# Способы автоматизации поиска уязвимостей в программном обеспечении на соответствующих уровнях его разработки

**Тайлаков Виктор Александрович** – старший преподаватель кафедры Электроники и робототехники Алматинского университета энергетики и связи.

**Израилов Константин Евгеньевич** – кандидат технических наук, доцент кафедры Защищенных систем связи Санкт-Петербургского государственного университета телекоммуникаций им. проф. М.А. Бонч-Бруевича.

*Аннотация:* В статье рассматривается проблема поиска уязвимостей в программном обеспечении. Предлагается деление уязвимостей по точкам их возникновения в соответствии с уровнями разработки продукта: концептуальный, архитектурный, логический, операционный, инструкционный. Описываются особенности каждого уровня, их взаимосвязь, а также возникающие на них уязвимости. Приводятся примеры способов автоматического поиска гипотетических уязвимостей на каждом уровне.

*Ключевые слова:* Поиск уязвимостей, безопасность кода, программное обеспечение.

## Введение

Наличие уязвимостей в программном обеспечении является одной из актуальнейших проблем информационной безопасности, поскольку ведет непосредственно к триаде нарушений конфиденциальности, целостности и доступности информации при ее обработке. Существование данной проблемы было установлено достаточно давно, в связи с чем было разработано достаточно большое количество способов поиска уязвимостей в программном обеспечении. При этом часть из них являются в основном ручными – требуют участия эксперта, а часть является полностью автоматическими – строятся на использовании средств поиска уязвимостей; естественно возможны комбинированные, то есть автоматизированные, варианты. Ручные способы считаются наиболее трудоемкими, хотя и дают наиболее точные результаты; автоматические же имеют хотя и низкие требования к ресурсам и высокую скорость выполнения, но также большую вероятность ошибок первого и второго рода. Разумное комбинированное применение автоматических средств экспертом при ручном анализе может оказаться наиболее востребованным, поскольку могут быть учтены положительные стороны

каждого из способов. Поиск же существенно различных уязвимостей может потребовать не только правильных пропорций такого комбинирования, но и разработку принципиально новых средств.

## Уровни программного обеспечения

Любое не тривиальное программное обеспечение (ПО) имеет протяженный жизненный цикл разработки, проходящий от создания общих представлений о будущем продукте, через проработку его алгоритмов к непосредственному кодированию для сборки в выполняемый образ. Ошибки (в том числе и уязвимости) могут возникать на каждом из этапов этого цикла, изменяясь и растворяясь в ПО в процессе его разработки. Так, ошибки в основах функционирования продукта будут плохо различимы по его представлению в виде машинного кода, поскольку строятся на более абстрактных понятиях и всеобъемлющих, чем операции языка программирования. При этом чем выше уровень абстракции вида ПО, тем более значимое участие человека необходимо.

Поделим весь жизненный цикл разработки ПО на несколько уровней разработки по мере формирования продукта от начального прообраза до конечного выполняемого образа. Каждый уровень потребует разного участия человека для обнаружения уязвимостей. Тем не менее, на каждом из уровней возможно применение собственных методов для автоматизации их поиска (что будет показано на примерах).

Инструкционный уровень. Данный уровень относится к самому последнему, в котором может находиться ПО, поскольку основан на непосредственных инструкциях процессора выполнения (или виртуальной машины в случае байт-кода). Например, машинный код функции сравнения двух чисел будет иметь вид инструкций процессора по пересылки значений между регистрами, их сравнения, переходов на метки, занесение результатов в отдельные переменные и выход из функции. Данный уровень не используется программистами, а получается при компиляции исходного кода. Также, возможно вложение скрытой информации на данном уровне [7].

Операционный уровень. Данный уровень относится к самому последнему, в котором разрабатываться ПО человеком, поскольку основан на непосредственных операциях языка программирования. Например, функция сравнения двух чисел кодируется с помощью заголовка функции, переменных и их типов, операторов сравнения и способа вывода. Уровень используется для непосредственного кодирования логики работы ПО перед сборкой в выполняемый машинный или байт-код. Также, возможно вложение цифровых водяных знаков на данном уровне [6].

Логический уровень. Данный уровень предназначен для описания логики работы ПО без привязки к конкретному языку программирования. Так, например, блок-схема функции сравнения двух числе будет абсолютно одинаковой при кодировании ПО на языках C, Java или Python – сравнение входных параметров с выводом результата. Уровень описывает логику взаимодействия элементов архитектуры ПО, задавая при этом общие требования по кодированию.

Архитектурный уровень. Данный уровень предназначен для описания общей структуры ПО, способов их взаимодействия на уровне управления и данных. Так, например, архитектура ПО для выполнения операций над числами может состоять из модулей их сравнения, сложения, вычитания и др. Уровень производит внутреннюю детализацию ПО, задавая рамки для возможных вариаций логики его работы.

Концептуальный уровень. Данный уровень предназначен для описания концептуальной модели ПО – то есть ее основных элементов и их взаимосвязей. Так, например, концепция ПО для работы над числами должна вводить объекты – числа и операции над ними – сравнение, сложение, вычитание и др. Уровень делает первоначальную формализацию самой задумки ПО.

При прохождении через каждый уровень ПО в процессе его разработки может приобретать уязвимости (случайно или злонамеренно), имеющие собственные особенности и уровни абстракции.

Основная сложность уязвимостей в реальных продуктах заключается в том, что такое ПО имеет вид как правило, только машинного или байт-кода, то есть находится на инструкционном уровне. Последний, очевидно наиболее подходит для обнаружения уязвимостей лишь его уровень – как правило, автоматическими средствами. Поиск же уязвимостей, внесенных на предыдущих более абстрактах уровнях, требует участия эксперта, который может частично произвести обратную разработку ПО – с инструкционного до концептуального, применяя соответствующие методы поиска на каждом из уровней. Рассмотрим примеры того, как возможно автоматизировать работу эксперта по поиску уязвимостей на каждом из уровней.

## Уязвимости в программном обеспечении

Уязвимости инструкционного уровня. Появление уязвимостей на данном уровне является достаточно редким, поскольку машинный или байт-код генерируется и исходного программными средствами – компиляторами, что исключает человеческий фактор. При этом как исходный уровень ПО (язык программирования), так и конечный (ассемблерный язык) являются полностью формализованными, что практически полностью исключает неправильность трактовки ПО на этих уровнях. Впрочем, ошибки в самих компиляторах (как случайные, так и злонамеренные) могут приводить к генерации уязвимого кода. Для обнаружения таких уязвимостей существует достаточное количество средств тестирования компиляторов и баз тестов. Таким образом, дополнительной автоматизации поиска уязвимостей на данном уровне не требуется.

Уязвимости операционного уровня. Данный уровень можно отнести к источнику случайных уязвимостей, возникших вследствие ошибок работы программиста. Поскольку некоторые правила кодирования не только заложены в стандарт языка, но и хорошо формализуются, то поиск уязвимостей экспертом может быть упрощен анализаторами, проверяющими эти правила. Например, выход индекса за пределы статического массива хотя и не является ошибкой с точки зрения языка C (доступ к элементу массива трактуется как разыменование суммы указателя на массив и указанного индекса), но, тем не менее, некоторые случаи могут быть выявлены статическими анализаторами исходного кода. Алгоритм определения некоторых таких ситуаций может вычислить диапазон возможных значений индекса массива на предмет его выхода за допустимые значения (менее 0 или более размера массива).

Уязвимости логического уровня. Данный уровень можно отнести к источнику злонамеренных уязвимостей, внесенных злоумышленником с целью поменять логику работы ПО на нужную. Одним из наиболее популярных таких уязвимостей может быть закладывание в логику уязвимости типа backdoor, позволяющей получить несанкционированный доступ к данным или управлению системой. Как правило, выявление таких мест требует непосредственного участия эксперта, который, зная требуемую логику работы ПО, должен сравнить ее с логикой, реализованной в продукте – отличия в логике будут сигнализировать об уязвимости логического уровня. Тем не менее, в ряде случаев возможна автоматизация их поиска [4]. Например, наличие в функции сравнения одного из параметров с константным значением и переход в другую, редко используемую, функцию – что определимо автоматическими средствами – может говорить об аномалии в логике работы кода, требующей непосредственной проверки эксперта на предмет уязвимости backdoor.

Уязвимости архитектурного уровня. Уязвимости на данном уровне можно считать наиболее редко изучаемыми, поскольку для их обнаружения необходимо не только восстановление архитектуры из машинного или байт-кода (что является крайне сложной

задачей), но и понимание того, какие в принципе аномалии на этом уровне можно считать уязвимостями. Все это требует от эксперта высочайшей квалификации. Впрочем, автоматизация поиска уязвимостей также возможна. Так, например, если множество модулей системы используют данные, расположенные в отдельном модуле, который при этом имеет каналы извне системы, то это может быть потенциальной уязвимостью по причине повышенной вероятности кражи данных в случае успешности атаки извне на модуль. Способ определения такой ситуации может быть основан на выделении модулей, анализе хранимых в них данных и взаимодействия их с другими модулями (в том числе внешними к системе). Впрочем, данная аномалия также должна быть предварительно проверена экспертом.

Уязвимости концептуального уровня. С некоторой уверенностью можно утверждать, что данный уровень в принципе практически не рассматривается при разработке ПО, переходя сразу от идеи продукта к архитектурному уровню – тем не менее, полагаясь на свой богатый опыт разработки ПО, авторы с уверенностью могут утверждать, что уровень существует. Следовательно, некоторые уязвимости могут быть заложены уже на нем – то есть практически в самом начале разработки. Анализ данного уровня требует от эксперта не просто профессиональной подготовки, а высочайшего уровня абстрагирования от каких-либо деталей реализации – языков программирования, шаблонов проектирования, технологий и т.п. Однако даже в этом случае некоторая автоматизация поиска уязвимостей может быть осуществлена. Впрочем, из-за крайне большого количества вариаций концептуальных моделей и их смысловой интерпретации может потребоваться уточнение специфики разрабатываемого ПО. Так, например, если концептуальная модель описывает информационную систему, то в ней должны присутствовать такие элементы, как входные и выходные данные ([1,2,5]). Если связь между последними (всеми или частью) отсутствует, то это означает уязвимость концептуального уровня, поскольку часть входных или вычисленных по ним данных не будут выданы системой, что будет означать нарушение доступности (в смысле потери информации, введенной пользователем информационной системы). Алгоритм выявления такой ситуации должен учесть специфику элементов и требуемую между ними взаимосвязь. Также, к данному уровню можно отнести и некоторые встроенные настройки криптографической защиты (например, длина ключей шифрования [3]).

## Заключение

Поиск уязвимостей в программном обеспечении является в высшей степени сложной задачей (а иногда и проблемой). Ситуация усложняется еще больше, когда возникает необходимость применения автоматизирующих способов поиска – упрощающих трудоемкий ручной способ. При этом для поиска различных уязвимостей необходимо применение совершенно разной степени подготовленности и участия эксперта, соответствующих уровню разработки программного обеспечения, на котором возникла уязвимость. Тем не менее, как показано на примерах, поиск уязвимостей на любом из

уровней подходит для автоматизации. Впрочем, современная область безопасности программного обеспечения, как правило, ограничивается поиском уязвимостей в исходном, машинном и байт-коде – операционный и инструкционный уровни, оставляя остальные без должного внимания, хотя в мировой практике уязвимости именно более абстрактных уровней – логического, архитектурного и концептуального – несут особую угрозу информационной безопасности. В связи с этим, а также собственной научной заинтересованности, будущие исследования авторов буду посвящены дальнейшему теоретическому исследованию способов поиска уязвимостей на каждом из уровней разработки программного обеспечения, а также практической реализации синтезированных алгоритмов поиска в виде статического анализатора машинного и байт-кода.

*Список литературы*

1. Буйневич М.В., Покусов В.В., Израилов К.Е. Гипотетическая схема информационно-технического взаимодействия модулей системы обеспечения информационной безопасности // III Международная научно-практическая конференция «Информатика и прикладная математика», 2018. С. 179-192.
2. Буйневич М.В., Покусов В.В., Ярошенко А.Ю., Хорошенко С.В. Категориальный подход в приложении к синтезу архитектуры интегрированной системы обеспечения безопасности информации // Проблемы управления рисками в техносфере. 2017. № 4 (44). С. 95-102.
3. Волкогонов В.Н., Голубев В.С., Ушаков И.А. Проблема криптозащиты в устройствах Cisco при использовании актуальных прошивок // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2017): сборник трудов VI Международной научно-технической и научно-методической конференции. 2017. С. 194-198.
4. Израилов К.Е., Покусов В.В. Утилита для поиска уязвимостей в программном обеспечении телекоммуникационных устройств методом алгоритмизации машинного кода. Часть 3. Модульно-алгоритмическая архитектура // Информационные технологии и телекоммуникации. 2016. Т. 4. № 4. С. 104-121.
5. Buinevich M., Izrailov K., Pokusov V., Sharapov S., Terekhin S. Generalized Interaction Model In The Information System // International Journal of Pure and Applied Mathematics (IJPAM-AP). 2018. Vol. 119. Iss. 17d. pp. 1381-1384.
6. Krasov A.V., Arshinov A.S., Ushakov I.A. Embedding the hidden information into Java byte code based on operands' interchanging // ARPN Journal of Engineering and Applied Sciences. 2018. Т. 13. № 8. С. 2746-2752.
7. Shterenberg S.I., Krasov A.V., Ushakov I.A. Analysis of using equivalent instructions at the hidden embedding of information into the executable files // Journal of Theoretical and Applied Information Technology. 2015. Т. 80. № 1. С. 28-34.

{social}