с переменным ключом и видоизменявший фрагмент расшифровки от копии к копии. Во фрагменте расшифровки переменными были только имена регистров и ключ шифрования-расшифровывания, общая же структура цикла оставалась постоянной.

```
?? 73 00
?? 41 09
8B ??
```

```
Mov регистр1, 73h
mov регистр2, 941h
mov cx, регистр 2
```

Простые сигнатуры

```
2E 80 ?? 9E FF ????          операция cs:[регистр1+9Eh], ключ
??                           inc регистр1
E2 F7                        loop $-0Ah
; Эта часть появляется только после расшифровки
1E                           push ds
0E                           push cs
F8                           clc
B8 FB DF                     mov      ax, DFFBh
CD 21                        int      21h
...
```

Простые сигнатуры

Это обстоятельство дает возможность составить сигнатуру, состоящую из двух отрезков длиной, например, по 6 байтов каждый. **S(SEAT.2389) = < ?? 73h 00 ?? 41h 09h, 0, 0 > , < 1Eh 0Eh F8h B8h FBh DFh, 11h, 15 >**.

Первый отрезок всегда присутствует в файле и памяти по нулевому смещению от начала вируса – то есть от той точки, куда вирус передает управление командой «JMP». В этом фрагменте встречаются как переменные, так и постоянные участки, поэтому для описания последовательности байтов используются символы-джокеры «?». Такие «прерывистые» сигнатуры иногда называют **масками**.

Простые сигнатуры

Второй отрезок изначально зашифрован с переменным ключом. Но поскольку инициализация цикла выполняется тремя командами и сам цикл тоже состоит из трех команд, повторяющихся **941h = = 2369 раз**, то после выполнения **3 + 3 * 2369 = 7110** вирусных команд в памяти окажется полностью расшифрованный образ вируса, в котором (начиная со смещения 11h) можно обнаружить второй отрезок сигнатуры. Впрочем, для расшифровки первых 6 байтов основного тела достаточно выполнить не 7110, а всего лишь **3 + 3 * 4 = 15** команд. Обычно это делается путем моделирования работы программы в эмуляторе, являющемся частью антивируса

Простые сигнатуры

Лет 15–20 назад, когда вирусов было мало, вирусолог мог позволить себе тщательно изучить код очередного представителя «электронной фауны» и выбрать оптимальный вариант сигнатуры. Пример подобного подхода можно найти в главе, посвященной загрузочным вирусам: мы использовали в качестве сигнатуры вируса Stoned. AntiEXE «узловой» фрагмент кода, что позволило антивирусу различать «здоровые», «больные» и «вылеченные» секторы и участки памяти. В современных условиях такой подход труднопригоден, так как, например, дежурной смене «дятлов» (как в Лаборатории Касперского величают вирусных аналитиков) приходится ежедневно иметь дело с сотнями и тысячами новых вирусов и троянских программ! Существует потребность в алгоритмах, позволяющих автоматизировать процесс выбора «хороших» сигнатур.

Простые сигнатуры

И такие алгоритмы есть. Например, можно вести большую базу данных со всевозможными сигнатурами всевозможных вирусов, а также наиболее типичных прикладных и системных программ. Тогда в качестве «хорошей» сигнатуры вируса можно выбирать любую цепочку его байтов, не встретившуюся в этой базе. Но, с одной стороны, такая база будет очень велика. С другой – нет никакой гарантии, что эта сигнатура не встретится в какой-нибудь прикладной программе, отсутствующей в базе. Как, например, произошло в 2011 г. с антивирусом Avira, который случайно внес в свои антивирусные базы сигнатуру, характерную для программного кода антивируса Avira.

Простые сигнатуры

Более корректный алгоритм выбора «хороших» сигнатур в 1994 году обнародовали сотрудники IBM Дж. Кепхарт и В. Арнольд. Предположим, что стоит задача выбора сигнатуры **B = B1B2…BS** длиной S байтов для вредоносной программы с постоянным фрагментом длиной Q. Нетрудно сообразить, что всего возможно **QS = Q – S + 1** различных вариантов сигнатуры.

Какой из них – лучший? Очевидно, тот, который не встречается ни в других вирусах, ни в «нормальных» программах, и вероятность появления его в пока еще не написанных программах тоже невелика.

Простые сигнатуры

Для того чтобы оценить эту вероятность, сигнатуру рассматривают как совокупность из **S – n + 1** всевозможных непрерывных цепочек длиной по n байтов – так называемых n-грамм. Например, в сигнатуре **B = B1B2B3B4B5** можно выделить три 3-граммы: **B1B2B3**, **B2B3B4** и **B3B4B5**. Очевидно, что сигнатура, состоящая из «типичных» n-грамм, хуже сигнатуры, состоящей из «редких» n-грамм.

Ее использование в антивирусе может приводить к ложным срабатываниям (false positives). Например, 2-грамма «MZ» характерна не только для сетевых червей, но и для любых EXE-программ, и поэтому использовать ее в составе сигнатуры нежелательно.

Простые сигнатуры

Чтобы оценить «типичность» n-грамм сигнатуры, создают «тестовый корпус» – огромный файл длиной T байтов, составленный изо всех известных вирусов и большого числа типичных «нормальных» программ: компонентов операционных систем, системных библиотек, утилит, приложений и т. п. Методика предусматривает поиск каждой из n-грамм сигнатуры в «тестовом корпусе» и подсчет количества f(B1B2…Bn) находок. Приблизительно оценить вероятность появления сигнатуры B в файле, не входящем в состав «корпуса», можно по формуле:

$$p(B_1B_2...B_S) \approx \frac{f(B_1B_2...B_n)f(B_2B_3...B_{n+1})...f(B_{S-n+1}B_{S-n+2}...B_S)}{f(B_2B_3...B_n)f(B_3B_4...B_{n+1})....f(B_{S-n+1}B_{S-n+2}...B_{S-1})(S-n+1)}.$$

Простые сигнатуры

В качестве «хороших» отбираются сигнатуры, у которых эта вероятность минимальна. Вместе с ростом n не только становится более точной оценка вероятности **p(B1B2…BS)**, но и возрастают затраты вычислительных ресурсов.

В середине 1990-х годов специалисты из IBM экспериментировали с 3-граммами и сигнатурами длиной от 12 до 24 байтов. В материалах фирмы Symantec, датированных 2009 г., сообщается о 5-граммах, 48-байтовых сигнатурах и упрощенных вариантах формулы расчета вероятности.

Простые сигнатуры

Еще одна проблема связана с поиском в файле сигнатуры для вирусов, не имеющих постоянной точки входа. Существуют алгоритмы, позволяющие несколько ускорить процедуру глобального поиска. Проиллюстрируем идею одного из таких алгоритмов – **Бойера-Мура-Хорспула** – на примере поиска образца «ЛОМ» в строке «ГОЛОВОЛОМКА».

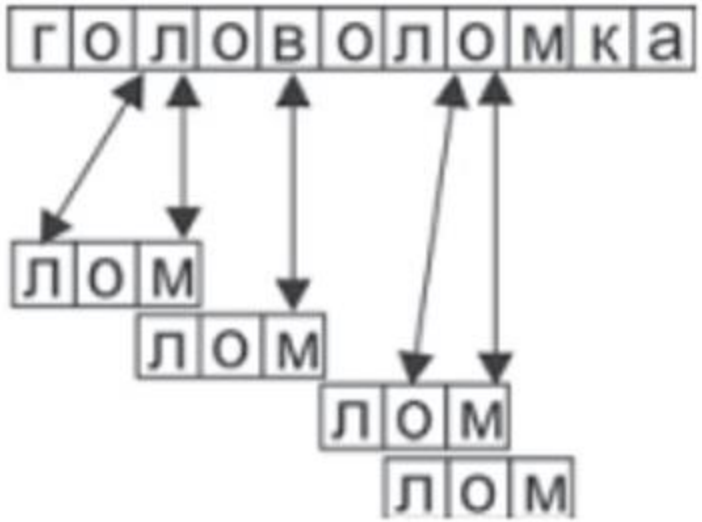
Она (идея) заключается в том, чтобы выполнять сравнение данных с образцом, начиная с его (образца) последней буквы – справа налево. Если расхождений не обнаружено, значит, образец в наборе найден. В противном случае (то есть если в какой-нибудь букве расхождение все-таки возникло) возможны две принципиально различные ситуации.

Простые сигнатуры

Первая ситуация возникает, когда в образце уже имеются буквы, равные той, которая в этот момент находится напротив «хвоста» образца (например, несколько букв «Л», как на первом шаге поиска, или несколько «О», как на третьем шаге). В этом случае необходимо сдвигать образец таким образом, чтобы напротив этой буквы строки оказалась «предпоследняя» в образце такая же буква (то есть напротив 3-й буквы строки – первая буква образца).

Вторая ситуация соответствует случаю, когда такая буква не встречается в образце. Следовательно, можно смело сдвигать образец на полную его длину – на 3 элемента (см. ситуацию на втором шаге сравнения).

Простые сигнатуры



Работа алгоритма
Бойера-Мура-Хорспула

Простые сигнатуры

Для повышения эффективности реализации этого алгоритма перед началом поиска для каждого вида элементов, которые могут встретиться в наборе данных, создается индекс – число позиций, на которое нужно сдвигать образец. Все индексы помещаются в общую таблицу и используются в процессе поиска. Для рассматриваемого примера в таблице для большинства букв должно находиться значение 3, для буквы «О» – 1, а для буквы «Л» – 2.

```
int bmh_search( char *s, int n, char *p, int m ) {
 int t[256], i, j, k;
 /* Подготовка таблицы */
 for (i=0;i<256;i++) t[i]=m; for (i=m-2;i>=0;i--) t[p[i]]=m-i-1;
 /* Собственно поиск */
 i=m-1;
 while (i<n) {
   k=i; j=m-1;
   while (j>=0) {
```

Простые сигнатуры

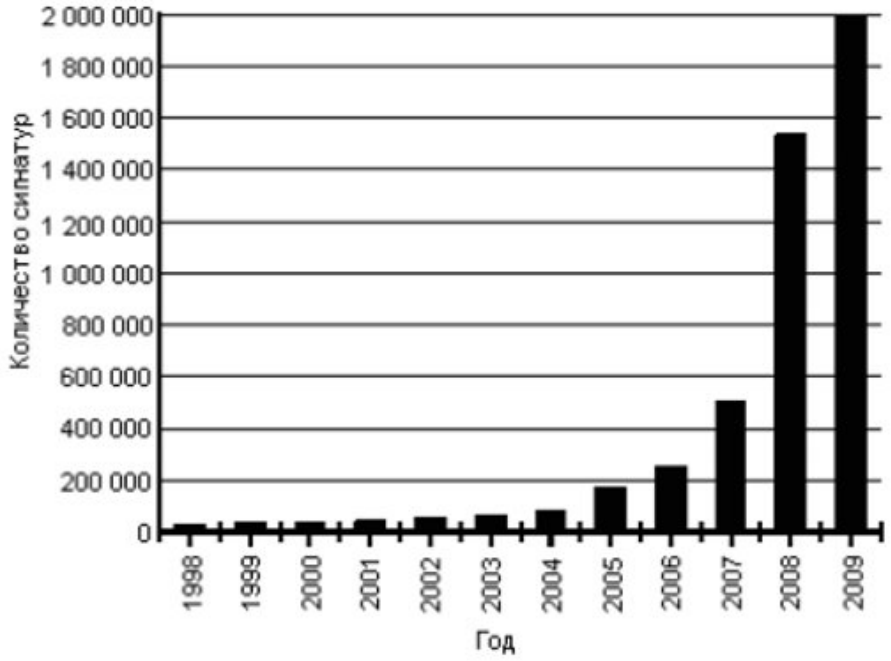
```
      if (s[i]!=p[j]) break;
      i--; j--;
     }
    if (j<0) return i+1; // Позиция совпадения
    i=k+t[s[k]];
   }
 return -1; // Ничего не найдено
}
```

Простые сигнатуры

Алгоритм **Бойера-Мура-Хорспула** работает тем эффективнее, чем более длинные образцы предлагаются ему для поиска. Альтернативой этому алгоритму может служить алгоритм **Кнута-Морриса-Пратта**. Если же речь идет о поиске целого семейства частично пересекающихся сигнатур, то лучше использовать алгоритм **Ахо-Корасик**. Основной недостаток сигнатурного детектирования – неспособность обнаружения «новых», еще не изученных вирусов.

Кроме того, неприятной особенностью сигнатурных антивирусов является большой объем справочных данных. Согласно материалам «Лаборатории Касперского», количество записей в антивирусных базах растет по экспоненте.

Простые сигнатуры



Рост объемов вирусных баз
в Антивирусе Касперского

Контрольные суммы

Существенно уменьшить объемы требуемой памяти позволяет отказ от сигнатур в пользу контрольных сумм. Контрольная сумма – результат применения к произвольному набору данных некой хешфункции, рассчитывающей короткий «дайджест» постоянной длины (например, всего 4 байта). В базе данных антивируса можно хранить не сами сигнатуры, а их короткие контрольные суммы.

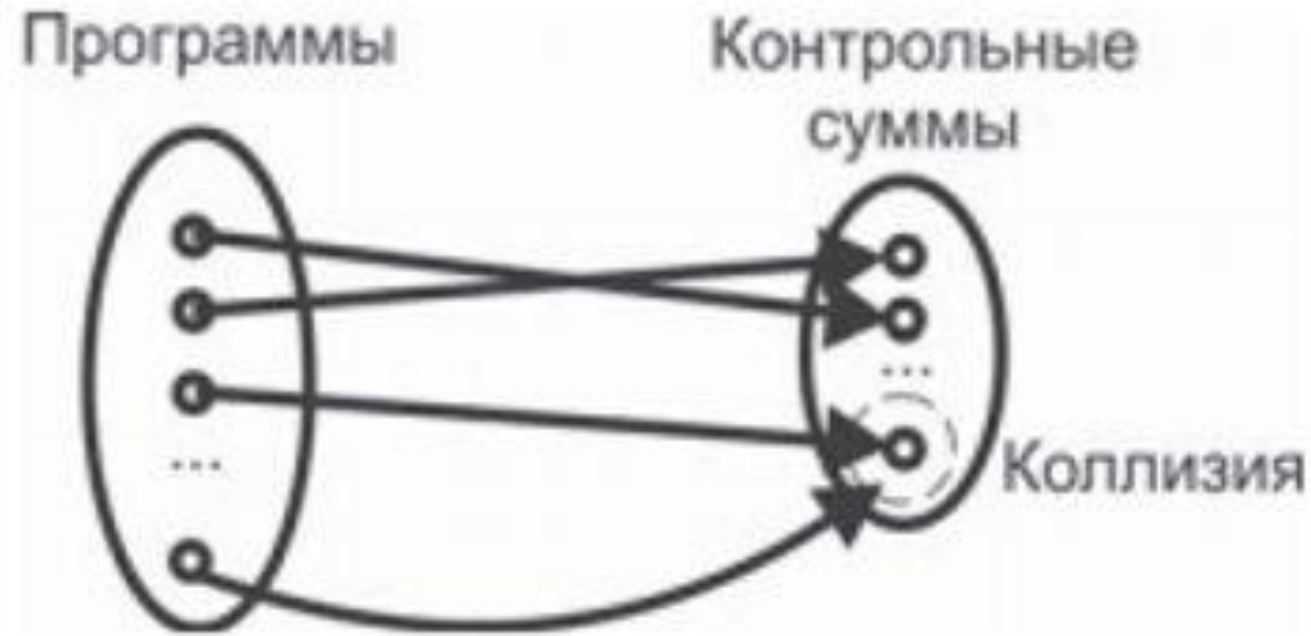
Такие же суммы рассчитываются «на лету» по содержимому тестируемых файлов. Несовпадение хранимого образца с результатом расчета означает отсутствие соответствующего вируса. А совпадение – лишь очень высокую (**но не единичную**) вероятность заражения. Причина кроется в сути контрольных сумм: они, как и любые другие хеш-функции, представляют собой отображение большого множества «объектов»-прообразов на малое множество «дайджестов»-образов.

Контрольные суммы

Соответственно, всегда возможна коллизия: несколько различных «объектов» могут иметь одинаковые «дайджесты», в частности «плохие» контрольные суммы могут оказаться не только у зараженных, но и у вполне «здоровых» файлов. Невысокая вероятность коллизий и трудность их целенаправленного генерирования – суть критерии качества того или иного метода расчета контрольных сумм. С этой точки зрения хороши так называемые «криптографические» хеш-функции: **MD5, SHA-1, RIPEMD, ГОСТ 34.11-94** и др.

К сожалению, эти алгоритмы довольно сложны с вычислительной точки зрения и не способны обеспечить высокую скорость расчета контрольных сумм. Поэтому на практике в антивирусах нашли применение несколько менее стойкие к коллизиям, зато гораздо быстрее вычислимые «технические» хеш-функции.

Контрольные суммы



Контрольные суммы

Чаще всего используются циклические избыточные коды (CRC – cyclic redundancy code). Метод использования основан на представлении блока данных в виде непрерывного полинома с битовыми коэффициентами. В качестве контрольного кода используется остаток от деления этого полинома на более короткий «порождающий» полином, имеющий длину N битов. Техника деления такова: 1) в конец блока данных добавляется N–1 нулевых битов; 2) вместо арифметического деления используется операция «сложение по модулю 2»; 3) сложение с «левыми» нулями промежуточных остатков не производится. Существуют как программные реализации, основанные непосредственно на делении «уголком» (пример на языке Ассемблера можно найти в главе, посвященной Win32-вирусам), так и оптимизированные по скорости – благодаря сдвигу сразу на 8 битов и использованию таблицы корректирующих коэффициентов. Вот пример реализации для «быстрого» вычисления CRC-32

Контрольные суммы

В качестве делителя выбираются не любые цепочки битов, а те из них, которые соответствуют «неприводимым» (то есть не раскладываемым на сомножители) полиномам. Например, для CRC-16 используются полиномы **0x1021** и **0x8005**, а во многих стандартах связи для CRC-32 зафиксированы **0x04C11DB7** и его «зеркальное отражение» **0xEDB88320**. Сравнительно недавние исследования показали высокое качество полиномов **0x1EDC6F41**, **0x741B8CD7** и **0x814141AB1** . Кроме того, иногда применяются CRC-48 и CRC-64. Нередко в стандартах на методы вычисления CRC упоминаются дополнительные операции, такие как, например, инвертирование битов результата. Распознающих свойств метода это не улучшает, но лишь упрощает аппаратную реализацию алгоритма в связном оборудовании.

Контрольные суммы

Существуют как программные реализации, основанные непосредственно на делении «уголком», так и оптимизированные по скорости – благодаря сдвигу сразу на 8 битов и использованию таблицы корректирующих коэффициентов. Вот пример реализации для «быстрого» вычисления CRC-32:

```
/* Расчет таблицы корректирующих коэффициентов */
make_crctable( void ) {
 int i, j;  DOUBLE r;
 for (i = 0; i <= 255; i++) {
  r = i;
  for (j = 8; j > 0; j--) if (r & 1)  r = (r >> 1) ^ 0xEDB88320;  else  r >>= 1;
  crctable[i] = r;
 }
}

/* Расчет CRC-32 для буфера buf длиной len */
DOUBLE crc32(unsigned char *buf, DOUBLE len) {
  DOUBLE crc = 0xFFFFFFFF;
  while (len--)  crc = (crc >> 8) ^ crctable[(crc ^ *buf++) & 0xFF];
  return crc ^ 0xFFFFFFFF; // Для облегчения аппаратной реализации
}
```

Вопросы эффективности

Считается, что одним из важнейших критериев качества современного антивируса является эффективность использования вычислительных ресурсов. Речь идет о быстродействии, требованиям к дисковой и оперативной памяти и т. п. На момент написания этих строк известны несколько миллионов разновидностей вредоносных программ, следовательно, антивирус в общем случае вынужден выполнять такое же количество детектирующих операций (например, сравнений сигнатур) по отношению к каждому файлу. Неудивительно, что антивирусный монитор способен вызывать многосекундные задержки при контроле доступа к файлам, а антивирусный сканер – затрачивать десятки часов на проверку диска. Дошло до того, что типичный пользователь при покупке выбирает не тот антивирус, который «знает больше» или «лечит лучше», а тот, который «работает быстрее». Надо сразу оговориться, что в современных условиях удовлетворительного решения проблемы, по-видимому, не существует. Современная антивирусная реализация – это клубок компромиссов: одни производители жертвуют надежностью детектирования, другие – быстродействием, третьи – количеством распознаваемых вредоносных программ, четвертые – возможностью лечения и т. п.

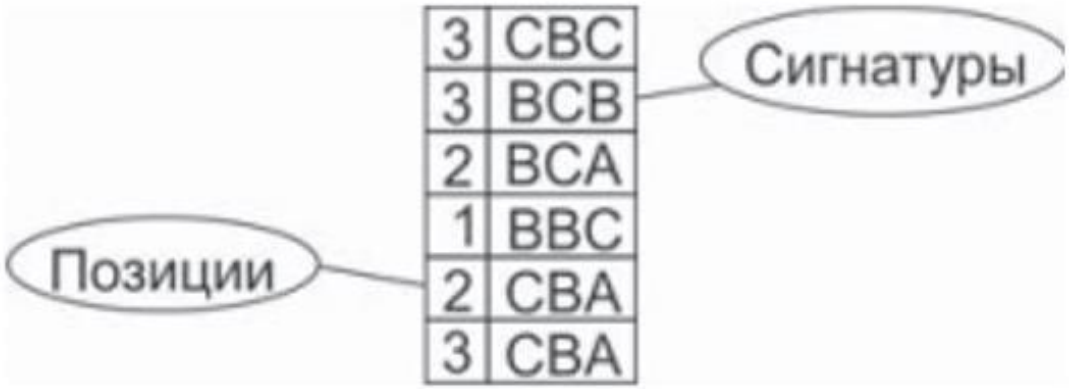
Выбор файловых позиций

Дальнейшие рассуждения проиллюстрируем на примере маленьких таблиц, моделирующих N = 6 различных вирусов. «Коды» этих вирусов формируются всего из трех различных байтов (символов): ‘A’, ‘B’ и ‘C’ . Записи, внесенные в антивирусную базу, выделены

«Табличные» модели отдельных вирусов

Выбор файловых позиций

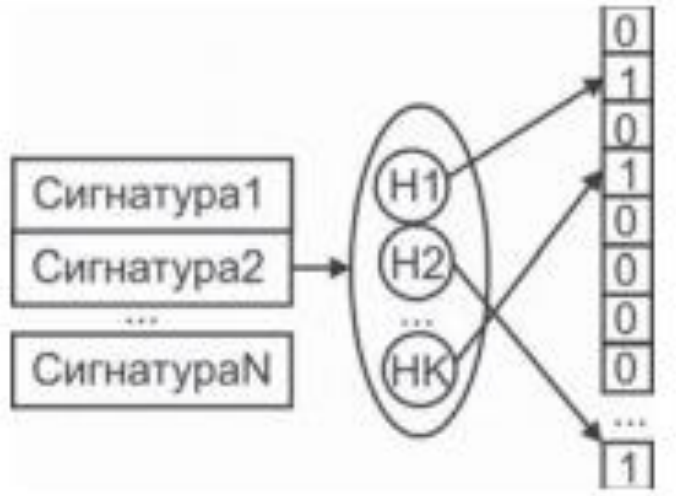
Разумеется, проще всего сформировать «антивирусную» базу, разместив в ней записи в том порядке, в каком изначально были перечислены вирусы.



«Табличная» модель антивирусной базы

Фильтр Блума

Это метод хеширования, позволяющий сразу отбрасывать записи, отсутствующие в вирусной базе данных. Идея заключается в том, чтобы вместе с базой данных, содержащей N записей о вирусах, хранить «битовую карту» – массив из M первоначально обнуленных битов. При добавлении к базе новой сигнатуры для нее рассчитываются K различных хеш-функций, и в «карте» устанавливаются в «1» K битов1 . Индексами (адресами в карте) для этих битов являются значения рассчитанных хешей. В итоге после завершения создания базы «карта» оказывается заполнена перемешанными значениями «0» и «1».



Метод половинного деления

Существенно ускорить поиск позволяет «дихотомия» («половинное деление») лексикографически упорядоченной таблицы с пронумерованными по порядку записями (. Если известно, что аргумент поиска меньше среднего ключа в таблице, то можно не проверять элементы в диапазоне от этого среднего и до конца таблицы. Исходная таблица окажется разделена на две примерно равные части, в одной из которых заведомо находится искомая запись. Эту «половину» можно вновь разделить пополам по средней записи и т. д., пока либо искомая запись не будет обнаружена, либо не будет доказано ее отсутствие.

Разбиение на страницы

Это концепция, предусматривающая разделение таблицы по какому-либо признаку на независимые части («страницы») и поиск только в некоторых из них. Например, наиболее естественно разделить таблицу вирусной базы данных на отдельные страницы в зависимости от значения поля «позиция» или комбинации **«позиция+начало_сигнатуры»**. Внутри страницы записи упорядочены лексикографически по значению поля «сигнатура». Такой подход позволит считывать целиком всю страницу в память и искать в ней нужную запись методом половинного деления, избежав многочисленных обращений к носителю. Кроме того, на компьютере с несколькими процессорами (или ядрами в процессоре) можно будет считывать в память сразу несколько страниц и выполнять поиск параллельно. Для ускорения доступа к базам данных, разбитым на страницы, обычно используют «индексные таблицы», содержащие номера страниц и их адреса на носителе

Разбиение на страницы