УДК 004.056

Н. В. Федотов

О ПОДХОДЕ К ОБНАРУЖЕНИЮ КОМПЬЮТЕРНЫХ АТАК, НАПРАВЛЕННЫХ НА УВЕЛИЧЕНИЕ ПРИВИЛЕГИЙ

Рассмотрена методология построения системы, предназначенной для определения компьютерных атак в исходных текстах программ, разработанных современными средствами программирования.

В настоящее время за рубежом проводятся исследования, направленные на совершенствование методов и средств обнаружения компьютерных атак.

В настоящей работе проведена оценка результатов исследований зарубежных специалистов, представленных в работе [1], по созданию системы, предназначенной для выявления атакующего кода (attack code), встроенного в исходные тексты программ, которые созданы на языке программирования СИ (C source code) или с помощью средств, входящих в состав общедоступных оболочек программных сред (shell code).

Система осуществляет сканирование исходного текста программы, разбивает его на составные части, каждая из которых описывается набором характеристик (features), подсчитывает статистики появления этих характеристик (feature statistics) и оценивает вероятность содержания в тексте программы атакующего кода.

Создание системы осуществлялось на основе анализа массивов нормального (без атакующего кода) и атакующего исходных текстов программ (normal and attack software). Каждый из этих массивов включал “обучающий” и “испытательный” наборы исходных текстов программ. При этом анализ текста и описание его характеристик осуществлялись на основе “обучающего” набора (training corpus), а тестирование исходного текста — на основе “испытательного” (test corpus).

Результаты испытаний показали, что атакующий код может быть достаточно точно определен.

Принципы построения системы, предназначенной для выявления атакующего кода. Различные компьютерные атаки выполняются за разное время и число шагов.

Атака, направленная на отказ в обслуживании (denial-of-service attack), как правило, осуществляется за продолжительное время (в те-

чение часов или дней). В течение этого времени система обнаружения вторжения может выработать сигнал тревоги.

Атака, направленная на увеличение привилегий (privilege-increasing attack), может быть произведена всего за несколько шагов в течение короткого интервала времени. В этом случае системами обнаружения вторжения, основанными на обработке данных аудита, атака обнаруживается с задержкой по времени. Это связано с тем, что необходимая для выработки сигнала тревоги информация накапливается в файлах аудита не сразу, а с течением времени. Нападающий (attacker) может использовать это время в целях получения дополнительных привилегий.

Среди атак, направленных на увеличение привилегий, можно выделить два типа атак. В случае атаки первого типа целью является обеспечение доступа в систему неавторизованного пользователя; в случае атаки второго типа осуществляется предоставление прав доступа привилегированного пользователя обычному пользователю.

Как правило, при нападении вначале на атакованном хосте (host) используют авторизованный доступ в систему. Для того чтобы произвести атаку, нарушитель должен выполнить следующие действия: загрузить или создать исходный текст программы, компилировать его и затем использовать для атаки.

Не все действия могут быть выполнены на атакованной машине. Нападающий может создать и компилировать атаку на другом компьютере, а затем загрузить исполнительный код на атакованном хосте. Однако, с точки зрения надежности и безотказного выполнения исполнительного модуля с атакующим кодом, целесообразно проводить его компиляцию и выполнение на одном компьютере. В силу этого атакующий может использовать исходный текст программы на атакованной машине.

В результате исследований, проведенных в работе [1], было определено, что атакующий исходный код, разработанный либо на языке СИ, либо средствами из состава оболочек, отличается от нормального и может быть точно определен в исходном тексте программы.

Системы обнаружения вторжения, в основном, строятся на основе методов машинного обучения и, в частности, нейронных сетей. Исследования таких систем направлены на обнаружение атак после того, как они произошли. Средства обнаружения вирусов определяют атаки перед тем, как они начали выполняться. Работа [1] в большей степени связана с исследованием по обнаружению вирусов.

И в системах обнаружения вторжения, и при обнаружении вирусов общим алгоритмом является метод проверки сигнатуры, при котором осуществляется сканирование модулей или выполняемых действий с

помощью известных сигнатур атак. В некоторых системах используются характеристики, описывающие состояния системы (шаги), которые соответствуют атаке.

Работа [1] уникальна тем, что в ней представлена система для определения в исходном тексте программы кода атаки, направленного на увеличение привилегий в UNIX-подобной системе. Принцип функционирования этой системы заключается в следующем. Во входящем потоке, полученном при сканировании файла, определяется запрос *write* ("запись"). После этого файл (модуль) классифицируется по типу языка, на котором он написан: языка СИ, языка из состава оболочки или другого. В последнем случае, если классификатор не может распознать тип языка, дальнейший анализ прекращается. Если язык распознан, дальнейший анализ выполняется детектором атаки для определенного языка. В систему могут быть добавлены дополнительные детекторы для расширения ее возможностей.

Каждый детектор включает в себя две подсистемы, работающие последовательно: экстрактор характеристики, который формирует вектор нормализованной статистики характеристики, и классификатор атаки на основе нейронной сети, который передает дополнительную информацию в систему обнаружения вторжения, когда атака распознана. При функционировании системы и выполнении модуля в данном файле запрос *write* блокируется, и система обнаружения вторжения вырабатывает тревогу.

Предварительные результаты применения описываемого метода, которые показывают эффективность данного подхода, опубликованы в работе [2]. Работа [1], в основном, посвящена дальнейшим исследованиям по построению точных классификаторов языка и атаки. В ходе этих исследований был значительно расширен (по сравнению с результатами работы [2]) состав "обучающего" и "испытательного" наборов. Число модулей с атакующим кодом было увеличено почти в десять раз. При этом "испытательный" набор включал содержащие атакующий код исходные тексты программ, созданные после проведенного анализа "обучающего" набора, что позволило оценить эффективность предложенного метода обнаружения новых атак. На основе нового "обучающего" набора расширен перечень характеристик и модифицирован метод нормализации статистик.

Система, предлагаемая в работе [1], позволяет снизить уровень ложных тревог в два раза и уровень пропусков в шесть раз (по сравнению с результатами, представленными в работе [2]).

Для оценки скорости, с которой осуществляется определение атакующего кода, авторами работы [1] создана интегральная система.

Исходные данные, используемые для построения системы. Необходимо собрать как можно больше образцов исходных текстов программ для анализа и построения системы, способной определить небольшое количество строк, содержащих атакующий код, среди большого количества строк, содержащих нормальный. С этой целью было собрано большое количество программ, содержащих нормальный и атакующий код из открытых проектов (open-source projects) и хакерских сайтов (hacker web sites). Эти программы были написаны специалистами из различных стран с различными уровнями подготовки, стилями программирования. Каждый отдельный файл, включенный в массив, анализировался экспертом по специальной методике.

Исходные данные для случая, когда исходный текст программ написан на языке СИ. Массив программ, содержащих нормальный код, состоял из модулей, которые выполняли большой набор различных задач, включая некоторые операции, которые может выполнять атакующий. В состав этого массива входили программы, состоящие как из одного файла, так и из взаимосвязанных модулей.

Набор "обучающих" программ, содержащих нормальный код и созданных на языке СИ, включал в себя 5271 файл. В состав этого массива входили: веб-сервер (apache_1.3.12); командная оболочка (bash-2.04); небольшие программы (fileutils-4.0, sh-utils-2.0); программа обработки почты (sendmail-8.10.0); инструментарий разработчика (binutils-2.10); компилятор (flex-2.5.4); отладчик (gdb-4.18); программное обеспечение интегральной пользовательской среды (emacs-20.6); библиотека машинно-ориентированных кодов (glibc-2.1.3).

Набор "испытательных" программ, содержащих нормальный код, включал в себя 3323 файла, которые были получены после разработки классификатора. В состав этого массива входили: ядро операционной системы (linux-2.4.0-test1); программа управления компакт-диском (eject-2.0.2); средства мониторинга использования системы (top-3.4) и использования сети (ntop v.0.3.1); инструментарий (ssh-2.4.0), обеспечивающий шифрование связей типа "точка–точка" (peer-to-peer communications).

Массив программ, содержащих атакующий код и созданных на языке СИ, включал файлы из различных источников сети Internet. После испытаний было обнаружено, что один и тот же файл, содержащий атакующий код, находился в различных источниках в разной форме. Например, текст программы был один и тот же, а секции комментариев — различные.

Анализ программ, содержащих атакующий код, показал, что не все исходные тексты этих программ работоспособны. Во многих случаях

модули были тривиально разрушены, и изменение некоторых символов позволило осуществить их компиляцию и выполнение.

При формировании каждого массива выполнялся тест на уникальность файлов: сравнивались друг с другом комментарии для различных модулей массива (или их отсутствие). Если файл был уникальным, он включался в массив. Благодаря этому тесту не в полной мере предотвращалось наличие в массиве несущественных модификаций файлов, но ограничивалось число их точных дублей. Уникальность требовалась для всех файлов массива.

При формировании каждого массива были использованы те программы, содержащие атакующий код, относительно которых предполагается, что их реализация будет успешной.

Массив программ, содержащих атакующий код, также разделялся на "обучающий" и "испытательный" наборы программ.

Набор "обучающих" программ включал в себя 469 файлов. Он состоял из уникальных программ, содержащих атакующий код, доступных на сайтах сети Internet, а также программ, использующих дефекты в программном обеспечении, заявленных с 1.01.2000 по 15.10.2000 (BugTraq).

Набор "испытательных" программ, содержащих атакующий код, включал в себя 67 файлов, полученных на сайтах [3, 4], и все программы, заявленные с 16.10.2000 по 31.12.2000 (BugTraq).

Оба набора программ, содержащих атакующий код, включали в себя реализации атак на UNIX-подобные системы, в том числе, на основе операционных систем Linux, HPUX, Solaris и BSD. Файлы исходных текстов этих программ содержали комментарии и различные слова на европейских языках.

Исходные данные для случая, когда исходный текст программ написан с помощью средств, входящих в состав оболочек. Набор "обучающих" программ, содержащих нормальный код, включал в себя 476 файлов, собранных из оболочки SHELLdorado [5], загрузочные скрипты операционной системы RedHat 6.1 (содержимое директорий init.d и rc*.d), скрипты (scripts) оболочек Borne, Bash, Korn [6–8].

Набор "обучающих" программ, содержащих атакующий код, включал в себя 119 файлов: программы, заявленные с 1.01.2000 по 15.10.2000 (BugTraq), файлы с сайтов, используемых для формирования массива программ, написанных на языке СИ [9–13], некоторые образцы атак из сети Internet начиная с 1996 г.

Набор "испытательных" программ, содержащих нормальный код, включал в себя 650 файлов Redhat 7.1. Все файлы дерева директорий (directory tree) были сканированы для проверки содержания в их пер-

вой строке символов "#!" и подтверждения их принадлежности составу оболочки. При этом каждый файл проверялся на уникальность.

Набор "испытательных" программ, содержащих атакующий код, состоял из 33 файлов для атак, направленных на увеличение привилегий (эти файлы были получены с тех же сайтов, что и файлы, написанные на языке СИ), а также программ, заявленных с 16.10.2000 по 31.12.2000 (BugTraq).

Дополнительные файлы. С целью проверки реакции системы на иные типы файлов в дополнение к массивам, описанным выше, использовались 545 файлов, которые не являлись написанными на языке СИ или созданными средствами, входящими в состав оболочек. Эти файлы включали обработанные архиватором и компрессором файлы (archived and compressed files) из набора "испытательных" программ, написанных на языке СИ и содержащих нормальный код. При создании этих файлов использовались программные средства TAR, GNU gzip, bzip, compress, zip.

В состав каждого массива также входили файлы документов специального формата (html, postscript, pdf, UNIX mbox) и зашифрованные файлы [14].

Общий обзор системы. С целью сокращения вычислений вначале определялся тип языка входящего потока. Применение такого подхода может привести к пропуску атаки, но, вероятно, не должно привести к увеличению числа ложных тревог.

После того как язык определен, используется детектор атаки, соответствующий этому языку. Он извлекает характеристики и классифицирует исходный код как нормальный или вредоносный.

Если наличие атаки определено, об этом оповещается система обнаружения вторжения. В дальнейшем эта информация используется таким образом, чтобы пользователь мог распознать атакующие действия и предполагаемую цель атаки.

Определение языка, на котором написаны файлы, содержащие атакующий код. Определение языка осуществлялось системой, построенной на основе правил. При этом использовалось описание структуры и синтаксиса анализируемого языка.

Принципы построения правил определения следующие. Исследуемый исходный текст классифицировался как созданный на языке СИ в случае присутствия в нем директивы препроцессора языка СИ, комментария или зарезервированного слова (не являющегося словом английского языка или оболочки), которые используются при написании программ в среде СИ или СИ++ (C++).

Исходный текст классифицировался как текст, созданный с помощью средств, входящих в состав оболочки, если в нем обнаружены

символы “#!”, символ комментария оболочки “#” или слово с префиксом “$”.

Кроме того, существовало несколько правил, в соответствии с которыми исходный текст классифицировался как нераспознанный (например, заголовок файла электронной почты, символы не ASCII- или ISO-кодов, заголовок html-файлов). Если заданное число проверяемых символов слова в файле исходного текста не было распознано, он также классифицировался как нераспознанный.

В соответствии с этими правилами файлы, созданные средствами makefile, Python, Perl, относятся к скриптам оболочки (shell scripts). Набор правил для классификации этих файлов может быть расширен. Однако, как показала практика, для таких файлов все же существует очень мало характеристик, соответствующих средствам оболочки, поэтому эти файлы классифицируются как обычный текст.

Кроме того, файлы, написанные на языках Java и СИ++, а также некоторые файлы, написанные на языке Fortran, в которых используется препроцессор языка СИ, будут относиться к написанным на языке СИ. Однако в этих файлах вектор характеристики имеет почти все элементы нулевые, и, таким образом, эти файлы классифицируются как не содержащие атакующий код.

В таблице представлены результаты работы классификатора языка на испытательном наборе модулей, описанном выше.

Результаты работы классификатора языка

Типы файлов	Общее количество обработанных файлов	Количество файлов, классифицированных как		
		созданные на языке СИ	созданные средствами, входящими в состав оболочки	нераспознанные
Созданные на языке СИ	$3323 + 67 = 3390$	3389	0	1
Созданные средствами, входящими в состав оболочки	$476 + 119 = 595$	0	594	1
Другие	545	0	0	545
Итоговые результаты	4530	3389	594	547

Результаты работы классификатора следующие. Из 3390 файлов, созданных на языке СИ, обработанных классификатором языка, один

файл был определен как нераспознанный, 3389 файлов были определены правильно. Общая ошибка определения составляет 0,04%.

Файл, неправильно определенный классификатором языка как созданный на языке СИ, входил в состав большого приложения и содержал единственную строку кода, написанного на языке СИ. Он также содержал некоторые зарезервированные слова, являвшиеся словами английского языка. Аналогично, файл, неправильно определенный классификатором как входящий в состав оболочки, не содержал каких-либо команд, используемых в средствах оболочки, и комментариев.

Система функционировала достаточно быстро. В эксперименте использовался компьютер SPARC Ultra 60 с тактовой частотой 450 МГц. При определении языка для обработки одного килобайта данных требовалось 90 мкс.

Детектирование атаки. Как только тип языка исходного текста определен, необходимо выполнить проверку текста на присутствие характеристик, соответствующих файлам, содержащим атакующий код.

Извлечение характеристик (feature extraction) осуществляется в два этапа. На первом этапе исходный текст разбивается на части (кодовые категории), каждая из которых проверяется своим набором характеристик, на втором — определяются статистики появления характеристик (feature statistics). Статистики появления каждого типа характеристик (feature type) определяются так называемым экстрактором характеристики (feature extractor).

Каждый тип характеристики представляет собой регулярное выражение (regular expression), с помощью которого сканируется каждая кодовая категория (code category) исходного текста. Каждый раз, когда регулярное выражение находится в исходном тексте, подсчитывается статистика появления характеристик по определенной схеме кодирования (encoding scheme).

Каждый тип характеристики может быть представлен набором троек (set of triples): регулярное выражение, кодовая категория, схема кодирования (регулярное выражение применяется к определенной кодовой категории с использованием определенной схемы кодирования). Большинство типов характеристик описываются одной тройкой, но могут содержать информацию из различных кодовых категорий и описываться набором троек.

Вначале экстрактор разделяет исходный текст на четыре кодовых категории: *comments* (*/** A comment **/*), *strings* (“A string”), *code-sans-strings* (printf( );), *code* (printf(“A string”);). Первые три кодовые категории не связаны между собой, четвертая включает предыдущие две.

Когда регулярное выражение применяется в определенной кодовой категории, оно использует одну из следующих схем кодирования: схему

once (определяет появление регулярного выражения в составе исходного текста); схему *count*: (определяет количество появлений регулярного выражения); схему *normalized*: (определяет нормализованное значение количества появлений регулярного выражения по отношению к длине исходного текста — результат деления количества появлений на длину в байтах кода файла).

Детектор атаки строится путем создания большого набора троек, соответствующих определенному языку. В дальнейшем процесс выявления характеристик (feature-selection process) определяется этими тройками (подробное описание этого процесса изложено в работе [15]). Из всех N характеристик выбирается одна, которая соответствует самой малой величине ошибки. Эта характеристика формирует одноразмерный вектор. После этого из оставшихся $N-1$ характеристик выбирается следующая, соответствующая минимальной ошибке. Таким образом, формируется двухразмерный вектор. Процесс продолжается до тех пор, пока не рассмотрены все характеристики и не определен вектор размерностью N. Вектор, который минимизирует общую ошибку, называется "лучшим" (best) и используется в дальнейшем при детектировании атаки. Авторами работы [1] использовался разработанный ими пакет программ LNKnet [16, 17]. При вычислении статистических данных использовалось правило нормализации.

Детектор атаки для исходных текстов, написанных на языке СИ. Для построения детектора в работе [1] были определены 19 типов характеристик, каждая из которых соответствовала некоторому типу атаки. В первую очередь, учитывалось присутствие в исходном тексте слов "реализовать" ("exploit") или "уязвимость" ("vulnerability") и строилось регулярное выражение, с помощью которого исходный текст сканировался для определения наличия этих слов в комментариях.

Авторы работы [1] пришли к выводу, что при атаке, направленной на получение привилегий, используется создание и ликвидация связей с другими файлами. Для сканирования текста с целью определения наличия функций *link, unlink, rmdir* в кодовой категории *code-sans-strings* в работе [1] было предложено соответствующее регулярное выражение.

Поскольку атакующие при реализации атаки, направленной на увеличение привилегий, могут использовать переменную среду (environment variable), в работе [1] было предложено регулярное выражение для определения функций, которые ее изменяют, и использовано для сканирования кодовой категории *code-sans-strings*.

Атакующие, как правило, пытаются загрузить для выполнения вредоносный код (malicious code) в привилегированных программах. В работе [1] предложены регулярные выражения для его определения (либо исполняемый код, встроенный в кодовые категории *code-sans-strings*

или *strings*, либо текст, написанный на языке СИ или средствами оболочки и встроенный в кодовую категорию *strings*), а также определения необходимых компонентов для реализации этой атаки (ключевого слова языка Ассемблер для включения в стек (stack) в кодовой категории *code-sans-strings*; вызовов функций управления системным журналом (syslog) в кодовой категории *code-sans-strings*; функций, подобных функциям strcpy и memcpy в кодовой категории *code-sans-strings*; функций ptrace в кодовой категории *code-sans-strings*).

В работе [1] также предложены регулярные выражения для определения самих атакующих действий, например вызовов функций *chown*, *setuid*, *passw*, *shadow*, *system*, *exec*.

Регулярные выражения были созданы для определения функций, используемых для получения локального имени хоста (local host name) и определения наличия главной функции (main function) в кодовой категории *code-sans-strings*.

Предложены также регулярные выражения для определения наличия в тексте обращений к локальным файлам.

Результатом формирования характеристики являлось значение вектора статистик характеристик (vector of feature statistics), которое поступало на вход детектора, построенного на основе нейронной сети. По комбинации элементов он определял наличие или отсутствие атакующего кода в исходном тексте. Детектор представлял собой многоуровневый перцептрон (multi-layer perceptron) нейронной сети (neural network), для которого размерность пространства характеристик равна 19.

Детектор атаки может быть двух типов: with-comment и sans-comment. Детектор первого типа может быть использован для защиты от неподготовленного нападающего, детектор второго типа — для защиты от более подготовленного и является прообразом детектора двоичного кода.

Детектор первого типа наиболее эффективно определял наличие атакующего кода, встроенного в файл, для характеристик, сформированных при обработке комментария, вызовов функции “выполнить” (exec), имен локальных файлов. Детектор второго типа наиболее эффективно определял наличие атакующего кода для характеристик, которые соответствовали встроенному исполнительному коду, вызовам функций типа “выполнить” (exec), вызовам для обращения к системным файлам.

Эксперимент показал, что в случае использования результатов детектирования атак система обнаружения вторжения может предотвратить значительное количество атак прежде, чем они начали осуществляться.

Детектор, предложенный в работе [1], функционировал на компьютере SPARC Ultra 60 с тактовой частотой 450 МГц и анализировал килобайт данных за 666 мкс. При этом на анализ исходного кода, представленного на языке СИ, было затрачено 77 % времени, на чтение файлов — 21 %, на идентификацию атаки — остальное время.

Детектор атаки для исходных текстов, созданных средствами оболочек. Исходный текст, как и ранее, разделялся на четыре части (кодовые категории). Были рассмотрены атакующие действия и скорректированы регулярные выражения для моделирования синтаксиса оболочек. Были созданы специфичные характеристики для кода, составленного средствами оболочек.

Атакующий может использовать модуль из состава оболочки для добавления в систему новых пользователя или гостевого пароля (guess password). Поэтому было создано регулярное выражение для определения доступа к файлам пароля (password file) и теневого пароля (shadow password file).

Атакующий может внести вредоносный код в файл функциональной среды или в стек с целью аварийного завершения привилегированной программы, перемещать эти файлы и обращаться к другим, чтобы скрыть их изменение. Было разработано регулярное выражение для определения этих действий. Осуществлялось также сканирование исходных текстов с целью определения изменения переменных среды (altering environment variables) и создание объекта в используемой совместно библиотеке (creating a sharing library object).

Атакующий иногда стремится получить доступ для использования программы из состава оболочки, работающей в привилегированном интерактивном режиме (privileged interactive shell). Для определения этих действий также было создано регулярное выражение.

Существуют атаки, которые направлены на изменение настроек локальной безопасности хоста, изменение содержимого файлов типа rhost или hosts. Данные в измененных файлах атакующий может использовать для подключения к хосту. Для определения этих действий были созданы регулярные выражения.

При детектировании атак применялся метод обратного выявления характеристик (backward feature selection). Метод описан в работе [15].

Для файлов, содержащих комментарии, наибольшая эффективность детектирования была достигнута при использовании 15 характеристик, сформированных для сканирования исходных текстов с целью определения наличия в нем следующих функций: *localhost, copy, passw, link, root, test, core, exec, trusted, chown, touchr, set[ug]id, interactive, shared*.

Для файлов, в которых отсутствовали комментарии, наибольшая эффективность детектирования была достигнута при использовании

12 характеристик, сформированных для сканирования текста с целью определения наличия в нем следующих функций: *code, localhost, set[ug]id, passw, root, link, chown, copy, exec, touchr*, а также функций обращения к другой оболочке и выполнения встроенного исполнительного кода.

Значение вектора статистики поступало на вход детектора, построенного на основе нейронной сети. Детектор представлял собой многоуровневый перцептрон нейронной сети (neural network), для которого размерность пространства характеристик равна 16 или 12 в зависимости от использования или неиспользования секции комментариев.

Атакующий код в файлах, входящих в состав оболочки, точно определить значительно труднее, чем в файлах, созданных на языке СИ.

Детектор, предложенный в работе [1], функционировал на компьютере SPARC Ultra 60 с тактовой частотой 450 МГц и анализировал килобайт данных за 1,071 мкс. При этом на анализ исходного текста потребовалось 77% времени, на чтение файлов — 21%, на детектирование атаки — 1% и на другие действия — 3%.

На ввод/вывод данных потребовалось большее время ввиду того, что модули из состава оболочки меньше по размеру (в байтах), чем модули, созданные на языке СИ. Процесс детектирования для модулей из состава оболочки также занимал больше времени, поскольку регулярные выражения в этом случае были более сложными.

Результаты испытаний системы. Для проведения экспериментов авторами работы [1] был создан инструментарий сканирования файлов.

Испытания системы проводились на сервере, на котором использовались разнообразные программные средства. Ожидалось большое количество ложных тревог, так как многие модули имели характеристики, похожие на характеристики программ, содержащих атакующий код.

Сервер содержал 4 880 452 файла общей длиной 137 044 936 Кб. Файлы длиной более 1 Мб сканированию не подвергались. Идентификатор языка определил 36 600 файлов с исходным текстом на языке СИ и 72 849 файлов, созданных средствами оболочек, т.е. 109 449 файлов были выделены для дальнейшего анализа.

Детектор атак определил 4836 файлов, содержащих атакующий код, из которых 3855 ($\sim$10%) являлись написанными на языке СИ и 981 ($\sim$1%) — созданными средствами оболочек.

Дальнейший анализ показал, что относительно 143 файлов детектором выработаны неправильные решения, при этом для файлов, написанных на языке СИ — в 33 случаях, а для файлов, созданных средствами оболочки — в 110 случаях.

На чтение файлов потребовалось 54% общего времени эксперимента, на определение языка — 16%, на определение присутствия атаку-

ющего кода в файлах, написанных на языке СИ, — 8,7% и в файлах, созданных средствами оболочки, — 14%, на другие действия — 6,8%. Большое количество времени, затраченное на идентификацию языка, объясняется тем, что многие файлы при испытаниях не являлись написанными на языке СИ и созданными средствами оболочки.

Выводы. По мнению авторов работы [1], вследствие того, что предложенный подход основан на анализе большого количества исходных текстов программ, содержащих общедоступный код, который используется в UNIX-подобных системах, разработанная система позволяет точно определить атаки, направленные на увеличение привилегий.

Гибкость синтаксиса оболочек затрудняет определение наличия атакующего кода в модулях из состава оболочек.

Правильно идентифицировать модули, входящие непосредственно в состав оболочки, сложнее, чем модули, созданные на языке СИ.

Исходные тексты, созданные средствами (скриптами) оболочек, менее структурированы, чем исходные тексты, созданные на языке СИ, поэтому перечень характеристик и время их обработки больше.

Как отмечается в работе [1], рассмотренная система для определения атак не является достаточно совершенной и атакующий может обойти ее. Например, нападающий может снизить значение статистики появления в файле определенной характеристики. Для этого он может создать в файле большое количество исходного текста, которое не соответствует ни одной характеристике, разбить атакующий код на подпрограммы и поместить эти подпрограммы в различные файлы. Он также может создать небольшую программу, которая будет удалять характеристики в файле, не позволяя проводить его дальнейший анализ.

Для устранения недостатков системы авторы работы [1] планируют провести исследования с целью расширения количества регулярных выражений и проведения мультифайлового анализа.

В работе [1] рекомендовано несколько способов использования рассмотренной системы. При применении сетевых систем обнаружения вторжения она может быть использована для мониторинга данных, передаваемых FTP-, www-сервисами и средствами e-mail.

Для систем обнаружения вторжения в хостах сканирование может периодически выполняться для файлов, находящихся на диске, или данных входящего трафика.

Например, в операционной системе FreeBSD периодически загружаются на исполнение средства проверки безопасности файловой системы. В этом случае может использоваться рассмотренная система.

Возможна также доработка ядра операционной системы с целью сканирования файла, когда запрошена операция его записи на диск.

Таким образом, атакующий код в файлах, содержащих исходные тексты, написанные на языке СИ или с помощью средств из состава оболочек, отличается от нормального кода и может быть точно определен. Если в файле имеется текстовая информация, то точность определения наличия атакующего кода повышается.

СПИСОК ЛИТЕРАТУРЫ

1. Cunningham R. K., Stevenson C. S. Accurately Detecting Source Code of Attacks That Increase Privilege // Recent Advances in Intrusion Detection. – 2001 (10–12 Oct.).
2. Cunningham R. K., Rieser A. Detecting Source Code of Attacks that Increase Privilege // Recent Advances in Intrusion Detecton. – 2000 (1–4 Oct.).
3. http://www.fakehalo.org/. – 20.12.2000.
4. http://www.uha1.com/. – 13.12.2000.
5. http://www.oase-shareware.org/. – 2000.
6. Blinn B. Portable Shell Programming: An Extensive Collection of Bourne Shell Examples. – London: Prentice Hall, 1995.
7. Newham C., Rosenblatt B. Learning the Bash Shell. – O'Reilly & Associates, 1998.
8. Rosenblatt B., Loukides M. Learning the Korn Shell. – O'Reilly & Associates, 1993.
9. http://www.rootshell.com/. – 15.10.2000.
10. http://www.hack.co.za/. – 30.10.2000.
11. http: //www.lsd-pl.net/. – 24.10.2000.
12. ftp://ftp.technotronic. com/. – 1.11.2000.
13. http://www.anticode.com/. – 15.10.2000.
14. http://www.gutenberg.net/. – 1990.
15. Fukunaga K. Introduction to Statistical Pattern Recognition. – Academic Press, 1990.
16. Kukolich L., Lippmann R.: LNKnet User's Guide // http://www.ll.mit.edu/IST/lnknet/. – 2000.
17. Lippmann R., Kukolich L., Singer E. LNKnet: Neural Network, Machine Learning, and Statistical Software for Pattern Classification // Lincoln Laboratory Journal. – 1993. – № 6. – P. 249–268.

Статья поступила в редакцию 25.03.2004

Николай Васильевич Федотов родился в 1945 г., окончил в 1977 г. Всесоюзный заочный электротехнический институт связи. Канд. воен. наук, профессор Академии военных наук. Автор 23 научных работ в области информационной безопасности.

N.V. Fedotov (b. 1945) graduated from the All-Union Electro-technical Communication Institute by Correspondence in 1977. Ph. D. (Military), professor of Academy of Military Sciences. Author of 23 publications in the field of information safety.