Министерство образования и науки РФ

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

КОМПЛЕКСНАЯ ЗАЩИТА
ОБЪЕКТОВ ИНФОРМАТИЗАЦИИ

КНИГА 19

Ю.М. МОНАХОВ, Л.М. ГРУЗДЕВА, М.Ю. МОНАХОВ

ВРЕДОНОСНЫЕ ПРОГРАММЫ В КОМПЬЮТЕРНЫХ СЕТЯХ

Учебное пособие

Рекомендовано Учебно-методическим объединением вузов Российской Федерации по образованию в области историко-архивоведения в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности 090104 «Комплексная защита объектов информатизации»

Владимир 2010

УДК 930.1
ББК 32.81
В81

Редактор серии – доктор технических наук М.Ю. Монахов

Рецензенты:
Зав. кафедрой оперативно-технической деятельности
Владимирского юридического института
Федеральной службы исполнения наказаний России
кандидат технических наук, доцент
К.Н. Курысев

Доктор технических наук, профессор
Владимирского государственного университета
О.В. Веселов

Печатается по решению редакционного совета
Владимирского государственного университета

Монахов, Ю.М.

В81 Вредоносные программы в компьютерных сетях : учеб. пособие / Ю.М. Монахов, Л.М. Груздева, М.Ю. Монахов ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2010. – 72 с. (Комплексная защита объектов информатизации. Кн. 19). – ISBN 978-5-9984-0087-2.

Это девятнадцатая книга из серии «Комплексная защита объектов информатизации». В ней представлен систематизированный материал, посвященный основным методам и средствам вредоносных программ.

Предназначено для студентов специальности 090104 – Комплексная защита объектов информатизации.

Табл. 3. Ил. 16. Библиогр.: 30 назв.

УДК 930.1
ББК 32.81

ISBN 978-5-9984-0087-2

© Владимирский государственный университет, 2010

Предисловие

Вредоносные программы – это прежде всего компьютерные вирусы (computer viruses), сетевые черви (worms) и троянские кони (трояны, Trojan horses). Кроме этого к вредоносным иногда относят средства несанкционированного доступа, а также разновидность троянов – программные закладки или «логические бомбы» (logic bombs).

Компьютерные вирусы и сетевые черви характеризуются в первую очередь способностью к саморазмножению (саморепродукции), для *компьютерных вирусов* – это саморепродукция в пределах отдельного компьютера, на другие компьютеры вирус сам не распространяется, а предоставляет делать это человеку, копирующему программы с одной машины на другую. Для *сетевого червя* саморазмножение – это именно процесс распространения с одного компьютера на другой по сети, на одной машине иметь более одной копии ему не только не нужно, но и вредно, так как каждая копия отнимает дополнительные ресурсы.

Троянские кони характеризуются прежде всего функцией нанесения ущерба – сразу после запуска или через некоторое время они уничтожают доступную информацию, форматируют жесткие диски, портят конфигурацию компьютера и т.п. Обычно они маскируются под игровые или развлекательные программы.

Программные закладки также содержат некоторую функцию, не оговоренную в спецификации, но эта функция, наоборот, старается быть как можно незаметнее, так как чем дольше программа не будет вызывать подозрений, тем дольше эта функция-закладка сможет работать. Примерами последних вредоносных программ могут быть:

- широко известная программа, осуществляющая атаку «салями» – зачисление остатков денег при округлении на определенный счет;

- программа, посылающая информацию о состоянии дел в конкурирующей фирме;

- закладка, внедренная в ПО военного комплекса, блокирующая его в некоторый момент по радиосигналу и т.п.

У средств несанкционированного доступа главной особенностью является способность к проникновению в защищенную компьютерную систему через ошибки или «дыры» в соответствующих системах безопасности или администрирования, а также «в лоб» – путем перебора или криптографической дешифровки паролей.

В данной книге мы будем рассматривать компьютерные вирусы, черви и трояны, называя их обобщенно вредоносные программы (ВП) или просто «вирусы», как часто принято говорить в специальной литературе.

Эпидемии названных ВП в настоящее время наносят существенный ущерб различным организациям и отдельным пользователям компьютеров. За последние 10 – 15 лет распространение вредоносного кода, носившее локальный характер, превратилось в глобальные эпидемии вредоносных программ, не требующих для распространения участия пользователей. Работа и функционирование многих структур и организаций тесно связана или полностью зависит от глобальных сетей. Сетевые ВП, размножающиеся в неограниченном количестве, забивают каналы передачи информации, тем самым нанося огромные убытки, не говоря уже о том, что код вредоносной программы может содержать деструктивные функции, что может привести к потере или утечке конфиденциальной информации.

Современная ВП может инфицировать большую часть вычислительных машин всего за несколько часов благодаря масштабам глобальной сети Интернет и огромным пропускным способностям современных каналов передачи информации. Многие исследователи вирусных компьютерных эпидемий отмечают, что современные ВП используют далеко не самые оптимальные стратегии поиска новых компьютеров для заражения. В будущем возможно появление вредоносного кода, способного распространяться за считанные секунды благодаря использованию усовершенствованных стратегий распространения. Моделирование эпидемий ВП

показывает, что большую часть времени, за которое доля инфицированных компьютеров достигает своего максимального значения, вредоносная программа тратит на заражение небольшого, в масштабах эпидемии, числа компьютеров. Таким образом, в будущем возможно ускоренное развитие эпидемий ВП благодаря предварительному анализу состояния сети передачи данных, т.е. составления списка уязвимых компьютеров, который будет использоваться для того, чтобы захватить некоторое «критическое» число уязвимых компьютеров. Если это «критическое» число компьютеров будет захвачено, дальнейшее распространение станет происходить лавинообразно.

Современное программное обеспечение также не способствует снижению числа новых эпидемий ВП. Новые уязвимости в различных программах находят практически каждый день, производители программного обеспечения не всегда оперативно их устраняют, а готовые заплатки устанавливаются очень медленно. Некоторые пользователи и технический персонал сетей никак не заботятся о безопасности собственных компьютеров.

Определенную роль в распространении вирусов играет и человеческий фактор: неопытные пользователи, не задумываясь, открывают все почтовые вложения, через которые распространяются многие вредоносные программы. Также многие пользователи не заботятся о регулярном обновлении антивирусных программ.

Таким образом, в настоящее время существуют множество факторов, способствующих появлению массовых эпидемий ВП. К тому же, современные сетевые черви (согласно многим работам в области компьютерной безопасности) используют далеко не весь свой потенциал; возможно появление вредоносного кода, распространяющегося за считанные минуты.

Задача противодействия распространению ВП крайне актуальна. Все организации, работа которых так или иначе связана с использованием сетей передачи данных, терпят убытки от постоянных вспышек эпидемий сетевых червей, новые модификации которых появляются каждый день.

Противодействие распространению и созданию вредоносных программ – очень сложная задача, которая имеет множество ас-

пектов, одним из них является моделирование и методы предсказания распространения вредоносных программ. С помощью математических моделей можно оценить масштабы возможной эпидемии, изучить динамику изменения числа зараженных компьютеров и т. д. Моделирование также может быть использовано для того, чтобы оценить эффективность тех или иных мер противодействия распространению, например, лечения уже зараженных компьютеров или предварительного устранения уязвимостей в ПО, используемых вредоносным кодом для инфицирования. Таким образом, эпидемиологические модели являются необходимым инструментом для изучения и противодействия распространению вирусных атак.

Самый простой подход – использование классических эпидемиологических моделей, разработанных еще в XIX веке для изучения эпидемий инфекционных заболеваний и основанных на системах дифференциальных уравнений. Однако эти модели достаточно примитивны и не учитывают некоторых особенностей распространения компьютерных вирусов. Характер эпидемий, предсказанных с помощью традиционных моделей, достаточно часто не совпадает со статистическими данными, поэтому актуальна задача создания новых, более подходящих математических моделей. Кроме классических моделей существуют модели распространения эпидемий, специально разработанные для изучения компьютерных вредоносных программ. Многие из них базируются на измененных системах дифференциальных уравнений, сформулированных в классических эпидемиологических моделях. Большинство моделей не вносит существенных изменений в традиционные модели, а некоторые из них предназначены только для конкретных видов ВП.

В первой главе дается историческая ретроспектива возникновения и совершенствования вредоносных программ, причем акцент изложения смещен не к глобальному охвату всех существующих ВП, а на основные идеи по их созданию и вредоносному действию. Во второй – излагается классическая теория формирования и обнаружения вредоносного программного кода, в третьей – рассматриваются основные математические модели распространения ВП в вычислительной среде.

Глава 1. КРАТКАЯ ИСТОРИЯ ВОЗНИКНОВЕНИЯ И СОВЕРШЕНСТВОВАНИЯ ВРЕДОНОСНЫХ ПРОГРАММ

Теоретические основы построения вирусов были заложены в 40 – 50-х гг. прошлого столетия в трудах по самовоспроизводящимся автоматам Джоном фон Нейманом (John von Neumann). В дальнейших исследованиях (С. Пенроуз (L. S. Penrose), Ф. Ж. Шталь (F. G. Stahl)) были описаны модели структур, способных к активации, размножению, мутациям, захвату. Отдельные модели были реализованы в виде программного кода. Первые самораспространяющиеся программы, написанные в исследовательских целях, не были вредоносными в настоящем понимании.

В 1983 г. Фред Коэн (Fred Cohen) в работе, посвященной исследованию самовоспроизводящихся компьютерных программ, ввел термин «компьютерный вирус». В этом же году был предложен проект по созданию самораспространяющейся программы (Университет Южной Калифорнии, США), которую тут же окрестили вирусом. По результатам этих исследований была опубликована работа «Computer Viruses: theory and experiments», обозначающая проблему. Видимо, с этого момента следует считать начало практики создания, совершенствования, обнаружения и уничтожения вредоносного программного кода

1.1. Первые вирусы и вирусные эпидемии

Pervading Animal (начало 70-х гг.) – первый известный вирус-игра для вычислительной машины UNIVAC 1108. Программа с помощью наводящих вопросов пыталась определить имя животного, задуманного играющим. Благодаря наличию функции добавления новых вопросов, когда модифицированная игра записывалась поверх старой версии, плюс копировалась в другие директории, через некоторое время диск становился переполненным.

Сетевой вирус CREEPER появился предположительно в 1973 г. в сети Arpanet. Программа была написана Бобом Томасом (Bob Thomas)

для модуля специализированной ОС, отвечающего за удалённое исполнение программ. CREEPER не был ни вирусом, ни червём, а самоперемещающейся программой, то есть когда на удалённом компьютере запускалась новая копия CREEPER, предыдущая копия прекращала свою работу. CREEPER выводил на терминал сообщение «I'M THE CREEPER… CATCH ME IF YOU CAN». Позже Рэем Томлинсоном была написана программа REAPER, которая точно так же перемещалась по сети, и в том случае, когда обнаруживала действующий CREEPER, прекращала его выполнение (прообраз антивирусной программы).

Возможности первых вирусов были сильно ограничены малой функциональностью существующих на тот момент (вычислительных машин) (ВМ). Только в конце семидесятых, вслед за выпуском нового поколения персональных компьютеров Apple (Apple II) и впоследствии IBM PC (1981 г.), стали возможны вирусные эпидемии. Появление BBS (Bulletin Board System) обеспечило быстрый обмен информацией между удаленными ВМ.

Вирус Elk Cloner (1981 г.) изначально использовался для распространения пиратских копий компьютерных игр. Поскольку жестких дисков тогда еще не было, он записывался в загрузочные сектора дискет. Проявлял себя переворачиванием изображения на экране и выводом текста.

Elk Cloner – один из первых компьютерных вирусов, распространившийся «in-the-wild», т.е. он был обнаружен на компьютерах сторонних пользователей, а не в системе, на которой он был разработан. Вирус был написан в 1981 г. Ричардом Скрента для компьютеров Apple II. Elk Cloner распространялся, заражая DOS Apple II, записанную на гибких дисках. После того как компьютер загружался с зараженной дискеты, автоматически запускалась копия вируса. Вирус не влиял на работу вычислительной машины. Когда происходил доступ к незараженной дискете, вирус копировал себя туда, заражая её, и далее медленно распространяясь с диска на диск, с компьютера на компьютер. Вирус не причинял вреда намеренно, хотя он мог повредить диски, содержащие нестандартный образ DOS, затирая резервные дорожки диска вне зависимости от их содержимого. После каждой 50-й загрузки вирус выводил на экран стишок:

Elk Cloner: The program with a personality
It will get on all your disks
It will infiltrate your chips
Yes it's Cloner!
It will stick to you like glue
It will modify ram too
Send in the Cloner!

Считается, что первые антивирусные утилиты были написаны Анди Хопкинсом (Andy Hopkins). Программы CHK4BOMB и BOMBSQAD (1984 г.) позволяли производить анализ загрузочного модуля с помощью контекстного поиска и перехватывать операции записи и форматирования, выполняемые через BIOS.

Brain (1986 г.) – первый вирус для IBM-совместимых компьютеров, вызвавший глобальную эпидемию. Он был написан Баситом Фарук и Амжадом Алви (Basit Farooq Alvi и Amjad Alvi). Вирус заражал загрузочные сектора, менял метку диска на «(c) Brain» и оставлял сообщение с именами, адресом и телефоном авторов. Отличительной чертой его была функция подмены в момент обращения к нему зараженного сектора незараженным оригиналом. Это дает право назвать Brain первым стелс-вирусом (stealth virus – вирус-невидимка, полностью или частично скрывающий свое присутствие в системе путем перехвата обращений к операционной системе, осуществляющих чтение, запись, чтение дополнительной информации о зараженных объектах (загрузочных секторах, элементах файловой системы, памяти и т.д.)). В течение нескольких месяцев эпидемия достигла глобальных масштабов. Заметим, что ничего деструктивного вирус не делал.

В этом же году Ральф Бюргер (Ralf Burger) открыл возможность создания программой своих копий путем добавления своего кода к выполняемым DOS-файлам формата COM. Образец программы, получившей название Virdem, был прототипом для создания тысяч компьютерных вирусов, частично или полностью использовавших описанные автором идеи.

1.2. Вредоносные вирусы и первые антивирусные программы

Lehigh (1987 г.) – первый по-настоящему вредоносный вирус, вызвавший эпидемию в Лехайском университете (США). Он зара-

жал только системные файлы COMMAND.COM и был запрограммирован на удаление всей информации на текущем диске. В течение нескольких дней было уничтожено содержимое сотен дискет из библиотеки университета и личных дискет студентов. Всего за время эпидемии было заражено около четырех тысяч компьютеров. Однако за пределы университета Lehigh не вышел.

Семейство резидентных файловых вирусов Suriv (1987 г.) – творение программиста из Израиля. Самая известная модификация – Jerusalem – стала причиной глобальной вирусной эпидемии, первой настоящей пандемией, вызванной MS-DOS-вирусом. Действие вирусов Suriv сводилось к загрузке кода в память компьютера, перехватыванию файловых операций и заражению запускаемых COM- и/или EXE-файлов. Это обеспечивало практически мгновенное распространение вируса по мобильным носителям. Jerusalem отличался от своих предшественников дополнительной деструктивной функцией – уничтожением всех запускаемых программ в пятницу, 13-го. Такой черной датой стало 13 мая 1988 г., когда в одночасье перестали работать компьютеры многих коммерческих фирм, государственных организаций и учебных заведений, в первую очередь Америки, Европы и Ближнего Востока.

Червь Морриса (1988 г.). С ним связана первая эпидемия, вызванная сетевым червем. Программа использовала ошибки в системе безопасности ОС Unix для платформ VAX и Sun Microsystems. С целью незаметного проникновения в вычислительные системы, связанные с сетью Arpanet, использовался подбор паролей. Это позволяло маскироваться под задачу легальных пользователей. Однако из-за ошибок в коде безвредная по замыслу программа неограниченно рассылала свои копии по другим компьютерам сети, запускала их на выполнение и таким образом забирала под себя все сетевые ресурсы. Червь Морриса заразил до 9000 компьютеров в США и практически парализовал их работу на срок до пяти суток. Общие убытки оценили минимум в 96 млн долларов. Ущерб был бы гораздо больше, если бы червь изначально создавался с разрушительными целями.

Отметим, что 4 мая 1990 г. впервые в истории состоялся суд над автором компьютерного вируса, который приговорил Р. Мор-

риса к трем годам условно, 400 часам общественных работ и штрафу в 10 тыс. долларов.

Эпидемия червя Морриса стала причиной создания организации CERT (Computer Emergency Response Team), в функции которой входит оказание содействия пользователям в предотвращении и расследовании компьютерных инцидентов, имеющих отношение к информационным ресурсам. На сайте этой организации (http://www.cert.org) оперативно публикуются самые последние сведения о новых вредоносных программах, обнаруженных уязвимостях в программном обеспечении, методах защиты сетей, аналитические статьи, а также результаты различных исследований в области компьютерной безопасности.

Dr. Solomon's Anti-Virus Toolkit (1988 г.) – первая широко известная антивирусная программа. Созданная Аланом Соломоном (Alan Solomon), она завоевала огромную популярность и просуществовала до 1998 г., когда компания Dr. Solomon была поглощена другим производителем антивирусов – американской Network Associates (NAI).

Datacrime (1989 г.) – вирус, который отличался тем, что с 13 октября по 31 декабря инициировал низкоуровневое форматирование нулевого цилиндра жесткого диска, что приводило к уничтожению FAT-таблицы и потере данных. В ответ корпорация IBM выпустила (1989 г.) коммерческий антивирус Virscan для MS-DOS, позволяющий искать характерные для ряда известных вирусов строки в файловой системе.

Ярким примером первых троянцев служит AIDS Information Disk (1989 г.). 20 000 дискет с этой троянской программой были разосланы по адресам, украденным компанией PC Cyborg у журнала PC Business World и Всемирной организации здравоохранения. Эти диски якобы содержали информацию о вирусе иммунодефицита человека, и авторы совершенно очевидно сыграли на всеобщей озабоченности проблемой СПИДа (AIDS). Когда пользователь запускал установочную программу, троянец записывал себя на жесткий диск, создавал собственные скрытые файлы и папки, а также изменял системные файлы. После того как компьютер был загружен 90 раз, троянская программа шифровала содержимое же-

сткого диска, делая информацию на нем недоступной. Единственный доступный файл на диске назывался README и содержал адрес почтового ящика в Панаме, куда пользователю предлагалось отправить деньги за расшифровку своих данных. Интересно, что в лицензионном соглашении, записанном на дискете с «дистрибутивом» троянца, честно сообщалось об использовании авторами «программных механизмов», способных «неблагоприятно повлиять на другие программы и приложения». Автор троянца, Джозеф Попп (Joseph Popp), был осужден за вымогательство. Фактически Aids Information Diskette – это первый и единственный вирус, для массовой рассылки использовавший настоящую почту.

Cascade (1989 г.) – резидентный зашифрованный вирус, вызывающий характерный видеоэффект – осыпание букв на экране. Примечателен тем, что послужил толчком для профессиональной переориентации Евгения Касперского на создание программ-антивирусов, будучи обнаруженным на его рабочем компьютере. Уже через месяц второй инцидент (вирус Vacsina) был закрыт при помощи первой версии антивируса – V, который несколькими годами позже был переименован в AVP – AntiViral Toolkit Pro.

Eddie (также известен как Dark Avenger, 1989 г.) – первый вирус, противодействующий антивирусному программному обеспечению. Он заражает новые файлы, пока антивирус проверяет жесткий диск компьютера. Это достигалось применением особой технологии, позволяющей заражать не только COM/EXE-программы в момент их запуска, но и любые файлы при попытке прочтения.

Начиная с 1990 г. проблема вирусов окончательно утрачивает связь с научными кругами и становится достоянием рядовых программистов, преследующих личные цели. В Болгарии открывается первая BBS (VX BBS), ориентированная на обмен вирусами и информацией для вирусописателей. Чуть позже начинают появляться конференции Usenet по вопросам написания вирусов.

Chameleon (1990 г.) – первый полиморфный вирус. Заметим, что полиморфизм заключается в формировании кода вируса «на лету» – уже во время исполнения, при этом сама процедура, формирующая код, также не должна быть постоянной и видоизменяется при каждом новом заражении. Большинство антивирусных программ

пытаются обнаружить вредоносный код, соответствующий компьютерному вирусу, или часть кода такого вируса посредством проверки файлов и данных, находящихся на компьютере или пересылаемых через компьютерную сеть или Интернет. Если изменить код вируса, отвечающий за поиск и заражение новых файлов, или какую-либо другую важную его часть, то антивирус не сможет обнаружить такой «измененный» вирус. Часто код вируса редактируют, добавляя операторы NOP или другие операторы, не изменяющие алгоритм. Таким образом, обнаружение по-настоящему полиморфных вирусов с помощью скан-строк невозможно. Автор Chameleon, Марк Уошбурн (Mark Washburn) усовершенствовал принципы самошифрации вируса Cascade – свойство изменять внешний вид как тела вируса, так и самого расшифровщика.

Только в 1992 г. был изобретен достаточно эффективный способ нейтрализации полиморфных вирусов – эмулятор процессора для дешифрации кодов. Эта технология является неотъемлемым атрибутом каждого современного антивирусного продукта.

DiskKiller (1990 г.). Этим вирусом была заражена дискета бесплатного приложения к журналу PC Today. В июле 1990 г. подписчикам разошлось около 50000 экземпляров. Действие DiskKiller сводилось к уничтожению всей информации на жестком диске. Отметим, что в этом же году вышла первая версия антивирусной программы Norton AntiVirus.

Dir_II (1991 г.) – вирус, использовавший принципиально новый способ заражения – link-технологию. На сегодняшний день он остается единственным представителем этого класса, который был обнаружен в «диком» виде.

MtE (MuTation Engine, 1991 г.) – первый известный полиморфик-генератор. Его главное предназначение – возможность интеграции в другие вирусы для обеспечения их полиморфизма. MtE поставлялся в виде готового объектного модуля и сопровождался подробной документацией.

В 1992 г. появился первый конструктор вирусов VCL (Virus Creation Laboratory), представляющий собой графическую среду для разработки вирусов и различных троянских программ для MS DOS.

Начиная с этого момента, любой человек мог легко сформировать и написать вирус.

Win.Vir (1992 г.) – первый вирус, поражающий исполняемые файлы Microsoft Windows 3.1. Эпидемии не вызвал, и его появление осталось практически незаметным. Однако именно Win.Vir положил начало эпохи вирусов для Windows.

Shifter (1994 г.) – первый вирус, заражающий объектные модули (OBJ-файлы). SrcVir (1994 г.) – семейство вирусов, заражающих исходные тексты программ (C и Pascal).

OneHalf (1994 г.) – резидентный файлово-загрузочный полиморфный вирус, вызвавший глобальную эпидемию во всем мире, в том числе в России. OneHalf заражал загрузочные сектора дисков и COM/EXE-файлы, увеличивая их размер на 3544, 3577 или 3518 байтов, в зависимости от модификации. При каждой перезагрузке зараженного компьютера зашифровывались два последних незашифрованных ранее цилиндра жесткого диска. Это продолжалось до тех пор, пока весь винчестер не оказывался зашифрованным. Встроенная стелс-процедура позволяла вирусу при запросе зашифрованной информации производить расшифровку на лету – следовательно, пользователь долгое время пребывал в неведении. Единственным визуальным проявлением вируса было сообщение «Dis is one half. Press any key to continue...», выводившееся в момент достижения количеством зашифрованных цилиндров диска половины от их общего числа. Однако при первой же попытке лечения, после вылечивания загрузочных секторов диска вся информация на винчестере становилась недоступной, без возможности восстановления. Популярности этого вируса в России поспособствовала компания Доктор Веб (Dr. Web), которая выпустила новую версию своего антивируса, анонсировав его как средство от OneHalf. Однако на практике после лечения загрузочных секторов от этого вируса Dr.Web «забывал» расшифровать информацию на диске и восстановить ее было уже невозможно.

Concept (1995 г.) – первый макровирус, поражавший документы Microsoft Word. Green Stripe (1995 г.) – первый вирус для AmiPro, популярного в то время текстового редактора.

Boza (1996 г.) – первый вирус для ОС Microsoft Windows 95. Вирус Win.Tentacle (1996 г.) вызвал первую эпидемию среди пользователей Microsoft Windows 3.x. Отличительная его черта в том, что это был первый Windows-вирус, вырвавшийся на свободу. До той поры все Windows-вирусы жили только в коллекциях и электронных журналах вирусописателей, а в «живом виде» встречались только загрузочные, MS-DOS и макро-вирусы.

Wazzu (1996 г.) – вирус, ставший причиной вирусного инцидента в Microsoft. Он был обнаружен в одном из документов Word на веб-сайте корпорации. Позднее Wazzu был найден также на компакт-дисках, распространенных Microsoft на выставке компьютерных технологий Orbit (г. Базель, Швейцария), и на компакт-дисках Microsoft Solution Provider. Win95.Punch (1996 г.) – первый резидентный вирус для Windows 95. Он загружался в систему как VxD-драйвер, перехватывал обращения к файлам и заражал их.

Linux.Bliss (1997 г.) – первый вирус для ОС Linux. Одновременно с выходом очередной версии пакета офисных приложений Microsoft Office 97 отмечается постепенное «переползание» макро-вирусов на эту платформу. Первые макро-вирусы для Office 97 на поверку оказались полными аналогами своих предшественников, всего лишь конвертированными в новый формат. Несмотря на это очень скоро появились настоящие макро-вирусы, нацеленные на работу исключительно в Office 97. ShareFun (1997 г.) – первый макро-вирус для MS Word 6/7, использующий для своего распространения возможности электронной почты, в частности почтовую программу MS Mail. Homer (1997 г.) – первый сетевой вирус-червь, использующий протокол передачи данных File Transfer Protocol (FTP).

В начале 1998 г., вскоре после разработки технологии IRC (Internet Relay Chat) образовался новый класс вредоносных программ – IRC-черви. Win95.HPS и Win95.Marburg (1998 г.) – первые полиморфные Windows32-вирусы. Marburg известен также тем, что им были заражены компакт-диски, сопровождавшие английскую, словенскую, шведскую и итальянскую редакции журнала PC Gamer. В этом же году был обнаружен вирус Win95.CIH, содержащий «логическую бомбу», нацеленную на уничтожение всей информации на жестких дисках и порчу содержимого BIOS на некоторых мате-

ринских платах. Дата срабатывания программы (26 апреля) совпадала с датой аварии на Чернобыльской атомной электростанции, вследствие чего вирус получил второе имя – Чернобыль (Chernobyl). Масштабность эпидемии выяснилась 26 апреля 1999 г., когда по различным оценкам пострадало около полумиллиона компьютеров по всему миру, а общий ущерб составил сотни миллионов долларов США. Центром эпидемии стала Южная Корея, где было заражено более 300 тыс. компьютеров.

BackOrifice, Backdoor.BO (1998 г.) – первая известная утилита скрытого администрирования удаленных компьютеров. Единственное отличие этого трояна от обычных программ для удаленного управления – несанкционированная установка и запуск. Действие утилиты сводилось к скрытому слежению за системой: ссылка на троянца отсутствовала в списке активных приложений, но при этом зараженный компьютер был открыт для удаленного доступа. Фактически открывался свободный вход на зараженные компьютеры для других вредоносных программ, впоследствии возник целый класс червей, размножение которых базировалось на оставленных BackOrifice дырах.

Во второй половине 1998 г. вирусы активно начинают осваивать новые технологии: Java.StangeBrew – первый вирус, который заражал выполняемые модули Java, VBScript.Rabbit – скрипты Visual Basic (VBS-файлы), HTML.Internal – первый HTML-вирус.

Следующий период вирусной истории прошел под знаменем массовых рассылок по электронной почте. Happy99 (также известный как Ska, 1999 г.) – первый интернет-червь, использовавший для своего распространения программу MS Outlook.

В марте 1999 г. началась глобальная эпидемия вируса Melissa – первого макро-вируса для MS Word, сочетавшего в себе также и функциональность интернет-червя. Сразу же после заражения системы он считывал адресную книгу почтовой программы MS Outlook и рассылал по первым 50 найденным адресам свои копии. Подобно Happy99 вирус Melissa делал это абсолютно незаметно для пользователя и, что самое страшное, от его имени. Microsoft, Intel, Lockheed Martin были вынуждены временно отключить свои корпоративные службы электронной почты. Автор вируса Melissa Дэвид Л. Смит

(David L. Smith) был признан виновным и осужден на десять лет тюремного заключения.

ZippedFiles (также известный как ExploreZip, 1999 г.) – первый упакованный интернет-червь. Его тело было упаковано утилитой сжатия Neolite, обращаться с которой в то время антивирусные продукты не умели, что привело к эпидемии. Bubbleboy (1999 г.) – первый вирус из поколения червей-невидимок, распространявшихся по электронной почте без использования вложенных файлов и проникавших на компьютеры сразу же после прочтения зараженного письма. Все вирусы этого типа используют различные уязвимости в системе безопасности Internet Explorer.

Babylonia (1999 г.) – первый вирус-червь, который имел функции удаленного самообновления: он ежеминутно пытался соединиться с сервером, находящимся в Японии, и загрузить оттуда список вирусных модулей.

LoveLetter – скрипт-вирус, в 2000 г. побивший рекорд вируса Melissa по скорости распространения. Всего в течение нескольких часов были поражены миллионы компьютеров – LoveLetter попал в Книгу рекордов Гиннеса. Успех гарантировали методы социальной инженерии: электронное сообщение имело тему «I love you» и интригующий текст, призывающий открыть вложенный файл с вирусом. Liberty (2000 г.) – первая вредоносная программа класса троянский конь для PalmOS карманных компьютеров Palm Pilot.

Stream (2000 г.) – первый известный вирус, способный манипулировать дополнительными потоками (ADS) файловой системы NTFS. Pirus (2000 г.) – первый вирус, написанный на скрипт-языке PHP. Fable (2000 г.) – первый вирус, скрывающийся в информационных файлах PIF. Hybris (2000 г.) – опасный и технологически совершенный вирус, главным нововведением которого было использование как веб-сайтов, так и электронных конференций (в частности alt.comp.virus) для загрузки новых модулей вируса на зараженные компьютеры. Кроме того, для идентификации настоящих вирусных модулей, т. е. действительно выпущенных автором, Hybris использовал 128-битный электронный ключ RSA для их шифрования.

1.3. Современные вредоносные программы

Самые известные вирусы 2001 – 2005-х гг., такие как CodeRed или Nimda, продемонстрировали сочетание способности к практически мгновенному распространению и существенно усложненную, многоуровневую структуру. Фактически можно выделить такие тенденции: расширение спектра путей и методов проникновения, использование новых платформ и технологий, обновление вирусных кодов через Интернет, новые вредоносные функции и активное противодействие антивирусным программам.

Ramen (2001 г.) – вирус, за считанные дни поразивший большое количество крупных корпоративных систем на базе ОС Linux. Sadmind (2001 г.) – первый известный интернет-червь, заражающий компьютеры Sun Sparc с ОС Solaris/SunOS. Для размножения использовалась брешь в службе системного администрирования /usr/sbin/sadmind. Червь также атаковал HTTP-серверы с установленным Microsoft Internet Information Server (IIS). CodeRed (2001 г.) – представитель нового типа вредоносных кодов, способных активно распространяться и работать на зараженных компьютерах без использования файлов. В процессе работы такие программы существуют исключительно в системной памяти, а при передаче на другие компьютеры – в виде специальных пакетов данных. Для проникновения на удаленные компьютеры CodeRed использовал брешь в системе безопасности IIS, которая позволяет злоумышленникам запускать на удаленных серверах посторонний программный код. Microsoft выпустила соответствующую заплатку, однако подавляющее большинство пользователей не успело вовремя обновить свое программное обеспечение. CodeRed вызвал эпидемию, заразив около 12000 серверов по всему миру и провел крупномасштабную DDoS атаку на веб-сервер Белого дома, вызвав нарушение его нормальной работы. 19 июля появилась новая модификация CodeRed, показавшая чудеса распространения – более 350000 машин за 14 часов (до 2000 компьютеров в минуту). Однако по замыслу автора 20 июля вирус прекратил свое распространение.

Следующая версия CodeRed.c (CodeRed II) была обнаружена в конце 2001 г. После заражения (использовалась все та же брешь

в системе безопасности IIS) вирус ничем не выдавал свое присутствие один-два дня, после чего перезагружал компьютер и начинал активные попытки распространения, продолжавшиеся 24 часа. Червь также устанавливал троянскую программу explorer.exe и использовал встроенную бекдор-процедуру (Backdoor).

В это же время был обнаружен почтовый червь Sircam. Этот вирус отличала необычная процедура выбора имени зараженного вложения. Для этого случайным образом на диске выбирался документ, к имени которого добавлялось расширение .pif, .lnk, .bat или .com. Полученная конструкция вида mydiary.doc.com служила темой рассылаемых писем и именем новой копии программы: к отобранному файлу и дописывался код червя. Таким образом, Sircam мог привести к утечке конфиденциальной информации. При рассылке использовался собственный SMTP-клиент, в поле «От» указывался один из адресов, найденных на зараженном компьютере, а сообщение содержало текст вида «Hi! How are you? I send you this file in order to have your advice. See you later. Thanks.» Кроме этого в определенный момент времени (в зависимости от системного времени и модификации вируса) на зараженном компьютере удалялись все файлы на системном диске.

Nimda (2001 г.) – вирус-червь, в течение 12 часов поразивший до 450000 компьютеров. Для распространения были задействованы пять методов: электронная почта (брешь в системе безопасности Internet Explorer, позволяющая автоматически выполнять вложенный исполняемый файл), по локальной сети, внедрение на IIS-серверы, заражение браузеров через javascript, а также с помощью бекдор-процедур, оставленных CodeRed.c и Sadmind. После заражения Nimda добавлял в группу «Администраторы» пользователя под именем Guest и открывал локальные диски на полный доступ для всех желающих.

Klez (2001 г.) – почтовый червь. Программа проникала на компьютер по сети или через электронную почту, используя брешь в защите IFrame браузера Internet Explorer, которая допускала автоматический запуск вложенного файла. Также вирус имел встроенную функцию поиска и подавления антивирусного программного обеспечения. Klez дописывал свой код к одному из документов на

зараженной машине и начинал массовую рассылку. В поле «От» подставлялся любой адрес, найденный на компьютере или же случайно сгенерированный. При этом список всех обнаруженных на зараженном компьютере адресов электронной почты также присоединялся к вложению. Кроме рассылки своих копий червь обнаруживал себя по 13-м числам четных месяцев или 6-м нечетных, в зависимости от модификации: в такой день все файлы на зараженных компьютерах заполнялись случайным содержимым.

Tanatos/Bugbear (2001 г.) – почтовый червь, устанавливающий бекдор-процедуру (Backdoor) и троян – клавиатурный шпион. Процедура распространения практически списана с Klez – копирование по сети, массовая рассылка с зараженным документом во вложении, использование уязвимости IFrame в Internet Explorer, подавление антивирусных программ. Кроме увеличения трафика вирус проявлял себя спонтанной печатью разнообразного мусора на сетевых принтерах.

В 2003 г. была зарегистрирована эпидемия интернет-червя Slammer, заражающего серверы под управлением Microsoft SQL Server 2000. Вирус использовал брешь в системе безопасности SQL Server, программная заплата к которой вышла в 2002 г. После проникновения червь начинает в бесконечном цикле посылать свой код на случайно выбранные адреса в сети – только за первые десять минут было поражено около 90 % (120 000 единиц) всех уязвимых серверов, при этом пять из тринадцати главных DNS-серверов сети Интернет вышли из строя.

Slammer имел крайне небольшой размер – всего 376 байтов (CodeRed – 4 кб, Nimda – 60 кб) и присутствовал только в памяти зараженных компьютеров. Более того, при работе червя никакие файлы не создавались, и червь никак не проявлял себя (помимо сетевой активности зараженного компьютера). Это означает, что лечение заключается только в перезагрузке сервера, а антивирусы в данной ситуации бессильны.

В конце 2003 г. около 8 млн компьютеров во всем мире оказались заражены интернет-червем Lovesan/Blaster. Для размножения использовалась очередная брешь – на этот раз в службе DCOM RPC Microsoft Windows. Кроме того, вирус включал в себя функ-

цию DDoS-атаки на сервер с обновлениями для Windows. В это же время новый вирус, Sobig.f, установил новый рекорд по скорости – доля зараженных им писем доходила до 10 % от всей корреспонденции. Это достигалось использованием спамерских технологий. Sobig.f также инициировал цепную реакцию: каждый новый вариант червя создавал сеть инфицированных компьютеров, которая позднее использовалась в качестве платформы для новой эпидемии. Однако конец эпидемии запрограммировал сам автор – 10 сентября 2003 г. Sobig.f прекратил размножение.

Swen (также известный как Gibe, 2003 г.) – пример удачного использования методов социальной инженерии. Этот вирус-червь распространялся по электронной почте в виде письма якобы от Microsoft Corporation Security Center и с темой «Internet Security Update». Во вложении находился файл с именем q216309.exe, а в самом сообщении говорилось о необходимости срочной установки вложенной заплатки.

Mitglieder (2004 г.) – троянский proxy-сервер, ссылка на зараженный этой программой сайт была разослана злоумышленником на тысячи адресов ICQ. Mitglieder проникал на компьютер через уязвимость в Microsoft Internet Explorer, позволявшую установить и запустить proxy-сервер на зараженной машине без ведома пользователя. После заражения открывался порт, используемый для рассылки спама. Таким образом, зараженные машины образовывали сеть машин-зомби (ботнет), которыми можно удаленно управлять, чем вскоре и воспользовались авторы новых вредоносных программ.

Bizex (также известный как Exploit, 2004 г.) – первый ICQ-червь. Для распространения использовалась массовая несанкционированная рассылка по ICQ сообщения «http://www.jokeworld.biz/index.html :)) LOL». Получив от знакомого человека такую ссылку, ничего не подозревающая жертва открывала указанную страницу, и в случае, если использовался браузер Internet Explorer с незакрытой уязвимостью, на компьютер загружались файлы червя и в некоторых случаях сопутствующего трояна. После установки в систему Bizex закрывал запущенный ICQ-клиент и подключившись к серверу ICQ с данными зараженного пользователя начинал рассылку по най-

денным на компьютере спискам контактов. Одновременно происходила кража конфиденциальной информации – различные логины и пароли.

Почтовый червь Bagle (2004 г.) для распространения использовал собственный SMTP-клиент, код вируса пересылался во вложении с произвольным именем и расширением .exe. Рассылка производилась на адреса, найденные на зараженном компьютере. Также Bagle содержал встроенную backdoor-процедуру, открывающую порт 6777 на запуск команд и загрузку любых файлов. Следующие модификации содержали процедуры распространения через P2P-сети, методы социальной инженерии и активно противодействовали антивирусному программному обеспечению.

Mydoom (2004 г.) известен прежде всего массированной DDoS-атакой на веб-сайт компании SCO. За пару часов работа сервера была полностью парализована и вернуться в нормальный режим сервер смог только через неделю. Для распространения Mydoom использовал почтовую рассылку через собственный SMTP-клиент, а также P2P-сети (Kazaa).

Модификация почтового червя NetSky (также известен как Moodown) для распространения кроме электронной почты задействовала P2P и локальные сети. Вторая модификация NetSky отличилась тем, что в силу человеческого фактора ею были заражены тысячи писем, отправленных известным финским производителем антивирусного программного обеспечения – компанией F-Secure – своим клиентам. Червь Sasser (2004 г.) для проникновения использовал уязвимость в службе LSASS Microsoft Windows.

Cabir (2004 г.) – первый сетевой червь, распространяющийся через протокол Bluetooth и заражающий мобильные телефоны, работающие под управлением OS Symbian. При каждом включении зараженного телефона вирус получал управление и начинал сканировать список активных Bluetooth-соединений. Затем выбирал первое доступное соединение и пытался передать туда свой основной файл caribe.sis. Ничего деструктивного Cabir не делал – только снижал стабильность работы телефона за счет постоянных попыток сканирования активных Bluetooth-устройств.

В 2005 г. наметился некоторый спад активности почтовых червей. Фактически после Mydoom, NetSky и Bagle до настоящего времени не наблюдалось ни одной сколько-нибудь значительной эпидемии. Это может свидетельствовать об успешности новых антивирусных технологий.

Поэтому пальму первенства перехватили сетевые черви и программы, использующие для распространения различные интернет-пейджеры (прежде всего MSN Messenger). Первые активно используют бреши в операционных системах Microsoft Windows – чаще всего это уязвимости в службах RPC DCOM и LSASS, а также дыры, оставленные прошедшими ранее эпидемиями: это позволяет создавать ботнеты, включающие тысячи компьютеров-зомби. Вторые пользуются некомпетентностью и отсутствием опыта создателей программ обмена сообщениями в упреждении вирусных инцидентов. Однако глобальных эпидемий зафиксировано не было. Это не означает уменьшение числа вирусов, наоборот, с каждым днем их появляется все больше. Но при этом можно отметить увеличение избирательности вредоносных программ – становятся популярны черви, главной целью которых является похищение определенной информации. Кроме уже ставших привычными краж номеров кредитных карт, участились случаи воровства персональных данных игроков различных онлайновых игр – Ultima Online, Legend of Mir, Lineage, Gamania. В России также зафиксированы случаи с игрой «Бойцовский клуб», где реальная стоимость некоторых предметов на аукционах достигает тысяч долларов США.

Развитие получили и вирусные технологии для мобильных устройств. Созданы вирусы всех трех типов – классические вирусы, черви и троянские программы. В качестве пути проникновения используются не только Bluetooth-устройства, но и обычные MMS-сообщения (червь ComWar).

В Интернете в среднем каждое тридцатое письмо заражено почтовым червем, около 70 % всей корреспонденции – нежелательна. С ростом сети Интернет увеличивается количество потенциальных жертв вирусных атак.

RedBrowser (2006 г.) – первая вредоносная программы для мобильных телефонов, способных исполнять Java-приложения (J2ME).

Потенциально этот троянец представляет опасность не только для смартфонов, но и для всех мобильных телефонов, поддерживающих платформу Java. Осуществляемые троянцем вредоносные действия – рассылка SMS-сообщений на платные мобильные сервисы. Появились «концептуальные» вирусы: Leap – первый червь для MacOS X. Червь рассылает себя по пейджинговым интернет-сетям и заражает файлы MacOS X.

Начиная с 2007 г. по настоящее время никаких сверхновых в идейном плане вредоносных программ обнаружено не было. По статистике Лаборатории Касперского к 2009 г. было зафиксировано 38190 уникальных вредоносных программ.

За последнее время были разработаны эффективные методики построения комплексов антивирусной безопасности, позволяющие снизить вероятность заражения компьютерной сети каким-либо вирусом или другой вредоносной программой практически до нуля. Однако стопроцентную безопасность не может гарантировать никто. Поэтому для сведения возможности заражения к минимуму необходимо четко понимать принципы работы антивирусного программного обеспечения и неукоснительно следовать всем правилам и рекомендациям.

Контрольные вопросы и задания

1. Рассмотрите перспективы создания вирусов для мобильных устройств на базе WindowsMobile, iPod/iPhone.
2. Каким образом можно выявить факт заражения компьютера вирусом и тип вируса, не используя анализаторы антивирусных программных комплексов?
3. Проанализируйте актуальность угрозы заражения вирусами в среде Internet-браузеров.
4. Опишите предполагаемый метод распространения вирусов через популярные социальные сети. Продумайте механизм защиты.
5. Проанализируйте способы применения технологий «сетевых червей» для целей, отличных от вандализма и атак на доступность ресурсов.

Глава 2. ОСНОВЫ ТЕОРИИ ВРЕДОНОСНЫХ ПРОГРАММ

Когда говорят о вредоносных программах (компьютерных вирусах), всегда подразумевают некий класс программ, обладающих определенными свойствами. Кроме того, для каждого нового вируса существует некая процедура, согласно которой антивирусные эксперты решают – вирус это или нет, прежде чем вносить его сигнатуру в вирусную базу.

В связи с этим возникает вопрос о возможности формализовать эту процедуру выяснения – относится ли программа к классу вирусов. Этой проблемой занимались многие исследователи еще с первой половины 80-х гг. XX века. И самым первым из них был Фред Коэн.

2.1. Общие положения

Результат Фреда Коэна.

В 1984 г. в своей работе "Computer Viruses – Theory and Experiments" Фред Коэн показал, что задача обнаружения компьютерных вирусов является неразрешимой. При этом он пользовался следующим определением компьютерного вируса:

Определение 2.1. Компьютерным вирусом является программа, способная заражать другие программы, изменяя их таким образом, чтобы они включали возможно измененную копию вируса.

Для того чтобы определить, является ли программа *P* вирусом, необходимо выяснить, модифицирует ли она согласно определению другие программы. Предположим, существует процедура *D*, возвращающая логическую единицу, если *P* является вирусом, и логический ноль в противном случае. Тогда можно составить программу *V*, которая будет включать такую процедуру обнаружения, заражающую другие программы только в случае, если эта процедура обнаружения не будет определять данную программу как вирус.

Соответственно если процедура *D*, будучи примененной к программе *V*, скажет, что это вирус, программа *V* на самом деле не бу-

дет заражать файлы и не будет вирусом согласно определению. И наоборот, если согласно процедуре D программа V не является вирусом, на самом деле она будет заражать другие программы, а значит будет вирусом.

Таким образом, предполагаемая процедура обнаружения D оказывается противоречивой и, следовательно, не существует.

Результат Д. Чесса и С. Вайта.

В работе "An Undetectable Computer Virus" Дэвид Чесс и Стив Вайт попытались развить идеи Коэна и показать, что задача обнаружения вирусов остается неразрешимой и при более слабых допущениях. Этой работой они попытались охватить класс полиморфных вирусов.

Рассмотрим множество программ. Множеством значений каждой из программ является одна или несколько других программ. При каждом выполнении программа производит одну программу из своего множества значений (в зависимости от входных данных). Пусть при этом выполнялась программа p, а результатом ее работы стала программа q. В этом случае будем говорить, что p производит q. Будем также говорить, что p со временем производит q, если p производит q либо непосредственно, либо после конечного числа итераций. Отношение "со временем производит" является транзитивным замыканием отношения "производит".

Определение 2.2. Множество V называется вирусным, если это максимальное множество, удовлетворяющее условию:

$(\forall\ p, q \in V)$[p со временем производит q, и q со временем производит p].

Максимальность понимается в том смысле, что не существует такой программы $r \in V$, чтобы при ее добавлении к множеству V все еще выполнялось указанное условие.

Множество V отождествляется с компьютерным вирусом. Программа p будет называться экземпляром вируса V, если $p \in V$. Программа p будет называться просто зараженной, если: (V – вирус) [$p \in V$].

Будем говорить, что программа p размножается, если она производит еще один экземпляр вируса.

Простейший вирус представляет собой множество из одного элемента и всегда производит сам себя. Более сложные множества соответствуют полиморфным вирусам, которые могут иметь несколько отличающихся форм, каждая из которых в процессе эволюции способна со временем произвести все остальные.

Предложенный формализм соответствует либо перезаписывающим вирусам, которые заменяют собой другие программы, принимая их имена, либо некоторым видам сетевых червей. Для описания вирусов в классическом определении Коэна необходимо усложнять модель. Однако основные результаты, которые будут получены ниже, могут быть распространены и на классические вирусы.

Обнаружение вирусов.

Определение 2.3. Алгоритм A обнаруживает вирус V тогда и только тогда, когда для любой программы p: $A(p)$ завершает работу и печатает 1 тогда и только тогда, когда $p \in V$.

Аналогично, алгоритм A обнаруживает множество вирусов S тогда и только тогда, когда для любой программы p: $A(p)$ завершает работу и печатает 1 тогда и только тогда, когда $p \in V \in S$.

Полученный Ф. Коэном результат о том, что не существует алгоритма A, способного обнаруживать множество всех возможных компьютерных вирусов, можно расширить и вывести, что, даже имея один из экземпляров вируса V, нельзя создать алгоритм, способный обнаруживать все экземпляры вируса V.

Рассмотрим такой полиморфный вирус, что для любого реализуемого алгоритма X будет существовать программа p типа «если $X(p)$, то прекратить работу, иначе размножаться», которая будет являться экземпляром этого вируса (при условии, конечно, что такая программа вообще способна размножаться).

Очевидно, не существует алгоритма B, который был бы способен обнаруживать все экземпляры такого вируса, поскольку для любого алгоритма B, претендующего на роль детектора, существует программа q типа «если $B(q)$, то прекратить работу, иначе размножаться», для которой алгоритм B будет возвращать неверный результат. Действительно, если $B(q)$ возвращает 1, значит q никогда не размножается и не является экземпляром описанного или любо-

го другого вируса. Если же $B(q)$ возвращает 0, тогда q в действительности размножается и является экземпляром вируса.

Возникает вопрос о существовании подобного полиморфного вируса. Рассмотрим следующий вирус W, одним из экземпляров которого является r:

```
если Sub1(r), то завершить работу, иначе {
  заменить текст подпрограммы Sub1 текстом произвольной программы;
  размножаться;
  завершить работу;
}
Sub1:
  Вернуть 0;
```

Для любого алгоритма C, претендующего на обнаружение всех экземпляров вируса W, найдется программа s:

```
если Sub1(s), то завершить работу, иначе {
  заменить текст подпрограммы Sub1 текстом произвольной программы;
  размножаться;
  завершить работу;
}
Sub1:
  Вернуть C(аргумент);
```

для которой алгоритм C возвращает ошибочный результат. Если $C(s)$ возвращает 1, значит s всегда просто завершает свою работу и не является экземпляром вируса W. Если же $C(s)$ возвращает 0, то s является экземпляром W. Соответственно не существует алгоритма C, способного безошибочно определять все экземпляры вируса W и только их.

Слабое определение обнаружения. Можно в некотором роде ослабить определение обнаружения, сохранив полученный выше результат в силе. В частности, можно допустить, чтобы алгоритм, претендующий на обнаружение вируса V, возвращал 1 не только на экземплярах этого вируса, но и на некоторых других программах при условии, что они тоже являются экземплярами (других) вирусов.

Определение 2.4. Алгоритм A слабо обнаруживает вирус V тогда и только тогда, когда для любой программы p $A(p)$ завершает

работу и возвращает 1, если $p \in V$, и возвращает отличный от 1 результат, если p не является экземпляром никакого вируса. При этом на экземплярах других вирусов алгоритм может возвращать любой результат.

Вполне очевидно, что для сконструированного вируса W не существует алгоритма, который мог бы слабо обнаруживать этот вирус, поскольку такой алгоритм будет либо возвращать 1 на программе, которая просто завершает свою работу, либо будет возвращать не 1 для программы, являющейся экземпляром W.

Практические следствия. На практике вирус W не представляет такой уж большой опасности и это связано с тем, что, как правило, не расценивается проблемой ложное срабатывание на программах вида:

```
если D(t), то завершить работу, иначе размножаться
```

или

```
если Sub1(u), то завершить работу, иначе {
  заменить текст подпрограммы Sub1 текстом произ-
вольной программы;
  размножаться;
  завершить работу;
}
Sub1:
  вернуть D(аргумент);
```

Несмотря на то что такие программы никогда не размножаются, их существование тесно связано с существованием определенных вирусов.

Здесь важно понимать значение полученного результата. Вывод Ф. Коэна важен в первую очередь тем, что можно считать ложными любые заявления о создании алгоритма, безошибочно детектирующего вирусы. Соответственно на основании результата Д. Чесса и С. Вайта можно считать ложными заявления о создании алгоритма, способного обнаруживать без ошибок все экземпляры полиморфного вируса по одному его экземпляру.

В реальной жизни антивирусная программа считается качественной, если она обнаруживает все жизнеспособные экземпляры вируса, желательно также его нежизнеспособные ответвления, и при

этом характеризуется сравнительно небольшим числом ложных срабатываний. Следует избегать лишь тех ложных срабатываний, которые затрагивают существующие программы, используемые в повседневной работе различными пользователями. Ложные срабатывания на искусственных примерах, появление которых на компьютерах пользователей близко к нулю, вполне допустимы.

Формализм Ф. Лейтольда.

В 2000 г. венгерский исследователь Ференц Лейтольд попытался дать новый импульс развитию математической теории компьютерных вирусов путем более точного моделирования вычислительной среды – компьютеров и операционных систем. Попутно он вывел некую общую классификацию вирусов и так же, как Д. Чесс и С. Вайт, рассмотрел в рамках предложенного формализма феномен полиморфных вирусов.

2.2. Вычислительные модели

Вычислительная машина с произвольной выборкой.

Для моделирования компьютерной среды Ф. Лейтольд отталкивался от вычислительной машины с произвольной выборкой (Random Access Machine, RAM), которая состоит из следующих компонентов:

- входная лента, представляющая собой последовательность ячеек, содержащих целые числа и доступных только для последовательного чтения – после чтения ячейки считывающая головка автоматически перемещается вправо;

- выходная лента, представляющая собой последовательность ячеек, доступных только для последовательной записи, по умолчанию эти ячейки пусты;

- программа – последовательность (возможно индексированная, т.е. с возможностью указать адрес некоторых или любых инструкций) инструкций, не находящаяся в памяти и соответственно неизменяемая, по умолчанию выполнение программы начинается с первой инструкции (можно ввести счетчик инструкций, указывающий на следующую к выполнению инструкцию, в этом случае по умолчанию счетчик указывает на первую инструкцию);

- память – неограниченная последовательность регистров r_0, r_1, …, r_i, …, способных хранить целые числа, по умолчанию значения регистров пусты, нулевой регистр r_0 используется для вычислений и часто называется аккумулятором, доступ к регистрам производится непосредственно по их индексам, откуда и происходит название машины.

Из сказанного выше следует, что после того как ячейка была прочитана или записана, она не может быть прочитана или записана еще раз. Неизменяемость программы означает в том числе и то, что программа не может менять себя в процессе выполнения. Один из возможных вариантов инструкций для вычислительной машины с произвольной выборкой представлен в табл. 2.1.

Таблица 2.1

Инструкция	Параметр	Описание
LOAD	Операнд	Загружает в память значение, определяемое операндом
STORE	Операнд	Копирует значение аккумулятора в ячейку памяти (регистр), определяемую операндом
ADD	Операнд	Прибавляет к аккумулятору значение, определяемое операндом
SUB	Операнд	Вычитает из аккумулятора значение, определяемое операндом
MULT	Операнд	Умножает аккумулятор на значение, определяемое операндом
DIV	Операнд	Делит аккумулятор на значение, определяемое операндом
READ	Операнд	Считывает значение с входящей ленты в регистр, определяемый операндом
WRITE	Операнд	Записывает на выходную ленту значение, определяемое операндом
JUMP	Метка	Присваивает счетчику инструкций значение метки
JGTZ	Метка	Присваивает счетчику инструкций значение метки, если аккумулятор содержит положительное число
JZERO	Метка	Присваивает счетчику инструкций значение метки, если аккумулятор равен нулю
HALT	–	Завершает работу машины

Допускается три вида операндов:

- i – обозначает собственно значение целого числа i;

- $[i]$ – для неотрицательных i обозначает значение регистра r_i;

- $[[i]]$ – косвенная адресация, обозначает значение регистра r_j, где j – значение регистра r_i. Если $j < 0$, машина завершает работу.

По умолчанию значение всех регистров равно нулю, счетчик инструкций указывает на первую инструкцию программы и выходная лента пуста. После выполнения k-й инструкции счетчик либо автоматически увеличивается на единицу и указывает на $(k + 1)$-ю инструкцию, либо, если была выполнена инструкция JUMP или выполнены условия инструкций JGTZ или JZERO, принимает значение метки, либо, если была выполнена инструкция HALT, машина прекращает свою работу.

Вычислительная машина с хранением программ в памяти с произвольной выборкой.

Развитием формализма вычислительной машины с произвольной выборкой является вычислительная машина с хранением программ в памяти с произвольной выборкой (Random Access Stored Program Machine, RASPM), отличие которой состоит только в том, что программа сохраняется в памяти, а значит может изменять сама себя в процессе выполнения.

Набор инструкций для RASPM в общем случае может не отличаться от набора инструкций RAM. Впрочем, некоторые инструкции могут быть упрощены. Так, вместо непрямой адресации можно использовать возможность модификации инструкций программы для получения того же эффекта.

Машина Тьюринга.

Большинство теоретических результатов относительно вычислительной способности различных автоматов (к которым относятся RAM и RASPM) получено для Машины Тьюринга (МТ). Следовательно, для применения к RASPM уже известных результатов необходимо установить отношение между этими формальными системами.

Определение 2.5. МТ с одной лентой может быть определена как совокупность семи элементов:

$$T = \langle Q, S, I, d, b, q_0, q_j \rangle, \qquad (2.1)$$

где Q – множество состояний МТ;

S – множество символов, которые могут быть записаны на ленте;

I – множество символов входящей последовательности, $I \in S$; $b \in S \mid I$ – пустой символ;

q_0 , q_f – начальное и конечное состояния МТ;

$d : Q \times S \rightarrow Q \times S \times \times \{l, r, s\}$ – множество функций перехода, которые состоянию и символу ленты ставят в соответствие новое состояние, новый символ и перемещение по ленте: на шаг влево, на шаг вправо, остаться на месте.

МТ начинает свою работу в состоянии q_0 и в дальнейшем меняет состояния согласно функциям перехода в зависимости от текущего состояния и значения ячейки ленты. Попутно ячейки ленты перезаписываются новыми значениями согласно тем же функциям перехода.

МТ может содержать и большее количество лент, но вычислительная способность такой машины будет находиться в полиномиальной зависимости от вычислительной способности МТ с одной лентой. Помимо этого имеет место следующая теорема:

Теорема 2.1. Вычислительные способности МТ и RASPM эквивалентны, а их затраты на вычисление полиномиально сравнимы, если стоимость выполнения инструкции является величиной постоянной или находится в логарифмической зависимости от длины операнда.

Кроме того, доказан фундаментальный результат.

Теорема 2.2. Не существует такой МТ, которая сможет определить, остановится ли произвольная МТ на произвольном входе (проблема остановки).

RASPM с присоединенным вспомогательным хранилищем.

Дальнейшее приближение модели к реальным компьютерам и операционным системам основывается на идее присоединенных внешних хранилищ (данных и программ).

Описанные в предыдущем пункте модели ограничены тем, что в чистом виде пригодны только для анализа отдельного алгоритма или программы. Для анализа взаимодействия алгоритмов потребовались бы значительные усилия. Чтобы упростить анализ та-

ких ситуаций, имеет смысл ввести в модель вычислительной машины специальную область или ленту, на которой будут храниться данные других программ. Назовем эту область присоединенным вспомогательным хранилищем (Attached Background Storage, ABS). Кроме этого имеет смысл потребовать, чтобы любая выполняющаяся программа имела полный доступ (чтение и запись) к этому хранилищу.

Определение 2.6. Вычислительная машина G с хранением программ в памяти с произвольной выборкой и с присоединенным вспомогательным хранилищем (RASPM с ABS) определяется шестью элементами:

$$T = \langle V, U, T, f, q, M_j \rangle, \qquad (2.2)$$

где V – алфавит, состоящий из входных и выходных символов, а также символов, которые могут быть записаны на присоединенном вспомогательном устройстве и в ячейках памяти (регистрах);
U – непустое конечное подмножество кодов инструкций;
T – непустое множество возможных действий процессора;
f – однозначная функция $f : U \rightarrow T$, ставящая в соответствие различным кодам инструкций различные действия процессора (действие процессора $f(x)$, соответствующее коду инструкции $x \in U$, будет называться командой);
q – стартовое значение счетчика операций;
M – стартовое наполнение памяти.

Без потери общности можно допустить, что существует взаимно однозначное кодирование символов алфавита V целыми числами. При этом каждая инструкция должна сопровождаться значением операнда, таким образом, каждая команда задается двумя ячейками: код инструкции в одной ячейке и значение операнда в следующей ячейке. Один из вариантов кодирования инструкций представлен в табл. 2.2.

Таблица 2.2

Инструкция	Параметр	Код	Значение
LOAD	Операнд	10	Загружает в аккумулятор значение, определяемое операндом

Окончание табл. 2.2

Инструкция	Параметр	Код	Значение
STORE	Операнд	20	Копирует значение аккумулятора в ячейку памяти, определяемую операндом
ADD	Операнд	30	Прибавляет к аккумулятору значение, определяемое операндом
SUB	Операнд	40	Вычитает из аккумулятора значение, определяемое операндом
MULT	Операнд	50	Умножает аккумулятор на значение, определяемое операндом
DIV	Операнд	60	Делит аккумулятор на значение, определяемое операндом
AND	Операнд	70	Выполняет побитовую операцию "И" между аккумулятором и значением, определяемым операндом
OR	Операнд	80	Выполняет побитовую операцию "ИЛИ" между аккумулятором и значением, определяемым операндом
XOR	Операнд	90	Выполняет побитовую операцию "исключающее ИЛИ" между аккумулятором и значением, определяемым операндом
READ	Операнд	A0	Считывает значение с входной ленты в ячейку, определяемую операндом
WRITE	Операнд	B0	Записывает на выходную ленту значение ячейки, определяемой операндом
GET	Операнд	C0	Считывает значение с вспомогательного хранилища в ячейку, определяемую операндом
PUT	Операнд	D0	Записывает на вспомогательное хранилище значение ячейки, определяемой операндом
SEEK	Операнд	E0	Перемещает считывающую/записывающую головку вспомогательного хранилища в позицию, определяемую операндом
JUMP	Метка	FC	Присваивает счетчику инструкций значение метки
JGTZ	Метка	FD	Присваивает счетчику инструкций значение метки, если аккумулятор содержит положительное число
JZERO	Метка	FE	Присваивает счетчику инструкций значение метки, если аккумулятор равен нулю

Обозначим значение *i*-й ячейки памяти как *c(i)*, где *i* – целое число. Допустимые коды операндов в таком случае представлены в табл. 2.3.

Таблица 2.3

Операнд	Код операнда	Значение
I	1	*i*
[*i*]	2	*c(i)*
[[*i*]]	3	*c(c(i))*

Полный код команды в этом случае будет складываться (в буквальном смысле) из кода инструкции и кода операнда. Например, команда ADD [*i*] будет иметь код 32, а команда GET [[*i*]] – код C3.

Поскольку программа в случае RASPM способна изменять себя в процессе выполнения, многие команды, в частности команды с операндом [[*i*]], могут быть эмулированы через другие команды.

Также нужно понимать, что не любой операнд подходит к каждой инструкции. Например, инструкция READ может иметь только операнды типов [*i*] и [[*i*]], поскольку записывает значение из ленты в память.

При включении RASPM с ABS счетчик инструкций принимает начальное значение *q* и процессор немедленно выполняет команду, расположенную по адресу, указанному в *q*. Дальнейшее выполнение программы определяется командами, записанными в памяти, и, таким образом, полностью определяется начальным содержанием памяти. Завершение работы машины происходит в следующих случаях:

- когда машина выключается пользователем;
- когда счетчик указывает на ячейку памяти, содержимое которой не является кодом команды ($x \in V$, но $x \notin U$);
- когда производится попытка выполнить деление на ноль.

Таким образом, в отличие от RAM специальная команда для завершения работы машины не используется.

Моделирование операционных систем.

Возникает естественное желание – использовать RASPM с ABS для запуска программ. Компоненты *V*, *U*, *T* и *f* машины *G* были определены ранее. Теперь, задав определенным образом *M* и *q*,

можно получить программу, которая и будет определять характер функционирования машины. Разумно потребовать, чтобы файлы программ и данных хранились на вспомогательном хранилище, порядок выполнения программ определялся входной лентой и чтобы выполняемая программа могла модифицировать как данные, так и программы на вспомогательном хранилище. Соответственно необходима программа-оболочка, способная работать с файлами данных и программ и выполнять другие программы.

Определение 2.7. Под операционной системой (ОС) понимается система программ, способная работать с файлами данных и программ и выполнять другие программы.

ОС может быть частью начального состояния памяти *M* или же может быть расположена на вспомогательном хранилище. Во втором случае начальное состояние памяти должно содержать специальную программу, которая будет загружать операционную систему и запускать ее. Такая служебная программа не будет считаться частью операционной системы.

Определение 2.8. Если ОС входит в состав начального состояния памяти *M* машины, то такая ОС будет называться машинозависимой.

Это означает, что, определив машину RASPM с ABS, мы определим также и ОС, так как *M* является частью определения машины.

Определение 2.9. Если ОС расположена на вспомогательном хранилище, такая ОС будет называться машиннонезависимой.

В таком случае ОС можно будет поменять вместе с вспомогательным хранилищем.

Определение 2.10. Если начальное состояние памяти *M* машины RASPM с ABS включает ОС, то такая машина будет называться ОС-зависимой машиной.

То есть машина может использовать только свою собственную ОС, хотя возможно создание программы, которая будет эмулировать какую-нибудь другую ОС.

Определение 2.11. Если ОС машины RASPM с ABS расположена на вспомогательном хранилище, то такая машина будет называться ОС-независимой машиной.

Из определений непосредственно следуют следующие теоремы.

Теорема 2.3. Если *O* – машинозависимая ОС машины *G*, то *G* – ОС-зависимая машина.

Теорема 2.4. Если *O* – машиннонезависимая ОС машины *G*, то *G* – ОС-независимая машина.

Для исследования общей задачи модификации одними программами других программ годится ОС любого типа. Если же поставить задачу исследования также и программ, способных изменять код ОС, необходимо использовать модель с машиннонезависимыми ОС.

2.3. Вирусы в RASPM с ABS

С учетом данного ранее определения ОС можно сформулировать определение вируса.

Определение 2.12. Компьютерный вирус определяется как часть программы, присоединенная к области программы и способная присоединять себя к областям других программ. Компьютерный вирус выполняется при выполнении программы, к области которой он присоединен.

В общем случае присоединение вируса к области программы не должно оставлять неизменным результат выполнения программы (без учета действий вируса), однако, такой способ присоединения часто используется вирусами для маскировки своего присутствия в системе (в области программы).

Способы размножения вирусов.

Как известно из практики, вирус может присоединяться к программе, используя различные технологии. Формы присоединения вирусов к различным программным областям будут называться способами размножения. Один и тот же вирус может обладать несколькими способами размножения.

Определение 2.13. Способ размножения вируса называется машинозависимым, если для использования этого способа вирусу требуется машина, обладающая определенными характеристиками или свойствами. В противном случае, когда характеристики и свой-

ства машины не влияют на размножение вируса определенным способом, способ размножения называется машиннонезависимым.

Аналогичным образом можно рассмотреть и зависимость способов размножения от ОС.

Определение 2.14. Способ размножения вируса называется ОС-зависимым, если для использования этого способа вирусу требуется ОС, обладающая определенными характеристиками или свойствами. В противном случае, когда характеристики и свойства ОС не влияют на размножение вируса определенным способом, способ размножения называется ОС-независимым.

Рассматривая совокупность способов размножения, можно распространить понятия машинозависимости и ОС-зависимости на вирусы.

Определение 2.15. Вирус называется машинозависимым, если все его способы размножения машинозависимы. Аналогично, вирус называется машиннонезависимым, если все его способы размножения машиннонезависимы.

Определение 2.16. Вирус называется ОС-зависимым, если все его способы размножения ОС-зависимы. Аналогично, вирус называется ОС-независимым, если все его способы размножения ОС-независимы.

В некоторых случаях вирусы могут присоединять себя не к программной области, а к файлу данных. Например, к исходному коду какой-либо программы. В этом случае для запуска вируса требуется, чтобы файл данных был преобразован третьей программой (компилятором) в файл программы. Дадим определение и такого способа размножения.

Определение 2.17. Способ размножения называется непосредственным, если вирус присоединяет себя непосредственно к программной области. Способ размножения называется косвенным, если вирус присоединяет себя к файлу данных.

Олигоморфные и полиморфные вирусы.

Везде ранее подразумевалось, что вирус никак не меняется при размножении, т.е. присоединенная вирусная часть одинакова при каждом заражении. Тем не менее многие вирусы не обладают таким свойством и способны меняться при размножении.

Определение 2.18. Способ распространения называется полиморфным, если можно найти две программы, зараженные с использованием одного способа распространения одного вируса, такие, что последовательности инструкций зараженных частей этих программ будут отличаться.

Определение 2.19. Вирус называется полиморфным, если он обладает полиморфным способом распространения.

Определение 2.20. Способ распространения называется олигоморфным, если можно найти две программы, зараженные с использованием одного способа распространения одного вируса, такие, что последовательности инструкций зараженных частей этих программ будут одинаковы (для любой пары), но хотя бы одна часть вирусного кода будет зашифрована с разными ключами.

Определение 2.21. Вирус называется олигоморфным, если он обладает олигоморфным способом распространения.

На практике выделяют также подвиды полиморфных и олигоморфных вирусов.

Определение 2.22. Вирус называется медленным полиморфным, если он обладает полиморфным способом распространения, но применяет этот способ очень редко.

Определение 2.23. Вирус называется медленным олигоморфным, если он обладает олигоморфным способом распространения, но ключ шифрования меняет очень редко.

Проблема обнаружения вирусов.

Вместе с выделением класса вирусов возникает и проблема обнаружения вирусов.

Определение 2.24. Проблема обнаружения вирусов является задачей теории алгоритмов: существует ли алгоритм, с помощью которого для любой программы можно было бы определить, содержит она вирус, способный к распространению, или нет.

Предполагается, что все данные о структуре программы и машины, на которой она выполняется, доступны. Неизвестными являются только данные о вирусе.

Общая проблема. Согласно тезису Тьюринга – Черча, если бы существовал алгоритм обнаружения вирусов, можно было бы соз-

дать МТ, которая бы выполняла этот алгоритм. Покажем, что такой МТ не существует.

Теорема 2.5. Не существует МТ, которая могла бы определять наличие вируса в произвольной программе для машины RASPM с ABS.

Доказательство. Доказано, что вместо МТ можно использовать эквивалентную ей RASPM или RASPM с ABS. Рассмотрим программу *P*, которая эмулирует работу МТ (произвольной). Программа *P* печатает на выходе 1, если эмулируемая МТ завершает свою работу.

Построим простой вирус, который сперва запускает программу *P* на случайном, но фиксированном (для данного вируса) входе *B*, после чего начинает выполнять собственно вирусную часть. При заражении вирус копирует целиком программу *P*, последовательность *B* и вирусную часть.

Используя эту конструкцию можно создать программу *V* в RASPM с ABS для любой МТ, и эта программа будет вирусом в том случае, если она сможет размножаться, т.е. в том случае, когда МТ завершает свою работу на фиксированном для программы *V* входе или, что тоже самое, если программа *P* печатает единицу для данной МТ и данного входа.

Предположим, что существует МТ, способная обнаруживать все вирусы, т.е. такая Машина Тьюринга *T*, которая читает код программы RASPM с ABS и печатает 1, если в коде этой программы содержится вирус, и печатает 0, если вируса нет. Применим машину *T* к программе *V*. Если *T* печатает 1, значит программа *P* и соответствующая ей МТ завершают свою работу на некотором входе *B*. Если *T* печатает 0, значит программа *P* и соответствующая ей МТ никогда не завершит свою работу на некотором входе *B*. Учитывая, что вход *B* и эмулируемая программой *P* Машина Тьюринга могут быть любыми, получаем, что Машина Тьюринга *T* способна для любой МТ и любого входа определить, завершит ли данная МТ работу на данном входе. Однако мы знаем, что это невозможно.

Полученное противоречие доказывает теорему.

Оценка сложности сигнатурного метода. Ф. Лейтольд в своих работах обращается к проблеме обнаружения не всех возможных вирусов, а некоторого подмножества "известных" вирусов.

Для этого предлагается сигнатурный метод – обнаружение вируса по строке или подстроке его кода. Естественно, таким образом невозможно обнаружить полиморфные вирусы. Но для обнаружения обычных или олигоморфных вирусов такой способ годится: олигоморфные вирусы могут обнаруживаться по строке или подстроке кода расшифровщика.

В случаях, когда обнаружение осуществляется по подстроке вирусного кода (либо даже по полному коду распаковщика для олигоморфных вирусов) возможны ложные срабатывания. Ф. Лейтольд приводит оценку вероятности ложных обнаружений при следующих допущениях:

- длина сигнатуры – *N;*
- общая длина анализируемых данных – $L >> N;$
- количество символов в алфавите – *n*, и встречаются они в анализируемых данных с равной вероятностью;
- количество известных вирусов *M*.

В этом случае оценка количества ложных обнаружений будет примерно равна:

$$p \approx LM \frac{1}{n^N}. \qquad (2.3)$$

Можно также подсчитать вычислительную сложность алгоритма, выполняющего поиск вирусов по сигнатуре. Для выполнения поиска требуется последовательно сравнить все ячейки анализируемой строки данных с первыми ячейками сигнатур. В общем случае это ($L \cdot M$) сравнений. Далее при обнаружении совпадения потребуется провести сравнение со второй ячейкой сигнатуры. Учитывая вероятность совпадения значения первой ячейки, количество сравнений второй ячейки будет $LM\frac{1}{n}$. Аналогично, третьей – $LM\frac{1}{n^N}$ и т. д. Общее количество сравнений составит

$$s = LM\left(1+\frac{1}{n}+\frac{1}{n^2}+\ldots+\frac{1}{n^{N-1}}\right) = LM\frac{\frac{1}{n^N}-1}{\frac{1}{n}-1}. \qquad (2.4)$$

В худшем сценарии, когда все сигнатуры полностью различны, а n и N достаточно велики, количество сравнений составит

$$S = LMN. \quad (2.5)$$

Следовательно, сигнатурный поиск может быть выполнен за полиномиальное время.

Необходимо понимать, что полученные оценки приблизительны и соответствуют худшему варианту поиска случайной подпоследовательности в случайной последовательности. На практике код программы и код вируса не являются случайными последовательностями и с учетом знания их структуры алгоритм поиска может быть существенно оптимизирован, оценка времени – уменьшена.

Результат Леонарда Адельмана. Леонард Адельман, известный математик, обладатель премии Тьюринга, участвовавший в свое время в разработке криптосистемы RSA (буква А в аббревиатуре), для исследования феномена вирусов решил использовать другой формализм теории алгоритмов – не формализм Машин Тьюринга, а формализм рекурсивных функций.

Теория Машин Тьюринга и теория рекурсивных функций в некотором смысле эквиваленты – алгоритмы, вычислимые в одной теории, будут вычислимыми в другой, и наоборот. Этот факт в совокупности с тезисом Тьюринга – Черча означает, что полученные Л. Адельманом результаты в той же степени применимы к современным реалиям, что и результаты Ф. Коэна, Д. Чесса, С. Вайта, Ф. Лейтольда и других исследователей, опиравшихся в своих работах на теорию Машин Тьюринга.

Принятые обозначения. В дальнейшем изложении будет считаться, что данные и программы, во-первых, различимы, как это обычно и бывает в компьютерных системах, а во-вторых, могут быть закодированы натуральными числами. Поскольку и данные, и программы всегда имеют ограниченный размер и ограниченный алфавит для их представления, принципиальная возможность взаимно однозначного кодирования натуральными числами сомнений не вызывает.

Исходя из сказанного, состояние некой вычислительной системы с точки зрения выполняющейся программы может быть описано как набор доступных программе данных и других программ,

что можно выразить как наборы или последовательности натуральных чисел. Действие любой программы заключается в том, чтобы из одного набора данных и программ получить другой набор данных и программ.

Ниже везде неявно подразумевается описанная интерпретация вычислительной системы.

1. Обозначим S – множество всех конечных последовательностей натуральных чисел, $S = \Omega(\infty)$. Таким образом, элемент S может обозначать либо набор файлов данных, либо набор файлов программ, закодированных при помощи натуральных чисел.

2. Обозначим e – вычислимую инъективную функцию из $S \times S$ на ∞ с вычислимой обратной функцией, $e : S \times S \to \infty$. По тому же принципу, что и программы и данные, натуральными числами могут кодироваться и состояния системы. Функция e взаимно однозначно ставит в соответствие набору данных и программ натуральное число. Иными словами e – однозначная нумерация состояний системы.

3. $\forall s,t \in S : \langle s,t \rangle := e(s,t)$. Нотация $\langle s,t \rangle$ предназначена для обозначения состояния системы и фактически является кодом этого состояния, представленным натуральным числом.

4. Для всех частичных функций $f : \infty \to \infty, \forall s,t \in S : f(s,t) := f(\langle s,t \rangle)$. Любая программа переводит систему из одного состояния в другое, учитывая, что состояния кодируются натуральными числами, можно считать любую программу функцией из ∞ в ∞.

5. Для всех частичных функций $f : \infty \to \infty, \forall n \in \infty$ примем обозначение $f(n)\uparrow$ тогда и только тогда, когда $f(n)$ определена.

6. Для всех частичных функций $f : \infty \to \infty, \forall n \in \infty$ примем обозначение $f(n)\downarrow$ тогда и только тогда, когда $f(n)$ не определена.

Определение 2.25. Для всех частичных функций $f, g : \Gamma' \to \Gamma', \forall s,t \in S, f(s,t) = g(s,t)$ тогда и только тогда, когда выполняется одно из условий:

1. $f(s,t)\uparrow$ и $g(s,t)\uparrow$ либо
2. $f(s,t)\downarrow$ и $g(s,t)\downarrow$ и $f(s,t) = g(s,t)$.

Определение служит для уточнения понятия тождественности программ, представленных в виде частичных функций.

Определение 2.26.

$\forall z, z' \in \Gamma', \forall p, p', q = q_z, q_2, \ldots q_z, q' = q'_1, q'_2, \ldots, q'_z \in S$ для всех частичных функций тогда и только тогда, когда:

$$h : S \to S : \langle p, q \rangle : \langle p', q' \rangle$$

при 1. $z = z'$ и

2. $p = p'$ и

3. $\exists i \in [l, z] : q_i \neq q'_i$, и

4. $\forall i \in [l, z]$ либо $q = q'$, либо $h\,(q_i)\downarrow$ и $h\,(q_i) = q'_i$

Если раньше для обозначения действия программы на систему использовались коды состояний, то в этом определении используется детализированное описание состояния и по сути вводится отношение между чистым и зараженным состоянием системы. Зараженное состояние характеризуется тем, что при прочих равных в нем некоторые программы изменены вирусом (функция h).

Определение 2.27. Для всех частичных функций

$f, g, h : \Gamma' \to \Gamma'$, $\forall s, t \in S : f(s,t) \overset{h}{:} g(s,t)$ тогда и только тогда, когда $f(s,t)\downarrow, g(s,t)\downarrow$ и $f(s,t) \overset{h}{:} g(s,t)$.

В этом определении уточняется введенное ранее отношение для состояний, получаемых в результате действия двух функций. Результатом действия функции f на состояние системы $\langle s, t \rangle$ является натуральное число, которое можно однозначно интерпретировать как новое состояние $\langle s', t' \rangle$. Точно так же $g(s,t) = \langle s'', t'' \rangle$. Соответственно запись $f(s,t) \overset{h}{:} g(s,t)$ обозначает, что обе программы завершают свою работу при запуске из состояния $\langle s, t \rangle$, и результат их выполнения будет отличаться фактом заражения некоторых программ: $\langle s', t' \rangle \overset{h}{:} \langle s'', t'' \rangle$.

Определение 2.28. Для всех частичных функций

$f,g,h:\Gamma'\to\Gamma'$, $\forall s,t\in S: f(s,t)\overset{h}{\cong}g(s,t)$ тогда и только тогда, когда $f(s,t)=g(s,t)$ или $f(s,t)\overset{h}{:}g(s,t)$.

Введенное обозначение призвано отразить тот факт, что зараженная программа не всегда выполняет заражение – в некоторых состояниях зараженная программа может выполнять ровно те же действия, что и незараженная, т.е. результат ее действия при некоторых состояниях системы будет точно таким же, как и у незараженной программы.

Определение 2.29. Для всех геделевских нумераций частичных рекурсивных функций φ_i общерекурсивная функция v будет вирусом по отношению к φ_i тогда и только тогда, когда $\forall d,p\in S$ выполняется либо:

1. Повреждение: $(\forall i,j\in\Gamma')\left[\varphi_{v(i)}(d,p)=\varphi_{v(j)}(d,p)\right]$,
2. Заражение или имитация: $(\forall j\in\Gamma')\left[\varphi_j(d,p)\overset{v}{\cong}\varphi_{v(j)}(d,p)\right]$.

В данном определении выбор переменных d и p в явном виде указывает на их природу – данные (информация, не подверженная заражению) и программы (информация, подверженная заражению).

Здесь необходимы комментарии.

Повреждение. Из определения следует, что результат действия зараженной программы при определенном состоянии системы определяется только типом вируса и состоянием системы независимо от того, какая программа была заражена. На самом деле выполняемое вирусом действие может и не быть деструктивным, важно то, что оригинальная программа не выполняется вовсе, т.е. выполняется функция, прописанная в самом вирусе.

Заражение или имитация. Здесь согласно определению ситуация прямо противоположная. Результат действия зараженной программы либо вовсе не отличается от результата действия исходной, либо отличается появлением в системе новых зараженных программ. Различное поведение определяется различиями в исходном состоянии системы.

2.4. Классификация вирусов

Определение 2.30. Для всех геделевских нумераций частичных рекурсивных функций φ_i, для всех вирусов v по отношению к $\varphi_i, \forall i, j \in \Gamma'$:

1) i вредоносна по отношению к v и j тогда и только тогда, когда $i = v(j)$ и $(\exists d, p \in S)\left[\varphi_j(d,p) \overset{v}{\cong} \varphi_i(d,p)\right]$. Существует состояние системы, при котором зараженная вирусом v программа j выполняет вредоносную функцию, т.е. результат действия зараженной программы не ограничивается заражением (появлением новых зараженных программ);

2) i заразна по отношению к v и j тогда и только тогда, когда $i = v(j)$ и $(\exists d, p \in S)\left[\varphi_j(d,p) \overset{v}{:} \varphi_i(d,p)\right]$. Существует состояние системы, при котором зараженная вирусом v программа j выполняет функцию размножения – в результате ее действия в системе появляются новые зараженные программы;

3) i безобидна по отношению к v и j тогда и только тогда, когда $i = v(j)$, i не вредоносна по отношению к j, i не заразна по отношению к j;

4) i является трояном по отношению к v и j тогда и только тогда, когда $i = v(j)$, вредоносна по отношению к j, i не заразна по отношению к j. Троян способен только к выполнению вредоносных функций и не может размножаться;

5) i является переносчиком по отношению к v и j тогда и только тогда, когда $i = v(j)$, не вредоносна по отношению к j, i заразна по отношению к j. Переносчик является антиподом трояна: он только размножается и не содержит вредоносных функций;

6) i является вирусом по отношению к v и j тогда и только тогда, когда $i = v(j)$, i вредоносна по отношению к j, i заразна по отношению к j.

Вирус – наиболее универсальный тип вредоносных программ, способный как к размножению, так и к выполнению вредоносных действий.

В тех случаях, когда существует единственная j такая, что (т.е. когда v инъективна) и i является вредоносной (заразной, безобидной, трояном, переносчиком, вирусом) по отношению к v и j, ссылка на j будет опускаться и i будет называться вредоносной (заразной, безобидной, трояном, переносчиком, вирусом) по отношению к v.

Таким образом, если по отношению к какому-либо вирусу инфицированная программа является безобидной, это значит, что она выполняет в точности те же действия, что и исходная неинфицированная программа и никаких других. Если она является трояном, значит она не может заражать другие программы, а может только имитировать их действия или наносить вред. Если программа является переносчиком, значит она не способна наносить вред, но при определенных обстоятельствах может заражать другие программы.

Определение 2.31. Для всех геделевских нумераций частичных рекурсивных функций φ_i, для всех вирусов v по отношению к φ_i:

1) вирус v безобиден тогда и только тогда, когда одновременно справедливо: $(\forall j \in \Gamma')$ [v (j) не является вредоносной по отношению к v и j], $(\forall j \in \Gamma')$ [v (j) не является заразной по отношению к v и j];

2) вирус v – троянский конь тогда и только тогда, когда одновременно справедливо: $(\forall j \in \Gamma')$ [v (j) является вредоносной по отношению к v и j], $(\forall j \in \Gamma')$ [v (j) не является заразной по отношению к v и j];

3) вирус v распространяется тогда и только тогда, когда одновременно справедливо: $(\forall j \in \Gamma')$ [v (j) не является вредоносной по отношению к v и j], $(\forall j \in \Gamma')$ [v (j) является заразной по отношению к v и j];

4) вирус v вредоносен тогда и только тогда, когда одновременно справедливо: $(\forall j \in \Gamma')$ [v (j) является вредоносной по отношению к v и j], $(\forall j \in \Gamma')$ [v (j) является заразной по отношению к v и j].

Следующая теорема отмечает ряд простых свойств, присущих различным типам вирусов.

Теорема 2.6. Для всех геделевских нумераций частичных рекурсивных функций φ_i, для всех вирусов v по отношению к φ_i:

- $(\forall j \in \Gamma'$ [v (j) безобидна по отношению к v и j];

- вирус v безобиден тогда и только тогда, когда $(\forall j \in \Gamma')$ [v (j) безобидна по отношению к v и j];

- если v – троян, то $(\forall j \in \Gamma')$ [v (j) безобидна по отношению к v и j] или [v (j) является трояном по отношению к v и j];

- если v способен только распространяться, то $(\forall j \in \Gamma')$ [v (j) безобидна по отношению к v и j] или [v (j) является переносчиком по отношению к v и j].

Доказательство. Первое свойство непосредственно следует из первой теоремы о рекурсии (теоремы о неподвижной точке). Остальные свойства следуют непосредственно из определений.

2.5. Обнаружение компьютерных вирусов

После определения компьютерного вируса естественным образом возникает вопрос об обнаружении такого рода программ, или о разрешимости множества компьютерных вирусов. Л. Адельман доказал следующую теорему.

Теорема 2.7. Для всех геделевских нумераций частичных рекурсивных функций φ_i:

$V = \{I \mid \varphi_i$ является вирусом$\}$ – Π_2 – полное множество.

Теорема приводится без доказательства.

Здесь Π_2 – класс множеств в арифметической классификации. Известно, что классы множеств Π_2 с индексом 1 и выше являются неразрешимыми. Следовательно, и множество вирусов является неразрешимым.

Контрольные вопросы и задания

1. Разработайте алгоритм V, подходящий под описание Коэна. Опишите этот алгоритм на псевдокоде.

2. Докажите лемму о возможности полиморфизма для кода RASPM с ABS. Приведите пример полиморфного кода для данной вычислительной машины.
3. Докажите лемму о возможности использования RASPM вместо машины Тьюринга.
4. Является ли слабое обнаружение вируса NP-полной задачей?
5. Приведите примеры различных программ $i = v\ (j)$ по классификации Адельмана на псевдокоде с обоснованием отнесения их к тому или иному классу.

Глава 3. МАТЕМАТИЧЕСКИЕ МОДЕЛИ РАСПРОСТРАНЕНИЯ ВРЕДОНОСНЫХ ПРОГРАММ

Математический аппарат для изучения динамики «обычных» биологических эпидемий был разработан достаточно давно (подробный обзор моделей распространения инфекционных заболеваний представлен в изданиях, приведенных в рекомендательном списке литературы). Биологический подход к моделированию вирусной проблемы по общему признанию начался с работ J. O. Kephart и S. R. White из фирмы IBM еще до изобретения WWW, однако модным это направление стало только в 2001 г. после вспышек вирусных эпидемий Code Red и Nimda. Наибольший резонанс получили работы N. Weaver и его коллег. Ими исследованы различные концептуальные алгоритмы размножения самовоспроизводящихся кодов, кардинально повышающие эффективность их распространения. «Вирусописатели» не заставили себя долго уговаривать и в 2003 г. был дан старт небывалым по размаху и последствиям сетевым эпидемиям, описанным в первой главе настоящего пособия (Slammer, MSBlast, Sobig, MyDoom и др.).

В данной главе рассматриваются математические модели сетевых эпидемий, от простых биологических моделей до усложнённых дискретных и с обратной связью. Показано влияние топологии сети на распространение атакующих процессов.

3.1. Упрощенная эпидемиологическая модель: SI-модель

Основные результаты можно получить уже из анализа классических моделей эпидемий, которые подходят для изучения компьютерных инфекций даже несколько лучше, чем для биологических аналогов.

Большинство существующих на сегодняшний день математических моделей подобных атак основываются на стандартных эпидемиологических моделях, используемых для описания поведения биологических вирусов. Однако такие модели не учитывают ряд фак-

торов, имеющих место в компьютерных сетях; например, изменяемость скорости заражения в связи с выходом из строя роутеров или динамическое противодействие пользователей. В связи с этим общие эпидемиологические модели проходят ряд существенных модификаций или от них вовсе отказываются.

Вообще эпидемические модели предполагают, что принципал последовательно принимает ряд дискретных состояний, причём переход из одного состояния в другое происходит с определённой вероятностью.

Таким образом, разработка подобной модели требует определения набора состояний принципала и вероятностей перехода.

Рассмотрим, к примеру, так называемую SI-модель (от Susceptible – Infected, т.е. Восприимчивый – Заражённый). В этой простой модели, состоящей только из двух состояний, каждый субъект может пребывать либо в "восприимчивом" (*S*) состоянии, либо в "заражённом" (*I*). Субъекты, находящиеся в состоянии *S*, здоровы и могут потенциально быть заражены. *I*-субъекты, в свою очередь, являются носителями вируса и могут заражать оставшиеся *S*-субъекты. Если субъект заражён, он остаётся в таком состоянии навсегда (рис. 3.1).

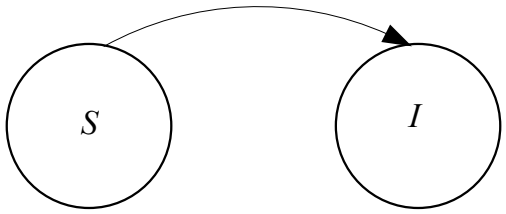

Рис. 3.1. Граф состояний субъекта в SI-модели

Следующее уравнение описывает динамику распространения вируса согласно такой модели:

$$\frac{dJ(t)}{dt} = \beta J(t)[N - J(t)], \qquad (3.1)$$

где $J(t)$ – число заражённых субъектов во время t; N – количество субъектов, и β – скорость заражения.

Начальные условия: $t = 0$, в сети $J(0)$ заражённых и $N - J(0)$ восприимчивых принципалов.

3.2. SIS-модель

SIS-модель (рис. 3.2) в отличие от предыдущей предусматривает факт естественного восстановления принципала и его переход обратно в «восприимчивое» состояние.

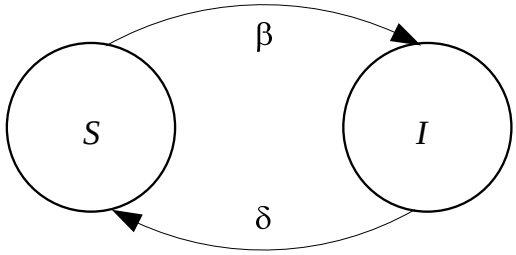

Рис. 3.2. Граф состояний субъекта в SIS-модели

Такой субъект не приобретает иммунитета и может быть повторно заражён. Вводится величина δ – скорость восстановления субъекта.

Динамика эпидемии описывается уравнением

$$\frac{d\rho}{dt} = \beta\rho(1-\rho) - \delta\rho, \qquad (3.2)$$

где ρ – $J(t)/N$ – распространённость.

3.3. Модель Кермака – Маккендрика (SIR-модель)

Модель Кермака – Маккендрика (рис. 3.3) учитывает процесс вывода заражённых принципалов из системы. Она базируется на предположении, что во время эпидемии некоторое количество заражённых субъектов либо избавляется от вируса, либо перестаёт функционировать. Как только субъект избавляется от вируса, он приобретает к нему иммунитет и остаётся в этом состоянии навсегда. Таким образом, субъекты принимают три состояния: «восприимчивый» (S), «заражённый» (I), «невосприимчивый»(R).

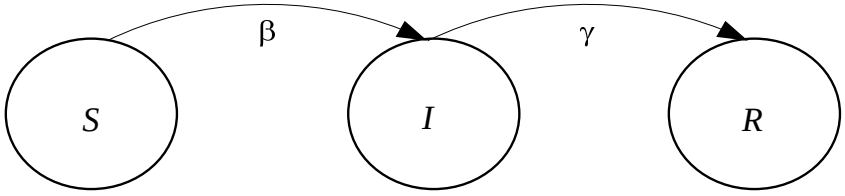

Рис. 3.3. Граф состояний субъекта в модели Кермака – Маккендрика

Такая модель будет описываться системой уравнений:

$$\begin{cases} \dfrac{dJ(t)}{dt} = \beta J(t)[N - J(t)], \\ \dfrac{dR(t)}{dt} = \gamma I(t), \\ J(t) = I(t) + R(t) = N - S(t), \end{cases} \qquad (2.3)$$

где β – скорость распространения, а γ – скорость вывода принципалов из сети.

Пусть $P = \gamma / \beta$ – относительная скорость иммунизации (выхода из системы). Тогда согласно вышеприведённой модели мы получаем интересный результат: $\frac{dI(t)}{dt} > 0$ тогда и только тогда, когда $S(t) < P$. Это означает, что если начальное количество уязвимых субъектов в сети меньше некоторой константы (порога эпидемии), то процесс заражения будет затухающим и в определённый момент времени прекратится. К сожалению, гораздо чаще происходит обратный процесс, так как скорость вывода субъекта из процесса иммунизации определяется запаздывающей человеческой реакцией и необходимостью загрузки громоздких обновлений, а скорость заражения – постоянно улучшающимися техническими характеристиками сети и улучшенными механизмами распространения, внедрёнными злоумышленником (он может вставить небольшую паузу в цикл размножения, чтобы не создавать катастрофического трафика и слегка понизить скорость инфицирования). Доля же уязвимых принципалов обычно очень велика. Поэтому практически всегда $P >> 1$: к примеру, при моделировании эпидемии CodeRed v2 согласие с реальными данными было достигнуто при $P \sim 10^6$.

Такая модель, несмотря на кажущуюся полноту, всё ещё не подходит для описания распространения вредоносных программ в какой-либо среде. Во-первых, противодействие различного рода (к примеру, устранение уязвимостей, иммунизация, фильтрация потенциально опасных потоков данных) приводит к выходу из процесса заражения не только инфицированных, но и восприимчивых к инфекции принципалов, тогда как модель Кермака – Маккендрика не предусматривает подобного явления. Во-вторых, эта модель принимает скорость инфицирования за константу, что в корне неверно для лавинообразного распространения червей типа MyDoom или Nimda.

3.4. Двухфакторная модель

Двухфакторная модель является попыткой сделать классические эпидемиологические модели более реалистичными, вводя два дополнительных условия:

- противодействие эпидемии приводит к тому, что из сети исключаются не только заражённые, но и восприимчивые субъекты. Например, во время эпидемии Code Red люди, узнавшие об угрозе, принимали определенные контрмеры, вплоть до отсоединения компьютеров от сетей;

- имеет смысл представить скорость заражения не константой, а убывающей функцией времени, так как крупномасштабное распространение угрозы заполняет каналы связи и нарушает сетевую инфраструктуру (скажем, выводит из строя роутеры), тем самым замедляя дальнейшее инфицирование субъектов (рис. 3.4).

Эта модель описывает распространение вредоносных программ, не учитывая ограничения, появляющиеся в связи с топологией сети. Под этими ограничениями мы понимаем тот факт, что принципал, являющийся распространителем атакующего ПО, не может напрямую заразить восприимчивый к данному роду атаки субъект – для этого ему требуется инфицировать все субъекты, находящиеся на пути к цели. Например, на скорость распространения большинства сетевых червей (вроде CodeRed) топология не оказывает существенного влияния. Напротив, вирусы, распространяющиеся по электронной почте (скажем, Melissa и LoveBug), инфицируют субъекты сети в прямой зависимости от логической топологии, устанавливаемой адресными книгами реципиентов данного вида вредоносного программного обеспечения.

Модель будет описываться дифференциальным уравнением вида

$$\frac{dI(t)}{dt} = \beta(t)[N - R(t) - I(t) - Q(t)]I(t) - \frac{dR(t)}{dt}, \qquad (3.4)$$

где $I(t)$ – число инфицированных субъектов, $R(t)$ – число инфицированных субъектов, исключённых из сети, $Q(t)$ – число восприимчивых субъектов, исключённых из сети.

Вообще говоря, распространение вредоносных программ по сети дискретно по времени, однако, создатели модели используют непрерывное дифференциальное уравнение для её описания. Такого рода аппроксимация даёт точные результаты для крупномасштабных систем, в связи с чем широко используется для моделирования потоков трафика в Интернет и подобных процессов. Чтобы получить решение этого уравнения, необходимо определить свойства

этих функций. Например, на основе некоторых предположений получают систему уравнений (3.5), полностью описывающую поведение модели:

$$\begin{cases} \dfrac{dS(t)}{dt} = -\beta(t)S(t)I(t) - \dfrac{dQ(t)}{dt}, \\ \dfrac{dR(t)}{dt} = \gamma I(t), \\ \dfrac{dQ(t)}{dt} = \mu S(t)J(t), \\ \beta(t) = \beta_0[1 - \dfrac{I(t)}{N}]^{\eta}, \\ N = S(t) + I(t) + R(t) + Q(t), \\ I(0) = I_0 << N;\ \ S(0) = N - I_0;\ \ R(0) = Q(0) = 0, \end{cases} \tag{3.5}$$

где γ – скорость перехода субъекта из $I \rightarrow R$, μ – скорость перехода из $S \rightarrow R$, η – параметр, задающий чувствительность скорости заражения к числу инфицированных принципалов (если $\eta = 0$, то скорость постоянна), $J(t) = I(t) + R(t)$.

Решение системы уравнений (3.5) представлено на рис. 3.5.

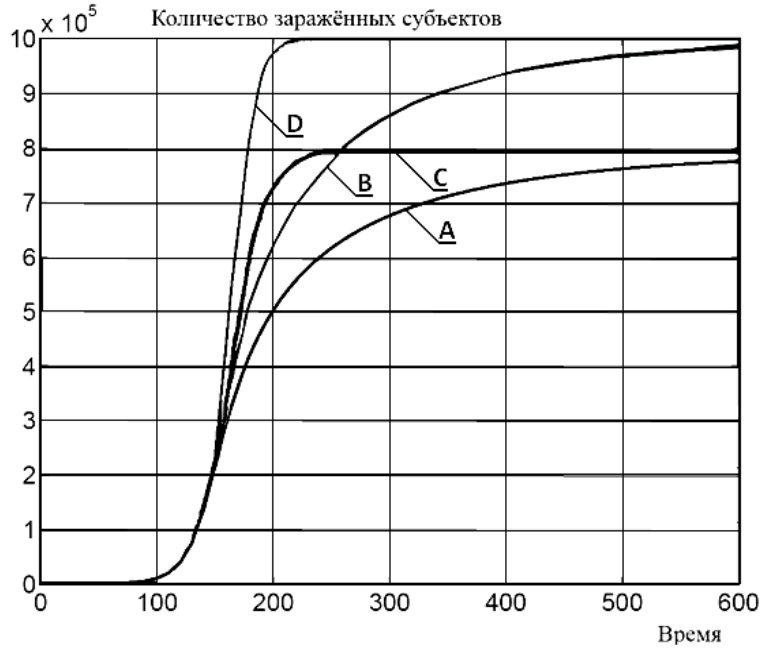

Рис. 3.4. Сравнение различных способов моделирования распространения вредоносных программ (*D* – стандартная эпидемиологическая SI-модель, *C* – модель, учитывающая контрмеры, *B* – модель с замедлением скорости распространения, *A* – двухфакторная модель)

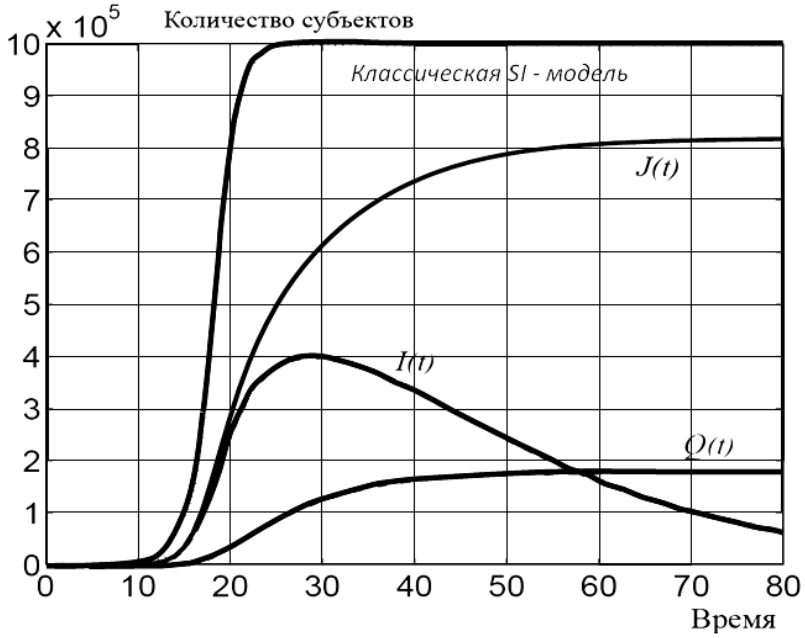

Рис. 3.5. Решение системы дифференциальных уравнений, описывающих двухфакторную модель

3.5. Модель AAWP

Когда активная вредоносная программа выпускается в сеть, она пытается одновременно просканировать несколько принципалов на предмет уязвимости. Когда такая программа находит уязвимый субъект, она удалённо исполняет зондирующий код, заражающий данную единицу сети. Если заражение прошло успешно, то копия вредоносной программы передаётся на заражённый субъект; в результате этот субъект пытается вышеуказанным методом инфицировать остальную сеть. Если атакующий процесс сканирует неуязвимый субъект или несуществующий сетевой адрес, то тогда он не представляет никакой угрозы. Во время распространения эпидемии субъекты сети могут перестать функционировать в нормальном режиме, вынуждая операторов этих субъектов к перезагрузке или, как минимум, к выгрузке процессов, могущих каким-либо образом угрожать системе, из памяти. После этой процедуры такие принципалы становятся опять восприимчивыми к эффекту атакующей программы, действующей в сети (рис. 3.8).

После обнаружения вредоносной программы люди пытаются уменьшить вред, наносимый сети, путём устранения уязвимо-

сти принципалов, в расчёте на которую и написана данная программа. Как только субъект получает обновление, гарантирующее «закрытие» такой уязвимости, он становится невосприимчивым к действиям вредоносного программного обеспечения. На рис. 3.7 показано поведение модели AAWP в зависимости от скорости выхода такого рода обновлений.

Weaver указывал на возможность значительного увеличения скорости распространения сетевых червей путём создания «чёрного списка» субъектов сети, которые *a priori* считаются уязвимыми и будут инфицированы в первую очередь (рис. 3.6). После того как все принципалы из этого списка заражены, атакующий процесс использует их, чтобы распространить инфекцию по остальной сети. Субъекты сети, содержащиеся в «чёрном списке», должны обладать достаточно большой степенью связности, чтобы эффективно распространить угрозу в распределённой системе.

Алгоритмы, используемые при создании сетевых червей, реализуют случайное сканирование, сканирование по подсетям, перестановочное сканирование, а также сканирование, учитывающее топологию распределённой информационной системы, подверженной заражению. В алгоритмах случайного сканирования предполагается, что принципалы в сети образуют собой гомогенный граф («однородную популяцию», если пользоваться терминами классической эпидемиологии).

Это означает, что каждый субъект сети имеет равные шансы на заражение. Сканирование по подсетям в своей основе также имеет положение об «однородной популяции», однако, вместо того, чтобы проверять на уязвимость всё адресное пространство сети, подобный алгоритм сканирования ищет восприимчивые субъекты внутри нескольких подсетей, состоящих из принципалов, непосредственно «соседствующих» с заражённым субъектом. Например, сетевой червь Nimda согласно известной статистике выбирал потенциально уязвимые субъекты следующим образом:

- с вероятностью 50 % выбираются IP-адреса, содержащие такие же два первых октета, как и заражённый принципал;

- с вероятностью 25 % выбирается IP-адрес с таким же первым октетом, как и адрес заражённого принципала;

- с вероятностью 25 % выбирается адрес, состоящий из псевдослучайно генерируемых октетов.

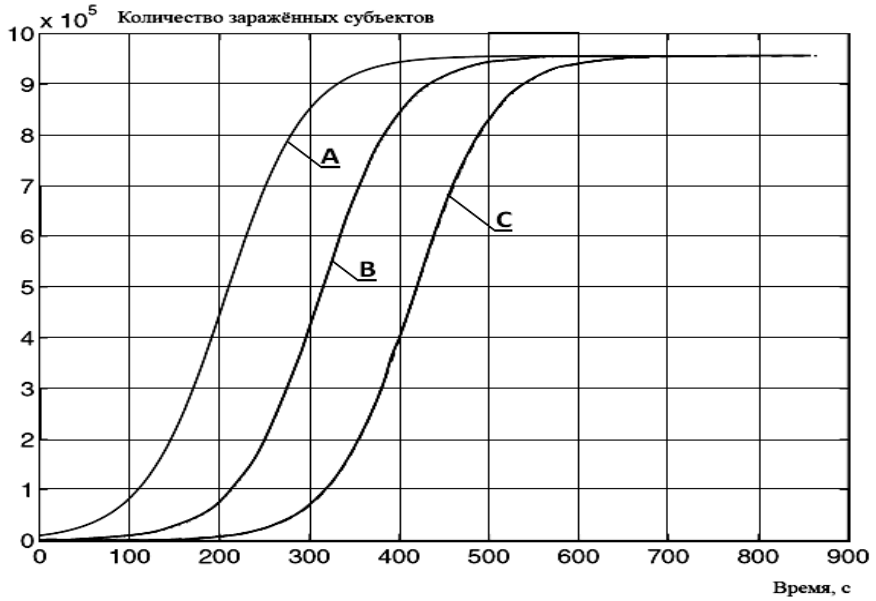

Рис. 3.6. Модель AAWP. Зависимость распространения инфекции от размеров «чёрного списка»: *A* – 10000 субъектов, *D* – 1000 субъектов, *C* – 100 субъектов

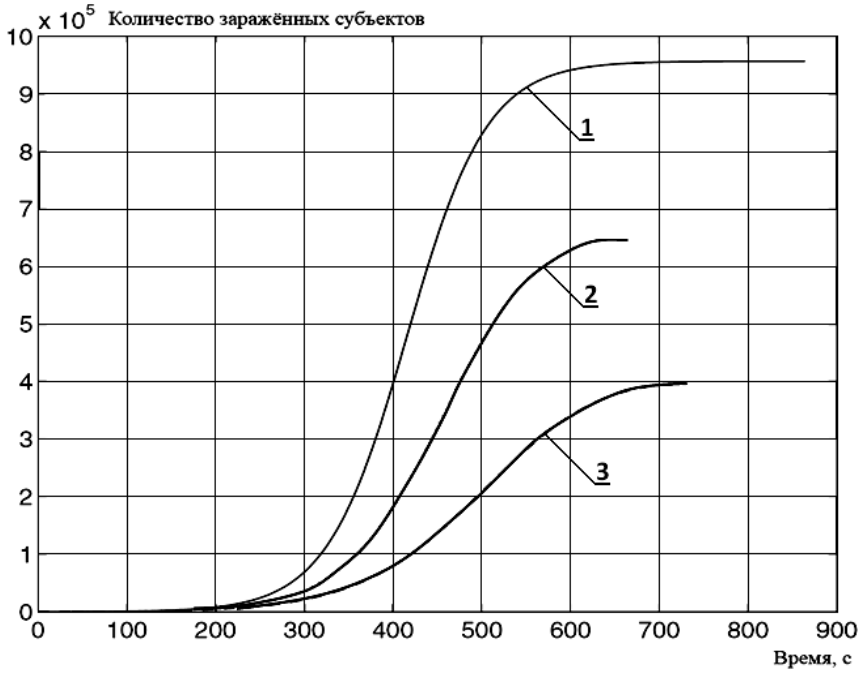

Рис. 3.7. Модель AAWP. Зависимость распространения инфекции от скорости вакцинации: 1 – вакцинация не производится, 2 – 0,0005 субъекта/с, 3 – 0,001 субъекта/с (субъекты заражаются за время 1 с, «чёрный список» состоит из 100 принципалов)

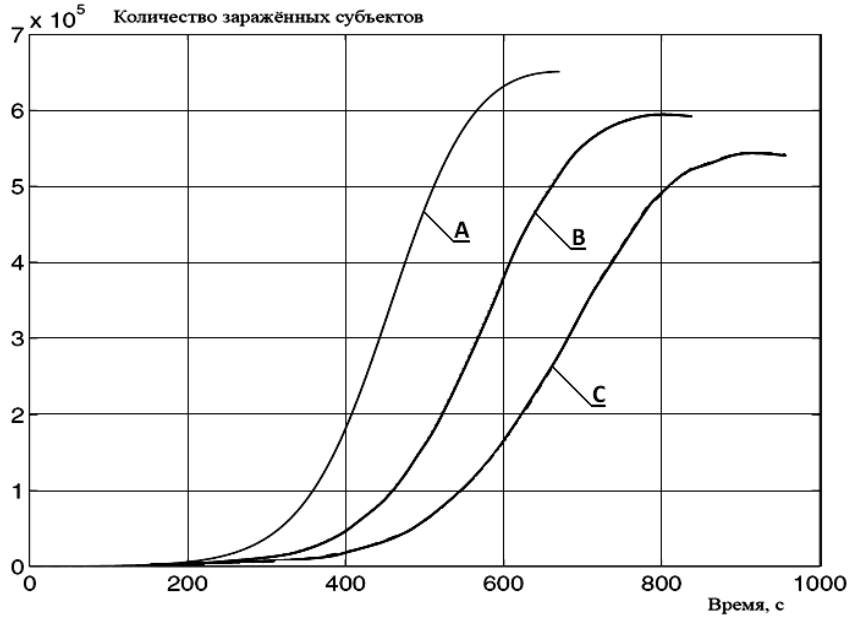

Рис. 3.8. Модель AAWP. Зависимость распространения инфекции от времени, требуемого для инфицирования одного субъекта сети: *A* – 1 с, *B* – 30 с, *C* – 60 с (субъекты вакцинируются со скоростью 0,0005 субъекта/с, «чёрный список» состоит из 100 принципалов)

Модель AAWP (Analytical Active Worm Propagation) описывает поведение сетевых червей, применяющих случайное сканирование сети. Она базируется на предположении, что каждый принципал сети может быть заражён с одинаковой вероятностью $1/2^{32}$ (так как черви сканируют всё адресное пространство). Эта модель дискретна по времени, и распространение вируса описывается рекуррентной формулой

$$n_{i+1} = (1-d-p)n_i + [(1-p)^i N - n_i][1-(1-\frac{1}{2^{32}})^{sn_i}], \qquad (3.6)$$

где n_i – количество заражённых принципалов на i-м тике, p – скорость вакцинации субъектов, d – скорость выведения субъектов из строя, s – частота сканирования.

Такая модель может быть легко видоизменена для симуляции червей с механизмами сканирования по локальной подсети (local subnet scanning).

3.6. Модель с противодействием: Progressive SIDR

Ни одна из приведённых выше моделей не учитывает наличие некоторого блокирующего фактора, скажем, антивирусного про-

граммного обеспечения, появляющегося по мере развития эпидемии (и, как следствие, всё большей информированности о ней субъектов). Laveille представлена модель, названная Progressive SIDR (PSIDR) и состоящая из двух фаз (рис. 3.9):

- *начальное заражение* – червь заражает одну машину в сети и затем в течение некоторого времени распространяется свободно, т.е. согласно SI-модели;

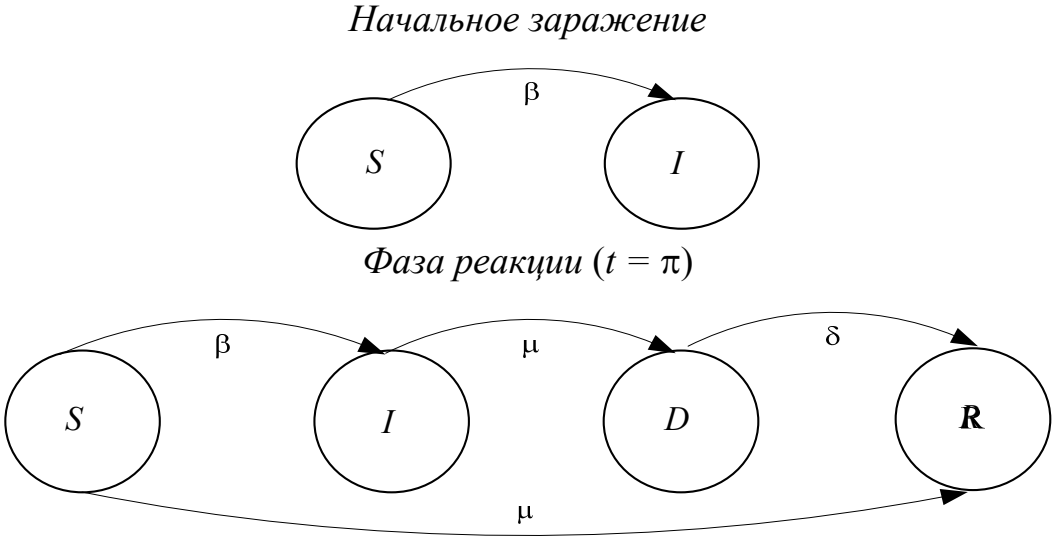

Рис. 3.9. Графы состояния субъекта в модели PSIDR

- *фаза реакции*, когда по прошествии некоторого времени червь обнаруживается и со стороны субъектов производятся немедленные действия.

Итак, все принципалы, оставшиеся незаражёнными, автоматически вакцинируются, а инфицированные – обнаруживаются с определённой скоростью, избавляются от инфекции и приобретают иммунитет. В этой фазе скорость распространения остаётся прежней, однако восприимчивые принципалы вакцинируются со скоростью μ, а инфицированные субъекты обнаруживаются со скоростью μ и «лечатся» со скоростью δ (μ представляет собой не что иное, как скорость обновления антивируса).

В первой фазе система ведёт себя согласно законам

$$\frac{dS}{dt} = -\beta SI, \quad \frac{dI}{dt} = \beta SI, \tag{3.7}$$

а во второй –

$$\frac{dS}{dt} = -\beta SI - \mu S, \qquad \frac{dI}{dt} = \beta SI - \mu I,$$

$$\frac{dD}{dt} = \mu I - \delta D, \qquad \frac{dR}{dt} = \delta D + \mu S, \tag{3.8}$$

где $D(t)$ – количество обнаруженных инфицированных субъектов. При этом начальные условия для системы таковы: $S > 0$, $I > 0$, $D = 0$, $R = 0$.

Анализ этой модели выявляет тот факт, что обновление антивируса должно происходить как можно раньше (рис. 3.10 – 3.15), потому что любая задержка обновления приводит ко всё большим и большим периодам «взрыва» эпидемии. Тем не менее даже если вакцина будет доступна немедленно, эпидемия будет распространяться довольно быстро, так как начнётся «борьба» за субъекты между сетевым червём и системой защиты. Этот феномен предполагает, что общая безопасность сети может быть значительно увеличена за счёт скорости доставки обновлений (например, это можно реализовать, регулируя частоту запроса клиентами «заплаток» с центральных серверов).

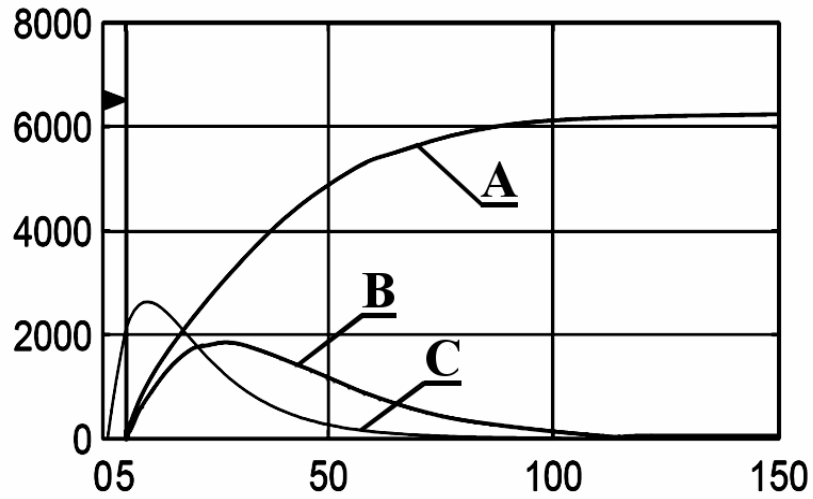

Рис. 3.10. Модель PSIDR. Поведение модели в однородных графах ($\pi = 5$, $\mu = 0{,}07$; $\delta = 0{,}05$; $\beta = 0{,}1$)

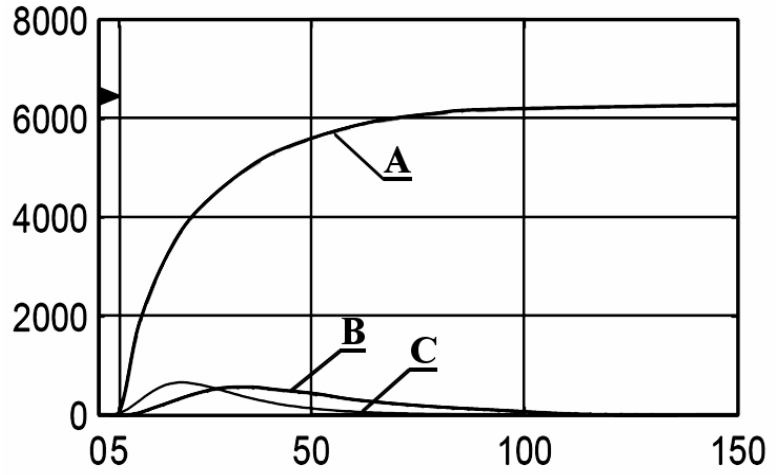

Рис. 3.11. Модель PSIDR. Поведение модели в SF-сетях ($\pi = 5$, $\mu = 0{,}07$; $\delta = 0{,}05$; $\beta = 0{,}1$)

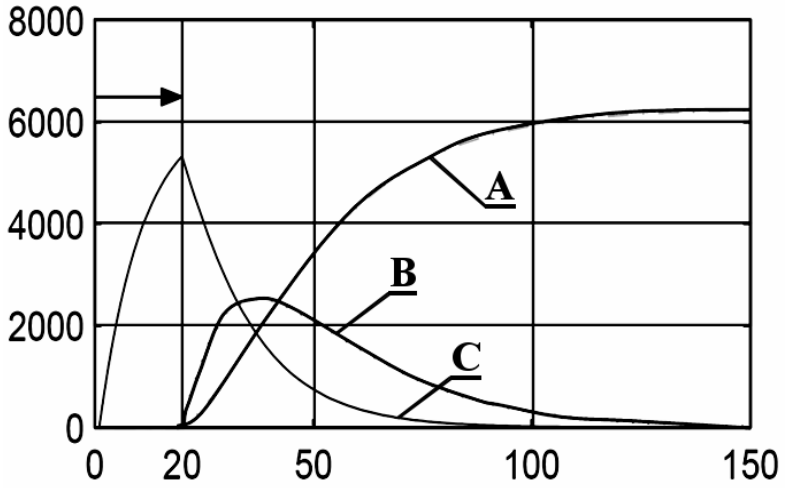

Рис. 3.12. Модель PSIDR. Поведение модели в однородных графах ($\pi = 20$, $\mu = 0{,}07$; $\delta = 0{,}05$; $\beta = 0{,}1$)

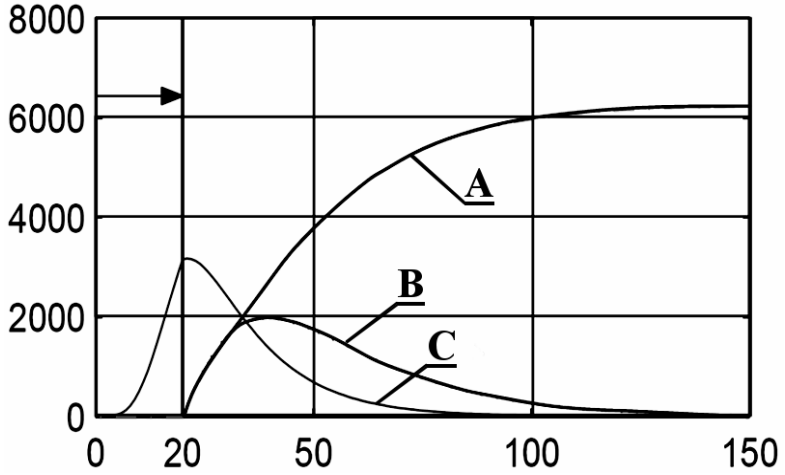

Рис. 3.13. Модель PSIDR. Поведение модели в SF-сетях ($\pi = 20$, $\mu = 0{,}07$; $\delta = 0{,}05$; $\beta = 0{,}1$)

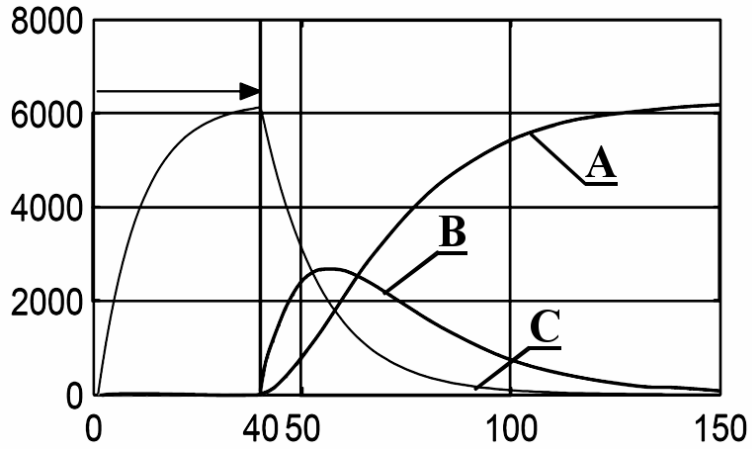

Рис. 3.14. Модель PSIDR. Поведение модели в однородных графах ($\pi = 40$, $\mu = 0{,}07$; $\delta = 0{,}05$; $\beta = 0{,}1$)

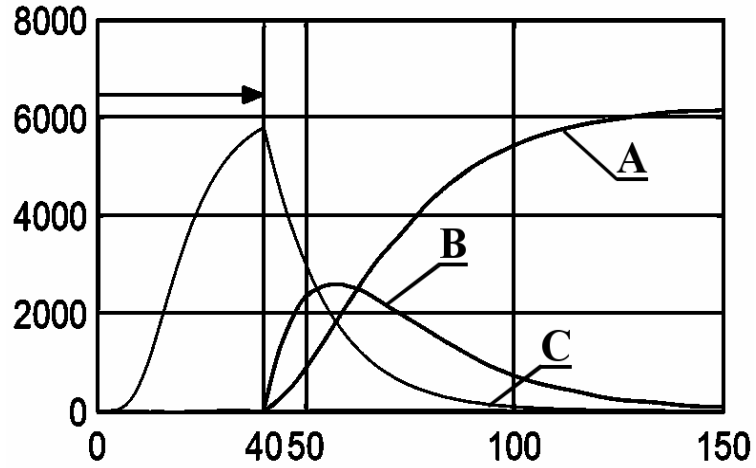

Рис. 3.15. Модель PSIDR. Поведение модели в SF-сетях ($\pi = 40$, $\mu = 0{,}07$; $\delta = 0{,}05$; $\beta = 0{,}1$)

Хотя эта модель и является наиболее подробной на текущий момент, есть аспекты, которые она не отслеживает, как-то:

- частота иммунизации δ должна быть переменной величиной, так как чем больше субъектов атаковано, тем интенсивнее борьба с атакой. Скорость иммунизации, таким образом, может быть функцией от $I(t)$; это отношение, скорее всего, будет существенно влиять на время, требуемое системе для полного восстановления после атаки;

- частота распространения угрозы должна быть переменной величиной, как, скажем, в двухфакторной модели. Внедрение этого положения в модель должно увеличить точность симуляции;

- некоторые сетевые черви выводят субъекты из строя в определённый период времени. Скажем, червь Klez.e наносит поврежде-

ния 6-го числа каждого нечётного месяца. Пока неясно, каким образом автор будет интегрировать этот факт в модель, вероятно, вводя несколько стохастических характеристик.

3.7. Поведение эпидемиологических моделей в сетях с различными типами топологий

Чтобы показать влияние топологии на распространение угрозы, мы рассмотрим два типа сетей: однородные графы и SF-сети. Структуру однородного графа имеет сеть электронной почты внутри LAN и некоторые P2P-сети. SF-сети отличаются специфическим распределением степени связности: у большинства вершин она очень мала, в то время как у относительно небольшого числа вершин степень связности большая. Такая структура наиболее близка к топологии Интернет и прочих больших сетей.

SIS-модель в однородных графах.

A. Число заражённых смежных принципалов незначительно влияет на вероятность заражения конкретного субъекта; таким образом, уравнение (3.2) упрощается до

$$\frac{d\rho}{dt} = \beta(1-\rho) - \delta\rho. \qquad (3.9)$$

B. Распространённость вируса при стабильном состоянии системы (когда соотношение заражённых и уязвимых субъектов не меняется) можно найти, решив (3.9) при условии $\rho` = 0$:

$$\rho_{eq} = \frac{\lambda}{1+\lambda}, \qquad (3.10)$$

где $\lambda = \beta / \delta$. Эпидемического порога нет, и (3.9) действительно до тех пор, пока в сети есть хотя бы один заражённый субъект.

SIS-модель в SF-сетях.

A. Эпидемии в таких сетях быстро достигают стабильного состояния с ненулевой распространённостью ρ_{eq}, однако, если ($\lambda = \beta/\delta$) мала (а именно меньше эпидемического порога), то $\rho_{eq} \rightarrow 0$ и эпидемия рано или поздно сойдёт на нет.

B. Распространённость возрастает при росте λ.

C. Согласно *Pastor-Satorras* в конечных SF-сетях присутствует эпидемический порог

$$\lambda_c = \frac{< s >}{< s^2 >}, \qquad (3.11)$$

где <*s*> – средняя степень связности, а <s^2> – среднеквадратическое отклонение степени.

D. Результаты, полученные Laveille, показывают, что ρ ~ <*s*> (это объясняет практически моментальное распространение вредоносных программ в корпоративных LAN). Также время, требуемое для достижения стабильного состояния, не зависит от размера сети. Это очень важно: отсюда следует, что эпидемия может поразить даже такую большую сеть, как Интернет за довольно малое время. Та же эпидемия Klez, если судить по статистике MessageLabs, свидетельствует о достижении порога самоподдержки вирусной инфекции. В наличии оказалось несколько факторов, обеспечивающих «бессмертие» вирусного кода.

Во-первых, появились экземпляры, обладающие серьезной изменчивостью поведения: они выбирают для поражения и маскировки множество типов файлов, при этом достаточно осторожно обращаясь с системными (за исключением дней уничтожения); способны переносить в себе другие вирусы и сосуществовать с ними к обоюдной выгоде (симбиоз, сразу же нашедший подражателей); эффективно подавляют антивирусные программы (борьба за существование); и, наконец, помимо изменчивых почтовых сообщений, активно оккупируют локальные сети, раскладывая свои копии во всех доступных местах.

Во-вторых, рост количества компьютеров соответственно увеличивает абсолютную величину пользователей с достаточным уровнем некомпетентности. Люди не понимают природы угрозы новых технологий и, естественно, не могут от нее защититься. Это обеспечивает факт присутствия в Сети значительного числа инфицированных систем. Таким образом, ситуация скачкообразно преобразовалась из периода отдельных вирусных вспышек в эпоху хронической заболеваемости.

E. Влияние топологии на поведение модели отчётливо видно на ранних стадиях эпидемии, ещё до её вхождения в фазу «быстрого распространения» (рис. 3.16).

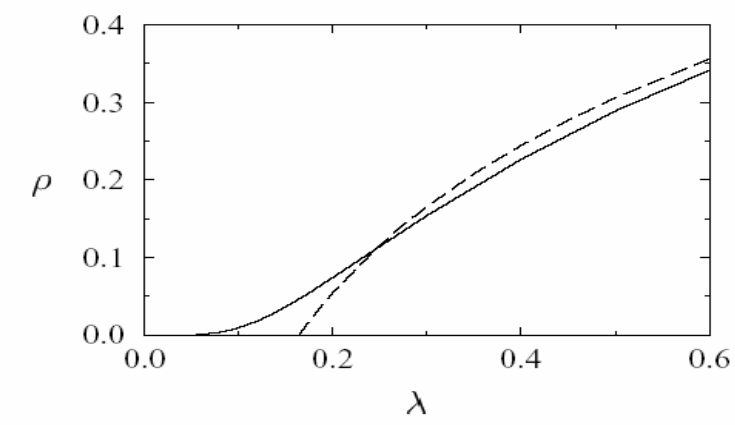

Рис. 3.16. Влияние топологии сети на поведение SIS-модели. Сплошной линией показана функция распространённости в SF-сетях, пунктиром – в однородных графах

Работы показывают, что протекание эпидемического процесса находится в весьма сильной зависимости от узлов сети с самой высокой степенью связности. Предполагается, что прицельная иммунизация таких субъектов значительно эффективнее случайной. Процесс обновления систем защиты демонстрирует именно случайную иммунизацию.

SI-модель в однородных графах.

A. Так как принципалы не восстанавливаются после атаки, то

$$\frac{\partial \rho}{\partial t} = \beta\rho(1-\rho). \qquad (3.12)$$

Очевидно, что при любой ненулевой скорости заражения вирус рано или поздно распространится по всем субъектам.

Решением этого уравнения будет

$$\rho(t) = 1-(1-\rho_0)e^{-\beta t}, \qquad (3.13)$$

где ρ_0 – начальная распространённость вируса.

B. Вирусы в таких сетях распространяются крайне быстро, поскольку заражённый принципал соседствует со всеми остальными принципалами в сети.

SI-модель в SF-сетях.

A. Принимая во внимание разброс степеней связности, модифицируем дифференциальное уравнение (3.12) в

$$\frac{\partial \rho_k}{\partial t} = \beta k (1 - \rho_k)\theta(\beta). \qquad (3.14)$$

где ρ_k – относительная распространённость вируса, k – число соседних вершин, $\theta(\beta)$ – вероятность заражения принципала с числом соседних вершин k.

Абсолютная распространённость рассчитывается как

$$\rho = \sum_k \rho_k . \qquad (3.15)$$

B. Распространённость инфекции на ранней фазе в больших SF-сетях меньше до определённого момента времени, затем она резко возрастает, так как степень связности «ключевых» вершин в больших сетях значительно превосходит таковую в сетях меньшего размера; эта разница в скорости становится очевиднее при малых β.

Контрольные вопросы и задания

1. Разработайте программу, реализующую симуляцию моделей, приведенных в данной главе.
2. В каком случае необходимо пользоваться поправками, введенными для распространения вредоносных программ в SF-сети?
3. Проведите исследование возможности замены фиксированных частот заражения, иммунизации и т.п. на функции от числа зараженных вычислительных машин. Используйте наиболее подходящий, на ваш взгляд, тип функции.
4. Соберите статистику и проведите анализ точности моделирования процесса распространения ВП с помощью моделей, приведенных выше. Сделайте выводы о возможностях применения данных моделей для описания реальных процессов.
5. Подходят ли данные модели для анализа распространения вируса по ресурсам одной вычислительной машины? Как можно адаптировать модели для данного случая?

ЗАКЛЮЧЕНИЕ

Процесс развития вредоносных программ и средств противодействия им – это постоянная война технологий. Регулярно в вирусах реализовываются оригинальные идеи, что требует адекватных действий от разработчиков антивирусного ПО. Приведем некоторые примеры, характеризующие состояние изучаемого вопроса на середину-конец 2009 г., когда заканчивалось оформление рукописи пособия.

Основной тенденцией было появление нескольких эксплойтов, использующих уязвимости ОС Windows и другого популярного ПО. В аналитическом обзоре компании «Доктор Веб» отмечалось распространение в России вируса JS.Gumblar, и эта угроза остается до сих пор актуальной. Начало осени ознаменовалось вредоносной активностью червей семейства Win32.HLLM.MyDoom, которые использовались в DDoS-атаках на южнокорейские и американские веб-ресурсы. Появился новый червь для мобильных телефонов под управлением ОС Symbian Series 60 3rd Edition, распространяемый с веб-ресурсов в виде заманчивого ПО для пользователя, а уже зараженные пользователи рассылают СМС-спам от своего имени по «своим» контактам. Была обнаружена уязвимость в Microsoft DirectX, использующемся MS Internet Explorer версии 6 и 7, которой оказались подвержены пользователи ОС Windows 2000/2003/XP. Уязвимость была использована для распространения ВП с помощью специально сформированного злоумышленниками веб-сайта, вызывающего переполнение стека и запускающего вредоносное ПО на целевой системе. (Отметим, что для перечисленных уязвимостей уже выпущены патчи). В конце лета увеличилась вредоносная активность червя Win32.HLLW.Facebook (Koobface) в социальной сети Twitter. И это далеко не полный перечень «новостей».

Рядовому пользователю авторы настоятельно рекомендуют следить за новостями на сайтах антивирусных компаний и прислушиваться к советам специалистов по информационной безопасности о необходимости обновления программного обеспечения (не только антивирусного) или выполнении специфических действий по улучшению защищённости компьютера.

Рекомендательный библиографический список

1. Брэгг, Р. Безопасность сетей. Полное руководство / Р. Брэгг, М. Родс-Оусли, К. Страссберг. – М. : Эком, 2006. – 912 с. – ISBN 5-7163- 0132-0.

2. Гордон, Я. Компьютерные вирусы без секретов / Я. Гордон. – М. : Новый издательский дом, 2006. – 320 с. – ISBN 5-9643-0044-8.

3. Гошко, С.В. Энциклопедия по защите от вирусов / С.В. Гошко. – Изд. 2-е, стер. – М. : СОЛОН-Р, 2005. – 352 с. – ISBN 5-98003-196-0.

4. Девянин, П.Н. Анализ безопасности управления доступом и информационными потоками в компьютерных системах / П.Н. Девянин. – М. : Радио и связь, 2006. – 176 с. – ISBN 5-256-01768-3.

5. Касперски, К. Записки исследователя компьютерных вирусов / К. Касперски. – СПб. : Питер, 2004. – 320 с. – ISBN 5-469-00331-0.

6. Касперски, К. Компьютерные вирусы: изнутри и снаружи / К. Касперски. – СПб. : Питер, 2005. – 528 с. – ISBN 5-469-01282-4.

7. Коваль, И.М. Как написать компьютерный вирус / И.М. Коваль. – М. : Символ-Плюс, 2000. – 192 с. – ISBN 5-93286-006-5.

8. Козлов, Д.А. Энциклопедия компьютерных вирусов / Д.А. Козлов, А.А. Парандовский, А.К. Парандовский. – М. : СОЛОН-Р, 2001. – 464 с. – ISBN 5-93455-091-8.

9. Норткатт, С. Обнаружение вторжений в сеть. Настольная книга специалиста по системному анализу / С. Норткатт, Д. Новак, Д. Маклахлен. – М. : ЛОРИ, 2003. – 384 с. – ISBN 8-5582-147-1.

10. Собейкис, В.Г. Азбука хакера 3. Компьютерная вирусология / В.Г. Собейкис. – М. : Майор, 2006. – 512 с. – ISBN 5-98551-013-1.

11. Столингс, В. Основы защиты сетей. Приложения и стандарты / В. Столлингс. – М. : Вильямс, 2002. – 432 с. – ISBN 5-8459-0298-3.

12. Daley, D.J. and Gani, J. Epidemic Modeling: An Introduction. Cambridge University Press, 1999. – 226 p.

13. Frauenthal, J.C. Mathematical Models in Epidemiology. – New York : Springer-Verlag, 1980. – 335 p.

Интернет-источники

14. http://citeseer.ist.psu.edu/anderson94next.html
15. http://www.ieee-infocom.org/2003/papers/46_03.PDF
16. http://europe.cnn.com/2003/TECH/internet/01/25/internet.attack/
17. http://www.renesys.com/projects/bgp instability/
18. http://citeseer.ist.psu.edu/chess00undetectable.html
19. http://www.arxiv.org/PS_cache/cond-mat/pdf/0107/0107420.pdf
20. http://www.ieee-infocom.org/2003/papers/46_01.PDF
21. http://www.esecurityplanet.com/trends/article.php/329246
22. http://lists.jammed.com/incidents/2001/07/0149.html
23. http://www.leshatton.org/RS_1001.html
24. http://nms.lcs.mit.edu/papers/scanworm.pdf
25. http://www.hpl.hp.com/techreports/2002/HPL-2002-287.pdf
26. http://www.suntimes.com/output/tech/cst-fin-lundy22w.html
27. http://www.caida.org/outreach/papers/2002/codered/codered.pdf
28. http://www.icir.org/vern/papers/radiation-imc04.pdf
29. http://www.hpl.hp.com/techreports/2003/HPL-2003-39.pdf
30. http://www.brandeis.edu/gradjournal

Оглавление