BIS JOURNAL

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ БИЗНЕСА

№4 (59) 2025

Арутюн АВЕТИСЯН
ИСП РАН
«Железа и софта недостаточно — нужны люди!»
→ 35

Денис БАТРАНКОВ
ГК «Гарда»
NGFW 2025: конец эпохи
→ 4

Сетевая безопасность: прочны ли наши сети?
→ 4–34

Три кита РБПО. 3. Архитектурный анализ
→ 41–68

Дмитрий ПОНОМАРЕВ
ООО НТЦ «Фобос-НТ»
Архитектурный анализ
→ 41

Алексей СМИРНОВ
CodeScoring
Композиционный анализ
→ 44

Евгений ТОДЫШЕВ
УЦСБ
ИТ и ИБ: рука об руку
→ 58

«ЖЕЛЕЗА И СОФТА НЕДОСТАТОЧНО — НУЖНЫ ЛЮДИ!»

НА ВОПРОСЫ BIS JOURNAL ОТВЕЧАЕТ АРУТЮН АВЕТИСЯН

Арутюн АВЕТИСЯН
академик РАН, доктор физико-математических наук, профессор РАН, директор Института системного программирования РАН

— Арутюн Ишханович, расскажите о создании Института системного программирования им. В. П. Иванникова РАН, который вы возглавляете. Почему из всех направлений кибернетики Виктором Петровичем Иванниковым было выбрано именно системное программирование?

— История Института системного программирования неразрывно связана с развитием отечественных информационных технологий. В 1994 году Виктор Петрович, тогда заведующий отделением Института проблем кибернетики РАН, принял решение создать самостоятельный научный центр, сфокусированный на системном программировании — области, которая обеспечивает основу для функционирования всей вычислительной техники. Это было время серьёзных перемен: распадались старые структуры, шло переосмысление роли науки. Но Виктор Петрович очень точно почувствовал, что в новых реалиях стране жизненно необходимо собственное развитие базового софта — компиляторов, операционных систем, инструментов тестирования и верификации.

При этом Виктор Петрович всегда был сторонником открытых подходов. Он понимал, что невозможно конкурировать, если всё держать в замкнутом контуре. Идея открытости знаний и технологий стала основой философии ИСП РАН.

Подход Иванникова можно назвать пророческим. Тогда мало кто осознавал, что системное программирование будет развиваться настолько стремительно и станет краеугольным камнем всей цифровой инфраструктуры — от суперкомпьютеров и космических систем до смартфонов и интернета вещей.

— В своё время было принято решение о встраивании в процесс международной ИТ-кооперации на подсобных, второстепенных ролях, что привело к утрате компетенций по ряду важных направлений, в том числе в системном программировании и средствах разработки. Как вы оцениваете наше отставание от лидеров в данной области? Что нужно сделать для достижения суверенитета в области разработки системного и прикладного ПО?

— С самого начала своего существования ИСП РАН активно конкурировал с ведущими мировыми центрами передовых исследований и разработок. Мы работали с такими гигантами IT-индустрии, как, например, HP и Dell. В те годы до 90% бюджета Института формировалось за счёт контрактов с зарубежными компаниями. Это позволило нам сохранить научную школу, удержать кадры и развивать собственные компетенции, когда в стране практически отсутствовал рынок заказов на сложное программное обеспечение. Так, в начале 2000-х ИСП РАН насчитывал чуть более 100 сотрудников. Сегодня нас уже более 800.

Да, в тот период наша страна действительно занимала вспомогательные позиции в мировой ИТ-кооперации, а по ряду направлений — в частности, системному программированию и средствам разработки — и вовсе утратила лидерство. Однако время показа-

ло: это отставание носило не фундаментальный, а организационный характер. За последние 20 лет мы прошли серьезный путь, и теперь Россия одна из лидирующих стран в области системного программирования. Это стало возможным не только за счет передовых научных школ и соответствующих компетенций, но и благодаря наличию индустриального открытого ПО.

— В чём, по-вашему, разница между «Импортозамещением» и «Технологической независимостью»? Какую роль в достижении технологической независимости играет подготовка кадров? Что делается в институте в этом направлении?

— Импортозамещение — это своеобразное «затыкание дыр». Технологическая независимость — способность создавать и долгосрочно развивать конкурентоспособные на мировом уровне продукты, необходимые внутри страны, несмотря на внешние факторы.

Безусловно, подготовка кадров играет здесь ключевую роль. Однако невозможно достичь технологической независимости, если действовать в отрыве от науки и индустриальной разработки. Важно, чтобы специалисты формировались внутри живых профессиональных экосистем, где знания передаются не только через учебные программы, но и благодаря участию в реальных проектах. Я уже говорил о том, что открытые коды позволяют реализовывать этот механизм эффективно. Если в нашей стране сформируется, к примеру, сообщество программистов, которые способны работать с исходниками Linux и внедрять туда свои инновации и решения — считайте, что у нас есть национальная операционная система, ведь де-факто Linux — это мировой стандарт для ОС общего назначения.

Именно такую модель подготовки кадров и развивает Институт. Мы не просто обучаем специалистов, а создаем сообщества вокруг ключевых технологий: компиляторов, операционных систем, средств анализа и верификации. ИСП РАН формирует экосистему молодых исследователей, инженеров и преподавателей, объединенных общими задачами и открытым обменом знаний.

Сегодня мы задаем этот тренд в академической и инженерной среде, формируя поколение разработчиков, для которых технологическая независимость — не лозунг, а естественная часть профессиональной культуры.

— Расскажите о работах по созданию доверенного ядра ОС Линукс, ведущихся в ИСП РАН. Не дублируются ли таким образом результаты разработки других ОС на базе ядра Линукс, внедряемые в России в рамках импортозамещения?

— В 2021 году при поддержке ФСТЭК России в нашем Институте был создан Технологический центр исследований безопасности ядра Linux. Его задача — объединить отечественных разработчиков и повысить качество исследований. Сегодня в центре участвуют более 30 организаций. На его базе проводятся исследования безопасности стабильных веток ядра с применением статического анализа, фаззинга и функционального тестирования. Исправленные уязвимости передаются в международное сообщество, а в основную ветку Linux уже включено более 650 патчей, разработанных российскими специалистами.

Эта работа не дублирует проекты других компаний, а наоборот, сокращает избыточные усилия и повышает качество отечественных решений на базе Linux.

Мы очень гордимся тем, что наш Институт смог организовать внутри страны целое сообщество из более чем 80 высококлассных специалистов, которые, будучи представителями конкурирующих компаний, работают вместе ради общего дела.

— Какие ещё крупные проекты в части разработки ПО ведет институт?

— Сегодня ИСП РАН ведет ряд крупных проектов, охватывающих полный цикл разработки надежного и безопасного программного обеспечения. Среди них — полный стек инструментов РБПО по компилируемым языкам. Мы создали и активно используем технологию Svace. Это статический анализатор исходного кода и необходимый инструмент жизненного цикла разработки безопасного ПО, который обнаруживает более 70 классов критических ошибок в исходном коде. Для обеспечения совместной работы и управления результатами, сервер просмотра предупреждений Svacer поддерживает интеграцию в CI/CD, предоставляет интерфейс командной строки для интеграции в типовые CI/CD-процессы. Также одним из важнейших инструментов является AstraVer Toolset — система дедуктивной верификации ключевых компонентов. Она позволяет разрабатывать и верифицировать модели политик безопасности, а также проводить доказательство корректности компонентов на языке C. Реализация новых возможностей в ОС Astra Linux Special Edition продолжает вери-

фицироваться с помощью AstraVer Toolset. Для верификации ядер и драйверов различных операционных систем успешно применяется система Klever, которая автоматически строит модели на основе исходного кода на языке программирования Си, позволяет применять наиболее точные методы анализа к большому объёму исходного кода.

Кроме того, в нашем активе есть программный комплекс ИСП Crusher, который комбинирует несколько методов динамического и статического анализа, в частности, фаззинг (ISP Fuzzer) и символьное выполнение. Комплекс Sydr-fuzz реализует гибридный фаззинг с использованием символьного выполнения для автоматической генерации тестов с целью увеличения покрытия кода и обнаружения ошибок.

Нами также создан Natch — инструмент для определения поверхности атаки, основанный на полносистемном эмуляторе QEMU. Он может быть встроен в CI/CD для интеграционного и системного тестирования, что позволит повысить эффективность технологий функционального тестирования и фаззинга в жизненном цикле безопасного ПО.

Помимо сферы РБПО, наш Институт работает в области доверенного ИИ, создавая инструменты и методы для разработки соответствующих технологий. Для этих целей существует Платформа Доверенного ИИ, которая обеспечивает автоматизацию процессов по выявлению и устранению угроз, возникающих на всех этапах жизненного цикла создания моделей машинного обучения. Мы разработали доверенную версию платформы Talisman для построения информационно-аналитических систем. Talisman — это комплекс взаимосвязанных программных инструментов для автоматизации типовых задач обработки данных, включая их сбор, интеграцию, анализ, хранение и визуализацию. Он обеспечивает быструю разработку специализированных многопользовательских интеллектуальных аналитических систем. А для хранения данных и проведения ресурсоёмких вычислений в научных, образовательных и коммерческих целях разработана платформа Asperitas.

ИСП РАН также работает и над другими проектами — например, по созданию операционной системы реального времени JetOS совместно с ГосНИИАС, а также ведёт сопровождение двух веток ядра Linux, основанных на стабильных версиях 5.10 и 6.1.

Важно отметить, что в последние годы наш Институт перешёл от разработки отдельных технологий к созданию сообществ вокруг них. Мы убеждены: именно такая тактика обеспечит цифровой суверенитет и безопасность программной инфраструктуры России.

— Расскажите об участии ИСП РАН в создании стандартов в области разработки ПО (ГОСТ 56939–2024 по разработке безопасного ПО, будущий ГОСТ по композиционному анализу). Как вы взаимодействуете с профильными сообществами?

— Благодаря научно-техническим достижениям Института в области разработки программного обеспечения ФСТЭК России вовлекла нас в процессы стандартизации разработки безопасного ПО в составе Технического комитета по стандартизации «Защита информации» (ТК 362) ещё в 2018 году. Для этой работы был создан Подкомитет № 2 «Разработка безопасного ПО», который объединил более 60 российских организаций. В него входят разработчики программного обеспечения, испытательные лаборатории, органы по сертификации и профильные государственные ведомства. Председателем Подкомитета является Вартан Падарян — наш ведущий научный сотрудник.

Была разработана концепция развития национальной системы стандартизации в области разработки безопасного ПО, которая поэтапно реализуется. Один из ключевых результатов — подготовка и обновление ГОСТ 56939–2024, задающего общие требования к процессу разработки безопасного ПО. Работа над ним велась в рамках рабочей группы из десяти организаций. Первую редакцию предложила «Лаборатория Касперского», которая обладала большим практическим опытом в этой области.

Система рабочих групп — основа деятельности подкомитета. В таком формате идет работа и над новыми документами, в том числе над стандартом по композиционному анализу программного обеспечения. Этот проект сейчас проходит публичное обсуждение.

Мы видим, что работа над стандартами помогает объединять профессиональное сообщество. В ней участвуют и ученые, и разработчики, и представители отрасли. Такое сотрудничество позволяет вырабатывать понятные и единые правила создания качественного и безопасного программного обеспечения.

— Какова роль института в рамках сертификации процессов разработки безопасного ПО?

— Роль нашего Института здесь многопланова. С запуском сертификации процессов

разработки безопасного ПО, ИСП РАН более года оставался единственным органом по сертификации в данной области. За год было проведено 7 сертификаций, которые позволили накопить и обобщить методическую базу. Сейчас этот опыт внедряется в проект национального стандарта «Методика оценки уровня реализации процессов разработки безопасного программного обеспечения». Его разработка ведется в рабочей группе в ТК 362.

— Ваше мнение о необходимости создания отечественных репозиториев? Какие задачи они должны решать? Почему недостаточно имеющихся репозиториев, создаваемых крупными российскими компаниями и их объединениями?

— Чтобы ответить на этот вопрос, необходимо для начала определить, какую ключевую задачу мы ставим перед репозиториями. Если рассматривать их в качестве аналога сервиса GitHub, то в России уже есть вполне зрелые платформы. Это, например, GitVerse от СберТеха. Однако я считаю, что основная задача — не «импортозамещать» зарубежный продукт, а продвигать нашу страну к технологической независимости. Именно поэтому репозиторий должен объединять не только технологии, но и специалистов. Железа и софта недостаточно — нужны люди. Ещё раз подчеркну: сегодня требуются не просто хранилища проектов, а правильно выстроенная организация совместной работы — по сути, система сообществ, где развивается код, обсуждаются идеи, формируются новые решения. Репозитории, в свою очередь, должны быть инструментом создания таких сообществ, и наш Институт успешно реализует эту идею на практике.

— Сейчас много говорят о технологиях искусственного интеллекта. ИСП РАН входит в «Консорциум исследований безопасности технологий искусственного интеллекта». Как данные работы связаны с основной тематикой института?

— ИСП РАН традиционно занимается фундаментальными и прикладными исследованиями в области эффективности, продуктивности и безопасности программных систем. С появлением технологий искусственного интеллекта эти направления получили новое измерение: теперь важно не только обеспечить корректность и защищённость программного кода, но и сформировать доверие к поведению моделей машинного обучения. Это включает анализ и валидацию датасетов, обеспечение безопасности и надёжности процессов обучения, контроль корректности применения моделей и многие другие аспекты. Для системного ответа на эти вызовы в 2021 году в нашем Институте был создан Исследовательский центр доверенного искусственного интеллекта, который объединил специалистов по программной инженерии, безопасности и методам машинного обучения.

Что касается Консорциума исследований безопасности технологий искусственного интеллекта, ИСП РАН стал одним из его основателей, совместно с НТЦ ТК и Академией криптографии. Участие в создании такого Консорциума стало для нас естественным продолжением основной научной деятельности. Мы рассматриваем ИИ как сложную программно-аппаратную систему, зачастую требующую формальных гарантий корректной, безопасной и предсказуемой работы на всех этапах жизненного цикла — от проектирования и обучения до эксплуатации. Консорциум создаёт возможность тесного взаимодействия с индустриальными партнёрами и позволяет получать обратную связь по применению наших технологий в реальных проектах.

— Нужно ли здесь регулирование со стороны государства? Какие нормативные документы по теме искусственного интеллекта необходимо разработать?

— Безусловно, регулирование в области искусственного интеллекта необходимо. Особенно, когда речь идёт о системах, способных повлиять на безопасность, права и жизнь человека. При этом важно, чтобы такое регулирование было разумным и сбалансированным. Его задача — не сдерживать развитие технологий, а формировать рамки доверия, ответственности и прозрачности их применения.

Сегодня требуется разработка единой системы терминов и классификаций рисков искусственного интеллекта, чтобы разработчики, заказчики и регуляторы говорили на одном языке. Также необходимы стандарты оценки доверия и безопасности ИИ-систем, включая методики тестирования, верификации и сертификации. Важно определить нормативные требования к ответственному использованию данных и обучению моделей, особенно в части прозрачности их происхождения и обработки. Кроме того, нужны рекомендации по безопасной интеграции ис-

кусственного интеллекта в критические и государственные системы.

Особое внимание, на мой взгляд, должно уделяться регулированию применения генеративных технологий ИИ.

Здесь важно учитывать не только технические, но и социальные аспекты — влияние дипфейков, культурно-этические особенности использования ИИ в образовании, а также анализ и предотвращение распространения деструктивного контента в интернете.

ИСП РАН уже принимает участие в ряде рабочих групп по подготовке соответствующих нормативных документов: мы предлагаем решения, основанные на научно обоснованных моделях и опыте Института в области доверенного ИИ. При этом фокус наших усилий не на разработке самих регуляторных актов, а на научно-методологической и инструментальной поддержке их формирования.

— Расскажите об участии института в проекте «Специализированный конвейер безопасной разработки приложений с моделями машинного обучения (MLSecOps)».

— Цель проекта — сделать так, чтобы безопасность и прозрачность присутствовали на всех этапах работы с моделями — от обучения до внедрения.

Мы разработали собственную MLOps-платформу, сопоставимую с известными зарубежными решениями вроде MLflow или Kubeflow, но с акцентом на безопасность. Она позволяет отслеживать версии моделей, управлять экспериментами, тестировать и внедрять их, а главное — контролировать надёжность и защищённость.

В систему входят инструменты для проверки моделей на уязвимости, защиту от атак, анализ предвзятости, интерпретацию решений и поиск вредоносных фрагментов в предобученных моделях. Все компоненты построены на доверенном программном обеспечении, включая адаптированные версии TensorFlow и PyTorch, разработанные в ИСП РАН.

Проще говоря, MLSecOps формирует основу экосистемы доверенного ИИ — той, где безопасность не «дописывается» потом, а встроена изначально.

— Нет смысла говорить о доверии к ПО, не имея своего «доверенного» железа. Сталкиваетесь ли вы с недостатком высокопроизводительных отечественных систем и проблемами отечественных программно-аппаратных комплексов? Если сталкиваетесь, то с какими, как их решаете?

— В настоящее время нет единого понимания того, что входит в понятие доверенности. Такие работы ведутся в данный момент — в частности, по стандартизации в сфере доверенных СБИС и ПАК в ТК 167 («Программно-аппаратные комплексы для критической информационной инфраструктуры и программное обеспечение для них») и ТК 642 («Защита информации»). Здесь требуется решить широкий спектр задач, среди которых: аппаратный корень доверия, доверенные среды исполнения, средства защиты информации, доверенные маршруты проектирования СБИС (САПР). Отдельные исследования и разработки выполняются в нашем Институте, а также в «Лаборатории Касперского», НПО «КИС», НТЦ «Атлас», «Аквариус», «Байкал Электроникс» и других организациях.

В отечественных высокопроизводительных системах используются либо зарубежные микропроцессоры, либо микропроцессоры, спроектированные в РФ, но произведённые за рубежом («Эльбрус», «Байкал»). Однако важно отметить: у нас есть значительный потенциал. В России уже работают дизайн-центры, созданы собственные процессоры. И если правильно выстроить взаимодействие науки, промышленности и образования, выясняется: для большинства задач у нас либо уже есть своё железо, либо мы можем создать его в короткие сроки. Исключение составят графические акселераторы, где пока действительно будет сохраняться зависимость от зарубежных решений.

— Имеется ли у Вас информация, что делается для решения проблем с элементной базой для высокопроизводительных вычислительных систем и что необходимо сделать?

— В России есть возможности для развития сектора бесфабричного проектирования микроэлектроники разного уровня сложности и назначения. Возможности в этом секторе выросли с появлением экосистемы свободно распространяемых компонентов (в том числе на базе архитектуры RISC-V).

Используемые на отечественных фабриках технологии не подходят для производства конкурентоспособных чипов. В 2020 году была принята «Стратегия развития электронной промышленности Российской Федерации на период до 2030 года», согласно которой

в РФ должны появиться фабрики, производящие цифровые СБИС по технологическим нормам 7–5 нм. Это крайне сложная (если не невыполнимая в обозначенные сроки) задача, но решать её необходимо.

Приоритетный вопрос — разработка отечественных САПР. В настоящее время по заказу Минпромторга России ведутся работы по созданию маршрутов проектирования цифровых и аналоговых СБИС. Головными исполнителями выступают АО «МНТЦ МИЭТ» и АО «НПО «КИС», в кооперацию входят ООО «Альфачип», МГУ имени М. В. Ломоносова, ООО «Эремекс» и другие. ИСП РАН разрабатывает инструменты логического синтеза и верификации цифровых СБИС.

Меры, которые необходимо принять для решения проблем с элементной базой для высокопроизводительных вычислительных систем:

- обновить программы подготовки инженеров в технических вузах страны, обеспечить возможность доведения студенческих проектов до фабрик, использовать в учебном процессе САПР отечественной разработки;
- стартовать НИОКР в перспективных областях проектирования и производства микроэлектроники, включая области, в которых за счёт применения ИИ можно кратно увеличить продуктивность работы, а также области, связанные с обеспечением доверенности;
- создать опытное производство микроэлектроники для проектных норм 28–22 нм (с последующим выходом на 16–12 нм и 7–5 нм), адаптировать отечественные САПР под соответствующие технологии.

— ИСП РАН проводит ежегодную открытую конференцию. Расскажите о её концепции? Какие задачи с ее помощью решает институт?

— Наша ежегодная конференция — это не просто научное мероприятие, а отражение философии Института. Главная идея неизменна, и это открытость. Мы уверены, что даже развивая технологическую независимость, нельзя замыкаться в себе. Поэтому мы приглашаем к участию университеты, компании, научные центры, госструктуры, и активно работаем с международным сообществом.

ИСП РАН всегда работал на стыке образования, науки и инноваций. Конференция же объединяет всё это в одном пространстве. Центральная ось — наука, поэтому на мероприятии мы представляем результаты фундаментальных исследований и технологий, которые внедряются в индустрию и госсектор. Особое внимание уделяется кибербезопасности: в программе присутствует организованный совместно со ФСТЭК блок, посвященный разработке безопасного ПО. В целом на конференции представлены разные направления: к примеру, цифровая медицина и цифровое моделирование технических систем.

Также наш Институт проводит выставку технологий ИСП РАН, которые внедрены в различные отрасли промышленности. К участию в ней мы приглашаем партнёрские организации и ведущие компании индустрии. Более того, нами выделен отдельный трек, связанный с образованием: ИСП РАН организует круглые столы с участием представителей ведущих вузов.

Мы видим, что выбрали верный подход. В 2024 году, когда Институт отметил 30-летие, конференция собрала полторы тысячи человек очно. Это доказывает: то, чем мы занимаемся, действительно востребовано.

— В организации каких ещё конференций участвует ИСП РАН?

— Помимо ежегодной открытой конференции, мы являемся соорганизаторами других научных мероприятий. Например, «OS DAY» — это важная площадка по операционным системам, «Иванниковские чтения» — в память о нашем основателе, Викторе Петровиче Иванникове. ИСП РАН также активно участвует в междисциплинарных и стратегических форумах, ведь наша работа выходит за рамки чистого программирования. В сфере медицины и ИИ мы, например, соорганизуем II Школу молодых учёных «Применение цифровых технологий в медицине» совместно с РЭУ им. Г. В. Плеханова. Мы активно поддерживаем молодёжь: проводим SYRCoSE, АПСПИ, молодёжную конференцию МПОИТЭС.

ИСП РАН не только участвует в крупных международных конференциях, но и иногда организует мероприятия в ходе них. Например, в рамках Международной конференции по компьютерным наукам и информационным технологиям «CSIT 2025» мы организовали форсайт-сессию, посвященную доверенному ИИ, а также провели семинар по цифровой кардиологии на «IEEE COMPSAC2024».

Все эти мероприятия вместе с участием наших специалистов в роли докладчиков на ведущих международных конференциях — ещё один важный элемент развития области системного программирования в России.

Вопросы задавал Сергей Пазизин, научный редактор BIS Journal

АРХИТЕКТУРНЫЙ АНАЛИЗ

ТРИ КИТА РБПО. КИТ № 3

Дмитрий ПОНОМАРЕВ
заместитель генерального директора — директор департамента внедрения и развития практик РБПО ООО НТЦ «Фобос-НТ», сотрудник ИСП РАН, преподаватель МГТУ им. Н. Э. Баумана

В этом блоке, завершающем серию «Три кита РБПО», речь пойдёт о методологиях и инструментах анализа, объединённых в категорию «архитектурный анализ», или «комплексный архитектурный анализ» (КАО) в терминах, принятых в Методике выявления уязвимостей и недекларированных возможностей ПО ФСТЭК России [1]. Данный термин не в полной мере отражает всё многообразие охватываемых методологий, а для некоторых конкретных видов, на первый взгляд, не подходит совершенно. Представляется полезным не только дать характеристику наиболее значимых видов анализа, относящихся к категории «архитектурный», но и сделать небольшой экскурс в историю с объяснением причин, приведших к возникновению сложившейся традиции именования.

Попробуем понять, что именно приходит в голову сотруднику испытательной лаборатории в системе сертификации ФСТЭК России с многолетним стажем, когда он слышит словосочетание «архитектурный анализ». В предыдущих статьях цикла я уже упоминал о неформальной терминологии — РБПО «вширь» и «вглубь» [2]. «Вширь» — подразумевает комплексный анализ наличия в компании процессов РБПО, определяемых ГОСТ 56939–2024 [3] (далее — ГОСТ РБПО), а также минимально необходимого набора артефактов, подтверждающих реализацию процессов с требуемыми характеристиками. «Вглубь» — больше про реализацию конкретных практик РБПО применительно к конкретной кодовой базе конкретных средств защиты информации (или иного программного обеспечения) в соответствии с Методикой выявления уязвимостей и недекларированных возможностей ФСТЭК России (далее — Методика ВУ и НДВ). Перечисленные в Методике ВУ и НДВ требования и рекомендации в первую и основную очередь относятся к разработчикам и испытательным лабораториям, зачастую совместными усилиями готовящим продукты к сертификационным испытаниям и данные испытания проходящим (проводящим). Однако они могут использоваться в качестве коллекции отечественных лучших практик и критериев успешности их выполнения при анализе любых продуктов, не ограничиваясь средствами защиты информации.

АРХИТЕКТУРНЫЙ АНАЛИЗ И ГОСТ РБПО

Какие процессы, описанные в ГОСТ РБПО, можно отнести к архитектурному анализу? Разумеется, в первую очередь это процессы № 6 «Разработка, уточнение и анализ архитектуры ПО» и № 7 «Моделирование угроз и разработка описания поверхности атаки». Нет смысла комментировать далее — названия процессов говорят сами за себя, без участия роли «архитектор» (а в ряде крупных компаний с высокой гранулярностью ролей — роли «архитектор ИБ») выполнение данных процессов не представляется возможным. Вопрос отнесения остальных процессов к категории «архитектурный анализ» — дискуссионный. Как это часто случается, не договорившись об аксиоматике и типовых моделях (ролей, процессов) в производственных командах, сложно и, скорее всего, контрпродуктивно утверждать о наличии единого (или, того хуже, единственно верного) подхода к категорированию. Например, относить ли к «архитектурной» категории действия, осуществляемые в рамках процесса № 3 «5.3 Формирование и предъявление требований безопасности к ПО»? Разумеется, хорошо проработанный глоссарий типовых требований, включающий в себя требования различных типов (к функционалу, к анализу), вероятно, будет формироваться и поддерживаться с привлечением архитектора команды (компании в целом). Однако, во-первых, в небольших компаниях это может быть реализовано и на уровне роли руководителя проекта либо ведущего разработчика, а во-вторых, роль архитектора не обязательно будет главной (определяющей) при реализации данного процесса. Вполне могут найтись аргументы, обосновывающие тезис: «Требования от бизнеса, реализация от разработчиков, инфраструктура трекера и тестов от девопсов — не стоит пихать везде несчастного архитектора». Или процесс № 16 «Использование инструментов композиционного анализа» — процесс,

во многом также зависящий от формирования контролируемого репозитория [4] и внедрения инструментов автоматизации данного процесса. Кто-то скажет, что формирование политик наполнения контролируемого репозитория сторонними компонентами невозможно без архитектурного комитета, а кто-то сочтёт, что данный функционал вполне по силам специалисту ИБ или РБПО. Таким образом, с позиции ГОСТ РБПО к процессам архитектурного анализа стоит относить в первую очередь именно те процессы, при реализации которых в выбранной организации специалисты, обладающие ролью «архитектор», принимают непосредственное и даже руководящее участие.

АРХИТЕКТУРНЫЙ АНАЛИЗ И МЕТОДИКА ВУ И НДВ

Если ГОСТ РБПО говорит об организации процессов в целом, то Методика ВУ и НДВ в первую и основную очередь посвящена тому, что именно и в каком объёме необходимо делать исследователям безопасности — неважно, сотрудник ли это команды разработки продукта или сотрудник испытательной лаборатории, осуществляющей сертификационные испытания. Там, где ГОСТ, на примере горячо любимого мною фаззинга, говорит примерно следующее: «Нужно фаззить то, что нужно, при этом регулярно и желательно с привлечением разных дополнительных инструментов, повышающих эффективность анализа», Методика ВУ и НДВ конкретизирует: «Необходимо взять X сэмплов, Y датчиков срабатывания ошибок и выполнять фаззинг-тестирование без роста покрытия в течение Z часов с использованием инструментов, удовлетворяющих комплекту функциональных требований T». Соответственно, и раздел КАО в Методике ВУ и НДВ — это в первую очередь описание конкретных практик анализа (порой с точностью до описания инструментов), а также описание критериев достаточности их применения в ходе конкретных исследований, с учётом глубины исследований (уровень контроля) и некоторых конкретных особенностей испытываемого продукта.

При этом необходимо подсветить важный «исторический» нюанс. Методика ВУ и НДВ, как и парадигма ФСТЭК-РБПО в целом, уделяет достаточно существенное внимание «тяжёлым» практикам анализа, предполагающим непосредственно работу с исходным или исполняемым (в том числе специально модифицированным) кодом. Данные «тяжёлые» практики описаны в разделах САО (статический анализ исходного кода) и ДАО (в Методике ВУ и НДВ выделяется три вида динамического анализа: модульное, фаззинг- и системное тестирование) — им посвящены прошлые номера рубрики «Три кита» [4, 5]. Наличие акцента на указанных практиках, с одной стороны, позволяет существенно повысить доверие к исследуемому программному обеспечению за счёт необходимости глубокого контроля над создаваемым кодом, что практически невозможно без системно отстроенных процессов анализа и наличия профильных специалистов РБПО. С другой же стороны, «тяжёлые» практики, как правило, весьма ограниченно применяются в легковесном, «коммерческом» РБПО, состоящем из большего числа лучше автоматизируемых практик, к числу которых относятся: композиционный анализ, анализ избыточных привилегий процессов и ресурсов, поиск захардкоженных приватных параметров (пароли, ключи, токены), различные варианты DAST [2] и автоматизированного пентеста и т.д.

В силу принятой парадигмы при создании ФСТЭК России первой версии Методики ВУ и НДВ в 2018 г. разделы САО и ДАО были посвящены вполне конкретным видам анализа — «тяжёлым», требующим наличия исходного кода и контроля над созданием из него кода исполняемого, с проработанными критериями выполнения и высоким профессиональным уровнем специалистов. Но видов анализа, полезных при проведении испытаний, существенно больше, тем более что новые виды/подвиды появляются достаточно динамично. Плодить для каждого собственный раздел — высокоуровневую структуру в составе Методики ВУ и НДВ — решение спорное и вряд ли оптимальное. Именно поэтому множество различных видов и конкретных частных практик анализа кода, конфигурации и архитектуры, весьма разношёрстных и разномасштабных, было решено «свалить» в раздел КАО, где они и пребывают по настоящее время. Исключение составляют методологии определения поверхности атаки и ручного направленного анализа, в силу своей значимости и специфичности также вынесенные в отдельные разделы — ПОД.1 и ЭКО соответственно.

ВИДЫ АРХИТЕКТУРНОГО АНАЛИЗА ПРИ ПРОВЕДЕНИИ СЕРТИФИКАЦИОННЫХ ИСПЫТАНИЙ

Разобравшись с историей возникновения категории «архитектурный анализ» (кстати, ей посвящена одна из трёх программ образовательных курсов ФСТЭК России, организуемых на базе ИСП РАН [6]), уделим немного внимания текущему и перспективному [7] наполнению данного раздела Методики ВУ и НДВ в формате тезисов:

- многочисленные виды архитектурного анализа можно делить на подкатегории различными способами, в том числе на статические и динамические;
- исторически к видам архитектурного анализа относятся: проверка наличия и эксплуатируемости известных уязвимостей в сторонних компонентах с открытым исходным кодом (подмножество задач композиционного анализа); поиск захардкоженных приватных параметров, а также поиск компрометирующих конструкций и комментариев в коде ПО; поиск ошибок конфигурирования сервисов в составе ПО; динамический анализ исследуемого ПО в формате «песочницы» — в попытке выявить нештатные/недекларированные взаимодействия и ряд иных практик;
- в раздел КАО включены задачи тестирования на проникновение, вы-

полняемого в ходе сертификационных испытаний;

- в связи с выходом информационного письма ФСТЭК России об уточнении порядка исследований СЗИ в контейнерном исполнении [8] в разрабатываемую сейчас версию Методики ВУ и НДВ будет добавлен раздел, аналогичный требованиям информационного письма;
- важность задачи минимизации привилегий различных видов, в разрабатываемую сейчас версию Методики ВУ и НДВ будет добавлен раздел, посвящённый необходимости выявления повышенных привилегий и их последующей минимизации либо обоснования безальтернативности данного решения;
- вопросы харденинга сборок с использованием возможностей компиляторов (в том числе, например, с использованием безопасного компилятора SafeC [9]), также относятся к ведению раздела «Архитектурный анализ».

Методика ВУ и НДВ не ограничивает исследователя перечисленными методами анализа. Раздел КАО является одним из наиболее «методичных» и наименее «требовательных», подсказывая исследователю возможные векторы и аспекты рассмотрения защищённости анализируемого продукта. Яркий пример — рекомендация по анализу безопасности конфигураций компонентов, относящаяся не только к продукту в целом, но и к конкретным компонентам с открытым исходным кодом, ключевым в составе продукта. Из многолетнего опыта работы известно, что в типовых СЗИ за формирование непосредственной поверхности атаки и реализацию основного функционала почти всегда отвечает подмножество из менее чем 100 компонентов с открытым исходным кодом (существенная часть которых уже исследуется в рамках деятельности Центра исследований безопасности системного ПО [10]). Для многих из этих компонентов — зачастую популярнейших, используемых во множестве продуктов по всему миру — созданы открытые или платные инструменты оценки безопасности их конфигурирования. По уровню своей технологичности они не всегда заслуживают даже отнесения к «линтерам»[1]. Даже их полноценный учёт (а уж тем более какая-то «сертификация» в качестве «типового» инструмента анализа) является задачей нетривиальной и, скорее всего, малополезной практически. Однако рекомендация поиска и применения актуальных «прямо сейчас» средств анализа конфигураций, вне зависимости от их «технологичности» и «регалий», приносит ощутимую практическую пользу. В данном типе инструментов мощность «движка», как правило, не принципиальна — на первый план выходит полнота и актуальность набора экспертных правил, определяющих эксплуатационные «мисконфиги» компонентов.

Ещё один пример — возможность применения инструментов, подобных Luntry [12], для комплексного исследования защищённости контейнерных решений, предназначенных для выполнения под управлением оркестратора k8s. Несмотря на то что основным предназначением инструмента является обеспечение безопасности функционирующих в проде кластеров, он вполне и с успехом может применяться и для выявления различных «избыточностей» в соответствии с информационным письмом [8].

Более подробные описания ряда методологий и инструментов архитектурного анализа представлены уважаемыми участниками в следующих статьях рубрики. Обсуждению вопросов архитектурного анализа посвящён чат @sdl_arch в семействе чатов РБПО-сообщества ФСТЭК России и ИСП РАН [13]. В качестве заключения стоит отметить, что архитектурный анализ при проведении сертификационных испытаний ограничивается именно исследованиями в области безопасности кода, конфигурации и архитектуры ПО. Анализ и критика эффективности выбранного архитектурного подхода (например, монолит или микросервисы), если только он не несёт угроз безопасности, не входит в область задач эксперта испытательной лаборатории.

[1] «Линтер» представляет собой классический пример термина, вроде бы понятного всем. Однако порой попытка чёткой классификации инструментов на «линтеры» / «не линтеры» может привести к небольшой ссоре между вами и вашим визави. Лучший из известных мне подходов к классификации содержится в разделе 3 [11]

ИСТОЧНИКИ

[1] Новость об утверждении Методики ВУ и НДВ: https://clck.ru/3Q3aWn

[2] Пономарев Д. Многоликий динамический анализ // BIS Journal.— 2025.— № 3 (58). https://ib-bank.ru/bisjournal/post/2540

[3] ГОСТ 56939–2024: https://docs.cntd.ru/document/1310017763

[4] Проект ГОСТ «Композиционный анализ программного обеспечения» на сайте ФСТЭК, раздел «Термины и определения»: https://clck.ru/3Q3ajR

[5] Пономарев Д. Испытания статических анализаторов //BIS Journal.— 2025.— № 2 (57). https://ib-bank.ru/bisjournal/post/2388

[6] Курсы ФСТЭК России на базе ИСП РАН https://cpkstzi.ru/pr_33

[7] Запись трансляции «Актуальные вопросы защиты информации» (https://www.tb-forum.ru/2025/program/information-security), ссылка на просмотр https://clck.ru/3NcrPB

[8] Информационное сообщение ФСТЭК России «О повышении безопасности средств защиты информации, в состав которых разработчики включают средства контейнеризации или образы контейнеров» https://clck.ru/3Q3aru

[9] Безопасный компилятор Си/Си++ https://www.ispras.ru/technologies/safecomp/

[10] Центр исследований безопасности системного программного обеспечения https://gitlab.community.ispras.ru/cc-portal/intro/-/blob/master/README.md

[11] Статья «Статический анализатор Svace как коллекция анализаторов разных уровней сложности» https://svace.pages.ispras.ru/svace-website/pdf/svace2015.pdf

[12] Евдокимов Д. Всё для SOC в контейнерных средах и Kubernetes // BIS Journal.— 2025.— № 3 (58). https://ib-bank.ru/bisjournal/post/2517

[13] Информационные ресурсы РБПО-сообщества ФСТЭК России и ИСП РАН https://t.me/sdl_community/7859

КОМПОЗИЦИОННЫЙ АНАЛИЗ КАК ФУНДАМЕНТ КАЧЕСТВЕННОЙ И БЕЗОПАСНОЙ РАЗРАБОТКИ ПО

О ВАЖНОСТИ И СПОСОБАХ ПРОВЕРКИ OPEN SOURCE НА БЕЗОПАСНОСТЬ И КАЧЕСТВО

Алексей СМИРНОВ
основатель
CodeScoring

Роман ЛАПИН
директор
по разработке
продуктов
CodeScoring

ПРЕДПОСЫЛКИ ПРОБЛЕМАТИКИ

Каждая первая статья про композиционный анализ начинается с очевидной истины: современную разработку сложно представить без использования открытого стороннего программного обеспечения. Благо очевидное — готовые библиотеки решают известные задачи, за счёт чего ускоряется выпуск ваших программных продуктов на рынок. Это на поверхности.

Сегодня в разработке открытого кода хоть раз поучаствовало уже более 100 млн человек, а регулярно в написании кода участвует около 10 млн разработчиков. Это очень разные люди с разной квалификацией и когнитивными способностями. А в последние годы плодовитыми соавторами открытых проектов стали ещё и роботы, они же ИИ-ассистенты.

В мире уже сейчас опубликовано более 300 млн проектов под флагом Open Source. Любой из этих проектов может стать частью вашего программного продукта, и вопрос привносимого качества и безопасности сегодня игнорировать нельзя.

Очевидно, сторонний код (ровно как и собственный) не лишён риска наличия в нём функциональных ошибок и ошибок, приводящих к отказу приложения.

Проблемы могут быть идентифицированы не сразу, и «тысяча глаз» открытого сообщества не всегда оказывается эффективной. Ярким примером является громкая ACE-уязвимость Log4Shell, обнаруженная в 2021 году в библиотеке Log4j, которая оказала эффект на 8% всей Java-экосистемы. Дефект в коде был выявлен спустя более 10 лет после того, как он был создан. Уязвимость позволяет производить удалённое выполнение кода путём нехитрого веб-вызова, чем в первые дни начали пользоваться злоумышленники, автоматически распространяющие шифровальщики и криптомайнеры. Тем не менее, несмотря на серьёзность проблемы и широкое медийное обсуждение, в 2025 году выходит больше библиотек, которые используют именно опасную версию вместо безопасной. Именно от этой уязвимости можно защититься наложенным средством (WAF), но станет ли ваше приложение от этого безопасным?

Недостаточный контроль за сторонними библиотеками влияет не только на вопросы безопасности. В гонке за повышением скорости выпуска продуктов разработчики не всегда успевают актуализировать версии используемых компонентов, что неизбежно приводит к возрастанию объёма технического долга: замораживаются старые версии библиотек и технологии. Их обновление влечёт решение задачи обратной совместимости, которое сопровождать становится всё дороже. Если не уделить внимание этой проблеме, то продукт может закончить свою жизнь раньше ожидаемого срока. В качестве примера здесь можно привести python-проекты, которые застряли на версии 2.7, а средств и времени для перехода на свежую ветку интерпретатора нет.

Отсутствие культуры работы со сторонними компонентами может привести не только к наличию уязвимостей в выпускаемом программном обеспечении, но и к риску потерь чувствительных данных, если просто всегда использовать последние версии библиотек. Примером подобной угрозы является червь Shai-Hulud, который завёлся в пакетной экосистеме NPM (пользуются JavaScript-разработчики). Червь успел заразить более 500 открытых библиотек, в том числе «миллионники». Как это работает: при установке пакета с червём запускается поиск конфиденциальной информации (паролей, токенов, ключей и иных) в коде разработчика, после чего они отправляются в сторону злоумышленника. Если у разработчика были собственные библиотеки в NPM, то червь себя туда копировал через выпуск новой версии, чем обеспечивал себе распространение.

Кроме вопросов безопасности и качества, нельзя обойти правовой аспект. Сторонний код написан третьими лицами и, как правило, сопровождается лицензионным соглашением, которое конечный пользователь (разработчик) должен соблюдать. В мире сегодня насчитывается более 2000 типов лицензион-

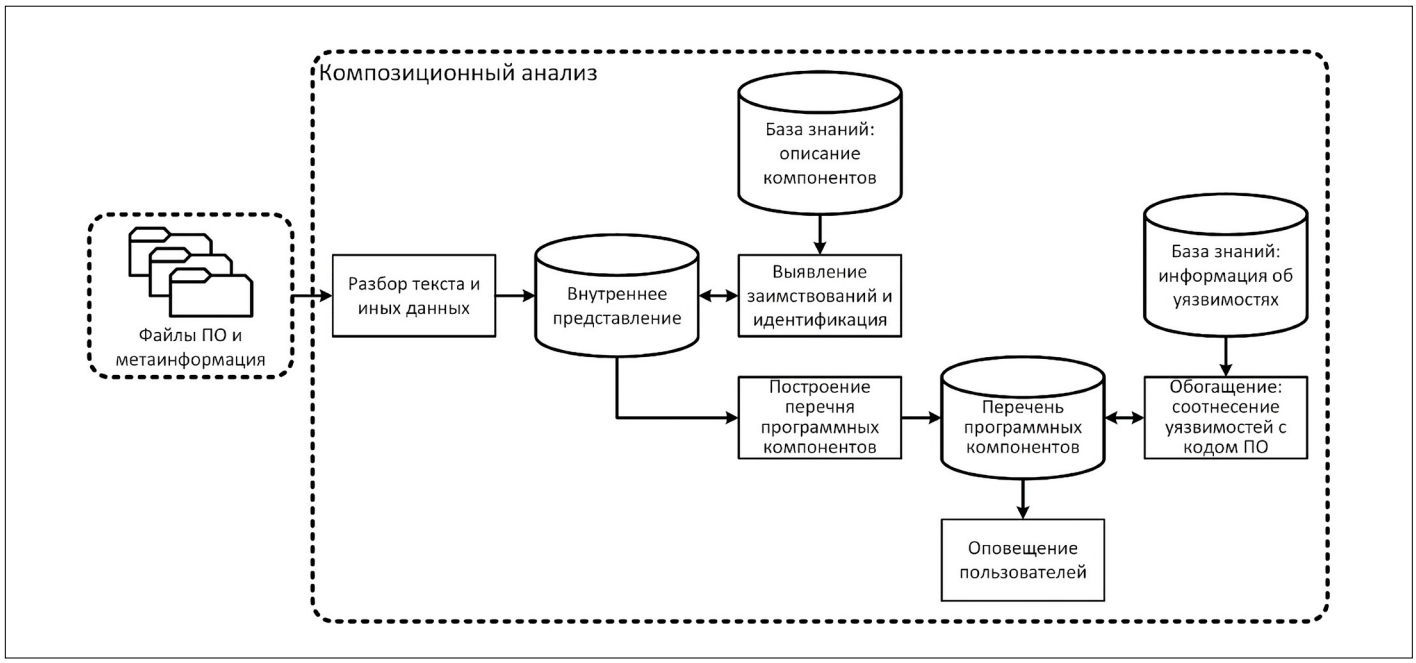


Рисунок 1.

ных соглашений, применяемых в открытых проектах. Каждая лицензия обладает своим набором разрешений и ограничений, которые могут конфликтовать с вашей политикой лицензирования ПО.

Современные приложения могут содержать сотни и тысячи сторонних компонентов, за которыми важно обеспечивать контроль на предмет известных дефектов, уязвимостей и особенностей применения. Автоматизацию этого процесса обеспечивает технология композиционного анализа.

ТЕХНОЛОГИЯ КОМПОЗИЦИОННОГО АНАЛИЗА

Композиционный анализ основан на инвентаризации сторонних компонентов ПО, определении особенностей их использования, составлении перечня известных уязвимостей и/или иных недостатков компонентов.

Композиционный анализ как технология включает в себя:

- выявление заимствований и идентификацию компонентов в составе ПО;
- построение перечня программных компонентов (ППК), также известного как Software Bill of Materials (SBOM);
- обогащение ППК и его актуализацию;
- оповещение о выявленных в ПО уязвимостях и нежелательных особенностях использования компонентов.

Схематичное отображение композиционного анализа ПО приведено[1] на рис. 1.

ППК — ЕДИНЫЙ СТАНДАРТ ОПИСАНИЯ КОМПОНЕНТОВ

В зависимости от применяемого языка программирования и пакетного менеджера разработчики могут фиксировать используемые для сборки ПО библиотеки в разных форматах в специальных файлах манифестов. Полнота такого описания контролируется не всегда. Зачастую компоненты фиксируются в виде названий и (не всегда) требуемых версий.

В контексте конкретного программного продукта все библиотеки можно разделить на две группы: директивные и транзитивные зависимости. Директивные вызываются напрямую из кода проекта, а транзитивные уже вызываются директивными. При этом разработчики, как правило, не указывают и не контролируют устанавливаемые версии транзитивных библиотек сами. Эта часть работы лежит на пакетном менеджере, который фиксирует версии в так называемых.lock-файлах. Если лок-файл не создан или не добавлен в репозиторий кода проекта, это приводит к тому, что при сборках в разное время, в состав продукта могут входить разные версии транзитивных зависимостей. Тем не менее в конечный продукт входят все такие зависимости. Для устранения такой неоднозначности разработчикам рекомендуется использовать условные. lock-файлы в паре с манифестами, в которых разрешаются версии библиотек, что позволяет точнее контролировать состав собираемого продукта.

К сожалению, описанный выше механизм контроля продуктовых зависимостей сильно зависит от используемого технологического стека, одного или многих. Такая разнородность вариаций описания привела к появлению понятия Software Bill of Materials (SBOM), или Перечня программных компонентов (ППК). Это единый машиночитаемый документ, содержащий в себе структурированную информацию о компонентах программного обеспечения и отношениях между ними. Информация

[1] Приводится цитата из проекта ГОСТ Р «Защита информации. Композиционный анализ программного обеспечения. Общие требования».

о компонентах может быть разного характера: от названия и версии компонента и информации о его авторах, до информации о лицензии компонента и, конечно, известных уязвимостях.

Первым общепризнанным стандартом описания SBoM стал SPDX (https://spdx.dev), разработка которого ведётся с 2010 года и поддерживается Linux Foundation. Изначально он был ориентирован на работу с лицензиями, а после был расширен возможностью описания уязвимостей. Выведен в международный стандарт ISO/IEC5962:2021 с версии 2.2.1.

Другим популярным и «модным» сегодня стандартом является CycloneDX (https://cyclonedx.org), разрабатывающийся под эгидой OWASP с 2017 года. Стандарт сразу был акцентирован на контроле безопасности, но не исключает возможности хранения дополнительных сведений о компонентах, включая данные о лицензиях. В 2024 году версия CycloneDX 1.6 опубликована в виде стандарта ECMA-4242.

Существующие открытые инструменты для работы со SBOM-файлами поддерживают оба формата описания, но перевес больше в сторону CycloneDX.

В рамках ТК362 ФСТЭК России ведётся работа по разработке проекта ГОСТ Р «Защита информации. Композиционный анализ программного обеспечения. Общие требования». В октябре 2025 года завершилось публичное обсуждение проекта, и сегодня авторами ведётся доработка по полученным замечаниям.

В проекте стандарта вводится определение Перечня программных компонентов (ППК), который может быть представлен в существующих общепринятых стандартах описания. Дополнительно вводятся усиленные требования к наполнению такого Перечня: указание принадлежности компонента к поверхности атаки, а также языков программирования, на которых он исполнен. Отметка о принадлежности к поверхности атаки позволяет специалистам более фокусно реагировать на обнаруживаемые уязвимости в сторонних компонентах и принимать решение об их исправлении или митигации.

Современные тенденции повсеместного внедрения практики машинного обучения в программные продукты отражаются и в развитии стандартов, приведённых выше, в формате MLBOM (реже — AIBOM). Появляются требования к детальному описанию (и фиксации!) наборов данных, библиотек и железу, которое участвует в формировании ML-моделей. В конечном счёте это тоже программное обеспечение.

Качественное формирование и регулярное отслеживание перечней программных компонентов от выпускаемых версий продуктов определяют основу практики композиционного анализа.

ПРАКТИКА ПРИМЕНЕНИЯ КОМПОЗИЦИОННОГО АНАЛИЗА И ИНСТРУМЕНТАРИЙ

Композиционный анализ может применяться на всех ключевых стадиях разработки ПО — от выбора сторонних компонентов и написания кода до сборки и контроля компонентного состава продукта, уже выпущенного в продуктивную среду.

Известно, что чем раньше проблема обнаружена, тем дешевле её исправить. Работа с известными уязвимостями предполагает, что информация о дефекте описана публично, а зачастую сопровождается proof of concept (PoC) и публичными эксплойтами, которые позволяют наглядно понять, как работает обнаруженная уязвимость. Эта информация является полезной при погружении разработчика в вопросы безопасной разработки. И этими знаниями ИБ должно делиться с ИТ.

Практика показывает, что грамотное донесение разработчику информации об уязвимостях на стадии выбора компонента и кодирования снижает количество «блокировок» в сборочном конвейере на порядок.

Такой сдвиг влево по циклу разработки может быть обеспечен путём выстраивания непрерывного образовательного процесса с разработчиками и предоставлением инструментов, которые следует запускать локально. Как правило, из открытого ПО это консольные агенты, которые умеют строить ППК и обогащать его знаниями об уязвимостях.

Консольные агенты могут также применяться и на этапе сборки в рамках конвейера безопасной разработки, который является обязательной точкой контроля качества и безопасности ПО.

Примерами таких агентов являются открытые инструменты: Trivy, Syft и cdxgen. Они поддерживают сканирование популярных языков программирования и Docker-образов.

Агенты могут выполнять задачу только инвентаризации компонентов либо обеспечивать также решение задачи обогащения данными об уязвимостях. Например, Trivy производит поиск известных уязвимостей по агрегированным базам, тогда как cdxgen формирует строго «инвентарь», который может быть обогащён при помощи других инструментов, например Dependency Track или dep-scan.

К сожалению, приведённые инструменты не поддерживают работу с БДУ ФСТЭК. Исключением является Trivy в отношении анализа системных пакетов. Он умеет обрабатывать данные в формате OVAL, которые формируют производители отечественных операционных систем (например, семейство ОС «Альт»).

Нельзя обойти вниманием и другие примеры агентов:

- OSV Scanner — работает с агрегированной базой уязвимостей от Google (https://osv.dev);
- OWASP dep-scan — реализует анализ возможности эксплуатации обнаруженной уязвимости для Java, JavaScript и PHP. Как правило, показывает результат для директивных зависимостей;
- OWASP-плагины для отдельных языков — следует внимательно отнестись к их настройке.

[2] https://ecma-international.org/publications-and-standards/standards/ecma-424/

Внимание: при проведении композиционного анализа на протяжении жизненного цикла важно помнить, что по пути от кода до выпуска в продукт могут быть добавлены новые компоненты средствами автоматизации (например, при упаковке приложения в контейнерный образ появляются системные пакеты).

При контроле сторонних компонентов всегда важно помнить, что **уязвимости в уже выпущенных вами продуктах сами по себе не появляются, а обнаруживаются со временем**. Поэтому после выпуска сборки полезно не только сохранить ППК, но и проводить его регулярное обогащение на предмет новых уязвимостей в компонентах. Верхнеуровневый контроль и регулярность анализа возможно обеспечить с применением открытого инструмента Dependency Track от OWASP. Инструмент предоставляет пользователю веб-интерфейс для работы с результатами композиционного анализа и поддерживает обогащение данными из баз NVD и OSV.

КАЧЕСТВО И ЭФФЕКТИВНОСТЬ КОМПОЗИЦИОННОГО АНАЛИЗА

Аналогично статическому анализу качество композиционного анализа при выявлении заимствований и идентификации состава ПО определяется тремя характеристиками: долей ложноотрицательных срабатываний, долей ложноположительных срабатываний и скоростью анализа.

На объём возможных ложных срабатываний ключевым образом влияют следующие факторы.

- **Качество инвентаризации компонентов:** нужно уметь обнаруживать не только директивные, но и транзитивные зависимости, разбирать слои образов и детектировать компоненты (или их части), включённые в программный продукт.
- **Полнота используемых источников информации об уязвимостях: чем больше источников** — тем лучше, но они должны быть релевантны вашим технологиям. Важно следить за тем, чтобы ваши «фиды» данных были от тех компонентов, которые вы используете. Например: системные пакеты, которые в разных дистрибутивах существуют в разных версиях, и не все уязвимости в одном дистрибутиве применимы в другом. Отсутствие таких знаний при анализе существенно испортит его качество.
- **Возможность проверки достижимости уязвимости** — не все обнаруживаемые уязвимости могут быть достижимы из собственного кода приложения. Автоматическая проверка трассы вызовов позволяет сократить время на приоритизацию рассматриваемых находок. К сожалению, не всегда можно точно сказать, что уязвимый метод недостижим. Но можно выделить уязвимости, которые точно достижимы. Эта та область, где композиционный и статический анализ сближаются, дополняя друг друга. Примером открытого инструмента, реализующего такую функцию, является dep-scan.
- **Качество и актуальность данных,** используемых при анализе, также влияют на конечный результат. Например, случаются ситуации, когда источник данных об уязвимости содержит ошибочную ссылку на версию компонента, что приводит к ложному срабатыванию.

На эффективность работы с результатами композиционного анализа также влияют:

- **возможность дедупликации находок** — полезная оптимизация, которая существенно снижает объём анализируемой информации;
- **возможность гибкой работы с отчётом** — выделение важных уязвимостей согласно политике безопасности организации сокращает путь специалиста в достижении целей безопасной разработки;
- **возможность встраивания** в инструменты разработчика — чем проще инструмент подключить в конвейер и управлять им, тем меньше уходит времени на сопровождение.

Композиционный анализ является одной из самых легковесных практик безопасной разработки с точки зрения ресурсоёмкости и времени выполнения, поэтому может выполняться в конвейере последовательно с другими этапами сборки.

ПОЛЬЗА КОМПОЗИЦИОННОГО АНАЛИЗА ДЛЯ ОТРАСЛИ

Несмотря на то что знание сторонних компонентов является логичным навыком разработки, применение композиционного анализа приносит большую пользу в решении задач информационной безопасности:

- понимание компонентной основы программных продуктов;
- своевременная информированность об уязвимостях и защита от актуальных угроз;
- понимание путей решения выявленных проблем;
- контроль ранее выпущенных сборок ПО;
- автоматизация процесса защиты цепочки поставки и упрощение процессов сертификации.

Для разработчика композиционный анализ является средством разработки и тестирования программ, которое позволяет:

- ответственно подходить к выбору сторонних компонентов и не раздувать кодовую базу, что положительно сказывается на процессах сборки и тестирования;
- использовать актуальные версии компонентов и точнее контролировать риски технического долга с точки зрения контроля качества закладываемой архитектуры и обратной совместимости.
- сократить объёмы возможных трений со службой информационной безопасности и начать говорить с ними на одном языке.

Юристы получают полный инвентарь и информацию о лицензионных соглашениях, нарушение которых может нести риски для бизнеса компании.

Правильное применение композиционного анализа позволяет не только повысить качество выпускаемых продуктов, но и наладить регулярный контроль и своевременное устранение трещин, которые могут появиться в фундаменте разрабатываемого ПО.

CODESCORING — ПЛАТФОРМЕННЫЙ ПОДХОД К БЕЗОПАСНОЙ РАЗРАБОТКЕ

Построение практик безопасной разработки уже давно не обходится только подходами наложенной защиты, потому что обнаружение проблем «справа» жизненного цикла слишком дорого обходится коллегам «слева» и замедляет производственный процесс.

Разработка всегда была процессом, который важно сделать не только безопасным, но и удобным для его основного потребителя — разработчика. Важно выстраивать правильную коммуникацию ИТ — ИБ с учётом привычной среды создания ПО.

При качественной реализации практики безопасной разработки оказывают оздоровительный эффект на весь процесс, повышают прозрачность и контролируемость.

У этих плюсов есть цена, которая часто выражена в снижении скорости выпуска продуктов на рынок. Но точно ли это должно быть всегда так?

Современные инструменты безопасности должны быть не просто качественными с точки зрения анализа, но и говорить с разработчиком на одном языке: от редактора кода, хранилища артефактов до конвейера сборки и далее по циклу.

Чем раньше и чем понятнее о вероятной проблеме будет рассказано разработчику, тем меньше ошибок обнаружится в конвейере сборки и, как следствие, — меньше возвратов задач перед релизом.

Создатели платформы безопасной разработки CodeScoring разделяют эти взгляды и ценности и применяют их при разработке своего продукта.

ПЛАТФОРМЕННЫЙ ПОДХОД

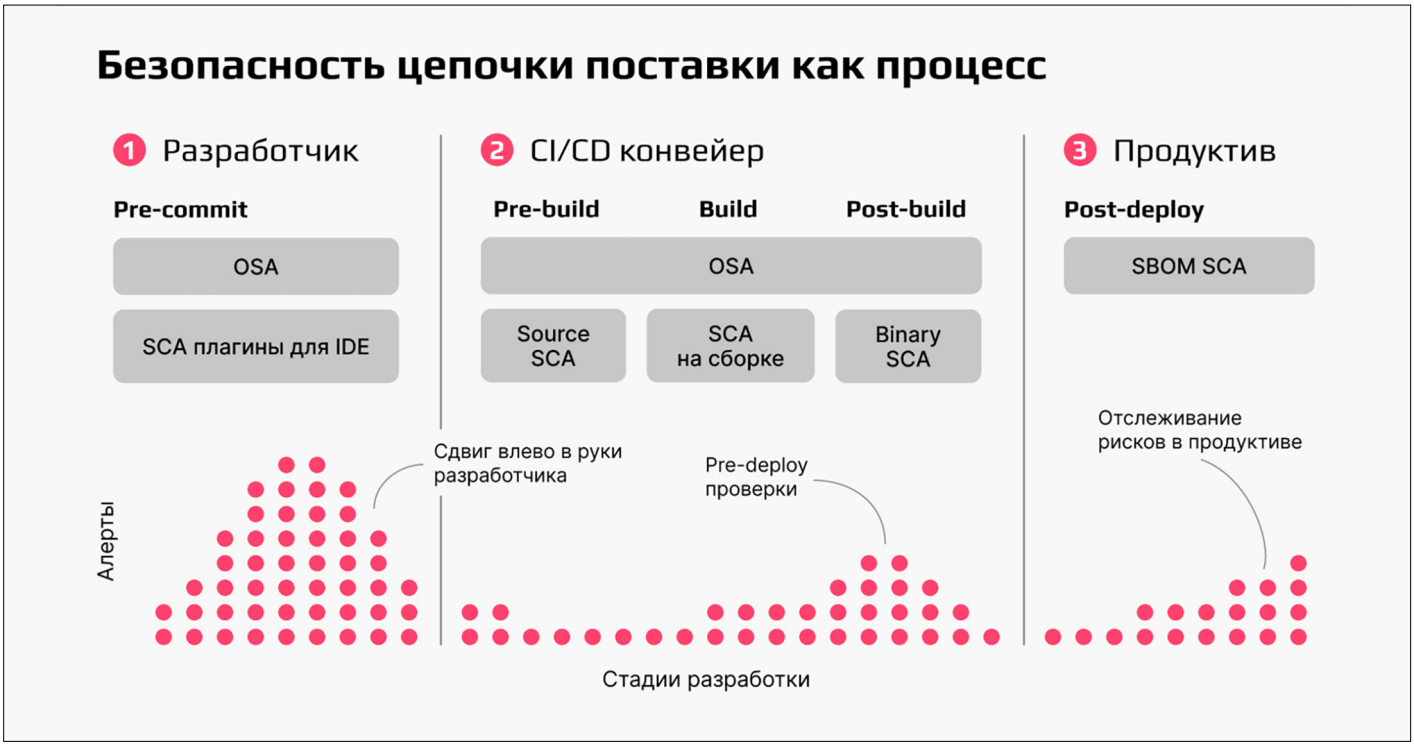
CodeScoring предлагает удобные для использования инструменты анализа ПО и возможность гибкой работы с результатами.

Модули CodeScoring:

CodeScoring.OSA — файрвол для вредоносного и уязвимого Open Source. Обеспечивает проверку компонентов, которые попадают в контур организации извне.

CodeScoring.SCA — помощник в контроле безопасности и лицензионной чистоты Open Source на всех этапах разработки. Обеспечивает проведение композиционного анализа в IDE, конвейере и отслеживание обнаруживаемых уязвимостей после выпуска продукта «в бой». За счёт собственных и дополнительно интегрированных в платформу технологий **ИСП РАН** обеспечивает более точный выбор приоритетных к исправлению проблем.

CodeScoring.TQI — помощник в анализе качества разрабатываемого кода в контексте команд разработки. Проверяет ключевые параметры технического долга, который накапливается в процессе разработки.



CodeScoring.Secrets — эффективная работа с поиском секретов в коде с применением машинного обучения для корреляции результатов и снижения числа ложноположительных срабатываний.

CodeScoring устанавливается у заказчика, анализирует более 15 языков программирования, интегрируется с нужными сервисами: IDE, CI/CD конвейеры, ASPM-платформы, IDP, менеджеры задач, почтовые сервисы и сервисы безопасности.

ЭФФЕКТИВНОЕ ВНЕДРЕНИЕ ПРОВЕРОК

CodeScoring обеспечивает выполнение политик безопасности организации с возможностью детальной настройки до отдельного проекта и команды.

Рассмотрим на примере контроля безопасности цепочки поставки.

Модуль CodeScoring.OSA обеспечивает многоуровневую защиту от вредоносного кода на этапе выбора стороннего компонента или в процессе сборки (CI/CD).

Модуль CodeScoring.SCA обеспечивает интегрированную в среду разработки проверку политик безопасности Open Source. Это позволяет разработчику выбрать безопасный компонент заранее, не дожидаясь блокировки от безопасности в пайплайнах. На этапе сборки проводится контрольная проверка, так как в процессе могли добавиться новые компоненты (чаще всего — системные). Также модуль позволяет рекуррентно отслеживать возможные проблемы безопасности в ранее выпущенных версиях продуктов.

ТЕХНОЛОГИИ

Архитектура платформы построена по принципу API-first, что позволяет интегрировать её в любые существующие процессы клиента и гибко масштабировать. Интеграцию также упрощает наличие собственного консольного агента.

Работа платформы опирается на комплексную систему обработки и анализа данных. Помимо сбора информации из открытых источников это множество механизмов очистки, нормализации, корреляции, дедупликации и модерации с постоянным мониторингом и контролем качества.

Технологическая база непрерывно развивается — добавляются новые источники данных, алгоритмы анализа и инструменты визуализации и работы с результатами. Вся ключевая функциональность платформы — собственной разработки.

О КОМПАНИИ

CodeScoring — петербургская компания, которая представлена на рынке уже более 10 лет и фокусируется на производстве удобных анализаторов ПО. Компания создала первое российское решение по анализу Open Source на безопасность.

Сегодня эффект от внедрения CodeScoring ощущает более 40 тысяч программистов в России, а экспертиза команды отмечена ИСП РАН и ТК362 «Защита информации».

Компания ведёт активную просветительскую деятельность: более 150 выступлений на конференциях, читаются лекции в ведущих вузах страны и публикуются экспертные материалы по проблемам качества и безопасности ПО.

Узнать больше о решении: https://codescoring.ru. Следите за нашими новостями:

КОНТАКТЫ

Пилотирование и приобретение решения: sales@codescoring.ru.

SBOM: КАК БИЗНЕСУ УВИДЕТЬ, ИЗ ЧЕГО СОСТОИТ ЕГО СОФТ, И СДЕЛАТЬ ЕГО БЕЗОПАСНЕЕ

ПОЧЕМУ ПРОЗРАЧНОСТЬ В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ СТАНОВИТСЯ НОВОЙ НОРМОЙ БЕЗОПАСНОСТИ — И КАК «СПИСОК ИНГРЕДИЕНТОВ» ДЛЯ КОДА ПОМОГАЕТ КОМПАНИЯМ ИЗБЕЖАТЬ ПРОБЛЕМ

Аркадий КОМИССАРЧУК
аналитик информационной безопасности, AppSec Solutions

ЦИФРОВОЙ БИЗНЕС СТРОИТСЯ НА ЧУЖОМ КОДЕ — И ЭТО РИСК

Сегодня почти любое программное обеспечение — от мобильного приложения до банковской системы — создаётся не с нуля. В каждом продукте десятки или сотни внешних библиотек, фрей мворков и модулей. Они ускоряют разработку, снижают издержки и позволяют быстрее выводить продукт на рынок.

Но есть обратная сторона: вместе с удобством приходит зависимость от чужого кода, а значит, и уязвимости, которые могут стать угрозой для бизнеса. Когда в 2021 году в популярной библиотеке Log4j нашли критическую уязвимость, тысячи компаний по всему миру спешно проверяли: «А мы то используем её или нет?» Во многих случаях ответить на этот вопрос было невозможно: у компаний просто не было списка компонентов, из которых состоит их продукт.

КАК КОМПАНИИ СЕЙЧАС РЕШАЮТ ПРОБЛЕМУ — ПРАКТИКА SCA

Когда компании осознали, что их программное обеспечение состоит из множества сторонних компонентов, стало очевидно: без системного контроля за их безопасностью и актуальностью риски становятся неконтролируемыми.

Так появился подход **SCA (Software Composition Analysis)** — анализ состава программного обеспечения. SCA-инструменты автоматически исследуют проект и формируют полную картину его зависимостей. Они позволяют:

- определить, какие библиотеки, модули и фреймворки используются в продукте;
- установить их версии и источники происхождения;
- выявить компоненты с известными уязвимостями;
- проверить соблюдение лицензионных условий.

SCA стал ключевым элементом современной практики DevSecOps, когда контроль безопасности встроен прямо в процесс разработки. Неотъемлемой частью практики SCA является формат SBOM.

ЧТО ТАКОЕ SBOM

SBOM (Software Bill of Materials) — это файл в формате JSON или XML (структурированное описание и перечисление объектов), который включает в себя инвентаризационный список всех компонентов (пакетов, библиотек), используемых в разрабатываемом приложении или необходимых для его работы. Простыми словами, это «список ингредиентов» программного обеспечения.

Он показывает:

- из каких библиотек и модулей состоит программа;
- какие версии этих компонентов используются;
- откуда они взяты (источники, лицензии);
- и как они связаны между собой (зависимости одних компонентов от других).

Фактически SBOM — это файл, который говорит:

«Наш продукт использует библиотеку X версии 2.3, библиотеку Y версии 1.4, и они зависят от библиотеки Z версии 0.9».

Поиск зависимостей в проекте является непростой задачей: разные экосистемы пакетов указывают использование тех или иных сторонних зависимостей по-разному, и к каждой нужен собственный подход для их определения. Например, в Java используется экосистема пакетов Maven и зависимости указываются в файле манифеста `pom.xml`. В Python применяется пакетный менеджер pip, а зависимости могут указываться по-разному — в файле `requirements.txt` или в `Pipfile` (если используется инструмент Pipenv).

Два наиболее популярных стандарта (формата) файла SBOM — CycloneDX и SPDX. Существует множество различных инструментов для автоматического поиска зависимостей и сборки SBOM, в их числе — Trivy, Syft, cdxgen. Они используют разные подходы к определению зависимостей проекта, но в среднем все справляются с этой задачей и выдают подробный и структурированный SBOM.

КАКИЕ ЗАДАЧИ SBOM РЕШАЕТ В СФЕРЕ ИБ?

SBOM позволяет решить две связанные задачи:

1. Инвентаризировать все используемые в продукте (исходном коде, артефакте, операционной системе) компоненты;

2. Автоматизировать анализ их безопасности с помощью инструментов SCA. Политики безопасности могут включать в себя проверку компонентов на уязвимости и лицензионную чистоту, а также на дату публикации, имя автора и другие элементы.

Решение обеих задач позволит снизить риск атаки на цепочку поставок ПО.

Первую задачу решают вышеуказанные инструменты, и в результате сканирования проекта формируется SBOM-файл, содержащий все используемые библиотеки, фреймворки и прочие сторонние программные модули, а также информацию о связях между ними (зависимостях).

Далее с помощью инструментов SCA проводится анализ компонентов, указанных в SBOM, и поиск информации об уязвимостях в открытых базах данных по идентификатору компонента. Также проверяется хеш библиотеки, если есть опасения, что её могут подменить или внести корректировки в исходный код. Эти данные позволяют найти все актуальные уязвимости в открытых источниках.

Стоит отметить, что наличие уязвимой библиотеки в составе ПО ещё не говорит о присутствии уязвимости, которую могли бы эксплуатировать злоумышленники. Для этого уязвимая функция должна быть достижима из кода, и при этом данные, которые в неё поступают, не должны быть защищены дополнительными мерами безопасности. Большинство инструментов SCA не отвечают на вопрос о возможности эксплуатации найденной уязвимости — для этого требуется ревью AppSec-инженера или разработчика. Однако сейчас некоторые решения уже начинают анализировать совместно и код, и набор библиотек, что помогает снизить количество ложноположительных (False Positive) срабатываний и объём ручной работы по их нахождению.

ЗАЧЕМ БИЗНЕСУ ЗНАТЬ СОСТАВ СВОЕГО КОДА

Понимание состава ПО — это не техническая прихоть, а вопрос управления рисками и ответственности.

Определение состава ПО позволяет:

1. Быстрее реагировать на угрозы. Когда появляется новая уязвимость (например, CVE-2025-xxxx), компания может мгновенно понять, затронута ли она, если есть SBOM. Без него на поиск ответа могут уйти дни или недели.

2. Минимизировать репутационные и финансовые потери. Утечка данных или сбой из-за «чужой» уязвимости — это не только техническая проблема. Это потеря доверия, уход клиентов, штрафы и возможные иски.

3. Контролировать цепочку поставок ПО. Если вы закупаете программное обеспечение или заказываете его у подрядчика, SBOM помогает убедиться, что внутри нет «сюрпризов» — старых библиотек, вредоносных модулей или компонентов без лицензии.

4. Упростить аудит и обеспечить соответствие отраслевым и регуляторным требованиям. Во многих странах SBOM становится обязательным элементом при поставках ПО государственным или критическим структурам. Например, в США после президентского указа о кибербезопасности (Executive Order 14028) поставщики федеральных систем должны предоставлять SBOM вместе с продуктом. Похожие инициативы появляются также в Европе и Азии.

Если говорить о практике регулирования и требований к SBOM в Российской Федерации, то можно отметить следующее:

1. По требованию ФСТЭК России (информационное письмо № 240/24/4436 от 26.09.2024) изготовители средств защиты информации (далее — СЗИ), испытательные лаборатории и органы по сертификации при проведении сертификации СЗИ, включающих в свой состав заимствованные программные компоненты с открытым исходным кодом, должны проводить сертификационные испытания СЗИ и осуществлять их поддержку безопасности с учётом заимствованных компонент.

2. ГОСТ Р 56939–2024, который описывает процессы разработки безопасного ПО, устанавливает, что процесс композиционного анализа должен включать в числе прочего процесс по формированию перечня заимствованного ПО, а также его постоянную актуализацию и контроль на предмет наличия известных уязвимостей.

ВЫВОД

Прозрачность состава ПО становится ключевым элементом цифровой безопасности и зрелости разработки. Сегодня компании не могут позволить себе иметь чёрные ящики в своих продуктах — слишком высока цена ошибок, и киберугрозы развиваются очень быстро.

SBOM — это не техническая мода, а инструмент управления рисками, сопоставимый по значимости с финансовым аудитом или контролем качества поставщиков. Он позволяет организациям видеть, из чего состоит их цифровая инфраструктура, быстро реагировать на уязвимости, подтверждать соответствие требованиям клиентов и регуляторов.

Внедряя практику формирования SBOM, бизнес делает шаг к более зрелой модели управления, где безопасность становится частью корпоративной культуры, а не реакцией «после инцидента».

* * *

SBOM — это новый язык доверия между разработчиками, заказчиками и пользователями. Те, кто научится говорить на нём первыми, выиграют в скорости, прозрачности и устойчивости своего бизнеса.

УГРОЗЫ ОТКРЫТОГО КОДА

ТЕПЕРЬ УЯЗВИМОСТИ ВСТРЕЧАЮТСЯ НЕ ТОЛЬКО В КОМПОНЕНТАХ РАЗРАБОТКИ, НО И В КОМПОНЕНТАХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И МАШИННОГО ОБУЧЕНИЯ

Константин КРЮЧКОВ
руководитель продукта AppSec.Track, AppSec Solutions

Более половины библиотек с открытым кодом, которые используют разработчики программного обеспечения, содержат серьёзные уязвимости. С их помощью злоумышленники могут перехватить данные, ослабить криптографическую защиту, выполнить произвольный код — всё, что называется «взломом» ИТ-системы. Масштаб проблемы показало ежегодное исследование AppSec Solutions, проведённое с помощью инструмента AppSec.Track. Если учесть, что, по оценкам экспертов, 90% программного кода разработчики заимствуют из открытых библиотек, масштаб риска трудно переоценить.

КОМПОЗИЦИОННЫЙ АНАЛИЗ КАК ИНСТРУМЕНТ ЗАЩИТЫ ЦЕПОЧКИ ПОСТАВОК

Предотвратить атаки на цепочку поставок ПО помогают инструменты композиционного анализа. Они позволяют установить компонентный состав приложения, которое используется в бизнес-процессах.

Компаниям, эксплуатирующим какие-либо ИТ-решения, необходимо понимать, из чего они состоят. Это касается как собственной разработки, так и готовых продуктов, получаемых, например, от подрядчиков. Зная состав ПО, можно проверить его компоненты на безопасность, целостность, наличие вредоносного содержимого, лицензионные ограничения и другие критерии.

Автоматизировать такую проверку позволяют инструменты класса OSA/SCA (Open Source Analysis / Software Composition Analysis), в том числе и наш сканер AppSec.Track. Он позволяет оценить безопасность и качество сторонних компонентов на всех этапах жизненного цикла разработки и эксплуатации ПО, начиная от поиска и загрузки компонентов разработчиком во внутренний контур организации и заканчивая мониторингом релизных версий приложения, где эти компоненты используются. Система интегрируется с популярными системами хранения артефактов — Nexus Repository Manager, JFrog Artifactory, Harbor и любыми git-совместимыми системами контроля исходного кода.

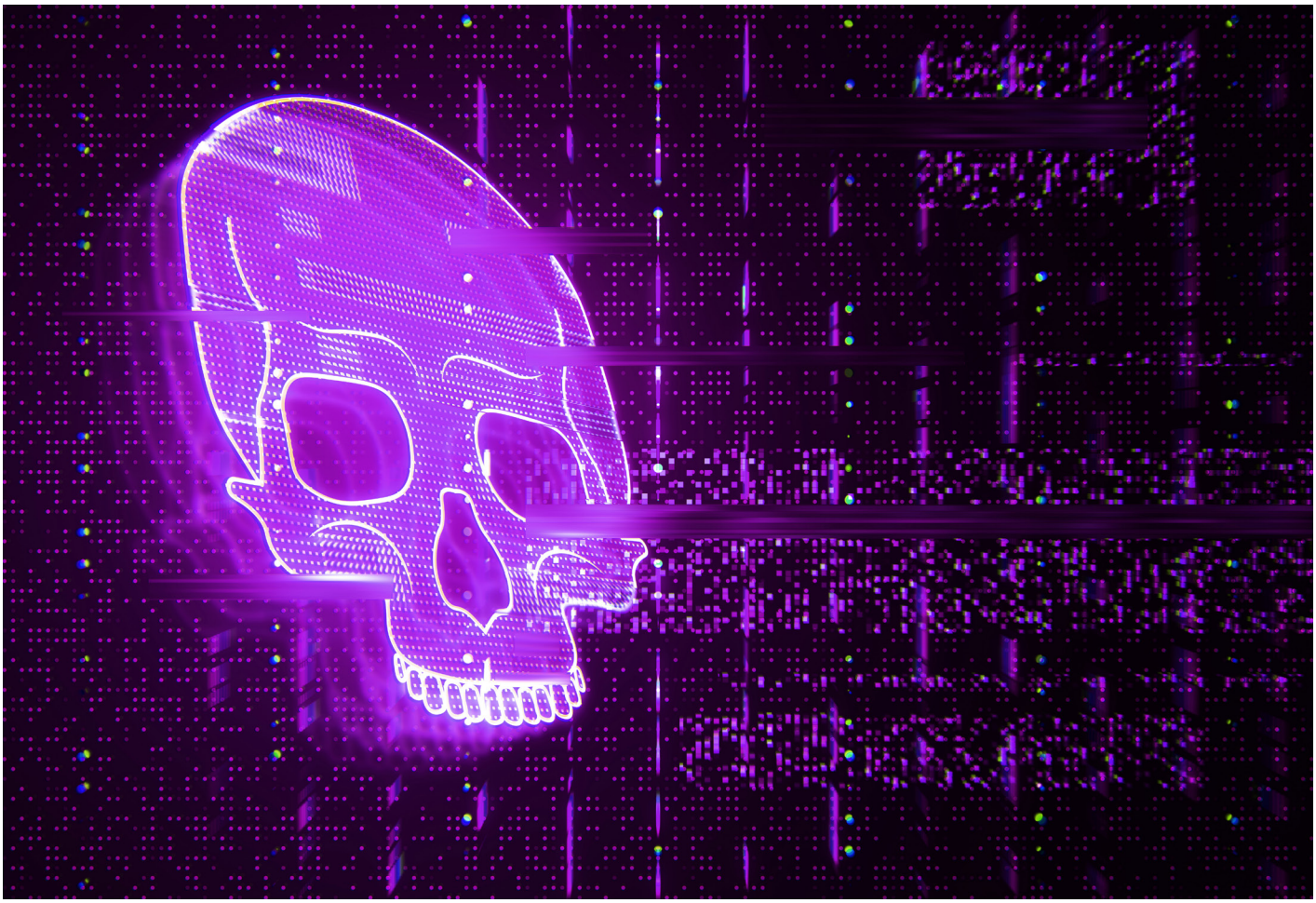
Инструмент работает с собственной базой знаний, курируемой экспертами AppSec Solutions и содержащей информацию об уязвимостях и компонентах. Это позволяет предоставлять полную и достоверную информацию об обнаруженных проблемах, а также рекомендовать безопасную версию пакетов с нарушениями для разработчиков и инженеров информационной безопасности.

В 2025 году в продукте появились важные функции. В частности, была доработана языковая поддержка, и теперь проверка компонентов доступна для языков Objective-C, Swift, Dart и более редких — Perl и R. Был добавлен функционал поиска секретов, которые могли быть оставлены в исходном коде или попасть в итоговую сборку, а также поиска ошибок конфигурации при создании образов Docker. Для инженеров безопасности, сопровождающих систему во время эксплуатации, мы создали новые разделы «Репозитории», «Исключения», «Компоненты» и «Дашборды», позволяющие управлять компонентами и результатами сканирований, а также делать это с помощью метрик. А для компаний, которым по требованиям безопасности ограничен или полностью запрещён доступ к ресурсам в сети Интернет, была выпущена функция развёртывания целиком On-Premise.

В ближайшем релизе AppSec.Track появится поддержка EPSS (Exploit Prediction Scoring System) — метрика для оценки вероятности эксплуатации уязвимости, которая позволит инженерам безопасности определять приоритет задач на исправление уязвимостей. Также появится возможность анализа Docker-образов по слоям для определения причины появления проблемы в компоненте.

АНАЛИЗ ДОСТИЖИМОСТИ УЯЗВИМОГО КОДА

Не каждая найденная уязвимость действительно представляет угрозу для приложения. Часто компо-



нент содержит потенциально опасную функцию, но она не используется в коде или к ней нет доступа извне. Поэтому для точной оценки риска важно понимать, достижим ли уязвимый участок кода в конкретном контексте работы программы.

Анализ достижимости уязвимого кода — технология, которая определяет, может ли уязвимость быть реально эксплуатирована злоумышленником. Такая проверка помогает сократить количество ложноположительных срабатываний и сосредоточиться на тех проблемах, которые действительно требуют исправления.

Сейчас инженеры AppSec.Track разрабатывают модуль для анализа достижимости уязвимого кода, который позволяет автоматически определять, может ли уязвимость быть реально эксплуатирована в данном контексте. Таким образом, модуль значительно сократит объём и ускорит работу ИБ-инженеров по разбору уязвимостей и определению проблемных мест.

ДЕТЕКТИРОВАНИЕ AI/ML-КОМПОНЕНТОВ

Сегодня технологии искусственного интеллекта становятся неотъемлемой частью большинства цифровых решений — от аналитических сервисов до промышленных систем. Вместе с тем встаёт новая задача для информационной безопасности: управление рисками, связанными с использованием моделей ИИ и компонентов машинного обучения.

У таких компонентов есть особенности: они зависят от качества данных, на которых обучались, могут содержать ошибки в логике, а также не всегда прозрачны с точки зрения лицензий и источников происхождения. Всё это формирует новый класс рисков, связанных не только с безопасностью, но и с доверенностью и устойчивостью цифровых систем.

Чтобы помочь компаниям справиться с этими вызовами, мы разрабатываем функцию распознавания компонентов AI/ML в AppSec.Track и будем работать с ними так же, как делаем это для системных пакетов и компонентов языков программирования. У моделей есть свои проблемы, которые связаны как с качеством данных, возвращаемых ими, так и с безопасностью и лицензионной чистотой. Начать управлять связанными рисками — это новый вызов, который встаёт перед командами ИБ. А мы со своей стороны делаем всё, чтобы они могли делать это максимально автоматизированно.

ВЫВОД

Современные цифровые экосистемы становятся всё более зависимыми от открытого кода и компонентов искусственного интеллекта, что делает вопрос их безопасности критически важным. Только комплексный и системный подход позволит организациям сохранить устойчивость перед лицом растущих угроз, где граница между ИТ-уязвимостью и ИИ-риском становится всё менее заметной.

COMPOSITION ANALYSIS TOOL. НОВЫЙ ВЗГЛЯД НА БЕЗОПАСНОСТЬ

ЭВОЛЮЦИЯ КОМПОЗИЦИОННОГО АНАЛИЗА ОТРАЖАЕТ ЗРЕЛОСТЬ ИНДУСТРИИ

Виталий ВАРЕНИЦА
заместитель генерального директора АО «НПО «Эшелон»

ВВЕДЕНИЕ

Современные программные системы состоят из множества компонентов, модулей и библиотек, создаваемых сообществом разработчиков по всему миру. Такой подход сделал возможным стремительное развитие технологий: вместо того чтобы писать всё вручную, команды используют готовые решения и собирают из них собственные продукты. Это ускоряет выпуск новых версий, снижает затраты и повышает гибкость разработки.

Однако вместе с выгодами пришёл и новый риск — зависимость от чужого кода. Ошибка или уязвимость в сторонней библиотеке может оказать влияние на приложение, даже если собственный код безупречен. Именно поэтому сегодня на первый план выходит композиционный анализ — подход, позволяющий выявлять и оценивать уязвимости в зависимостях. Но простой перечень компонентов уже не даёт полного ответа на вопрос: насколько эти уязвимости действительно опасны для конкретного продукта? Чтобы понять реальный уровень угрозы, необходимо видеть контекст — то, как именно используется сторонний код.

ВСЕГДА ЛИ БИБЛИОТЕКА С УЯЗВИМОСТЬЮ НЕСЁТ РИСКИ

Современные приложения создаются не изолированно, а из множества готовых компонентов. Библиотеки, фреймворки и открытые модули стали неотъемлемой частью разработки: они позволяют ускорять работу, снижать издержки и концентрироваться на функциональности, а не на базовых механизмах. Однако вместе с этим возникла и обратная сторона — зависимость от чужого кода означает унаследованные риски. Если в стороннем компоненте обнаружена уязвимость, под вопросом оказывается безопасность всей системы[1, 2].

На практике всё не так однозначно. Наличие уязвимости в библиотеке ещё не означает, что она представляет реальную угрозу для конкретного продукта. Проблемный участок может существовать в коде, но никогда не выполняться, если приложение не использует связанную с ним функциональность. Некоторые ошибки активируются только при специфических сценариях или типах данных, которые в данном контексте просто отсутствуют.

Тем не менее многие инструменты композиционного анализа и сегодня работают по тому же принципу. Их подход остаётся прямолинейным: если библиотека с определённой версией указана в базах уязвимостей, она автоматически считается требующей обновления или замены. Такой механизм основан на механическом сравнении списка зависимостей с публичными базами данных известных проблем. В итоге каждая версия компонента, где когда-либо фиксировалась ошибка, помечается как потенциально опасная.

Для разработчиков и команд безопасности это оборачивается значительным объёмом ложных срабатываний. Отчёты содержат десятки и сотни «уязвимых» зависимостей, из которых лишь малая часть действительно может повлиять на безопасность продукта. Реальные риски теряются среди формальных предупреждений, а ресурсы расходуются на анализ малозначимых инцидентов.

Главная причина этого — отсутствие контекста. Без понимания того, используется ли уязвимый участок кода в реальности, невозможно определить, есть ли фактическая угроза эксплуатации. Даже если компонент формально содержит уязвимость, она может оставаться недостижимой и не влиять на поведение приложения.

По мере усложнения систем стало очевидно, что такой подход требует пересмотра. Без понимания того, каким образом библиотека применяется в конкретном проекте, оценка её рисков остаётся поверхностной. Индустрия постепенно перешла от простого сопоставления зависимостей с базами CVE к анализу реальной достижимости уязвимостей при выполнении кода. Риск теперь определяется не фактом существования ошибки, а тем, используется ли она на практике.

МЕТОДЫ, ФОРМИРУЮЩИЕ КОНТЕКСТ В КОМПОЗИЦИОННОМ АНАЛИЗЕ

Чтобы оценить, насколько уязвимость в библиотеке действительно затрагивает приложение, необходимо понять, каким образом этот код участвует в работе системы. Недостаточно просто знать, что компонент присутствует в проекте — важно определить, как он взаимодействует с другими элементами и какие его функции реально вызываются. Для этого современные инструменты композиционного ана-

[1] Арустамян С. С., Вареница В. В., Марков А. С. Методические и реализационные аспекты внедрения процессов разработки безопасного программного обеспечения // Безопасность информационных технологий. — 2023. — Т. 30, № 2. — С. 23–37. DOI: 10.26583/bit.2023.2.01

[2] Вареница В. В., Марков А. С., Цирлов В. Л., Арустамян Р. С., Арустамян С. С., Антипов И. С., Магакелова Н. А. Как избежать ошибок при безопасной разработке программного обеспечения. — М.: Квант-медиа, Москва, 2025. — 344 с.

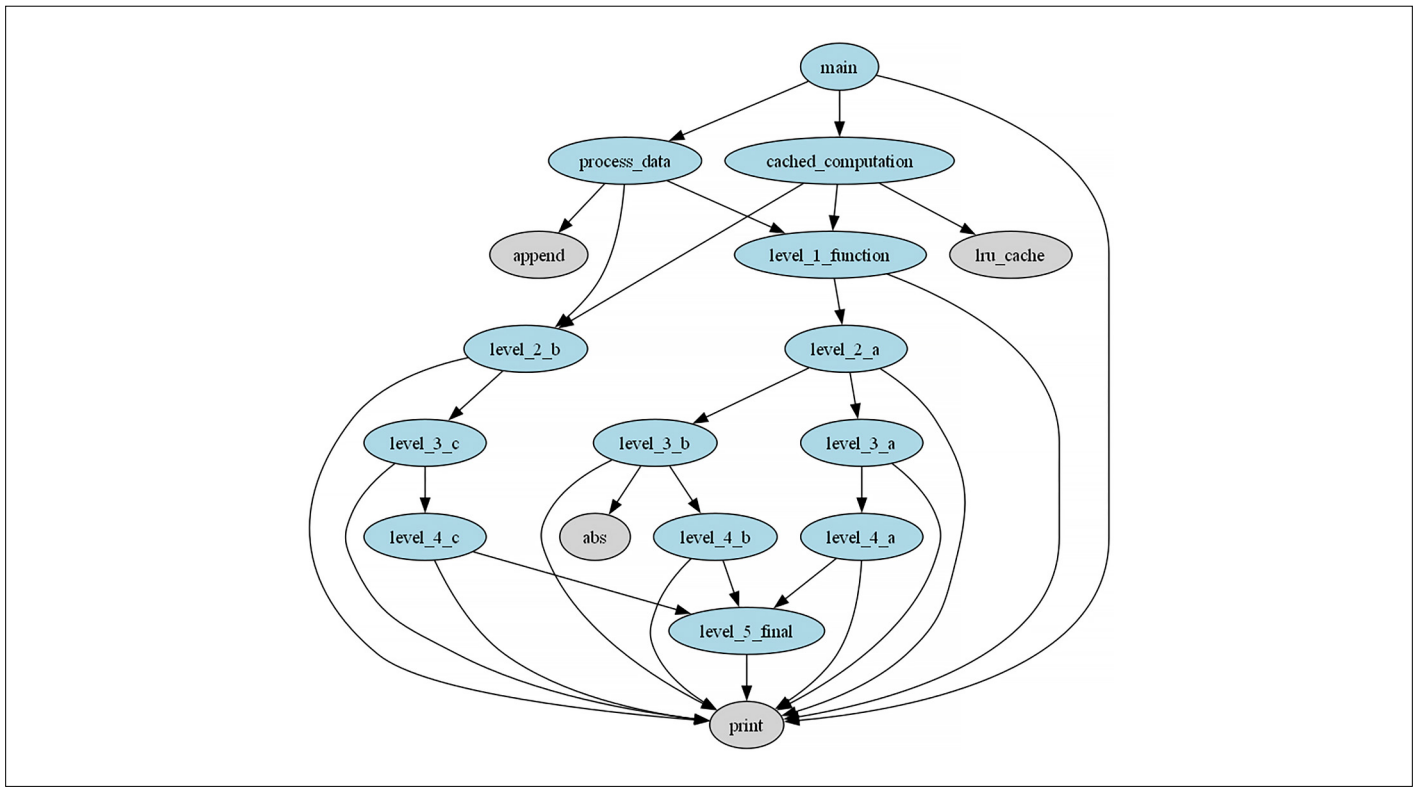


Рисунок 1. Граф вызова функций

лиза применяют структурные и семантические методы разбора кода.

Ключевым элементом стал синтаксический разбор с построением абстрактного синтаксического дерева (AST). Это дерево описывает структуру программы: функции, классы, зависимости и взаимосвязи между ними. Анализируя такую структуру, система может определить, какие участки библиотеки связаны с исполняемым кодом, а какие остаются неиспользованными.

Следующий шаг — построение графа вызовов[3]. Он отражает последовательность обращений между функциями, включая связи между собственным кодом и внешними библиотеками. С его помощью можно проследить, ведёт ли цепочка вызовов к уязвимой функции или нет. Если уязвимый участок не имеет ни прямого, ни косвенного пути вызова, вероятность эксплуатации минимальна (рис. 1).

Третий важный элемент — анализ потоков данных. Он показывает, какие значения проходят через функции, как они изменяются и откуда поступают. Если данные, обрабатываемые уязвимым кодом, не поступают извне и не могут быть изменены пользователем, риск практически отсутствует. Таким образом, инструмент не просто фиксирует факт уязвимости, а оценивает её достижимость и эксплуатируемость.

Совокупность этих методов формирует новое качество анализа. Инструмент видит не просто перечень библиотек, а взаимосвязанную систему, где каждая связь имеет значение. Такой подход помогает отличить реальные угрозы от формальных, определить приоритеты и сократить количество ложных срабатываний.

Особенно важен этот уровень детализации в масштабных продуктах, где используются сотни сторонних компонентов. Здесь невозможно проверять каждую библиотеку вручную — нужна автоматизированная оценка контекста. Благодаря структурному анализу код становится прозрачным, а риски можно оценивать не в отрыве, а в системе.

По сути, композиционный анализ превращается из проверки списков зависимостей в исследование поведения приложения. Он повышает точность, снижает избыточные предупреждения и помогает командам концентрироваться на действительно опасных сценариях.

СНИЖЕНИЕ КОЛИЧЕСТВА ЛОЖНОПОЛОЖИТЕЛЬНЫХ СРАБАТЫВАНИЙ

Переход к контекстному анализу зависимостей стал шагом к более зрелой и управляемой модели безопасности. Теперь цель — не просто обнаружить уязвимость, а понять её роль и влияние в реальном окружении приложения. Уязвимость перестаёт быть формальной записью в отчёте: она становится элементом системы, который можно оценить, локализовать и контролировать.

Такой подход помогает не только точнее определять риски, но и принимать более взвешенные инженерные решения. Иногда устранение проблемы требует не обновления библиотеки, а корректировки связей или ограничений доступа к данным. Это

[3] Марков А. С., Антипов И. С., Арустамян С. С., Магакелова Н. А. Сравнительный анализ и выбор статических анализаторов безопасности кода // Вопросы кибербезопасности. 2024. № 5 (63). С. 79–88.

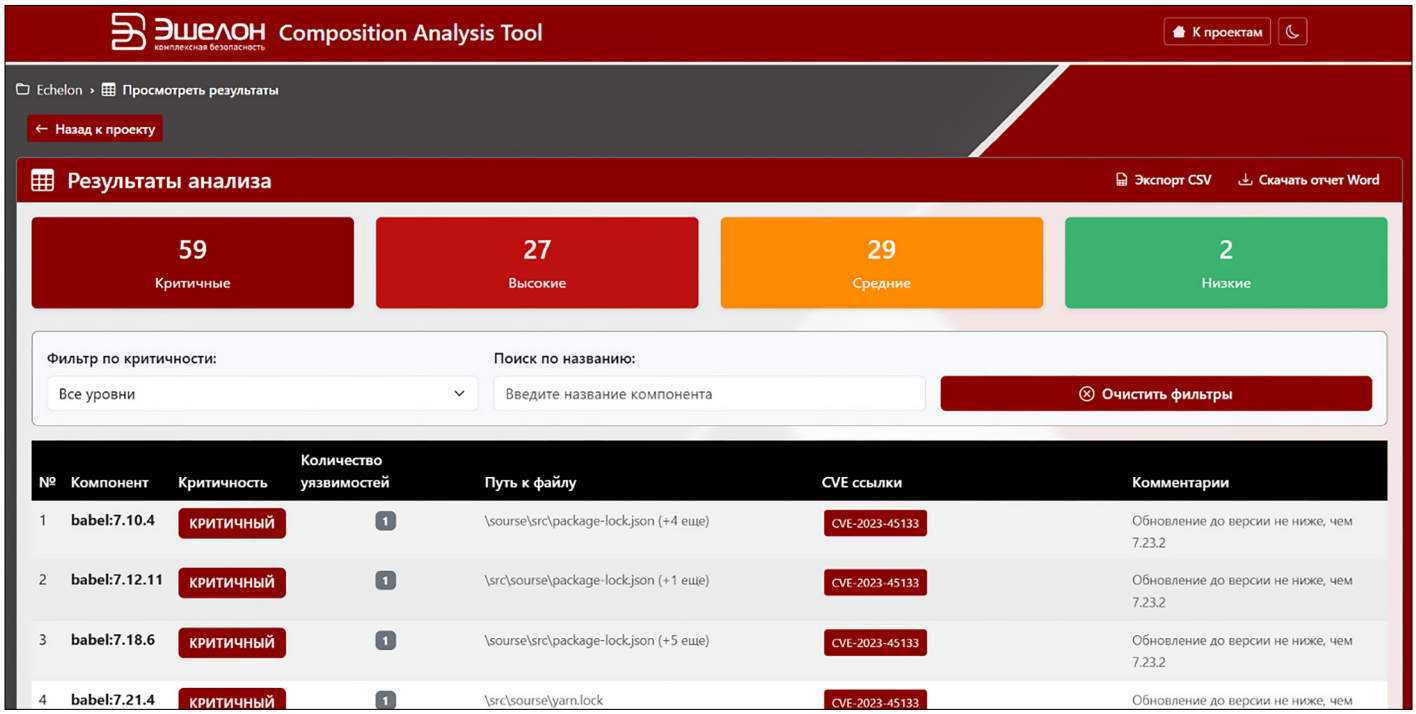


Рисунок 2. Интерфейс программы

делает безопасность адресной и экономичной, избавляя от избыточных обновлений, которые могут нарушить стабильность продукта.

Контекстное понимание создаёт баланс между безопасностью и скоростью разработки. Оно позволяет планировать изменения осознанно, не тормозя процессы и не жертвуя качеством. Безопасность перестаёт быть барьером — она становится частью стратегического управления кодом, встроенной в цикл создания продукта.

На уровне отрасли это отражает переход к новой культуре работы с открытым кодом. Когда большинство решений строится на внешних библиотеках, важно понимать не только их состав, но и контекст использования. Ценность теперь определяется не объёмом собранных данных, а умением выделить действительно значимые связи между компонентами.

Именно вокруг этой идеи сегодня формируются новые инструменты. Их задача — объединить структурный анализ, построение графов вызовов и исследование потоков данных в единую систему, способную наглядно показать связи между уязвимостями, кодом и архитектурой приложения. Это переход от пассивного мониторинга к активному управлению безопасностью — к «умному» анализу, работающему на опережение.

В настоящее время ведётся разработка решения, использующего этот подход на практике. Его цель — создать инструмент, который помогает видеть полную картину: где именно в коде проявляется уязвимость, каким образом она может быть достигнута и какие пути её устранения наиболее эффективны. Фрагмент интерфейса разрабатываемой системы приведён на рисунке 2.

ЗАКЛЮЧЕНИЕ

Эволюция композиционного анализа отражает общую зрелость индустрии. Если раньше безопасность сводилась к проверке списков зависимостей, то сегодня она строится на понимании структуры и контекста. Такой подход меняет саму природу работы с рисками — от механического устранения уязвимостей к стратегическому управлению ими.

Контекстный анализ помогает по-новому взглянуть на безопасность: он показывает систему в целом, выявляя взаимосвязи между уязвимостями, кодом и архитектурой. Такой подход позволяет сосредоточиться не на количестве найденных проблем, а на их реальном влиянии. Это переход от реагирования к управлению, от фрагментарных мер — к целостному пониманию защищённости.

В мире, где программные продукты строятся из тысяч взаимосвязанных элементов, именно контекст становится ключом к устойчивости. Он превращает анализ уязвимостей из формальной процедуры в инструмент осмысленного управления безопасностью — и именно это понимание становится основой защиты в современной экосистеме программного кода.

* * *

Для отслеживания последних новостей в сфере информационной безопасности подписывайтесь на наш телеграм-канал Echelon Eyes (*t.me/EchelonEyes*):

СТАТИЧЕСКИЙ АНАЛИЗАТОР PVS-STUDIO ЗА ПРЕДЕЛАМИ РБПО

Андрей КАРПОВ
CBDO компании ООО ПВС (pvs-studio.ru)

Публикации этого журнала во многом ориентированы на тему разработки безопасного ПО (РБПО), рассматривая методологии и продукты именно с этой точки зрения. Мы сами ранее упоминали PVS-Studio в журнале именно в контексте РБПО. Однако статический анализ — это не только про безопасность.

Вернее, статический анализ — это всегда про безопасность, качество и надёжность кода. Но работа над этими характеристиками может происходить вне классических понятий и ГОСТов. Взглянем на PVS-Studio за пределами описания:

PVS-Studio — статический анализатор кода, совместимый с ГОСТ Р 71207–2024, выявляющий критические ошибки в коде на языках C, C++, C#, Java. Включён в Реестр российского ПО: запись № 9837. Может применяться для построения РБПО согласно ГОСТ Р 56939–2024. Соответствует требованиям «Методики выявления уязвимостей и НДВ в программном обеспечении».

Теперь посмотрим на инструмент с других сторон.

В PVS-Studio реализовано множество диагностик для поиска опечаток и дефектов, связанных с написанием кода методом copy-paste. Особенность детекторов опечаток в том, что многие из них построены на эмпирических алгоритмах. Это их и слабая, и сильная сторона.

Такие детекторы генерируют большее количество ложных срабатываний по сравнению с алгоритмами определения нулевых указателей, деления на ноль и т.д. Зато они очень хорошо дополнят человека в обзорах кода, позволяя находить незаметные и нетипичные ошибки.

Например, C++ анализатор может выявить ошибки формирования строк с HTML-кодом. Выявит незакрытые теги или их дублирование, хотя этот баг не имеет никакого отношения к языку C++ (см. диагностику V735).

В общем случае статические анализаторы не подходят для задачи оптимизации кода, так как не могут получить информацию о том, как часто будет выполняться тот или иной его фрагмент. Это задача для инструментов профилирования.

И всё же PVS-Studio помогает оптимизировать код благодаря диагностикам «микрооптимизации». Отдельный найденный фрагмент неоптимального кода не ускорит программу, но исправление сотен таких предупреждений в сумме может давать заметный результат.

Пример срабатывания диагностики V813 на C++ код реального приложения. Контейнер передаётся по значению, но его элементы используются только для чтения:

```
void
  addDescriptions(
    std::vector<std::pair
    <int, std::string>> toAdd)
{
  if (m_descCount +
  toAdd.size() >
  MAX_POLICY_DESCRIPTIONS)
  ....
  for (const auto &it : toAdd)
  ....
}
```

Мы получали положительные отзывы от клиентов, что, исправив несколько таких недочётов в коде, они получали заметное ускорение приложения в десятки процентов.

Для помощи разработчикам игр инструмент PVS-Studio интегрируется с такими игровыми движками, как Unreal Engine (C++) и Unity Engine (C#). Анализатор не только удобно встраивается в соответствующие CI/CD-конвейеры, но и учитывает особенности движков, реализуя узкоспециализированные диагностики.

Пример диагностики для Unity Engine: V3205.
Анализатор обнаруживает нежелательное создание экземпляра классов `'MonoBehaviour'` или `'ScriptableObject'` с помощью оператора 'new'. Объекты, созданные таким образом, не будут связаны с движком, поэтому такие специфичные Unity-методы, как `'Update'`, `'Awake'`, `'OnEnable'` и прочие, вызываться не будут.

На практике при выборе инструмента приходится учитывать удобство интерфейса и интеграцию с другими системами. Нашей командой проделана большая работа в этом направлении. В настоящее время PVS-Studio интегрируется:

- с IDE (Visual Studio, IntelliJ IDEA, Rider, OpenIDE, CLion, Visual Studio Code, Qt Creator);
- сборочными системами (MSBuild, CMake, Ninja, Gradle, Maven, JSON Compilation Database;
- облачными CI (CircleCI, Travis CI, GitLab, Azure DevOps, GitHub Actions);
- и т.д.

Запускается на большом количестве ОС, в том числе отечественных: Windows, macOS, Arch Linux, Astra Linux, CentOS, Debian GNU/Linux, Fedora, Linux Mint, openSUSE, Ubuntu, РЕД ОС.

«ИТ-АРХИТЕКТОР И ИБ-СПЕЦИАЛИСТ ДОЛЖНЫ ИДТИ РУКА ОБ РУКУ»

НА ВОПРОСЫ BIS JOURNAL ОТВЕЧАЕТ ЕВГЕНИЙ ТОДЫШЕВ

Евгений ТОДЫШЕВ
руководитель направления безопасной разработки УЦСБ

— Что вы можете сказать о базовых принципах безопасного проектирования программного обеспечения на разных его этапах? Что поменялось в подходах к безопасности разработки ПО в последние годы?

— Подходы к проектированию ПО заметно эволюционировали и продолжают изменяться в лучшую сторону. Если ещё лет десять назад вообще не особо задумывались о том, что решение должно быть безопасным, то сейчас это стало императивом на протяжении всего жизненного цикла разработки, начиная с проектирования. В частности, security by design представляет собой такой подход к проектированию ПО, при котором требования безопасности учитываются на самых ранних стадиях разработки, когда только закладывается бизнес-функционал.

Но есть и другая сторона медали. Разработка сейчас активно идёт в сторону использования микросервисного софта. Для того чтобы им успешно централизованно управлять, необходимо использовать специальные средства автоматизации, такие как контейнеры и системы оркестрации. И это сильно расширяет поверхность атаки, усложняя задачу проектирования.

— Как моделирование угроз применяется при разработке ПО?

— В рамках данного метода мы представляем себя на месте злоумышленника, причём ещё на этапе проектирования софта. Для начала надо в целом описать, как данные перемещаются в системе. Затем можно применить различные методологии для моделирования угроз. Например, мы в своей практике чаще всего используем методологию STRIDE. С её помощью дальше описываем угрозы, которые могут появиться при реализации нового функционала.

При моделировании угроз необходимо придерживаться базовых принципов — в частности, это принципы наименьших привилегий и безопасности настроек по умолчанию.

На завершающем этапе необходимо приоритизировать устранение угроз по степени риска: уже на этапе проектирования мы закладываем средства защиты против самых критичных из них. Моделирование угроз позволяет нам проиграть сценарий атаки на бумаге, выявив и заложив устранение рисков. И это будет намного дешевле, нежели исправлять эти угрозы в продакшене, а ещё хуже — если уязвимость будет проэксплуатирована и компания понесёт определённые финансовые потери.

— Насколько выбор конкретного стека технологий, языка программирования влияет на безопасность итогового продукта?

— Прямого ответа на этот вопрос не существует. Безопасность — это всё-таки свойство применения технологии и во многом зависит от того, кто или как эту технологию применяет. С этой точки зрения нельзя выделить какой-то язык программирования с исключительным правом использования.

Например, для высокоуровневых приложений можно отдавать предпочтение языкам, которые управляют памятью автоматически, что в ряде случаев исключает целый класс критических уязвимостей, таких как переполнение буфера и утечки памяти. Эти проблемы часто встречаются в программном обеспечении, которое написано на C, C++.

Важно сказать, что языки программирования часто уже определены и сильно повлиять на этот процесс в большинстве случаев не получается. Поэтому выход у службы безопасности следующий: необходимо уметь жить с теми технологиями, которые нам предоставляет бизнес. Можно разрабатывать стандарты и правила безопасного кодирования и контролировать их выполнение. Также следует проявлять осмотрительность при работе с open-

Security by design представляет собой такой подход к проектированию ПО, при котором требования безопасности учитываются на самых ранних стадиях разработки, когда только закладывается бизнес-функционал

source-библиотеками, так как многие из них содержат уязвимости.

— Какие архитектурные решения сегодня считаются более предпочтительными (например, микросервисы против монолитов)?

— В монолитной архитектуре риски сконцентрированы. Если злоумышленник находит уязвимость, которая позволяет сломать периметр защиты, он сразу получает доступ ко всей системе и всем данным сразу. Преимущество в том, что единый периметр можно усиленно контролировать и защищать, чему службы ИБ уже давно научились.

В микросервисной же архитектуре риски распределены. Новые вызовы связаны с тем, что увеличивается поверхность атаки и возникает угроза горизонтального перемещения. Таким образом, перед злоумышленником стоит задача скомпрометировать один слабый сервис и использовать его как плацдарм для атак на другие, более критичные сервисы.

Именно поэтому микросервисы требуют принципиально другого подхода к безопасности. Основные требования, которые необходимо закладывать в архитектуру, — это строгая сетевая сегментация, использование service mesh, индивидуальной аутентификации и, конечно же, принципа наименьших привилегий для каждого сервиса. Главное архитектурное преимущество микросервисов с точки зрения ИБ — это возможность изолировать атаку и ограничить ущерб за счёт создания безопасной среды исполнения на стадии проектирования.

— Какими принципами необходимо руководствоваться при проектировании АПИ?

— Есть несколько основных паттернов, которые необходимо использовать при проектировании API. В соответствии с принципом минимальных привилегий на уровне API каждый пользователь получает лишь необходимый и достаточный набор прав. Не надо создавать универсальный эндпоинт. Также необходимо строго валидировать API-схему и парсинг данных.

Следующий паттерн — это идемпотентность и безопасность используемых методов. Это предполагает следование стандартам HTTP. GET-запросы не должны изменять состояние системы. Применение методов POST или PUT позволяет выстроить дополнительную защиту от случайных повторов и ряда команд. Для предотвращения сбоя в доступности следует установить лимиты. API без лимитов запросов — это потенциальное появление DOS-атак и различных видов злоупотребления.

Необходимо закладывать версионирование API с самого начала, что позволит нам вносить критические изменения в безопасность в новой версии без ущерба для старых клиентов

Ещё один важный принцип — это безопасность по умолчанию. Необходимо закладывать версионирование API с самого начала, что позволит нам вносить критические изменения в безопасность в новой версии без ущерба для старых клиентов. Важно проектировать политику устаревания для прежних, менее безопасных версий. Конечно, за версиями API необходимо строго следить, чтобы не получилось так, что из-за человеческого фактора (например, разработчик забыл отключить старый эндпоинт при выпуске нового) не появлялись возможности проэксплуатировать какие-то уязвимости.

— Как привить культуру безопасности в команде разработки?

— Тут можно идти несколькими путями: либо сверху вниз, либо снизу вверх. Что я под этим подразумеваю? В первом варианте мы обсуждаем внедрение практик культуры на уровне руководства. И в целом в любом случае наступают моменты, когда приходится это делать. Речь может идти о разъяснении значимости этих практик и необходимого времени для их внедрения. После обоснования перед руководством практики спускаются вниз и так далее.

В другом варианте мы первым делом начинаем общаться с разработчиками. Ситуация выглядит идеально, когда в командах разработки есть люди, которые понимают ценность данных практик и готовы занимать проактивную позицию. Хуже, когда таких людей нет. И тут можно проводить различные митапы с разработчиками, небольшие хакатоны.

— Какие рекомендации можно дать при проектировании системы аутентификации и авторизации?

— Не создавайте собственные системы, а используйте проверенные отраслевые практики (OAuth 2.0, OIDC). Многофакторная аутентификация должна быть стандартом, и не стоит использовать только SMS, лучше переходить на Push. Также MFA может быть адаптивной и применяться только в рискованных операциях.

При управлении сессиями и токенами необходимо использовать краткосрочные JWT-токены с механизмом обновлений и должна быть обеспечена возможность централизованного отзыва сессий. Отдельное внимание стоит уделять процессу восстановления пароля.

При проектировании сервиса авторизации необходимо использовать централизованный подход с единой точкой принятия решений, то есть все запросы должны проходить через единый сервис (или библиотеку), где содержатся все правила авторизации. При этом рекомендуем использовать гибридный подход RBAC+ABAC. Важно помнить о том, чтобы не допускать наследования привилегий по умолчанию для новых сущностей и осуществлять проверку нарушений прав доступа в автоматическом режиме.

Вопросы задавал
Усам Оздемиров

«БЕГ НА МЕСТЕ» ИЛИ «ДОРОГУ ОСИЛИТ ИДУЩИЙ»?

АНАЛИЗ МИРОВОГО ОПЫТА И ОПЫТА РОССИИ ПО СОЗДАНИЮ СОБСТВЕННЫХ СРЕД РАЗРАБОТКИ АСУ И ИНФОРМАЦИОННЫХ СИСТЕМ

Владимир ЛАПТЕВ
президент Фонда развития программной инженерии, к. т. н., профессор

Сергей СЕЛЕЗНЁВ
вице-президент Фонда развития программной инженерии, к. т. н.

Андрей БУГАЕНКО
заместитель генерального директора АО «НПО Ангстрем», к. т. н.

ВВЕДЕНИЕ

Проекты цифровой экономики реализуются во многих странах мира. Отдельные страны ставят целью создание не только национальной, но и глобальной цифровой экономики, а также поэтапный переход от национальных к транснациональным цифровым платформам экономики. Однако создать цифровую модель экономики, обеспечить планирование, в том числе стратегическое, долгосрочные инвестиции и так далее, без установления порядка в цифровом пространстве фактически невозможно.

Известно, что средства производства — это основа, фундамент информационных систем. Среда разработки является единственным средством производства цифровых продуктов. Разрозненные проекты по типовым средам разработки информационных систем, в том числе для систем с элементами искусственного интеллекта, продолжаются в рамках ряда государственных проектов, однако современные отечественные среды разработки пока отсутствуют.

О необходимости создать отечественные платформы для совместной разработки ИТ-проектов руководством страны заявлялось неоднократно. Давно понятна актуальность и жгучая необходимость иметь на федеральном уровне репозитории не только отдельных крупных и мелких мероприятий национальных проектов со средствами обеспечения жизненного цикла, соответствующей базой знаний, которые системно взаимоувязаны с предоставляемыми государством льготами, субсидиями и тому подобными мерами поддержки отрасли микроэлектроники и программной инженерии, как это сделано в ряде стран. Современные среды разработок — репозитории — должны быть базовыми технологическими платформами очередного этапа национального проекта «Экономика данных и цифровая трансформация государства» и других национальных проектов в области образования и науки, обеспечивать современные технологии программирования, реальное тестирование и последующую поддержку жизненного цикла программного обеспечения и аппаратных решений до уровня продуктов с целью подтверждения качества и защиты прав потребителей, а также способствовать успешному внедрению существующих лучших отечественных программных продуктов внутри страны и их экспорту.

Планомерное исключение участия России в международной ИТ-кооперации за последние годы явно показало критическую зависимость отечественных разработчиков АСУ и информационных систем (ИС) от зарубежных технологий и средств разработки.

Отсутствие ключевых классов российских полнофункциональных средств проектирования и разработки российской радиоэлектронной продукции, а также программного обеспечения АСУ и ИС привело к острой необходимости решить данную проблему. Пока многолетние усилия ответственных министерств и ведомств не принесли результатов.

Технологическая независимость критической информационной инфраструктуры (КИИ) без собственной производственной базы (технологии и средства производства), находящейся на территории и в юрисдикции Российской Федерации, невозможна в принципе и практически.

Время и опыт решения поставленной задачи за последние десятилетия показали, что технологическая независимость критической информационной инфраструктуры не может быть обеспечена рыночными методами без долговременного планомерного государственного управления и поддержки создания российских репозиториев с учётом требований качества и информационной безопасности.

В соответствии с Указом Президента РФ утверждены «Основы государственной политики по обеспече-

нию технологической независимости критической информационной инфраструктуры Российской Федерации на период до 2030 года» (далее — «Основы»). Минпромторг России подготовил план мероприятий по «Основам» для согласования и принятия решений в соответствии с Указом Президента РФ.

В первом разделе «Создание условий для устойчивого функционирования объектов критической информационной инфраструктуры за счёт обеспечения потребности отраслей экономики в безопасных, надёжных и качественных российском программном обеспечении и доверенных программно-аппаратных комплексах» пунктом 1 выделено мероприятие: «*Создание российской безопасной среды разработки и поддержка развития российских репозиториев программных пакетов и библиотек как инфраструктуры разработки общесистемного, прикладного и промышленного программного обеспечения с учётом требований информационной безопасности*».

Предлагается детально рассмотреть, что сделано и не сделано за последние десятилетия и что предлагается сделать до 2030 года.

ПОНЯТИЕ СИСТЕМЫ РЕПОЗИТОРИЕВ

В настоящее время понятие репозитория тесно связано с понятиями фонда алгоритмов и программ, с автоматизированным проектированием архитектуры информационных систем, коллективными средами автоматизации программирования, с коммуникацией разработчиков, подбором программистов, обеспечением быстрой и безопасной разработки программ, поддержкой управления качеством информационных и программных систем, обеспечением информационной безопасности и доверенности программных систем на всех этапах жизненного цикла. Наконец, с созданием операторов децентрализованной системы репозиториев. Без этих связанных понятий системной и программной инженерии понятие отдельного репозитория часто сводится к банальному понятию «облачного хранения файлов для коллективной работы через Интернет» или распределённой инфраструктуры разработки программного кода, включая библиотеки программ, с возможностью контроля версионности, дополненная функциями совместной работы над кодом, багтрекинга, автоматической проверки и тестирования, а также другими инструментами разработки и обеспечения целостности.

И наконец, понятие репозитория прямо связано с понятием собственности, интеллектуальной собственности, авторской собственности, обеспечением взаимодействия покупателя и продавца на всех этапах подготовки и выполнения контракта.

Поэтому операторов репозиториев можно классифицировать по типу собственности (личная, международная, коммерческая, государственная), по функциональным характеристикам (перечислены выше), а также по технологической независимости от других стран и собственников.

В ряде развитых стран наряду с репозиториями открытого кода (open source) поддерживаются и финансируются:

- «открытые» межгосударственные репозитории программного обеспечения и знаний для коллективных научных исследований в различных областях науки (научные межгосударственные репозитории типа Европейской инфраструктуры исследований);
- коммерческие репозитории типа GitHub компании «Майкрософт»;
- государственные репозитории программного обеспечения, разработанного и поддерживаемого за счёт денег налогоплательщиков типа code.gov;
- защищённые платформы репозиториев для обороны и безопасности типа forge.mil.

Рекомендуем для большего понимания понятия системы репозиториев ознакомиться с учебником и статьёй[1].

[1] Сухомлин В. А., Романов В. Ю., Гапанович Д. А. Введение в модельно-ориентированную системную и программную инженерию. Москва: Фонд «Лига интернет-медиа»; МАКС Пресс, 2024. — 672 с.

АНАЛИЗ ОПЫТА РОССИИ по созданию собственных сред разработки АСУ и информационных систем

Минцифры России реализует ряд проектов по созданию репозиториев:

- репозиторий «Экосистемы разработки и тестирования программно-аппаратных комплексов в НИИ «Восход»;
- репозиторий ФГИС «Отечественный реестр ПО» в НИИ «Восход»;
- репозиторий ГОСТЕХ;
- репозиторий, создаваемый в соответствии с новым положением по Национальному фонду алгоритмов и программ;
- репозиторий «маркетплейсов» российских магазинов мобильных приложений;
- репозиторий и создание оператора репозитория на базе АНО «Открытый код» при поддержке Фонда развития ИТ; Реестр доверенного отечественного программного обеспечения (готовится постановление Правительства);
- создание нового оператора реестра российского ПО как некоммерческой организации.

Минпромторг России продвигает «Стратегию создания российского репозитория программных пакетов и библиотек, находящегося в юрисдикции и на территории Российской Федерации». Ассоциация документальной электросвязи и компания ИВК с 2019 года продвигают свои наработки по СИЗИФ для других разработчиков операционных систем, но состыковать репозитории десятков «отечественных» операционных систем до 2–3 пока никто не решился.

ФСТЭК России заказала две опытно конструкторские работы:

- по созданию доведенного ядра ОС Линукс, работы ведутся в Институте системного программирования им. В. П. Иванникова Российской академии наук (ИСП РАН);
- по «Разработке унифицированной среды разработки безопасного отечественного программного обеспечения». Репозиторий и его оператор должны были быть готовы к 2025 году. В 2025 году ИСП РАН продолжает работать по данной ОКР.

«Ростелеком» и **ИСП РАН** подписали соглашение о сотрудничестве,

направленном на создание инновационных решений в области разработки безопасного программного обеспечения (РБПО). Этот шаг соответствует стратегическим задачам государства по повышению уровня информационной безопасности в цифровой экономике России и стимулирует развитие технологий, связанных с искусственным интеллектом.

Подписание соглашения с участием старшего вице-президента по коммерческим ИТ-продуктам «Ростелекома» Давида Мартиросова и директора ИСП РАН, академика Арутюна Аветисяна состоялось в рамках международного форума Kazan Digital Week 2025. Основная цель сотрудничества — создание отечественных автоматизированных систем (конвейеров), которые обеспечивают постоянный контроль качества и безопасность разрабатываемого ПО.

Стороны планируют разработать полностью российских решения:

- автоматизированный конвейер для разработки и тестирования безопасных программных продуктов и сервисов на базе проекта «Создание унифицированной среды разработки безопасного отечественного ПО»;
- специализированный конвейер безопасной разработки приложений с моделями машинного обучения (MLSecOps) в рамках исследовательских работ Центра доверенного искусственного интеллекта.

Такие конвейеры востребованы в государственных структурах, банках, энергетических и других критичных секторах экономики, где надежность и защищенность разрабатываемых и эксплуатируемых информационных систем являются приоритетом. Стороны договорились о безвозмездном предоставлении решений образовательным учреждениям для учебных целей.

Дорожная карта проекта до 2028 года будет представлена на конференции ИСП РАН в конце года. В совместной работе над конвейерами ИСП РАН будет отвечать за создание методологии, и предоставит собственные инструменты статического и динамического анализа архитектуры и кода ПО — Svace, Crusher, Sydr, Natch.

Сборка и тестирование конвейеров будут проводиться с использованием продуктов ГК «Базис» и ИТ-системы «Лукоморье», входящих в коммерческий ИТ-кластер «Ростелекома». Платформа управления динамической инфраструктурой Basis Dynamix Enterprise и платформа автоматизации с поддержкой микросервисных приложений Basis Digital Energy станут инфраструктурной основой конвейеров, а система управления проектами «Яга» и ESM-системы «Диво» обеспечат процесс разработки программных продуктов и сервисов.

Госкорпорацией Росатом в соответствии с указом Президента РФ № 166 от 30.03.2022 учреждено АО НПО «КИС», и оно же определено производителем доверенных программно-аппаратных комплексов для критической информационной инфраструктуры распоряжением Правительства РФ от 29.03.2023 № 757-р. Росатом уже более 10 лет пытается иметь собственные среды разработок, в том числе для создания «Цифровых двойников».

Для кредитно-финансовой сферы **Банком России** поддержан проект создания собственного доверенного репозитория для обеспечения технологического суверенитета и ускорения процесса цифровизации с использованием отечественных ИТ-решений.

На уровне отдельных министерств и ведомств — Минобороны, ФТС, ФНС, МИД, Минспорта, Миннауки, Минтранса и других — ведутся попытки использовать репозиторий и «конвейер» ГОСТЕХ от Сбертеха.

В Российской Федерации продолжает действовать «Межведомственная концепция создания доверенной программно-аппаратной среды для автоматизированных систем управления (АСУ) органов военного и государственного управления (ОВГУ)», которая определяет основные научно-технические и организационные принципы создания доверенной программно-аппаратной среды для АСУ органов военного и государственного управления, обеспечивающей информационную безопасность и технологическую независимость АСУ ОВГУ. Концепция разработана в соответствии с протокольным решением заседания научно-технического совета Военно-промышленной комиссии при Правительстве Российской Федерации по проблемным научно-техническим вопросам формирования отечественной доверенной аппаратно-программной среды для АСУ органов военного и государственного управления от 4 октября 2011 г. Однако современных репозиториев в России пока нет для нужд обеспечения национальной безопасности.

В конце декабря 2019 года состоялось заседание секции научного совета при Совете Безопасности Российской Федерации, которая поддержала предложение Минпромторга России и заинтересованных организаций о необходимости создания отечественного репозитория (доверенного государственного фонда алгоритмов и программ). Советом безопасности России в Минцифры России были направлены ряд писем с предложением подготовить и внести в установленном порядке в Правительство Российской Федерации проект постановления по конкретному решению ведения системы доверенных государственных репозиториев на базе государственных предприятий МНИИ «Интеграл» и НИИ «Восход» с учётом опыта Минобороны. Очередным решением Совета безопасности от 15 ноября 2021 года рекомендовано Минцифры России с участием Минпромторга России, ФСБ России, ФСТЭК России, ГК «РОСТЕХ», других заинтересованных федеральных органов исполнительной власти и организаций — «до 1 июня 2022 года ... представить в Правительство Российской Федерации предложения по внесению изменений в федеральный проект „Цифровые технологии“ национального проекта „Цифровая экономика Российской Федерации“, о необходимости создания находящихся в юрисдикции России отечественных доверенных репозиториев». Указы Президента России «О мерах по технологической независимости и безопасности КИИ РФ» — № 166 от 30 марта 2022 года

и «О Межведомственной комиссии Совета безопасности по вопросам обеспечения технологического суверенитета РФ в сфере КИИ» — № 203 от 14 апреля 2022 года.

Репозитории были необходимы не только для ГОСТЕХ (D6.10.32, 44,27 млрд в 2021–2024 гг.), но и для десятков других крупных и мелких мероприятий программы Цифровой экономики, например, D6.11.01, 52,76 млрд в 2021–2024 гг., D6.10.93, 30,14 млрд в 2021–2024 гг., D6.10.07, 21,03 млрд в 2021–2024 гг., D4.05.90, 0,28 млрд в 2021–2024 гг., D5.09.28, 0,4 млрд в 2021–2024 гг. (из Паспорта мероприятий программы Цифровой экономики). Проект Цифровой экономики закончился, написана стратегия развития открытых репозиториев, но создание отечественной системы репозиториев не завершено.

В конце 2024 года после Указа Президента Российской Федерации от 29 октября 2024 г. № 927, которым утверждены «Основы государственной политики в области обеспечения технологической независимости критической информационной инфраструктуры Российской Федерации на период до 2030 года» и поставлены задачи по критической информационной инфраструктуре, в том числе:

- создание российской безопасной среды разработки и поддержка развития российских репозиториев программных пакетов и библиотек как инфраструктуры разработки общесистемного, прикладного и промышленного программного обеспечения с учётом требований информационной безопасности;
- разработка российских и локализация иностранных технологий для производства российской высокотехнологичной продукции для критической информационной инфраструктуры.

...Разговоры продолжились на уровне Военно-промышленной комиссии Правительства.

Коллегия Военно-промышленной комиссии Российской Федерации в рамках своей координационной деятельности всесторонне прорабатывает данную проблематику, в том числе проблемные вопросы создания российской инструментальной среды для разработки доверенных программно-аппаратных комплексов и специализированного программного обеспечения для нужд обеспечения национальной безопасности. Так, соответствующие научно-технические подходы и исходные данные были рассмотрены в марте 2025 года на заседании секции № 4 НТС ВПК. По итогам этого заседания заинтересованным организациям, обладающим передовыми компетенциями в сфере развития критической информационной инфраструктуры Российской Федерации, было рекомендовано до 1 июля 2025 года выработать соответствующие предложения о целях, задачах и основных направлениях деятельности для создания такой российской инструментальной среды, как государственная организационно-техническая система, поддержаны предложения по четырёхуровневой системе репозиториев и создания с государственным участием операторов репозиториев. Были также обсуждены предложения о создании закрытого реестра — системы репозиториев отечественных АСУ и ИС, программных продуктов КИИ, возможности государственного финансирования вертикально интегрированной группы компаний по созданию российской безопасной среды разработки и поддержки развития российских репозиториев программных пакетов и библиотек с учётом требований информационной безопасности.

Жизненный цикл любой автоматизированной системы управления (АСУ) начинается с замысла (или концепции, архитектуры) и включает все этапы от требований к АСУ, разработки, внедрения, сопровождения и до утилизации. В России по сей день действует ГОСТ 34 — основа создания АСУ в рамках НИОКР. При этом в каждой НИОКР разработка программного обеспечения (ПО), как правило, избыточна, неотторжима, заблокирована поставщиками и интеграторами, несовместима с другим ПО и имеет небольшую команду сотрудников, которые знают, как его поддерживать и дорабатывать на своей технологической базе.

Проблема заключается в сложившейся технологии разработки АСУ. По созданной в НИОКР документации практически невозможно вести поддержку, сопровождение и интеграцию этой АСУ и ПО, особенно сторонними разработчиками, поскольку средства разработки не передаются заказчику, а остаются у исполнителя. И это повсеместно делается для получения последующего дохода от сопровождения.

Команды разработчиков в своих НИОКР действуют автономно и очень слабо взаимодействуют друг с другом (если и вообще взаимодействуют по причине конкуренции или секретности) с разработчиками в других НИОКР, общаясь иногда на семинарах и конференциях. Ни о каком свободном программном обеспечении и обмене кодами друг с другом и в помине не было и нет (хотя ранее, в СССР, это было повсеместно, программисты делились и документацией, и кодом).

Сегодня, как правило, свободное программное обеспечение используется «втихую» и выдаётся за свою разработку. Программисты используют GitHub, GitLab и другие системы управления версиями, выкачивают оттуда чужие программы и приступают к их модернизации и перелицовке. Поэтому команды разработчиков должны как бы заново изобретать колесо с каждым новым приложением, по сути, начиная с нуля с каждой новой НИОКР, и это очень трудоёмко, неэффективно и в конечном итоге — дорогостоящий процесс без наследования опыта.

Анализируя сегодняшнюю ситуацию с выполнением НИОКР по созданию государственных информационных систем и информационных систем в бизнесе, следует отметить, что практически ничего не изменилось. Все «куют» ПО на своих кузницах, на своих мелких подворьях, согласуя действия только через зарубежные фонды open source программного обеспечения. Своих OMG, Apache Software Foundation, Linux Foundation и других у нас пока нет, хотя раздаются отдельные предложения о необходимости перехода коли-

чества (в реестр отечественного ПО на июнь 2025 года включено 26 705 продуктов 9965 правообладателей) в качество и создания отечественных «майкрософтов».

АНАЛИЗ ОПЫТА США по созданию мировых сред разработки АСУ и информационных систем для ВПК

Рассмотрим, как обстоят дела с этим у наших конкурентов — команд разработчиков-программистов в США. Рассмотрим их «кузницу» выполнения НИОКР на примере Минобороны США и его платформы для разработчиков под именем forge.mil.

До недавнего времени (начало 2000-х) разработчики действовали так же, как и в России, за исключением того, что все военные АСУ сдавались заказчику в Архив Минобороны с исходными кодами, хорошо документированными и со средствами разработки и модернизации (в Минобороны России и других силовых структурах по сей день такого фонда нет).

Как известно, Минобороны США по-хорошему «помешалось» на сетецентрических операциях, в которых различные части каждой военной системы могут работать вместе, по единому протоколу, для более быстрого и тщательного реагирования. Сеть передачи данных (как правило, IP) связывает все эти отдельные части вместе. Опираясь на эти сети, Агентство оборонных информационных систем (DISA) решило в 2009 году создать платформу Forge.mil, идеально вписывающуюся в сетецентрическую среду разработки НИОКР, считая, что традиционный военный процесс приобретения и разработки не очень подходит для того, как программное обеспечение разрабатывается сегодня, когда большие АСУ разрабатываются изолированно.

В качестве создателя платформы DISA выбрала поставщика средств разработки программного обеспечения CollabNet и федерального ИТ-интегратора Carahsoft Technology Corp.

Сегодня платформа Forge.mil служит центральным узлом для разработки программного обеспечения в рамках НИОКР Минобороны США. Команды разработчиков из различных организаций — создателей ПО могут хранить ночные сборки проекта программного обеспечения в Forge.mil, а затем, для тестирования своего кода, договариваться с DISA на загрузку образа приложения и поддерживающей операционной системы в вычислительную среду Rapid Access для проведения тестирования в реальном масштабе времени. Модель совместного репозитория Forge.mil основана на гигантском репозитории программного обеспечения с открытым исходным кодом SourceForge [3], который разработчики используют для хранения и управления кодом для проектов программного обеспечения с открытым исходным кодом. DISA поощряет заказывающие военные службы или подрядчиков, разрабатывающих приложения для военных, использовать Forge.mil в качестве среды разработки. Предлагая эту платформу в качестве сетевой службы, она избавляет разработчиков от необходимости создавать и настраивать собственные стенды, что особенно актуально, если разработчики географически рассредоточены. Кроме того, размещая проекты с открытым исходным кодом на платформе, внешние участники также могут участвовать, что означает, что программное обеспечение может быть создано быстрее и с большей надёжностью. В качестве открытого репозитория Forge.mil может сократить дублирование усилий, поскольку службы DISA могут проверять, был ли компонент уже разработан в другом месте, прежде чем вводить его в эксплуатацию заново или писать новое ТЗ. Услуги могут быть сосредоточены на разработке компонентов, а не целых систем, каждый компонент может быть запущен в собственном темпе, не задерживая прогресс любой системы в целом. Репозиторий также может содержать программный код, который был разработан в разных НИОКР и может быть повторно использован другими разработчиками в соответствии с первоначальным лицензионным соглашением. Многие программы доступны для повторного использования, но, как правило, не используются, поскольку агентства не знают, что они доступны.

Forge.mil на самом деле состоит из ряда различных элементов (входит в TeamForge [3]), каждый из которых имеет отношение к определённому аспекту процесса разработки:

- SoftwareForge — это репозиторий для публичных проектов;
- ProjectForge будет содержать инструменты управления жизненным циклом приложения, такие как контроль версий и отслеживание ошибок, которые могут использоваться другими сервисами за определённую плату;
- CertificationForge предоставит рабочий процесс для сертификации приложений по Common Criteria или другим государственным программам сертификации;
- TestForge предоставит среду тестирования по требованию для опробования новых приложений;
- StandardsForge станет центральным местом встречи для различных сервисных служб для взаимодействия с органами по стандартизации.

Пока что SoftwareForge, кажется, продвинулась дальше всех. В 2009 году DISA объявила, что платформа открыта для несекретных военных проектов. Все проекты на SoftwareForge были доступны для просмотра посторонним разработчикам, чтобы привлечь разработчиков в надежде на расширение сотрудничества. Закрытые проекты в конечном итоге размещаются в другом разделе Forge.mil для работы в секретной сети маршрутизаторов с протоколом Интернет (SIPRNet). Все разработчики используют протокол IPSec для доступа к ресурсам Forge.mil (рис. 1).

Чтобы получить доступ к платформе, участникам либо потребуется выданная военными карта общего доступа, либо их спонсирует кто-то из Министерства обороны. Каждый проект имеет два уровня участников:

- все участники проекта получают доступ к коду и документации и смогут отправить изменения;
- часть команды проекта, называемая коммиттерами, сможет объе-

динить изменения в существующую базу кода.

Каждый проект SoftwareForge проходит проверку. Специальные подразделения DISA контролируют, чтобы любые новые проекты не дублировали другие проекты или не были ответвлением (выявляют воровство кода) от существующего проекта.

Всего за несколько месяцев до стадии бета-тестирования Forge.mil (2009 год) уже имела 60 проектов и около 1300 зарегистрированных пользователей. Через 5 лет Forge.mil может похвастаться 24 000 зарегистрированных пользователей, 900 проектов, 200 активных групп, более 2900 приложений и более 150 000 загрузок — и этот сервис растёт с каждым днём.

Основными целями развития платформы Forge.mil было создание более открытого и прозрачного процесса разработки, который мог бы устранить барьеры для повторного использования программного кода, поощрять сотрудничество и/или препятствовать использованию проприетарных или закрытых систем. Для создания такой платформы для совместной работы потребовалась мощная и адаптируемая технология Application Lifecycle Management (ALM), позволяющая повторно использовать код и улучшать качество, а также сокращать время выхода на рынок новых приложений.

Поддерживая контекстную видимость и прослеживаемость между цепочками инструментов и сохраняя контекст проекта и события, TeamForge интегрирует сторонние коммерческие и открытые инструменты разработки программного обеспечения, такие как JIRA, Jenkins, HP Quality Center и Git/Gerrit. TeamForge поддерживает методологии разработки, такие как Agile, Waterfall и гибридный подход, а CollabNet гордится тем, что TeamForge является единственной платформой, предлагающей как централизованные (Subversion/SVN), так и распределённые (Git) системы контроля версий. Пользователи могут определять и измерять проекты на нескольких настроенных пользователем трекерах и делегировать задачи, используя папки планирования, выпуски и команды. Средства визуальной организации TeamForge включают полный спектр рабочих процессов, доски Agile и Kanban.

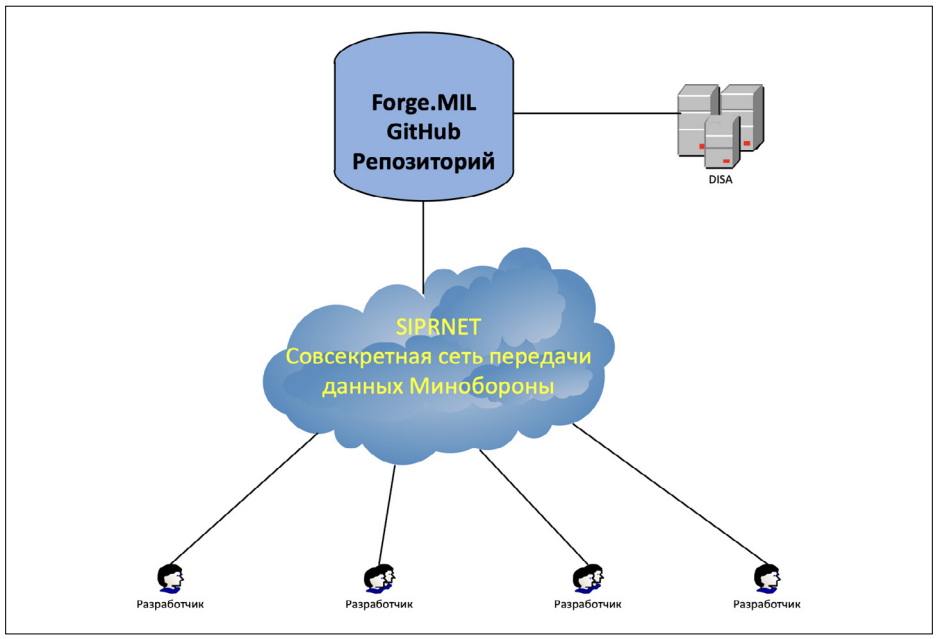


Рисунок 1. Все проекты на SoftwareForge были доступны для просмотра посторонним разработчикам, чтобы привлечь их к сотрудничеству

Внедрение Forge.mil привело к ощутимому сокращению времени цикла разработки и снижению затрат. Forge.mil также способствует быстрому внедрению новых проектов и ускоряет переход с устаревших платформ.

По экспертным оценкам, производительность труда разработчиков АС ВН класса КСБУ повысилась в 50 раз. Преимущества повторного использования кода были впечатляющими. В Министерстве обороны произошли ощутимые улучшения качества кода и заметное ускорение выхода на рынок новых приложений. Эти преимущества неоднократно реализовывались в проектах, начиная от систем вооружения и заканчивая оперативными потребностями в сфере бизнеса. Кроме того, говорить, что экономическая выгода является «существенной», — это, конечно, преуменьшение. По оценкам DISA, экономия варьируется от 18 000 USD на проект для небольших групп (1–15 разработчиков) до 1,2 млн USD на проект для корпоративной группы (300–2000 разработчиков).

В дополнение к этим очень реальным и ощутимым результатам Forge.mil предоставляет ряд нематериальных преимуществ. DISA удалось сформировать сообщество разработчиков, стимулировать творчество и инновации, которых иначе было бы невозможно достичь. Forge.mil также несёт в себе многие социальные и технологические преимущества сообществ программного обеспечения с открытым исходным кодом, которые повышают качество программного обеспечения, гибкость и инновации.

Это согласуется с подходом «Совместно используемой платформы» Стратегии цифрового правительства, который позволяет федеральным сотрудникам работать вместе как внутри, так и между агентствами для сокращения затрат, оптимизации процесса разработки, применения единых стандартов и обеспечения согласованности при создании и предоставлении информации.

Стэн Шнайдер, генеральный директор RTI, заявил, что единственной необходимостью контролировать стоимость программного обеспечения является создание стабильной команды, работающей на базе высокотехнологичной платформы.

В государственном секторе создание информационных систем также

идёт под патронажем портала code.gov, где хранятся исходные коды, созданные привлекаемыми правительством США разработчиками.

Создание сообщества разработчиков на такой платформе, как Forge.mil, способствует решению и таких задач. Агентства должны будут обмениваться специально разработанным кодом друг с другом, чтобы предотвратить дублирование контрактов на разработку программного обеспечения в соответствии с новым законом, подписанным президентом США Д. Байденом. Закон о гармонизации и повторном использовании исходного кода в информационных технологиях (H.R. 9566), или SHARE IT Act, направлен на сокращение примерно 12 миллиардов долларов, которые, по оценкам законодателей, федеральное правительство тратит каждый год на закупку программного обеспечения, требуя от агентств публичного перечисления пользовательского кода и обмена этим кодом с другими агентствами. По словам авторов законопроекта, это решит проблему неэффективности, которая может произойти, когда агентства по незнанию нанимают подрядчиков для разработки кода, который уже был разработан для другого агентства. Согласно новому закону, метаданные включают информацию о том, был ли пользовательский код разработан в рамках контракта или опубликован в репозитории, номер контракта и гиперссылку на репозиторий, где код был передан. Согласно объявлению Лэнгуорти о внесении законопроекта в Палату представителей в сентябре, компании-разработчики программного обеспечения Atlassian и GitLab Inc. поддержали законопроект.

АНАЛИЗ ОПЫТА КИТАЯ по созданию и развитию средств (сред) разработки программного обеспечения

С 90-х годов КНР, учитывая процессы развития мировой экономики, последовательно реализовывала стратегию создания собственных средств производства и производительных сил, способных использовать их в создании конечного продукта как в гражданской сфере, так и сфере обороны и безопасности. Не исключение и создание информационных систем. На основе стратегии изучения и копирования иностранного опыта (в первую очередь США), предприняты множественные попытки создания инструментов (сред) разработки программного обеспечения для государственных нужд и частного сектора. Результатом накопления опыта и создания достойных нишевых инструментов, в том числе в рамках «Программы 863», стал фактический перенос гражданских и военных проектов в области создания сред разработки под эгиду Госпрограммы долгосрочного и среднесрочного развития науки и техники (на 2006–2020 гг.).

В КНР для обеспечения развития средств производства информационных систем и продуктов последовательно и под государственным присмотром осуществлялся перенос иностранных технологий, точечное освоение «китаизированных» технологий в технологических корпорациях, полный рефакторинг или создание своих инструментов и внедрение безопасных инструментов в государственные структуры, включая реализующие задачи обороны и безопасности. То есть, фактически последовательно осуществляется трансфер технологий от доверенных коммерческих инновационных компаний Китая в закрытый сектор. Завершив в конце 10-х — начале 20-х годов трансфер и рефакторинг западных технологий государственный сектор Китая перешел к обеспечению собственных нужд на базе уже отечественных инструментов и технологий производства программного кода. Регулирование осуществляет MIIT (Министерство промышленности и информационных технологий) и CAC (управление киберпространства Китая).

В рамках китайской регуляторики сферы управления безопасностью кода и разработки информационных систем фактически создан прототип forge.mil, с централизованным общегосударственным распределенным репозиторием «открытого кода» с собственной лицензией, одобренной международным сообществом Open Source Initiative и его закрытыми клонами в ведомствах.

Репозиторий работает на основе принципов: вендор ПО — китайская компания, обязательное размещение кода и поддержка версионности в репозитории, лицензия Mulan. Технически, он базируется на собственном инструменте Gitee, являющимся функциональным аналогом GitHub.

Фактическим оператором является определенная MIIT структура, под патронажем которой продвигается в рынке и навязывается для нужд обороны и безопасности система продуктов от собственных языков программирования, компиляторов, отладчиков и сред сборки пакетов, до операционных систем, сред виртуализации и СУБД.

Учитывая наличие у Китая собственных процессоров и «железа» можно предположить, что в ближайшем будущем Китай завершит формирование обособленной и самодостаточной экосистемы средств производства программного обеспечения.

ПРЕДЛОЖЕНИЯ по организации работ по созданию российской безопасной среды разработки и поддержки российских репозиториев и программных продуктов с учётом требований информационной безопасности

Указом Президента Российской Федерации от 29 октября 2024 г. № 927, которым утверждены «Основы государственной политики в области обеспечения технологической независимости критической информационной инфраструктуры Российской Федерации на период до 2030 года» и поставлены задачи, в том числе:

- создание российской безопасной среды разработки и поддержка развития российских репозиториев программных пакетов и библиотек как инфраструктуры разработки общесистемного, прикладного и промышленного программного обеспечения с учётом требований информационной безопасности;

◆ разработка российских и локализация иностранных технологий для производства российской высокотехнологичной продукции для критической информационной инфраструктуры.

Фактический отказ России, начиная с 1993 года, от развития собственной ИТ-индустрии и выбор курса на вхождение в международную ИТ-кооперацию на правах, в первую очередь, пользователя и потребителя современных информационных продуктов, привёл к возникновению крайне высокой зависимости отечественных разработчиков АСУ и информационных систем (ИС) от зарубежных технологий и средств разработки. Санкционные действия, предпринятые против России в течение последних трёх лет по ряду направлений, продемонстрировали эту зависимость в самом явном и критическом виде.

Следует учитывать, что существенные возможности для дальнейшего усиления такого давления и затруднения выполнения задач, определённых Указом Президента РФ, сохраняются и могут быть использованы противниками РФ в целях нанесения ущерба обороноспособности, безопасности и правопорядку. Это в первую очередь касается прекращения доступа к средствам проектирования и разработки, а также внедрения уязвимостей в разрабатываемое ПО.

В условиях нарастания санкционного давления наиболее серьёзными рисками при производстве операционных систем и программных продуктов, созданных на базе иностранных репозиториев, являются:

◆ технологическая доступность внедрения противником недекларированных возможностей в разрабатываемые российские дистрибутивы программных продуктов;

◆ отсутствие возможности определять жизненный цикл (версию) программного продукта, находящегося в иностранном репозитории;

◆ невозможность оперативно устранять выявленные уязвимости;

◆ отсутствие возможности разрабатывать программные пакеты и библиотеки, соответствующие требованиям, установленным Российской Федерацией;

◆ невозможность свободного внесения отечественных разработок в проекты открытого программного обеспечения;

◆ сложность обеспечения обратной совместимости произведённых ранее программно-аппаратных средств и отсутствие современных драйверов для их запуска.

Полную технологическую независимость критической информационной инфраструктуры (КИИ) невозможно обеспечить без развития собственной производственной базы отечественного ПО на основе полнофункциональных средств проектирования и разработки российской радиоэлектронной продукции, находящихся на территории и в юрисдикции Российской Федерации.

Кроме того, отсутствие механизма целенаправленного финансирования разработки ПО для нужд обороны и безопасности, при котором множество учреждений, входящих в сектор ОПК (ВПК), обороны страны и безопасности государства, размещают заказы на разработку аналогичных по функциональности и техническому содержанию информационных систем, приводит к существенному увеличению стоимости и сроков разработки ПО.

Время и опыт решения поставленной задачи за последние 30 лет показали, что задача обеспечения технологической независимости критической информационной инфраструктуры не может быть решена без долговременного целевого государственного управления задачей создания российской доверенной безопасной среды разработки, включая поддержку развития российских репозиториев программных пакетов и библиотек как инфраструктуры разработки общесистемного, прикладного и промышленного программного обеспечения с учётом требований информационной безопасности.

С учётом этого создание национальной инструментальной среды для разработки доверенных ПАК и специализированного ПО для нужд обороны страны, безопасности государства и обеспечения правопорядка (далее НИС) на основе репозиториев программного обеспечения, находящегося в юрисдикции и на территории Российской Федерации, является важным фактором обеспечения безопасности и жизнедеятельности государства.

Данная доверенная платформа также позволит сформировать среду общения между предприятиями ОПК, академической наукой и коммерческими организациями, взаимодействующими с оборонным сектором, сообществом разработчиков.

Основными целями создания, функционирования и последующего развития НИС определить:

◆ восстановление технологического суверенитета Российской Федерации в сфере создания АСУ и ИС для нужд обороны и безопасности;

◆ создание инфраструктуры разработки и поддержки полного жизненного цикла российского программного обеспечения, в том числе для отечественных аппаратных платформ;

◆ обеспечение устойчивости и безопасности функционирования объектов информационной инфраструктуры обороны и безопасности Российской Федерации.

Основными задачами НИС можно считать:

◆ обеспечение предприятий ОПК системным, прикладным, специальным и встраиваемым программным обеспечением для производства доверенных ПАКов, применяемых для реализации задач в сфере обороны и безопасности;

◆ формирование единых стандартов, устанавливающих требования и рекомендации к средствам поддержки жизненного цикла всех классов программного обеспечения и технических средств функционирования российских репозиториев;

◆ формирование экосистемы российских кроссплатформенных продуктов с использованием свободного и проприетарного программного обеспечения, совместимых друг с другом и с отечественными аппаратны-

ми архитектурами, независимыми от зарубежных решений;

◆ определение методов (включая критерии и процедуры) и средств проверки совместимости программных продуктов между собой и с аппаратными средствами, включая процедуры тестирования и проверки производительности программных продуктов;

◆ определение порядка и средств оценки доверенности и безопасности программного обеспечения в российских репозиториях;

◆ обеспечение соответствия программных продуктов установленным требованиям к безопасности для объектов обороны и безопасности;

◆ формирование доверенных каналов распространения программного обеспечения и доверенного оператора платформы;

◆ совершенствование нормативной и методической баз;

◆ создание системы обучения пользователей, разработчиков программного обеспечения и специалистов в области информационных и телекоммуникационных технологий.

Предложенная четырёхуровневая инфраструктура национальной инструментальной среды одобрена участниками заседания ВПК. К созданию такой инфраструктуры необходимо приступать немедленно, начиная с сокращения количества репозиториев операционных систем, поддерживаемых на государственном уровне, поэтапного объединения разрозненных команд разработчиков.

Для этого нормативным документом Правительства:

◆ создать небольшую (не более 10–15 человек) высокопрофессиональную работоспособную экспертную рабочую группу из представителей заказчиков, представителей организаций, доказавших реальными результатами и опытом свою компетенцию, и обеспечить её достаточное финансирование;

◆ определить порядок финансирования работ начиная с 2025 года, обеспечив достойные условия и оплату экспертов на мировом уровне как самозанятым, не зависящим от организаций и ведомств;

◆ определить базовую государственную межведомственную организацию для подготовки и проведения работ подготовительного этапа, обеспечить закрытый характер работ;

◆ утвердить примерный план создания Системы, предусматривающий три этапа:

- **начальный,** на котором разрабатывается, согласовывается и утверждается концепция Системы, разрабатываются предложения по совершенствованию нормативной базы, разрабатывается ТЗ на систему в целом и ЧТЗ на составные части, оцениваются потребности в финансовом обеспечении, уточняется план и создаётся пилотный проект. Ориентировочный срок — полгода с момента начала финансирования;
- **основной** этап разработки, на котором в соответствии с концепцией создаются компоненты Системы, утверждаются нормативные документы. Ориентировочный срок — 2 года с начала финансирования;
- **Поэтапный ввод в эксплуатацию.** Срок — постоянно с момента ввода в эксплуатацию. Финансирование — в соответствии с законом, аналогичным по смыслу Закону о гармонизации и повторном использовании исходного кода в информационных технологиях (H.R. 9566), SHARE IT Act;

◆ подготовить пилотный проект — «Создание информационной системы по контролю субъектов и объектов КИИ в соответствии с указом Президента от 8.11.2023 по расширению полномочий ФСТЭК на основе защищённого репозитория ПАК и ПО с обеспечением требований по информационной безопасности на уровне гостайны». В рамках поддержки отечественной промышленности 19 июня 2015 года принят Федеральный закон № 188-ФЗ и постановление Правительства от 16 ноября 2015 г. № 1236 об установлении запрета на допуск программного обеспечения и баз данных, происходящих из иностранных государств для целей осуществления закупок для обеспечения государственных и муниципальных нужд. ВПК необходимо подготовить дополнение в постановление Правительства о ведении общего закрытого репозитория — реестра программных продуктов и баз данных, средств радиоэлектронной промышленности для обеспечения обороны страны и безопасности государства, защиты внутреннего рынка Российской Федерации, развития национальной экономики, поддержки российских товаропроизводителей в рамках гособоронзаказа по аналогии с имеющейся нормативной базой.

ВПК совместно с федеральными структурами и государственными корпорациями также необходимо подготовить постановление Правительства о поддержке финансирования существующих российских научных школ для решения задач создания российской безопасной среды разработки и поддержки развития российских репозиториев программных пакетов и библиотек как инфраструктуры разработки общесистемного, прикладного и промышленного программного обеспечения с учётом требований информационной безопасности.

Как показала реальная жизнь, согласованную межведомственную концепцию отечественной системы репозиториев и отдельное мероприятие по системе отечественных репозиториев в федеральном проекте «Цифровые технологии» национального проекта «Цифровая экономика» Российской Федерации оказалось невозможно разработать и обосновать без соответствующего статуса рабочей группы с последующим долговременным финансированием и межведомственным научно-техническим сопровождением, контролем на уровне Правительства.

* * *

«Дорогу осилит идущий...» Пока в практическом решении проблем отечественной системы репозиториев программного обеспечения на федеральном уровне уже который год наблюдается «бег на месте».