ПРОБЛЕМЫ И ПЕРСПЕКТИВЫ ИСПОЛЬЗОВАНИЯ СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ В ЗАДАЧАХ ЗАЩИТЫ ИНФОРМАЦИИ

Чепик Н.А., Зайчик А.Ю., Козырский Б.Л., Комаров Т.И., Максутов А.А., Смирнов А.А.

Национальный исследовательский ядерный университет «МИФИ» (НИЯУ МИФИ)

Введение

Информационная безопасность давно стала самостоятельным направлением исследований и разработок, в рамках которого идет непрерывная война между теми, кто стремится, например, украсть секретные сведения или получить управление над каким-либо объектом, и теми, кто ставит своей целью обеспечение сохранности данных и исправной работы сервисов.

Развитие суперкомпьютерных технологий (СКТ) оказало существенное влияние на сферу информационной безопасности.

С развитием СКТ стало намного проще решать задачи полного или частично-полного перебора, а именно таковыми являются задачи, связанные с компрометацией систем защиты информации, например, задачи взлома криптоалгоритмов и криптопротоколов. Полный перебор вариантов это универсальный метод решения подобных задач, вероятность его успеха всегда равна единице.

Новый стимул в развитии получают атаки, основанные на использовании криптографии против криптографии. Многие скрытые каналы утечки информации из криптосистем для своего использования требуют именно решения задач частично-полного перебора. С появлением суперкомпьютеров требования к пропускной способности таких каналов существенно снижаются [4, 5]. РПВ, использующие скрытые каналы воздействия на объект, а также приема и передачи информации, уже появились.

Совершенствование методов поиска уязвимостей компьютерных систем (КС), создающих предпосылки для проведения атак, основанных на вставке кода, привело к увеличению фактов обнаружения РПВ, использующих уязвимости нулевого дня (Zero day vulnerabilities). Совершенно очевидно, что в самое ближайшее время будут обнаружены РПВ, которые при функционировании для затруднения своего выявления и нейтрализации применяют СКТ, в частности, гибридные, т.е. основанные на совместном использовании классических процессоров (CPU) и специализированных вычислительных модулей (NVIDIA Tesla, Intel Xeon Phi). В результате антивирусы, использующие традиционные реактивные методы защиты, окажутся не в состоянии справиться с новой угрозой. Необходимо опережающее совершенствование методов и средств защиты от РПВ.

И, разумеется, нельзя забывать о том, что суперкомпьютеры – это не только уникальный инструмент, который активно применяется в сфере информационной безопасности, но и объект потенциальных атак, защиту которого необходимо обеспечивать.

Операционные системы, под управлением которых работают суперкомпьютерные системы, используемое прикладное программное обеспечение – обычно это очень большие и сложные программные продукты, которые всегда содержат в себе некоторое количество ошибок и уязвимостей. Подобные изъяны в безопасности необходимо своевременно обнаруживать и устранять. Одним из способов решения данной проблемы является фаззинг – технология, которая заключается в автоматическом поиске уязвимостей методом «чёрного ящика». Ее основными преимуществами являются доступность, простота и воспроизводимость, которые невозможно получить при использовании других методов [1-3].

Фаззинг

Существует множество методов поиска уязвимостей ПО, каждый из которых обладает своими достоинствами и недостатками. Их можно разделить на три большие группы:

- метод белого ящика;
- метод чёрного ящика;

– метод серого ящика.

Метод белого ящика подразумевает наличие полного доступа к исходным кодам исследуемого программного приложения. Исходный код можно изучать вручную или же при помощи средств, осуществляющих автоматизацию данного процесса. Ручное исследование подходит для приложений небольших размеров, но вследствие того, что наиболее интересными объектами для поиска уязвимостей, как правило, являются сложные программы с большим объемом исходного кода, возникает необходимость в автоматических анализаторах. Широкому внедрению данного метода мешает тот факт, что большинство приложений распространяется без предоставления исходных кодов.

Метод черного ящика предполагает, что аналитик знает только то, что может увидеть собственными глазами как конечный пользователь программы. Чаще всего данный метод используется для анализа приложений с закрытым исходным кодом, веб-приложений и веб-сервисов. Как и в методе белого ящика допустимы два подхода: ручное и автоматизированное тестирование. Ручное тестирование в данном случае чрезвычайно трудоёмко и, как правило, осуществляется большим коллективом. Данных подход по силам только крупным компаниям и используется при выпуске дорогостоящих коммерческих продуктов (альфа- и бета-тестирование).

Среди существующих средств автоматизированного тестирования, использующих подход черного ящика, особо следует отметить фаззинг. Основной идеей данного метода является исследование поведения тестируемой программы при передаче ей на вход данных, которые могут быть совершенно любыми. И хотя приложения обычно создаются для работы со строго определенными входными данными, они должны быть устойчивыми, чтобы штатно обрабатывать любую входную информацию. Тенденция к значительному росту производительности и распараллеливанию вычислений делает фаззинг особенно эффективным. Исследователю для работы нужны лишь поверхностные знания о внутренних процессах и устройстве программ, т.е. фаззинг – это использование метода чёрного ящика в чистом виде.

Метод чёрного ящика, хоть и не всегда является наилучшим, практически всегда возможен. Его основными преимуществами является доступность, простота и воспроизводимость, которые невозможно получить при использовании других методов тестирования.

Основным недостатком метода чёрного ящика является невозможность проведения всестороннего анализа приложения: необходимо принимать непростое решение о том, как провести тестирование, когда его закончить и как оценить его результаты. Программы обычно обладают весьма сложной внутренней структурой, поэтому для проведения успешной комплексной атаки на приложение необходимо знание его устройства, что невозможно при данной методике проведения тестов.

Метод серого ящика является возможным компромиссом между методами чёрного и белого ящика. Основная концепция данного метода состоит в том, что исследуемый объект тестируется аналогично методу чёрного ящика, но наряду с этим, для понимания внутренней структуры тестируемого приложения, осуществляется реверс-инжиниринг при помощи различных утилит (дизассемблеров, отладчиков и декомпиляторов).

Основная идея фаззинга. Фаззинг – это способ обнаружения ошибок в ПО, в основе которого лежит подача на вход самых разнообразных данных и последующий анализ поведения приложения. Метод требует творческого подхода, а эффективность тестирования определяется лишь его результатом. Каждая отдельная задача проверки ПО требует индивидуального решения, которое учитывает особенности исследуемого объекта, имеющиеся аппаратные возможности, свойства конкретной ОС, наличие временных ресурсов и т.д.

Процесс тестирования методом фаззинга можно разделить на следующие этапы:

– определение цели;
– определение направлений ввода;
– подготовка данных;
– запуск программы с определенными данными;
– мониторинг исключений;
– исследование ошибок.

Определение цели. В том случае, когда аналитик имеет конкретное задание на исследование конкретного приложения, данный этап отпадает сам собой. Однако если речь идет о проверке целого программного комплекса (например, операционной системы), данный этап имеет решающее значение. В первую очередь необходимо проверить привилегированные приложения, наиболее распространённое ПО, а также те коды, которые входят в состав библиотек, используемых повсеместно.

Определение направлений ввода. На данном этапе потребуется подробное изучение документации по тестируемому приложению. Необходимо проанализировать каждый способ, при помощи которого приложение получает информацию извне. Направлениями ввода для обычной программы могут являться командная строка, переменные среды, имена и содержимое файлов, системные вызовы, данные, порождаемые другим ПО и т.д.

Подготовка данных. На этом этапе происходит подготовка данных, которые затем будут переданы тестируемому приложению. Существует множество подходов, каждый из которых имеет право на жизнь. Готовые наборы данных можно встроить в фаззер на этапе разработки или же добавить необходимые функциональные возможности для их генерации. Также следует обратить особое внимание на то, что многие программы проводят самые разнообразные проверки вводимых данных. Следовательно, данные, генерируемые фаззером, должны успешно проходить данные тесты для осуществления более полноценного и эффективного тестирования.

Запуск программы с определенными данными. Наиболее важный этап тестирования. Задачей разработчика системы фаззинга является максимальная оптимизация именно этого этапа. Чем выше частота, с которой проверяется работа приложения на различных наборах данных, тем эффективнее и полноценнее будет тестирование. Именно поэтому оправдано выделение максимальных вычислительных ресурсов для проведения фаззинга. Например, компания Google для тестирования Chrome создала свой кластер ClusterFuzz, на котором одновременно запущено около 6000 тысяч процессов данного браузера, а в сутки обрабатывается порядка 50 миллионов тестовых случаев.

Мониторинг исключений. Наиболее важен в случае тестирования больших и сложных приложений, так как недокументированное поведение приложений не всегда является очевидным.

Исследование ошибок. Данный шаг является самым сложным из всех перечисленных. Он не поддается автоматизации и требует от аналитика немалого мастерства. Ручное воспроизведение ошибки не составит труда, однако, детальное исследование её причин и выдача заключения о возможности или невозможности эксплуатации данной уязвимости могут стать весьма непростыми.

Виды фаззинга. Существующие на сегодняшний день виды фаззинга можно условно разделить на 2 категории.

Локальный:

- фаззинг аргументов командной строки;
- фаззинг переменной среды;
- фаззинг оперативной памяти;
- фаззинг формата файла;
- фаззинг системных вызовов.

Удаленного доступа:

- фаззинг сетевого протокола;
- фаззинг веб-приложений;
- фаззинг веб-браузеров.

Методы локального фаззинга активно применяются для анализа защищенности вычислительного комплекса Эльбрус. В докладе рассматриваются особенности реального применения данной технологии поиска уязвимостей.

Использование криптографии против криптографии

Для определения направления, связанного с использованием криптографии против криптографии, в 1996 г. был введен термин «клептография» (kleptography). Клептографическая атака – это создание «закладки», внедряемой в разрабатываемую криптосистему, которая может

быть реализована в виде устройства или программы. При этом не должно существовать возможности выявить наличие закладки по внешним признакам. Иначе говоря, если рассматривать криптосистему как «черный ящик», невозможно определить, осуществил разработчик клептографическую атаку или нет.

Кроме того, атакующая сторона, создающая «закладку», получает эксклюзивное право ее использования, т.е. даже при наличии у третьей стороны исчерпывающей информации о реализации системы, она в лучшем случае сможет выявить наличие скрытого канала, но не сможет им воспользоваться.

Базовыми элементами любых клептографических атак являются скрытые канал передачи информации. Именно они в сочетании с дополнительными криптографическими методами обеспечивают необнаруживаемость этих атак. Клептографические методы позволяют довести до совершенства сокрытие факта передачи дополнительной информации. В этом смысле прослеживается сходство со стеганографией.

Скрытые клептографические каналы являются частью криптоалгоритма и позволяют незаметно передавать информацию из криптографической системы или, наоборот, в криптографическую систему. Например, дополнительная информация может содержаться в электронной цифровой подписи (ЭЦП) или в открытом ключе шифрования [7, 8].

В докладе рассматривается новая клептографическая атака на схему ЭЦП, основанную на свойствах эллиптических кривых (ECDSA).

Кривые, использующиеся для шифрования, могут быть определены как в простом конечном поле, так и в расширенном характеристики 2. Эллиптические кривые над конечными простыми полями определяются уравнением:

E: $y^2 = x^3 + ax + b \bmod p$,

где $a, b \in \mathbf{F}(p)$ – коэффициенты эллиптической кривой и $4a^3 + 27b^2 \neq 0 \bmod p$.

Эллиптические кривые E над конечным полем задается множеством точек на плоскости. Над точками из $E(\mathbf{F}(p))$ определены операции сложения, скалярного умножения и отрицания.

Проблема дискретного логарифмирования на эллиптической кривой (ECDLP): возьмем точку P порядка n на эллиптической кривой E над конечным полем $\mathbf{F}(p)$ и точку Q на E, ECDLP заключается в том, чтобы найти целое число k, где $0 \leq k \leq n - 1$ и $Q = kP$, если такое число вообще существует.

Вычислительная проблема Диффи-Хеллмана на эллиптической кривой (ECDHP): возьмем точку P порядка n на эллиптической кривой E над конечным полем $\mathrm{F}(p)$ и две точки kP *и* lP, где $0 \leq k, l \leq n - 1$, ECDHP заключается в том, чтобы найти точку $k \times lP$.

Рассмотрим схему электронной цифровой подписи на эллиптических кривых (ECDSA).

Пусть m − это сообщение, которое необходимо подписать. Параметры схемы:

эллиптическая кривая E над конечным полем $\mathrm{F}(p)$; базовая точка G порядка n;

секретный ключ d; открытый ключ $Q = dG$; $h(x)$ – безопасная хеш-функция.

Алгоритм формирования подписи

Входные данные: сообщение m.

Выходные данные: подпись (r, s).

Выберем случайное число $1 \leq k \leq n - 1$, вычисляем

$R = kG = (x_1, y_1)$, $r = x_1 \bmod n$, $s = k^{-1}(h(m) + d \times r) \bmod n$.

Если $s = 0$ или $r = 0$, то (r, s) не принимается для данного сообщения m и вычисления надо повторить заново. В другом случае (r, s) является подписью сообщения.

Алгоритм проверки подписи

Входные данные: подпись (r, s), сообщение m.

Выходные данные: результат проверки подписи (r, s) сообщения m.

Вычисляем $R' = s^{-1}(h(m)G + rQ) = (x_1', y_1')$.

Если $r = x_1' \bmod n$, то подпись (r, s) принимается, в противном случае − отвергается.

Клептографическая атака на задачу нахождения дискретного логарифма была описана Янгом и Юнгом в работе. Они ввели понятие секретной встроенной лазейки с универсальной защитой (SETUP), которая видоизменяет криптографический алгоритм таким образом, чтобы

атакующий мог получить секретную информацию пользователя. Лазейка обеспечивает существование скрытого канала передачи информации. Некоторые свойства SETUP описаны ниже.

Существует C − криптосистема с объявленной спецификацией, SETUP-механизм — это такая модификация криптоалгоритма C в алгоритм C_1, что:

– Параметры входа C_1 согласуются с входными параметрами C;
– Параметры выхода C_1 соответствуют параметрам выхода C, но при этом выход C_1 содержит секретные биты, которые легко извлекаются разработчиком;
– По выходам алгоритмы C_1 и C полиномиально неразличимы для всех, кроме разработчика;
– Выход C_1 эффективно вычисляется с использованием встроенной в C_1 функции шифрования E;
– Секретная функция расшифрования D, обратная к E, не содержится в C_1 и известна только разработчику;

После обнаружения в реализации алгоритма SETUP-механизма ни пользователи, ни злоумышленники (иначе говоря, никто за исключением разработчика) не могут определить использованные секретные ключи пользователя.

Рассмотрим SETUP атаку на ECDSA.

Пусть v – это закрытый ключ атакующего и $V = vG$ – это его открытый ключ. Хеширование точки эллиптической кривой определим как хеширование ее x-координаты. Устройство работает следующим образом.

Алгоритм формирования цифровой подписи с SETUP

Входные данные: сообщение m.

Выходные данные: подпись (r, s).

Выбираем целое число $1 \leq k_1 \leq n - 1$;

Вычисляем

$R_1 = k_1G = (x_1, y_1)$; $r_1 = x_1 \bmod n$; $s_1 = k_1^{-1}(h(m_1) + d \times r_1) \bmod n$;

Сохраняем k_1 в энергонезависимой памяти.

Вычисляем

$Z = a \times k_1G + b.k_1V + h \times jG + e \times uV$,

где $a, b, h, e < n$ − фиксированные целые числа; $j, u \in \{0, 1\}$ – случайные;

Вычисляем

$k_2 = h(Z)$; $R_2 = k_2G = (x_2, y_2)$; $r_2 = x_2 \bmod n$; $s_2 = k_2^{-1}(h(m_2) + d \times r_2) \bmod n$;

Сохраняем k_2 в энергонезависимой памяти.

Каждый может проверить подпись в любое время с помощью открытого ключа Q. Злоумышленник может получить d и сообщение с помощью (r_1, s_1) и (r_2, s_2).

Алгоритм расшифрования SETUP

Входные данные: (r_1, s_1), (r_2, s_2), m_2.

Выходные данные: секретный ключ d.

Если предположить, что полученные подписи действительны, используем уравнение кривой E над конечным полем $F(p)$ для расчета возможных точек R_1' на кривой, чьи координаты x удовлетворяют соотношению $x \bmod n = r_1$. Существует две таких точки. Для каждой возможной точки R_1' выполняем следующие действия:

{

$Z_1 = aR_1' + b \times vR_1' = a \times k_1'G + b \times v \times k_1'G = a \times k_1'G + b \times k_1'V$

Для каждого возможного значения j, u вычисляем:

{

$Z_2 = Z_1 + h \times jG + e \times uV$

$k_2' = h(Z_2)$

$R_2' = k_2'G = (x_2', y_2')$

$r_2' = x_2' \bmod n$

Если $r_2' = r_2$, тогда $k_2' = k_2$, и выходим из цикла

}

}

$d = (s_2k_2 - h(m_2)) \times r_2^{-1} \bmod n$

Таким образом, вычисляется секретный ключ d, что позволяет атакующему подделывать подписи и расшифровывать сообщения.

SETUP атака безопасна в том смысле, что пользователь, не зная случайного числа k_1, не может рассчитать секретный ключ k_2 (проблема ECDHP). Кроме того, противник, не знающий закрытого ключа атакующего, не может рассчитать Z и, следовательно, не может рассчитать k_2. Поэтому SETUP атаки обладают универсальным свойством защиты. Однако ее можно назвать неустойчивой, потому что пользователь, который знает собственный секретный ключ, может восстановить выбранное k и обнаружить SETUP. Случайные значения j и u используются в качестве меры предосторожности, если случайный параметр k доступен для пользователя. Благодаря этим параметрам пользователь не сможет обнаружить SETUP в устройстве даже после многократного использования.

Антивирусная защита

"Лекарства" даже от простейшего вида РПВ, компьютерных вирусов, не существует. Всегда можно создать вирус, который не сможет нейтрализовать ни одна из существующих антивирусных программ. Основная идея в том, что если разработчик КВ знает, что именно ищет антивирусная программа, он всегда способен разработать РПВ, незаметное для нее.

Технологии разработки вирусов и антивирусных программ развиваются параллельно. По мере усложнения вирусов, антивирусное ПО становится сложнее и изощреннее. Можно выделить четыре поколения антивирусных программ: обычные сканеры, использующие для идентификации КВ характерные для них маски или сигнатуры; эвристические анализаторы; антивирусные мониторы; полнофункциональные системы защиты.

Можно выделить следующие принципиальные недостатки существующих средств защиты от КВ:

- использование методов, при реализации которых нападающие всегда находятся в более выигрышном положении, чем защищающиеся;
- принципиальные ограничения существующих методов АВЗ - каждый из них в отдельности легко обмануть, т.е. всегда можно создать вредоносную программу, которая не будет обнаружена;
- мало внимания уделяется самозащите от вредоносных программ, самоконтролю целостности и обеспечению гарантированности свойств компонентов системы защиты;
- недостаточно внимания уделяется проверке на отсутствие уязвимостей программного кода типа «переполнение буфера», наличие таких уязвимостей позволяет противнику подменять алгоритмы функционирования средств защиты, вызывать отказ в обслуживании со стороны этих средств и выполнять произвольный код на атакуемой машине;
- используются методы, при реализации которых нападающая сторона всегда оказывается в выигрышном положении по сравнению со стороной защищающейся;
- отсутствует оперативная реакция со стороны разработчиков средств АВЗ на появление принципиально новых вирусных методик.

В докладе рассматривается структура и принципы функционирования компонентов полнофункционального программного комплекса программных средств антивирусной защиты (КПС АВЗ), в которой реализованы все известные методы антивирусной защиты (АВЗ).

Практически любая антивирусная система базируется на нескольких общепринятых техниках поиска и обнаружения компьютерных вирусов. Этими техниками являются:

- Сигнатурный анализ - анализ исполняемого файла и поиск в нем некоторых заранее известных последовательностей байт, позволяющих определить наличие вредоносного кода.
- Отслеживание контрольных сумм исполняемых файлов. Первоначально считается хэш-образ от незараженного файла. В случае заражения файла его хэш-образ изменится. Это обстоятельство позволит однозначно определить факт заражения.
- Эмуляция выполнения подозрительного кода. Эмулятор разбирает программный код на команды и каждую команду запускает в виртуальной исполняемой среде. Это позволяет антивирусу наблюдать за поведением программы, не ставя под угрозу операцион-

ную систему и данные пользователя, что неизбежно произошло бы при исполнении программы в реальной среде.

- Песочница (виртуализация выполнения подозрительного кода) - логическое продолжение эмуляции. Выполнение подозрительного кода производится в реальной системе, но объекты атаки заменяются фиктивными (копия реестра и копия атакуемых файлов). Песочницы используют для запуска кода из неизвестных источников, как средство проактивной защиты от вредоносного кода, а также для обнаружения и анализа вредоносных программ. Очень часто песочницы используются в процессе разработки программного обеспечения для запуска «сырого» кода, который может повредить систему.
- Мониторинг системных событий - отслеживание всех системных событий в системе и обработка их аналитическим компонентом антивируса.

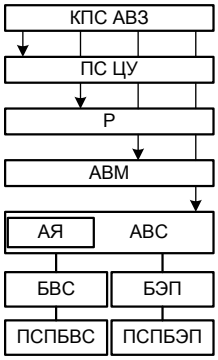

Рис. 1. Структура КПС АВЗ

Возможности реализации гибридного взаимодействия в данных технологиях:

1. Гибридная реализация механизма отслеживания контрольных сумм исполняемых файлов принесет эффективный прирост во времени, т.к. подсчет контрольной суммы по алгоритму MD5 или SHA2 легко распараллеливается.

2. Гибридная эмуляция выполнения подозрительного кода. Стоит отметить высокую сложность и системную требовательность реализации эмулятора. Но эмуляция на гибридной системе позволит ускорить этот процесс, и сделать его тем самым доступным.

3. Повышенная безопасность исполнения кода в песочнице связана с большой затратой ресурсов системы. Поэтому реализация гибридной песочницы уменьшит время прохождения этапа тестирования потенциально вредоносного и «сырого» кода. И позволит использовать эту систему при каждой проверке, а не только не отлаженного или подозрительного кода. Что в результате приведет к повышению показателя обнаружения вирусного кода.

4. Песочница для гибридных КВ (в перспективе) - реализация такой песочницы имеет смысл, т.к. становится возможно произвести анализ "основной" и "гибридной" части вируса в "естественной среде обитания".

Новые криптоалгоритмы

Большинство используемых на данный момент алгоритмов защиты информации разработаны более 10 лет назад. С учетом прогресса в области информационных технологий это означает моральное устаревание практически всех механизмов шифрования. Еще более усугубляется положение активным развитием гибридных вычислительных технологий: методы криптоанализа, которые были недоступны ранее из-за больших требований к вычислительным ресурсам системы становятся вполне реальны.

С другой стороны активно развиваются новые методы анализа стойкости алгоритмов, основанные на особенностях выполнения данного алгоритма в рамках той или иной архитектуры. Примером такой атаки может служить cache-timing, как разновидность временной атаки.

Таким образом появляется необходимость в разработке новых криптоалгоритмов, которые сохранят преимущества используемых ныне алгоритмов и будут адаптированы к использованию в условиях развития новых технологий и методов криптоанализа.

Основные направления совершенствования криптоалгоритмов:

- повышения уровня параллелизма для оптимизации использования на гибридных вычислительных системах;
- обфускация алгоритмов с целью исключения атак по сторонним каналам;
- поиск новых архитектур криптосистем.

В докладе рассматриваются методы модификаций криптосистем на примере RIJNDAEL. Также в докладе рассматриваются новые архитектуры, выделяются те или иные их особенности и сходство с уже существующими алгоритмами.

Повышение уровня параллелизма

Ни для кого не секрет, что используемый повсеместно AES состоит из четырёх основных операций над данными: SybBytes, ShiftRows, MixColumns и AddRoundKey. Все эти операции отлично распараллеливаются, что обеспечивает большую производительность на суперкомпьютерных вычислительных системах. Тем не менее, в классическом AES есть одно узкое место - функция расширения ключа. Помимо того, что многие атаки, в том числе и SQUARE (атака, предложенная самими авторами RJINDAEL), основаны на идее восстановления ключа по известному фрагменту, так и само выполнение этой функции довольно сильно зависит от последовательности выполнения итераций.

Есть много вариантов решения данной проблемы и все из них обладают своими плюсами и минусами. Одним из самых интересных является использование хэш-функций для разворачивания ключа. Данное решение обладает сразу несколькими преимуществами и не лишено недостатков:

+ оно обеспечивает стойкость к атакам, основанным на восстановлении ключа по его фрагменту;

+ обладает лучшей способностью к распараллеливанию;

+ не теряет своих свойств при масштабировании;

- требует большего количества вычислительных ресурсов.

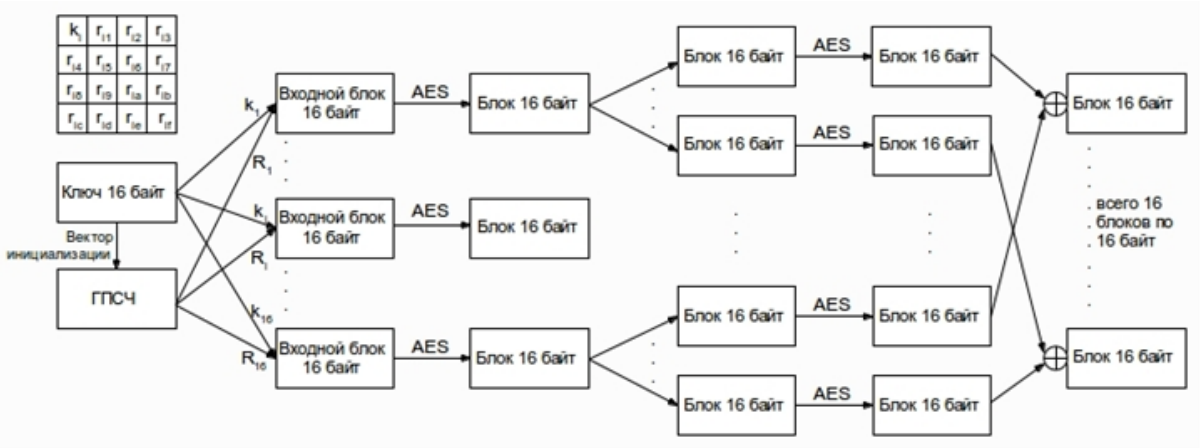

Рис. 2. Пример использования хэш-функции в качестве функции разворачивания ключа

Обфускация алгоритмов

В криптографии атакующий и объект атаки могут находиться в следующих состояниях:

- атакующий имеет полное представление об алгоритмах функционирования объекта атаки;
- атакующий имеет неполное (не имеет) представление об алгоритмах функционирования объекта атаки, но может наблюдать результат работы этих алгоритмов;
- атакующий имеет неверное представление об алгоритмах функционирования объекта атаки;

- атакующий не может наблюдать результата работы алгоритмов защиты информации объекта атаки.

Изначально атакующий может находится в состояниях, описанных в пунктах 1(при условии, что использован некий известный криптоалгоритм и атакующий обладает информацией о том, что использован именно он) или 2 (если информация о используемом криптоалгоритме недоступна атакующему). Для того, чтобы полностью скрыть сообщение (или шифротекст) от атакующего (состояние 4)используются стеганографические методы (обфускация которых также очень важна ввиду простой обратимости большинства методов). Обфускация криптоалгоритма может поставить атакующего в состояние 3. Это хорошо со стороны объекта атаки тем, что любые попытки атаковать криптоалгоритм, основываясь на его особенностях, будут безрезультатны.

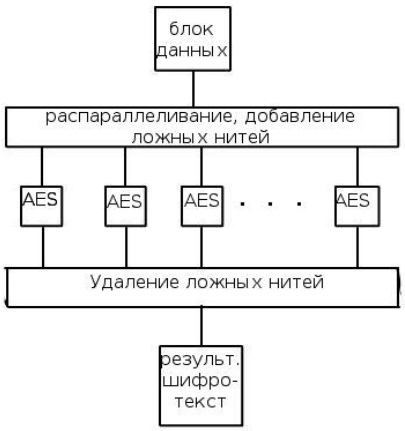

Рис. 3. Пример обфускации с использованием распараллеливания

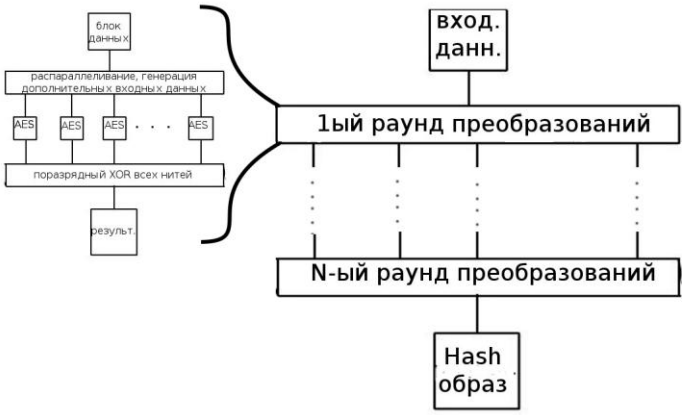

Рис. 4. Обфускация на примере реализации хэш-функции

Новые архитектуры криптосистем

Помимо попыток улучшить существующие криптосистемы ведется поиск новых архитектур. Примерами могут служить гибридная архитектура Сеть Фейстеля-Квадрат и Куб.

Сеть Фейстеля-Квадрат представляет собой классическую Сеть Фейстеля с использованием криптоалгоритмов с архитектурой Квадрат в качестве функции F. Данная архитектура является довольно интересной для изучения свойств и аккумулирует в себе достоинства обеих систем. Однако, с точки зрения использования суперкомпьютерных систем, более интересны многомерные архитектуры криптосистем, в частности - Куб.

Основным мотивом создания архитектуры Куб стала идея перехода от двумерной архитектуры, используемой в AES, к новой, предположительно более стойкой, трехмерной. Результатом стало целое семейство криптосистем, в котором блоки данных представляются в виде трехмерной матрицы, и классические для AES операции выполняются над слоями, образующимися путём «разрезания» исходной матрицы плоскостями, перпендикулярными той или

иной координатной оси. Такой подход позволяет выполнять операции параллельно внутри квадратов-слоёв и параллельно для нескольких слоёв (выполнение операций преобразования внутри слоя не зависит от данных соседних). Таким образом, криптосистемы с архитектурой Куб отлично подходят для реализации в рамках задач параллельного программирования.

Выводы

В докладе рассмотрены:

- технология поиска уязвимостей КС, называемая фаззингом;
- механизмы создания и использования скрытых каналов в криптографических алгоритмах;
- структура полнофункционального комплекса ПС АВЗ;
- методы совершенствования современных криптоалгоритмов.

Основной тезис работы: эффективная система защиты – это не фиксированный набор методов и средств защиты, это непрерывный процесс, который включает в себя: анализ защищенности системы на всех ее уровнях (элементная база, архитектура, системное ПО, сетевое ПО, прикладное ПО) и опережающее совершенствование методов и средств защиты. Таким образом, важнейшую роль в обеспечении информационной безопасности играет разработка методики комплексного анализа защищенности КС, и в первую очередь проведения важнейшего его этапа – теста на проникновение.

Именно такой подход авторы используют при анализе защищенности ВК Эльбрус.

Литература

1. Sutton M., Green A., Amini P. Fuzzing. Brute Force Vulnerability Dicovery. – Addison-Wesley, 2007
2. Fuzzing: Brute Force Vulnerability Discovery. URL http://www.fuzzing.org/
3. Kozirsky B.L., Komarov T.I. Fuzzing – a perspective method of searching software vulnerabilities. Proceedings of RES-2013, Moscow, pp. 160-163.
4. Young A., Yung M. Cryptovirology: Extortion-Based Security Threats and Countermeasures // IEEE Symposium on Security&Privacy, 1996. P. 129–141.
5. Young A.L., Yung M.M. Kleptography: Using cryptography against cryptography // Advances in Cryptology, EUROCRYPT'97, Springer-Verlag, Lecture Notes in Computer Science. № 1233. 1997. P. 62–74.
6. Chepik N.A. Kleptographic attacks on ECDSA. Proceedings of RES-2013, Moscow, pp. 163-166.
7. Клептографические атаки на криптосистемы с открытым ключом / З.Р. Гарифуллина, М.А. Иванов, А.В. Ковалев, Н.А. Чепик // Вестник НИЯУ МИФИ, 2012, том 1, № 2.
8. Vavrenyuk A.B., Zaychik A.U., Smirnov A.A., et. al. Heuristics Database of Antivirus Software Suite for Computer Systems Running Linux // Proceedings of RES-2013, Moscow, pp. 167-169.

Аннотация

В рамках данного доклада рассматриваются некоторые из аспектов информационной безопасности: фаззинг – метод автоматического поиска уязвимостей, использование криптографии против криптографии, перспективные пути развития существующих криптоалгоритмов, а так же возможности по совершенствованию современных антивирусных решений. Особое внимание уделяется роли, которую должны исполнять суперкомпьютерные технологии, в предлагаемых решениях.

Ключевые слова: фаззинг, клептография, ЭЦП, антивирус, cache-timing, RIJNDAEL.

Сведения об авторах

Авторы статьи – студенты кафедры Компьютерных систем и технологий факультета Кибернетики и информационной безопасности НИЯУ МИФИ.

Список авторов: Чепик Н.А., Зайчик А.Ю., Козырский Б.Л., Комаров Т.И., Максутов А.А., Смирнов А.А.

Контактная информация

Комаров Тимофей Ильич, студент кафедры №12 НИЯУ МИФИ.

Тел.: 8(916)127-98-57, эл. почта: happycorsair@yandex.ru.