УДК: 004.056.2

Алексей А. Тимаков[1], Михаил М. Фадеев[2], Илья Г. Рыжов[3], Александр В. Лысиков[4]
[1]МИРЭА – Российский технологический университет,
пр-кт Вернадского, 78, Москва, 119454, Россия
[2]Национальный исследовательский ядерный университет «МИФИ»,
Каширское ш., 31, Москва, 115409, Россия
[3]ПАО «Сбербанк»,
ул. Вавилова, 19, Москва, 117997, Россия
[4]ООО «ЭНИ.РАН»,
ул. Свободы, 10, Ульяновск, 432017, Россия
[1]e-mail: timakov@mirea.ru, https://orcid.org/0000-0003-4306-789X
[2]e-mail: mmfadeev@mephi.ru, https://orcid.org/0000-0003-0327-9759
[3]e-mail: ryzhov.ilgen@gmail.com, https://orcid.org/0000-0001-6014-6982
[4]e-mail: a.ly5ikov@yandex.ru, https://orcid.org/0000-0002-2432-924

ОСОБЕННОСТИ ПРОГРАММНОЙ АРХИТЕКТУРЫ ЯЗЫКОВОЙ ПЛАТФОРМЫ PLIF ДЛЯ АНАЛИЗА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ В АВТОМАТИЗИРОВАННЫХ СИСТЕМАХ

DOI: http://dx.doi.org/10.26583/bit.2023.2.03

Аннотация. В области формальных моделей безопасности компьютерных систем к настоящему времени выполнен значительный объем исследований. Одним из наиболее важных направлений является контроль информационных потоков в программных средах автоматизированных систем. Соответствующие механизмы сегодня активно исследуются, предпринимаются попытки их внедрения в языковые платформы, предназначенные для создания как системного, так и прикладного программного обеспечения. На данном этапе научной проработки указанное направление можно отнести к академическому, характеризуемому теоретическими изысканиями и появлением множества прототипов. По мнению авторов, основной проблемой использования обозначенных механизмов при создании промышленных автоматизированных систем и включения соответствующих этапов в цикл безопасной разработки является сложность ручного инструментирования программного кода на основе меток безопасности и проверка выявленных нарушений. В работе представлена разработанная технологическая платформа выявления запрещенных информационных потоков в программных блоках систем управления базами данных, предложена общая процедура ее применения, которая предполагает разделение функций разработчиков программного обеспечения и аналитиков информационной безопасности.

Ключевые слова: информационные потоки, контроль информационных потоков, формальная верификация, языковая платформа, политика безопасности.

Для цитирования: ТИМАКОВ, Алексей А. и др. ОСОБЕННОСТИ ПРОГРАММНОЙ АРХИТЕКТУРЫ ЯЗЫКОВОЙ ПЛАТФОРМЫ PLIF ДЛЯ АНАЛИЗА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ В АВТОМАТИЗИРОВАННЫХ СИСТЕМАХ. Безопасность информационных технологий, [S.l.], т. 30, № 2, с. 53–62, 2023. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1496. DOI: http://dx.doi.org/10.26583/bit.2023.2.03.

Aleksei A. Timakov[1], Mikhail M. Fadeev[2], Ilya G. Ryzhov[3], Aleksandr V. Lysikov[4]
[1]MIREA – Russian Technological University,
Vernadsky Ave., 78, Moscow, 119454, Russia
[2]National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Kashirskoe sh., 31, Moscow, 115409, Russia
[3]PJSC "Sberbank",
Vavilova str., 19, Moscow, 117997, Russia
[4]PLC "ENI.RAN",
Svobody str., 10, Ulyanovsk, 432017, Russia

[1]*e-mail: timakov@mirea.ru, https://orcid.org/0000-0003-4306-789X*
[2]*e-mail: mmfadeev@mephi.ru, https://orcid.org/0000-0003-0327-9759*
[3]*e-mail: ryzhov.ilgen@gmail.com, https://orcid.org/0000-0001-6014-6982*
[4]*e-mail: a.ly5ikov@yandex.ru, https://orcid.org/0000-0002-2432-9244*

Features of the software architecture of the PLIF language platform for the analysis of the security of information flows in automated systems

DOI: http://dx.doi.org/10.26583/bit.2023.2.03

Abstract. A lot of research in the field of formal security models have been conducted so far. Information flow control in automated systems software represents one of important directions. Appropriate mechanisms are being investigated in attempt to embed them into modern programming platforms designated for system and application software creation. Today all such studies are of academic type, they have theoretical meaning and usually end up at stage of prototype. According to those authors the main problem in adopting such mechanisms for industrial use and including respective steps into security development lifecycle is complexity of manual code markup with security labels and security warnings checking. We present a new platform for detecting illegal information flows in database program units and outline the respective testing procedure which explicitly separates the responsibilities of developers from that of security analysts.

Keywords: information flows, information flow control, formal verification, language platform, security policy.

For citation: TIMAKOV, Aleksei A. et al. Features of the software architecture of the PLIF language platform for the analysis of the security of information flows in automated systems. IT Security (Russia), [S.l.], v. 30, no. 2, p. 53–62, 2023. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1496. DOI: http://dx.doi.org/10.26583/bit.2023.2.03.

Введение

В настоящее время все больше специалистов признает необходимость развития формальных моделей безопасности, построенных на основе контроля информационных потоков (КИП) [1–6]. Наиболее перспективной областью применения таких моделей видится обеспечение конфиденциальности и целостности данных, обрабатываемых в прикладном программном обеспечении (ПО). Следует отметить, что в настоящее время появились первые языковые платформы, в которых реализованы полноценные механизмы КИП: Joana[1] [7], JIF [8], Paragon [9–10], Flow Caml [11]. К сожалению, ни один проект не получил широкого распространения.

Разработанная авторами платформа PLIF (PL/SQL Information Flow) для анализа безопасности информационных потоков в автоматизированных системах в среде баз данных Oracle позволяет строго разделить функции программной реализации алгоритмов и анализа безопасности информационных потоков, возникающих при их выполнении.

PLIF основана на темпоральной логике действий [12–13] и технологии проигрывания моделей. Языковая часть представляет адаптированный язык политик безопасности Paralocks [14].

Paralocks – язык для построения выразительных, но статически проверяемых политик безопасности информационных потоков в программном обеспечении. Paralocks основан на идее блокировок [9] и позволяет задавать ограничения по чтению и записи для элементов среды вычислений (переменных, параметров функций, возвращаемых значений, исключений и т.д.) в соответствии с правилами вышестоящей политики управления доступом и дополнительных требований прикладного уровня. Отличительной особенностью языка, предопределившей его выбор, является гибкость – с его помощью

[1]Analysis of Security APIs. URL: https://www.dsi.unive.it/~focardi/ASA8/ (дата обращения: 02.02.2023).

можно адаптировать КИП в приложении к любым применяемым на системном уровне моделям управления доступом: дискреционному, ролевому, мандатному.

1. Особенности программной реализации языковой платформы PLIF

Рассмотрим основы семантики языка Paralocks [15]. В общем виде некоторая политика безопасности P_{k} определяется формулой логики предикатов, представимой в виде конъюнкции определенных дизъюнктов Хорна: $P_k \triangleq C_1 \wedge C_2 \wedge \dots C_n$, где $C\_\{n\}$ – предложение политики вида: $\forall x_1, \dots, x_m: l_1(\cdot) \wedge l_2(\cdot) \wedge l_3(\cdot) \dots \Rightarrow Flow(u)$ или $\forall x_1, \dots, x_m: \neg l_1(\cdot) \vee \neg l_2(\cdot) \vee \neg l_3(\cdot) \dots \vee Flow(u)$, где *Flow(u)* – предикат, обозначающий поток данных к u (где u – связанная переменная или константа, обозначающая пользователя), $l_1..l_n$ – условия (блокировки), выполнение которых требуется для того, чтобы *Flow(u)* принял значение *TRUE*. Предикаты $l_1..l_n$ могут быть параметрическими или непараметрическими. Множество возможных блокировок определяется заданными на системном уровне элементами политики безопасности (например, ролями) и дополнительными ограничениями.

Предположим, есть программа, которая имеет дело с двумя действующими субъектами – поставщиком программного обеспечения и покупателем. Программа имеет доступ к чувствительным данным поставщика – ключу активации программного обеспечения, который не должен передаваться покупателю до подтверждения факта оплаты лицензии. Для моделирования платежного акта может быть введена специальная блокировка isPaid. Открытие блокировки isPaid необходимо связать с фактическим подтверждением платежа в коде.

Политика безопасности для программного ключа может быть записана следующим образом: $\{\forall x: Supplier(x) \Rightarrow Flow(x); isPaid() \Rightarrow Flow(x)\}$.

Данные, содержащиеся в переменной с такой политикой, могут свободно передаваться поставщику, но не должны передаваться покупателю до тех пор, пока блокировка isPaid не будет открыта.

Язык Paralocks не привязан к конкретной модели управления доступом – он не предполагает использование уровней конфиденциальности (целостности) или иерархии пользовательских ролей. Основная идея состоит в том, чтобы логически формализовать условия, при которых данный субъект в системе может получить доступ к информации.

Механизм блокировок позволяет аналитику безопасности описать допустимые информационные потоки в программном обеспечении и получить гарантию того, что программа соответствует спецификации. Корректная разметка исходного кода программы с использованием политик Paralocks является нетривиальной задачей.

В отличие от известных подобных механизмов реализации КИП [7–9], основанных на методах статического и динамического анализа ПО и требующих от программистов решения дополнительных нетривиальных задач, связанных с разметкой исходного кода, а также интерпретации результатов анализа, платформа PLIF позволяет строго разделить функции написания кода и контроля корректности его логики с учетом специфики предметной области и с привязкой к принятой в системе политике безопасности за счет переноса механизма проверки условий безопасности вычислений в область формальной верификации моделей программ.

В общем виде этапы анализа информационных потоков в программном обеспечении, реализующем бизнес-логику на основе хранимых программных блоков баз данных, представлены на рис. 1.

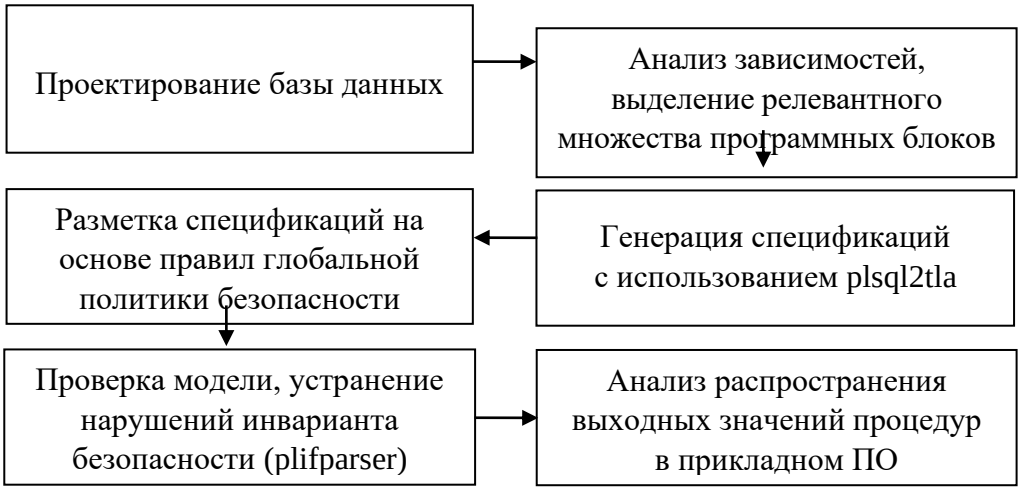

Рис. 1. Этапы анализа безопасности информационных потоков
Fig. 1. Stages of information flow security analysis

Выбор программной среды для реализации механизма КИП − хранимые программные блоки базы данных (блоки PL/SQL) − объясняется следующими особенностями: близость к данным, малый объем кода (в среднем размер хранимой процедуры (функции) не превышает 60 строк), отсутствие вспомогательных вычислений (PL/SQL код, как правило, хорошо оптимизирован и сфокусирован на основном алгоритме), наличие развитых средств анализа зависимостей – позволяет эффективно выделять релевантное относительно чувствительной информации подмножество хранимых программных блоков – сериализация транзакций с использованием различных стратегий управления блокировками − позволяет сократить количество состояний модели, описывающей параллельные вычисления. Для простоты в описании технологии используется подмножество языка PL/SQL, BNF-грамматика которого представлена в [14].

Первым этапом анализа является проектирование базы данных. Его необходимо осуществлять таким образом, чтобы конфиденциальные данные размещались компактно, в ограниченном наборе таблиц. Данное требование не противоречит общепринятым принципам безопасной разработки, и при этом позволяет более эффективно изолировать критичные вычисления от общего кода PL/SQL.

Следующим этапом является анализ зависимостей и выделение релевантного множества программных блоков. На этом этапе предполагается определение тех блоков PL/SQL, которые имеют отношение к манипулированию данными, подлежащими защите. Важной особенностью современных промышленных систем управления базами данных, используемых в автоматизированных системах, является наличие встроенных механизмов выявления прямых и косвенных зависимостей.

Генерация, разметка и применение алгоритма проверки TLA+ спецификаций являются ключевыми и наиболее сложными этапами.

Генерация спецификаций для хранимых процедур и функций осуществляется с помощью программного средства plsql2tla в соответствии с разработанной абстрактной семантикой информационных потоков [14].

Пользовательские сеансы взаимодействия с базой данных посредством хранимых программных блоков представляются параллельными процессами. Последовательные вычисления в рамках одного процесса можно описать системой переходов состояний вида: $\langle PC, c, M \rangle$, где: PC − метка безопасности счетчика команд (program counter) в

текущем сеансе; c − исполняемая команда текущего сеанса; M − абстрактное состояние среды, которое задает отображение локальных и глобальных переменных, входных и выходных потоков на соответствующие им метки безопасности. Правила абстрактной семантики описаны в [14].

Разметка спецификаций с использованием грамматики Paralocks осуществляется в соответствии с заданными на системном уровне правилами управления доступом (объектными привилегиями, назначенными ролям) и дополнительными ограничениями.

За основу формального определения безопасности семантики принимается понятие прогресс-зависимого информационного невлияния [14]. В отличие от классической схемы информационного невлияния, здесь проверка условий безопасности информационных потоков выполняется после каждого шага вычислений, при этом сами условия ослабляются (контролируется только вывод данных в выходные потоки).

Для проверки спецификаций и устранения нарушений инварианта безопасности разработан алгоритм проверки TLA+ спецификаций, основанный на методе проигрывания моделей [14]. Предварительный шаг алгоритма предполагает инициализацию констант, определяющих количество сеансов, множество конкретных пользователей, множество имен связанных переменных, множества имен параметрических и непараметрических блокировок. При проигрывании модели проверяются два свойства: главный инвариант безопасности ParalocksInv и свойство перехода CompInv. В случае нарушения ParalocksInv принимается решение о деклассификации данных (контролируемом раскрытии информации) в некоторых случаях требующей исправления кода или изменения привилегий на доступ к объектам базы данных. При нарушении свойства CompInv в начальном состоянии модели повышается политика соответствующей глобальной переменной − столбца. Для детального анализа проблемных трасс используется утилита графического отображения траектории, приводящей к нарушению инварианта plifparser. Алгоритм является конечным. Доказано в [14], что успешное завершение алгоритма гарантирует безопасность вычислений в смысле прогресс-зависимого информационного невлияния при неограниченном количестве программных блоков, выполняемых в каждом пользовательском сеансе.

Контроль распространения выходных значений верифицированных процедур и функций в прикладном программном обеспечении осуществляется с использованием стандартного анализа помеченных данных (Taint Tracking), рекомендуемой средой проверки является CodeQL [16].

2. Практическая реализация языковой платформы PLIF

Генератор TLA+ спецификаций plsql2tla является программным модулем, написанном на языке Java 11, позволяющим создавать TLA+ спецификации на основе поданных на вход процедур и функций (далее по тексту программные блоки) PL/SQL.

Реализация plsql2tla опирается на генератор нисходящих анализаторов для формальных языков ANTLR, с помощью которого в генераторе спецификаций с использованием грамматики языка PL/SQL[1] строится абстрактное синтаксическое дерево.

Опишем более детально алгоритм работы утилиты plsql2tla (рис. 2). Программа принимает на вход путь до папки, содержащей внутри себя процедуры и функции PL/SQL, а также описания таблиц, и путь до директории, в которой будут находиться сгенерированные файлы TLA+ спецификаций. После считывания входных данных

[1]Анализатор Oracle(c) PL/SQL 11g. URL: https://github.com/antlr/grammars-v4/blob/master/sql/plsql/PlSqlParser.g4 (дата обращения: 22.01.2023).

создаются объекты DatabaseSchema и ProgrammBlockDataHolder, содержащие все необходимые для создания спецификаций данные о таблицах и программных блоках соответственно. Экземпляры классов DatabaseSchema и ProgrammBlockDataHolder внедряются в объекты MainSpecificationCreator и ParametersSpecificationCreator, вызов методов createSpecification которых затем создает основной файл спецификаций и файл спецификаций с описанием параметров.

Рис. 2. Схема алгоритма программы plsql2tla
Fig. 2. Diagram of the algorithm of the plsql2tla program

Схематично упрощенная внутренняя структура программы может быть описана с помощью UML-диаграммы классов (рис. 3).

Инструмент проверки моделей TLC (TLA Checker) используется для проверки свойств безопасности[1].

TLC [17] поддерживает две структуры данных: множество всех наблюдаемых состояний seen, о которых известно, что они достижимы, и очередь (FIFO) SQ, содержащую элементы из seen, чьи последующие состояния (состояния-потомки) не были проверены. Элементы SQ являются фактическими состояниями, тогда как множество seen содержит только контрольные суммы состояний.

В начале работы TLC инициализирует множество seen и очередь SQ состояниями, удовлетворяющими начальному предикату. Затем TLC выполняет поиск в ширину графа состояний[2], модифицируя на каждом шаге множество seen и очередь SQ, при этом состояние-потомок s' с контрольной суммой k проверенного состояния s, такое что $s' \in$ seen проверяться не будет. Об ошибке сообщается, если обнаруживается состояние, не удовлетворяющее инварианту, или если у текущего состояния отсутствуют состояния-потомки.

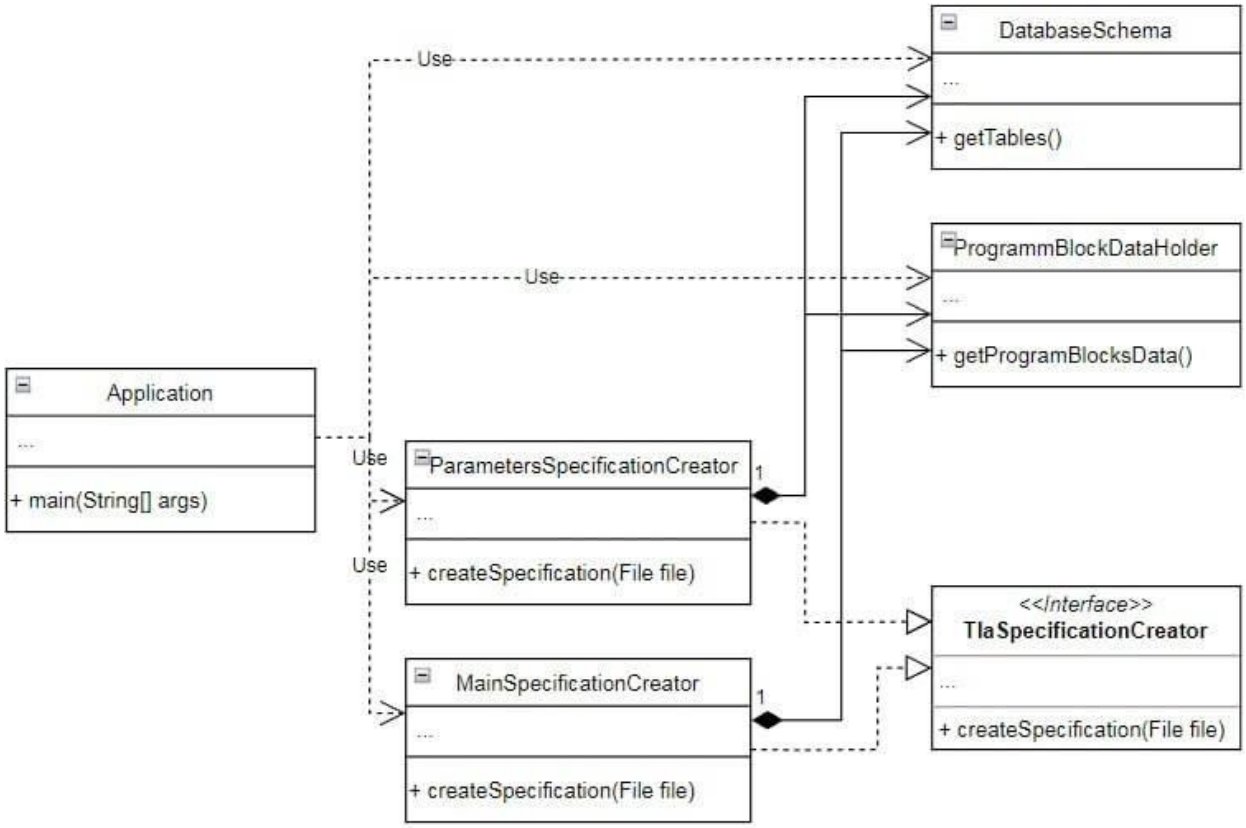

Рис. 3. Упрощенная UML-диаграмма классов plsql2tla
Fig. 3. Simplified UML diagram of plsql2tla classes

[1]LamportL. TLA+ Tools. URL: https://lamport.azurewebsites.net/tla/tools.html (дата обращения: 22.01.2023).
[2]Apalache vs TLC. URL: https://mbt.informal.systems/ docs/tla_basics_tutorials/apalache_vs_tlc.html (дата обращения: 31.01.2023).

Утилита графического отображения траектории plifparser написана на языке JavaScript и использует библиотеку D3.js. Представляет собой Web-консоль, позволяющую исследовать графы информационных потоков, возникающие в трассах, приводящих к нарушению инварианта безопасности. На начальном этапе происходит разбор выходного файла в формате DOT, сгенерированного на основе TLA+ спецификаций, после чего проводится графическая интерпретация иерархической структуры. Пример работы утилиты представлен на рис. 4.

Узлы графа представляют собой отдельные элементы среды вычислений: слева – фактические параметры хранимых процедур и функций, входные потоки; в центре – локальные переменные программных блоков, исключения, литералы, выходные значения; справа – глобальные переменные (столбцы таблиц и представлений базы данных), выходные потоки, ассоциированные с сокетами и файлами операционной системы. Ребрами графа являются информационные потоки, возникающие между элементами среды вычислений при выполнении операций присваивания, записи/чтения в сокеты, файлы, выполнении операций DML с объектами базы данных. Для каждого узла отображаются текущие и ранее присвоенные абстрактные значения – политики безопасности.

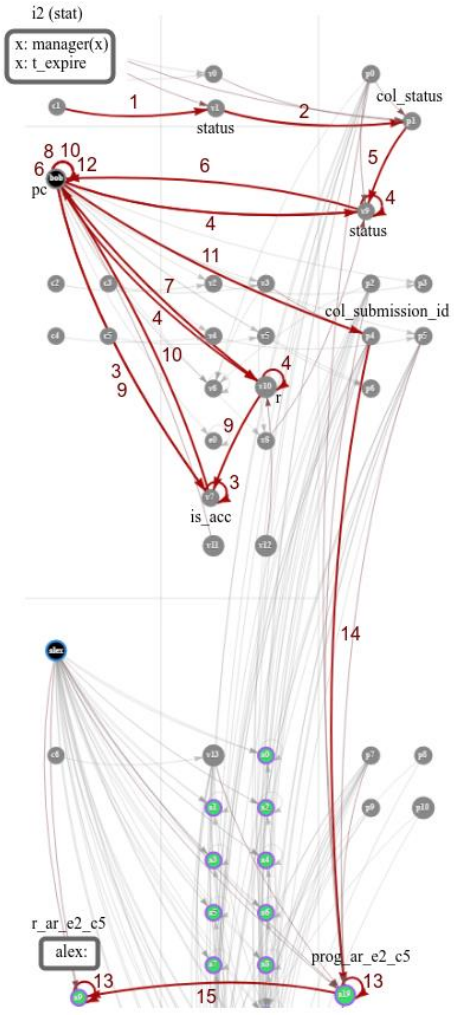

Рис. 4. Граф информационных потоков, сгенерированный утилитой plifparser
Fig. 4. Graph of information flows generated by the plifparser utility

Заключение

Таким образом, в работе предложена общая процедура исследования безопасности информационных потоков в программном обеспечении автоматизированных систем с использованием новой платформы PLIF. Представлены особенности ее программной архитектуры. В качестве перспективных направлений дальнейшей научной работы видится проведение отдельных исследований в отношении последнего этапа представленной процедуры анализа информационных потоков − анализа распространения выходных значений критичных процедур и функций в прикладном программном обеспечении.

СПИСОК ЛИТЕРАТУРЫ:

1. Adam Chlipala. 2015. Ur/Web: A Simple Model for Programming the Web. SIGPLAN Not. 50, 1 (January 2015), p. 153–165. DOI: https://doi.org/10.1145/2775051.2677004.
2. Arden O., George M.D., Liu J., Vikram K., Askarov A. and Myers A.C. Sharing Mobile Code Securely with Information Flow Control. IEEE Symposium on Security and Privacy, San Francisco, CA, USA. 2012, p. 191–205. DOI: https://doi.org/10.1109/SP.2012.22.
3. Daniel Schoepe, Daniel Hedin, and Andrei Sabelfeld. 2014. SeLINQ: tracking information across application-database boundaries. SIGPLAN Not. 49, 9 (September 2014), p. 25–38. DOI: https://doi.org/10.1145/2692915.2628151.
4. Schultz D.A. (2012). Decentralized information flow control for databases (Doctoral dissertation, Massachusetts Institute of Technology). URL: http://hdl.handle.net/1721.1/78363 (дата обращения: 01.02.2023).
5. Guarnieri M., Balliu M., Schoepe D., Basin D. and Sabelfeld A. Information-Flow Control for Database-Backed Applications. IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden. 2019, p. 79–94. DOI: https://10.1109/EuroSP.2019.00016.
6. Kozyri E. et al. Expressing Information Flow Properties. Foundations and Trends® in Privacy and Security. 2022, v. 3, no. 1, p. 1–102. DOI: http://dx.doi.org/10.1561/3300000008.
7. Graf J., Hecker M., Mohr M. & Snelting G. (2015). Checking Applications using Security APIs with JOANA. URL: https://www.semanticscholar.org/paper/Checking-Applications-using-Security-APIs-with-Graf-Hecker/3ed693338777cb5d4203ff8616efa62ec359673c (дата обращения: 01.02.2023).
8. Andrew C. Myers and Barbara Liskov. 1997. A decentralized model for information flow control. SIGOPS Oper. Syst. Rev. 31, 5 (Dec. 1997), p. 129–142. DOI: https://doi.org/10.1145/269005.266669.
9. Broberg N., Sands D. (2006). Flow Locks: Towards a Core Calculus for Dynamic Flow Policies. In: Sestoft, P. (eds) Programming Languages and Systems. ESOP 2006. Lecture Notes in Computer Science, vol 3924. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/11693024_13.
10. Niklas Broberg and David Sands. 2010. Paralocks: role-based information flow control and beyond. SIGPLAN Not. 45, 1 (January 2010), p. 431–444. DOI: https://doi.org/10.1145/1707801.1706349.
11. François Pottier and Vincent Simonet. 2003. Information flow inference for ML. ACM Trans. Program. Lang. Syst. 25, 1 (January 2003), p. 117–158. DOI: https://doi.org/10.1145/596980.596983.
12. Leslie Lamport. 1994. The temporal logic of actions. ACM Trans. Program. Lang. Syst. 16, 3 (May 1994), p. 872–923. DOI: https://doi.org/10.1145/177492.177726.
13. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers [Book Review], in Computer, vol. 35, no. 9, p. 81–81, Sept. 2002. DOI: https://doi.org/10.1109/MC.2002.1033032.
14. Тимаков А.А. Контроль информационных потоков в программных блоках баз данных на основе формальной верификации. Программирование. 2022, № 4, с. 27–49. – EDN: WEMCXC.
15. Niklas Broberg and David Sands. 2009. Flow-sensitive semantics for dynamic information flow policies. In Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security (PLAS '09). Association for Computing Machinery, New York, NY, USA, p. 101–112. DOI: https://doi.org/10.1145/1554339.1554352.
16. Moor O. d. et al. Keynote Address: .QL for Source Code Analysis. Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007), Paris, France. 2007, p. 3–16. DOI: https://doi.org/10.1109/SCAM.2007.31.
17. Yu Y., Manolios P., Lamport L. (1999). Model Checking TLA+ Specifications. In: Pierre, L., Kropf, T. (eds) Correct Hardware Design and Verification Methods. CHARME 1999. Lecture Notes in Computer Science, vol 1703. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/3-540-48153-2_6.

REFERENCES:

[1] Adam Chlipala. 2015. Ur/Web: A Simple Model for Programming the Web. SIGPLAN Not. 50, 1 (January 2015), p. 153–165. DOI: https://doi.org/10.1145/2775051.2677004.

[2] Arden O., George M.D., Liu J., Vikram K., Askarov A. and Myers A.C. Sharing Mobile Code Securely with Information Flow Control. IEEE Symposium on Security and Privacy, San Francisco, CA, USA. 2012, p. 191–205. DOI: https://doi.org/10.1109/SP.2012.22.

[3] Daniel Schoepe, Daniel Hedin, and Andrei Sabelfeld. 2014. SeLINQ: tracking information across application-database boundaries. SIGPLAN Not. 49, 9 (September 2014), p. 25–38. DOI: https://doi.org/10.1145/2692915.2628151.

[4] Schultz D.A. (2012). Decentralized information flow control for databases (Doctoral dissertation, Massachusetts Institute of Technology). URL: http://hdl.handle.net/1721.1/78363 (accessed: 01.02.2023).

[5] Guarnieri M., Balliu M., Schoepe D., Basin D. and Sabelfeld A. Information-Flow Control for Database-Backed Applications. IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden. 2019, p. 79–94. DOI: https://10.1109/EuroSP.2019.00016.

[6] Kozyri E. et al. Expressing Information Flow Properties. Foundations and Trends® in Privacy and Security. 2022, v. 3, no. 1, p. 1–102. DOI: http://dx.doi.org/10.1561/3300000008.

[7] Graf J., Hecker M., Mohr M. & Snelting G. (2015). Checking Applications using Security APIs with JOANA. URL: https://www.semanticscholar.org/paper/Checking-Applications-using-Security-APIs-with-Graf-Hecker/3ed693338777cb5d4203ff8616efa62ec359673c (accessed: 01.02.2023).

[8] Andrew C. Myers and Barbara Liskov. 1997. A decentralized model for information flow control. SIGOPS Oper. Syst. Rev. 31, 5 (Dec. 1997), p. 129–142. DOI: https://doi.org/10.1145/269005.266669.

[9] Broberg N., Sands D. (2006). Flow Locks: Towards a Core Calculus for Dynamic Flow Policies. In: Sestoft, P. (eds) Programming Languages and Systems. ESOP 2006. Lecture Notes in Computer Science, vol 3924. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/11693024_13.

[10] Niklas Broberg and David Sands. 2010. Paralocks: role-based information flow control and beyond. SIGPLAN Not. 45, 1 (January 2010), p. 431–444. DOI: https://doi.org/10.1145/1707801.1706349.

[11] François Pottier and Vincent Simonet. 2003. Information flow inference for ML. ACM Trans. Program. Lang. Syst. 25, 1 (January 2003), p. 117–158. DOI: https://doi.org/10.1145/596980.596983.

[12] Leslie Lamport. 1994. The temporal logic of actions. ACM Trans. Program. Lang. Syst. 16, 3 (May 1994), p. 872–923. DOI: https://doi.org/10.1145/177492.177726.

[13] Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers [Book Review], in Computer, vol. 35, no. 9, p. 81–81, Sept. 2002. DOI: https://doi.org/10.1109/MC.2002.1033032.

[14] Timakov A.A. Control of information flows in software blocks of databases based on formal verification, Programming. 2022, vol. 4, p. 27–49 (in Russian) – EDN: WEMCXC.

[15] Niklas Broberg and David Sands. 2009. Flow-sensitive semantics for dynamic information flow policies. In Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security (PLAS '09). Association for Computing Machinery, New York, NY, USA, p. 101–112. DOI: https://doi.org/10.1145/1554339.1554352.

[16] Moor O. d. et al. Keynote Address: .QL for Source Code Analysis. Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007), Paris, France. 2007, p. 3–16. DOI: https://doi.org/10.1109/SCAM.2007.31.

[17] Yu Y., Manolios P., Lamport L. (1999). Model Checking TLA+ Specifications. In: Pierre, L., Kropf, T. (eds) Correct Hardware Design and Verification Methods. CHARME 1999. Lecture Notes in Computer Science, vol 1703. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/3-540-48153-2_6.

Поступила в редакцию – 10 февраля 2023 г. Окончательный вариант – 19 апреля 2023 г.
Received – February 10, 2023. The final version – April 19, 2023.