МАТЕМАТИЧЕСКИЕ МОДЕЛИ
В ИНФОРМАТИКЕ
(Вычислительные системы)
2002 год Выпуск 169

УДК 519.722

МОДЕЛИРОВАНИЕ СИСТЕМЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ СИСТЕМ

В.А.Черников

Проблема распознавания компьютерного вируса алгоритмически неразрешима. Предлагается применение секретной модели Шеннона для анализа взаимодействия вируса и антивируса. Получен метод анализа и вероятностные критерии оценки надежности работы антивируса. Указан алгоритм построения и схема эффективной стратегии поиска вируса.

§ 1.Компьютерные вирусы и алгоритмическая неразрешимость

1. Определение. Определим *компьютерный вирус* как программу, модифицирующую другие программы путем добавления своей возможно измененной копии [1,2].

Напишем простейшую программу — вирус V:

```
program virus{
1234567;
  subprogram infect-executable{
    loop:file = get-random-executable-file;
    if first-line-of-file = 1234567 then goto loop;
    prepend virus to file;
  }
```

```
    subprogram do-damage{
      whatever damage is to be done
    }
    subprogram trigger-pulled{
      return true if some condition holds
    }
    main{
      infect-executable;
      if trigger-pulled then do-damage;
      goto next;
    }
  next:
  }
```

2. Полиморфность. Главной проблемой, возникающей при распознавании зараженной программы, является возможность существования практически неограниченного числа различных версий одного и того же вируса[1]. Покажем это на примере. Добавим в тело вируса V произвольное число утверждений; получим новый вирус EV:

```
program evolutionary-virus{
  subprogram print-random-statement{
    print random-variable-name, " = ",
    random-variable-name;
    loop:if random-bit = 0 then{
      print random-operator, random-variable-name;
      goto loop;
      }
    print semicolon;
  }
  subprogram copy-virus-with-random-insertions{
    loop: copy evolutionary-virus to virus till
    semicolon-found;
    if random-bit = 1 then print-random-statement;
```

[1]Оригинальное использование данного свойства для создания "трудновзламываемых" программ показано в [3].

```
    if not end-of-input-file goto loop;
  }
  main{
    copy-virus-with-random-insertions;
    infect-executable;
    if trigger-pulled do-damage;
    goto next;
  }
next:
}
```

Таким образом, при каждом срабатывании данного кода, генерируется новая модификация исходного вируса. Следствием этого является то, что точное определение какая именно программа-вирус использовалась при заражении невозможно.

3. Алгоритмическая неразрешимость. Действительно, предположим, что существует некая процедура D, которая позволяет отличить некую программу P от вируса V. Таким образом, процедура D дает ответ на вопрос и о том, являются ли две программы $P1$ и $P2$ разновидностями одной и той же программы P или нет. Тогда рассмотрим программу-вирус UEV следующего вида:

```
program undecidable-evolutionary-virus{
  subprogram copy-with-undecidable-assertion{
    copy undecidable-evolutionary-virus to file till
    line-starts-with-zzz;
    if file = P1 then print ''if D(P1,P2)
    then print 1;'' ;
    if file = P2 then print ''if D(P1,P2)
    then print 0;'' ;
    copy undecidable-evolutionary-virus to file till
    end-of-input-file;
  }
  main-program{
    if random-bit = 0 then file = P1 otherwise
    file = P2;
    copy-with-undecidable-assertion;
```

```
    zzz:  infect-executable;
    if trigger-pulled do-damage;
    goto next;
  }
next:
}
```

Пусть существуют два типа программ, зараженных вирусом UEV, — типа $P1$ и типа $P2$. Тогда, если тип программы $P1$, то строка, помеченная меткой "zzz", будет

$$\text{if } D(P1, P2) \text{ then print 1.}$$

Если же тип программы P2, то соответствующая строка —

$$\text{if } D(P1, P2) \text{ then print 0.}$$

Таким образом, обе зараженные программы будут вызывать процедуру D для определения эквивалентности. Если процедура D даст заключение об их эквивалентности, то $P1$ напечатает 1, в то время как $P2$ выдаст на экран 0, а значит процедура D противоречива. Если же процедура D говорит, что программы разного типа, то обе программы ничего не выведут на экран и будут одинаковы.

Итак, гипотетическая решающая процедура D внутренне противоречива, а значит, точное определение эквивалентности двух программ по их внешнему виду — неразрешимая задача. Если обе программы $P1$ и $P2$ — две модификации одного и того же вируса, то задача распознавания вируса — неразрешимая задача (см. также [1]).

4. Моделирование системы обеспечения безопасности. Пусть система M состоит из множества (достаточно большого) вычислимых функций m_i. Данное предположение сделано в силу соображений, изложенных в [4–6]. Злоумышленник пытается "вывести из строя" данную систему, для этого ему требуется нарушить работу хотя бы одного компонента. Злоумышленник в

состоянии выбирать какой-то элемент m_i и аргумент x. Далее, для реализации атаки злоумышленнику необходимо найти пару (i, x) такую чтоб $m_i(x)$ не вычислялась. Значит для реализации атаки требуется элемент множества $\{\langle i, x\rangle; m_i(x)\uparrow\} = \overline{K}$.

Напомним, что K — (Канторовское множество) является рекурсивно перечислимым, но не рекурсивным, а значит его дополнение по теореме Поста не рекурсивно и не рекурсивно перечислимо. Это означает, что подсистема контроля не может по виду запроса $\langle i, x\rangle$ определить, что данный запрос отсылает злоумышленник (так как K не рекурсивно).

Заметим, что и злоумышленник не имеет точного алгоритма поиска уязвимостей (дополнение K — не рекурсивно перечислимо). Схема взаимодействия показана на рис.1.

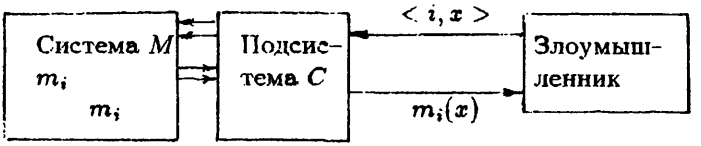

Рис.1.

§ 2. Модель вирус–антивирус

1. Базовые определения. Для рассмотрения процесса обнаружения вируса программой–антивирусом предлагается следующая модель взаимодействий. Пусть M — множество программ, записанных на некотором языке L в алфавите A. Таким образом, буквами алфавита являются команды определенного языка программирования, а программы — слова или предложения языка L. Учитывая конечность алфавита, количество различных программ M_i длины N (различных в смысле написания) определяется величиной $S(N) = N^{|A|}$ для размера алфавита A. Учитывая, что для большинства вычислительных систем и языков программирования смысл имеют цепочки команд, записанных по некоторым правилам (которые определяются особенностями конкретного языка и системы), будем считать, что множество программ M_i разбито на на два класса. Первый класс *выполняемые*

программы K_{EX} — это основной класс для нашего рассмотрения, в него входят те программы, которые могут выполнятся в нашей вычислительной системе. Все остальные цепочки попадут в другой класс *потенциально выполняемые программы* K_{PEX} : $|K_{PEX}| + |K_{EX}| = S(N)$.

Припишем каждой программе M_i в нашей модели некоторую вероятностную величину q_i, характеризующую вероятность того, что именно эта программа будет выполняться или появится в нашей системе в определенный промежуток времени. Для введенного разбиения на классы очевидны следующие выражения: $\sum_{M_i \in K_{PEX}} q_i \approx 0, \sum_{M_i \in K_{PEX} \cup K_{EX}} q_i \approx 1$.

Распределение q_i для M_i из класса K_{EX} считаем равномерным, т.е. все выполняемые программы равновероятны в нашей системе.

Введем понятие *вируса* в нашу модель. Для этого предлагается рассматривать процесс заражения программы M_i вирусом T_j как отображение $E_{ji} = T_j(M_i)$ пространства программ в себя. Согласно определению вируса, вирус модифицирует исходное предложение — программу, добавляя свое тело. Будем рассматривать лишь такие вирусы, которые не нарушают работоспособность программы. То есть, фактически, преобразование T_j есть отображение пространства M в класс K_{EX}. Кроме того, введенное условие требует, что часть кода программы — тело вируса должно обладать способностью восстановить исходную последовательность выполнения программы–жертвы, чтобы скрыть сам факт инфицирования. То есть инфицированная программа должна выполнить как минимум те же действия, что и неинфицированная версия. Для выполнения этого требования должно существовать обратное отображение T_i^{-1}, выполняемое телом вируса внутри инфицированной программы, для извлечения исходной программы в процессе выполнения инфицированной программы. Для того, чтобы такое функциональное описание вируса совпадало с общепринятым понятием вируса как программы, заражающей другие программы, достаточно считать, что программа–вирус есть ни что иное как: $E_{j0} = T_j(M_\epsilon)$, где M_ϵ есть программа нулевой длины. Добавим к множеству вирусов в нашей системе тождественное отображение I, означающее

отсутствие заражения программы каким бы то ни было вирусом. Аналогично программам–предложениям, поставим в соответствие каждому вирусу–отображению T_j величину p_j определяющую вероятность заражения некоторой программы M вирусом T_j.

Заметим, что для большинства существующих вычислительных систем величина $p_i = P(I)$ много больше любой другой $p_j = P(T_j)$, для T_j не равного I. Это следует из того, что суммарное число зараженных программ много меньше числа неинфицированных. Наличие тождественного отображения I позволяет утверждать, что $\sum p_i = 1$.

Обозначим "взвешенную сумму" всех отображений через "сумму вирусов" $K = p_0 I + \sum p_j T_j$.

В таком случае, рассматриваемая система, состоящая из множества программ M_i с вероятностями q_i и семейства отображений T_j с вероятностями p_j, образуют *секретную систему Шеннона* [7]. Нами предлагается применить аппарат, разработанный для оценки характеристик секретных систем, применяемых в криптографии, и предложенный Шенноном, для изучения систем–антивирусов. В частности данная методика позволяет, хотя бы теоретически, получить количественные характеристики для оценки трудоемкости обнаружения вирусов и зараженных ими программ.

В системе Шеннона есть две стороны. Для нашей системы первая играющая сторона — это заражающий процесс. Этот процесс выбирает некоторую программу M_j из множества M с вероятностью q_j, выбирает конкретный вирус T_i с вероятностью выбора p_i и "шифрует" данную программу, заражая ее, с помощью отображения T_i, таким образом получая E_{ij}. В дальнейшем, эта сторона способна по зараженной версии программы точно восстановить исходную программу. Второй стороной в нашей "игре" выступает процесс–антивирус, который по заданной зараженной программе E_{ij} пытается определить, какой конкретно вирус T_i использовался для заражения, а также извлечь неинфицированную версию программы–жертвы, т.е. определить исходное предложение M_j. Будем считать, что обе стороны знают

все параметры нашей системы, т.е. множества M и $\{T_i\}$, а также вероятности p_i.

2. Оценка единственности решения. Введенные Шенноном понятия *ненадежность сообщения* и *ненадежность ключа* [7] как характеристические меры надежности и устойчивости секретной системы в нашем случае приобретают естественную интерпретацию. Будем считать, что выражения [2]

$$H_E(K) = \sum P(E,K)\log P_E(K),$$

$$H_E(M) = \sum P(E,M)\log P_E(M),$$

задают численную меру, определяющую вероятную ошибку антивируса, при определении использовавшегося вируса и заражаемой программы. Например, если при работе антивируса вычисленное значение $H_E(M)$ обрабатываемой программы, потенциально зараженной вирусом, дало значение ноль, то мы предполагаем , что антивирус точно определит исходную программу.

В таком случае, по известным E и M, учитывая полную определенность исходной информации, антивирус даст нам точный вид вируса T_i, использованного при заражении. Аналогично, если ненадежность суммы вирусов $H_E(K)$ равна нулю, то это так же гарантирует точность решения.

3. Характеристические параметры системы. Для применения методики Шеннона необходимо, помимо знания исходных параметров системы, вычислить значения $H(K)$ — *неопределенность суммы вирусов*, $X(N)$ — *характеристика появления вируса* и $D(N)$ — *избыточность* используемого языка. Для тривиальных случаев эти величины могут быть определены довольно точно, в других случаях разумно, видимо, использовать введенное Шенноном аналитическое приближение *случайный шифр* [7] и полученные для этого объекта результаты. Заметим только, что функция $D(N)$, определяющая избыточность языка, должна иметь вид похожий на избыточность естественного языка [3]

---

[2] Данные выражения определяют *энтропию*, свойства которой подробно рассмотрены в [8].

[3] Программу на любом языке программирования можно про-

(но эта величина может быть подсчитана и точно для каждого конкретного машинного языка).

В качестве простого примера рассмотрим вирус T, который будучи активирован в системе, находит любую программу и записывает свое тело после некоторой строки. Программа–жертва после инфицирования выглядит следующим образом:

```
program{
      GOTO START_VIRUS
  START_PROGRAM:
      program_body{}
      GOTO CONTINUE_PROGRAM
  START_VIRUS:
      BODY_OF_VIRUS{}
      GOTO START_PROGRAM
  CONTINUE_PROGRAM:
      program_body{}
}
```

Пусть тело вируса, вместе с необходимыми командами перехода и метками составляют t, тогда для полученной антивирусом анализируемой программы длины N число разновидностей этого вируса составляет $N - t + 1$. Здесь мы учитываем, что возможна программа–жертва нулевой длины. Добавление тождественного отображения дает значение

$$X(N) = \begin{cases} \log(N - t + 2), & N \geq t, \\ \log 1, & N < t. \end{cases}$$

Для определения величины $K(N)$, используем следующий факт: оператор перехода goto имеет ограниченную зону действия [4], поэтому существует максимальное число строк N_m, которое определяет максимально возможное число разновидностей

читать используя естественный язык, так же как и алгоритм на естественном языке можно описать на языке программирования.

[4] Для процессора семейства 8086 один сегмент памяти равен 64 Кбайт

нашего вируса. Тогда $K(N) = \log(N_m - t + 2)$. Это ограничение обусловлено видом рассматриваемого вируса, но и для других возможных вирусов существуют подобные разумные ограничения на число их вариаций. Графическое решение для данной системы схематически представлено на рис.2.

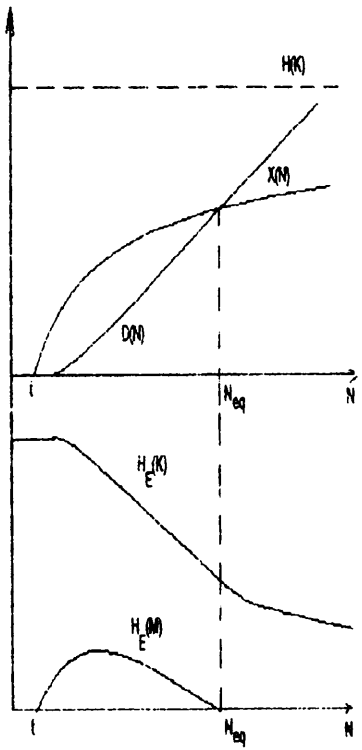

Рис.2.

Для рассматриваемого примера нулевое значение ненадежности программы при длине зараженной программы меньше t следует из того, что единственно возможный вирус — тождественное отображение. Возможное ненулевое значение $H_E(K)$

при таких значениях аргумента есть следствие как концевых эффектов, так и приближенным характером определения исходных характеристик системы.

4. Выбор оптимального пути анализа. В приведенном выше примере анализ зараженных программ проводится построчно, начиная с первой строки программы. В то же время известно, что большое число вирусов дописывают свой код именно в конец программы–жертвы. Кроме того, существуют вирусы, использующие свободное дисковое пространство между строками программы. В таком случае, желательно ввести еще один промежуточный шаг в процесс анализа. На этом шаге антивирус выбирает последовательность строк программы $\{i_1, \ldots i_n\}$, которые будут анализироваться им последовательно, как в приведенном примере. Для того, чтобы выбрать наиболее короткий путь, антивирус должен построить по каждому предлагаемому пути $TR_j = \{i_1, \ldots i_n\}$ соответствующие ему функции $H_j(K)$, $X_j(N)$, $D_j(N)$, провести необходимый анализ полученных результатов и выбрать наиболее оптимальный путь. Далее антивирус проводит "дешифровку программы" уже по этому пути. Введем понятие *уровень доверия* $L(K)$, определяющее наибольшее значение $H_E(K)$, при котором антивирус определяет вирус с необходимой нам точностью[5]. Наименьшая длина пути, при котором $H_E(K)$ опускается ниже уровня доверия, назовем *точкой доверия* $D_L(K)$, которая в силу невозрастания $H_E(K)$ [7,8] определяется соотношением:

$$D_L(K) = \min_{N>0}\{H_{E,K}(N) \geq H(K) \cdot L(K)\}.$$

Учитывая, что точка единственности

$$D_N(K) = \min_{N>0}\{H_E(M,N) = 0\}$$

получаем *критерий оптимальности пути*

$$D_{TR}(K) = \min\{D_L(K), D_N(K)\}.$$

---

[5]Фактически, данная величина определяет относительную погрешность нахождения вируса.

Поэтому, среди всех возможных путей, антивирус в первую очередь будет проверять те пути TR_j, для которых значение D_{TR_j} будет минимальным. Возможные варианты приведены на рис.3.

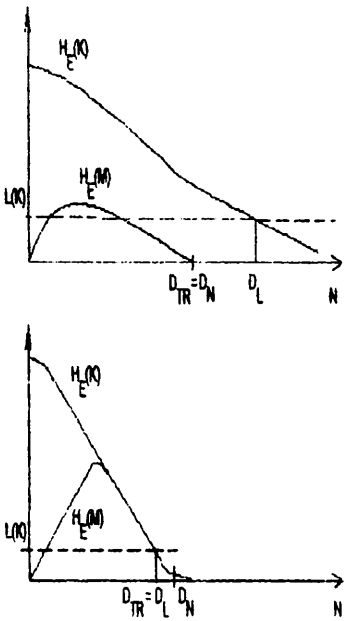

Рис. 3

В качестве первого приближения данный критерий выбора пути достаточно эффективен. Естественно, лучшим возможным критерием можно выбрать такое условие на вид функций $H_E(K)$ и $H_E(M)$, которое учитывало бы характер их поведения (скорость стремления к нулю) с ростом N.

5. Анализ программы — метод вероятных слов. Для непосредственного анализа инфицированной программы возможно использование самых разных методов и приемов "дешифрования"

сообщений. В качестве одного из них приведем пример использования "метода вероятных слов".

Вероятные слова — это куски кода программы ("слова" в определении языка), которые можно ожидать в зараженной программе вследствие того, что они характерны для данного набора программ, языка, вычислительной системы и набора отображений — вирусов. Для приведенного выше примера вируса таким словом, очевидно, может являться последовательность команд, составляющих тело вируса. После выбора части зараженной программы, которая, предположительно, и является нашим *вероятным словом* мы определяем использовавшийся вирус. Применяя это предположение к остальным частям нашей программы, мы проверяем правильность нашего предположения. В общем случае данный метод чрезвычайно трудоемок (верхняя грань — полный перебор), но для многих типов вирусов этот метод дает точный результат достаточно быстро. Возвращаясь к нашему примеру, этот метод дает точный ответ уже после первой проверки. Если антивирус находит указанное "слово" в проверяемой программе, то эта программа инфицирована нашим вирусом, в противном случае считаем, что к ней было применено тождественное отображение, т.е. программа не инфицирована вирусом. Модификация данного метода — это поиск не точного соответствия некоторому блоку кода, а некому "шаблону" из последовательности команд, в которых удалены служебные символы.

6. Алгоритм работы антивируса. Приведем всю последовательность действий, выполняемую антивирусом.

1. Первый шаг — это разработка антивируса. На этом шаге определяется рабочий язык L, пространство K_{EX}, множество отображений–вирусов T_j, а также все необходимые вероятностные параметры p_j, q_i.

2. Для каждой анализируемой программы строятся необходимые служебные функции. Используя сведения о свойствах данного множества вирусов, определяют набор оптимальных путей TR_j, а также необходимые функции $H(K)$, $X(N)$, $D(N)$.

3. Делается выбор того множества путей, для которого критерий оптимальности D_{TR} наименьший.

4. Для каждого выбранного пути применяется один из методов анализа кода программы (например, метод вероятных слов). В случае положительного результата для одного из методов, анализ прекращается[6].

5. Извлечение исходного кода программы из зараженного кода на основании полученных о вирусе сведениях.

Необходимо дать некоторый комментарий по п.2 алгоритма. В простейшем случае все необходимые операции выполняет сам разработчик антивируса, получая, таким образом, систему, наиболее эффективно работающую с заданным набором вирусов и весьма ограниченным по свойствам количеством анализируемых программ. Более интересный результат получается, если антивирус сам выполняет эти действия. Например, зная длину анализируемой программы, антивирус отбирает во множество вирусов только наиболее вероятные для данной длины отображения вирусы. Это позволяет уменьшить как число возможных вирусов T_j, так и число испытываемых путей TR_j. Для реализации данного свойства, антивирус должен, как минимум, уметь строить служебные функции $H(K)$, $X(N)$, $D(N)$. К сожалению, известные на данный момент статистические методы построения функций $X(N)$, $D(N)$ требуют весьма значительных затрат времени и системных ресурсов [7]. Поэтому, наиболее эффективным является совмещение этих подходов, когда на стадии разработки автор, основываясь на экспертных оценках специалистов, ограничивает возможное число вариантов. После этого антивирус использует только те пути, которые наиболее вероятны для данного типа вируса, а значит и уже заложенные разработчиком $H(K)$, $X(N)$, $D(N)$.

Литература

1. COHEN F. Computer Viruses — Theory and Experiments //IFIP-TC 11 "Computers and Security". 1987. — Vol. 6. — P. 22–35.

[6]Видимо, наилучший результат даст возможность распараллеливания этого шага

2. COHEN F. Computer Viruses // Dissertation at the University of Southern California,Pittsburg. ASP Press. 1985.

3. COHEN F. Operating Systems Protection Through Program Evolution // IFIP-TC11 "Computers and Security". 1994.

4. TURING A. On Computable Numbers, with an Application to the Entscheidungsproblem // London Math Soc Ser 2, 1936.

5. COHEN F. Models of Practical Defenses Against Computer Viruses // IFIP-TC11, "Computers and Security". 1988. — Vol.7. —P. 6.

6. BELL, La PADULA. Secure Computer System: A Mathematical Model // The Mitre Corporation, 1973. — P.

7. SHANNON C. Communication theory of secrecy systems // Bell System Techn. 1949. — Vol. 28. — P. 656-715.

8. SHANNON C. A mathematical theory of communication // Bell System Techn. 1948. — Vol. 27. — P.

Поступила в редакцию
20 декабря 2001 года