УДК 004.052.42+004.942 Вестник УрФО № 2(36) / 2020, с. 43–48

Моргунов Д. А.

DOI: 10.14529/secur200207

ВЫЯВЛЕНИЕ СКРЫТЫХ УЯЗВИМОСТЕЙ В ИСХОДНОМ КОДЕ МНОГОПОТОЧНЫХ ПРОГРАММ ПОСРЕДСТВОМ АНАЛИЗА ФУНКЦИОНАЛЬНЫХ ПЕРЕХОДОВ

В статье представлены новая теоретико-множественная модель и процедуры, позволяющие уменьшить временные затраты на обнаружение скрытых уязвимостей в исходном коде многопоточных компьютерных программ, а также результаты проведенного математического моделирования. Под скрытыми уязвимостями в статье понимаются уязвимости, приводящие к ситуациям «гонок» и взаимоблокировкам, поскольку они имеют стохастический характер проявления во время тестирования, что значительно усложняет их выявление. Представленная модель описывает состояния каждого потока многопоточной компьютерной программы исполняемой в настоящее время функцией и содержимым стека вызова функций. При этом сохраняется возможность использования модели в верификации методом Model Checking, а также исключается необходимость решения задачи поиска инварианта модели. Представленные процедуры позволяют формировать спецификации для метода проверки на модели, выполнение которых позволяет выявить уязвимости, приводящие к ситуациям «гонок» и взаимоблокировкам, в исходном коде многопоточных программ.

Ключевые слова: *верификация, математическое моделирование, выявление уязвимости, ситуация «гонки», взаимоблокировка.*

Morgunov D. A.

IDENTIFICATION OF HIDDEN VULNERABILITIES IN THE SOURCE CODE MULTI-THREAD PROGRAMS BY ANALYSIS OF FUNCTIONAL TRANSITIONS

The article presents a new set-theoretic model and procedures that reduce the time required to detect hidden vulnerabilities in the source code of multi-threaded computer programs, as well as the results of mathematical modeling. Hidden vulnerabilities in the article are understood as vulnerabilities leading to data races and deadlocks, since they have a stochastic nature of manifestation during testing, which greatly complicates their identification. The presented model describes the state of each thread of a multi-threaded computer program currently executing a function and the contents of the function call stack. At the same time, it remains possible to use the model in verification by the Model Checking method, and also eliminates the need to solve the problem of searching for the model invariant. The presented procedures make it possible to formulate specifications for the verification method on models, the implementation of which makes it possible to identify vulnerabilities leading to data races and deadlocks in the source code of multithreaded programs.

Keywords: *verification, mathematical modeling, vulnerability identification, data race, deadlock.*

Многопоточные компьютерные программы могут содержать в себе уязвимости, приводящие к отказам, имеющим случайный характер обнаружения при тестировании (скрытым отказам [1]). В данной статье такие уязвимости называются скрытыми. Скрытые уязвимости приводят к ситуациям «гонок» и взаимоблокировкам, которые в международной базе уязвимостей Common Weakness Enumeration имеют идентификаторы CWE-362 и CWE-833 соответственно.

Причиной возникновения уязвимостей, приводящих к ситуациям «гонок», является одновременный доступ различных потоков к одним и тем же функциям исходного кода без применения механизмов обеспечения монопольного доступа. Причиной возникновения уязвимостей, приводящих к взаимоблокировкам, является неверная организация доступа потоков к функциям исходного кода, в которых используются механизмы обеспечения монопольного доступа [2, 3].

Наиболее эффективным методом обнаружения скрытых уязвимостей является верификация проверкой на модели (Model Checking) [4, 5, 6], поскольку позволяет описывать свойства программы посредством темпоральной логики и проверять наличие описанных свойств без реального исполнения программы. Но при этом верификация многопоточной программы требует значительных временных затрат.

В целях уменьшения времени выполнения верификации была разработана теоретико-множественная модель, названная моделью функциональных переходов. Под функциональными переходами в данной статье понимается передача управления между функциями исходного кода. Состояние потока многопоточной программы в модели функциональных переходов описывается исполняемой в текущее время функцией и содержимым стека вызовов функций. Под переходом между состояниями понимается передача управления от одной функции к другой. Состояния и переходы между ними описываются отдельно для каждого потока.

В общем виде модель на основе функциональных переходов M_p многопоточной программы описывается выражением

$M_p = (M^i_t, Tb)$, $i = \overline{0, K_t - 1}$,

где M^i_t – модель на основе функциональных переходов i-го потока, *Tb* – множество метаданных о множестве *I* наборов инструкций исходного кода, K_t – количество потоков в программе при минимальном количестве потоков, исполняющих одинаковые наборы инструкций исходного кода (однородных потоков).

Модель M^i_t включает в себя:

– множество состояний i-го потока программы $S^i \subset S$, где *S* – множество, состоящее из множеств состояний каждого потока, S = {Sj}, $j = \overline{0, K_t - 1}$;

– множество переходов между состояниями i-го потока программы $R^i \subset R$, где *R* = {*Rj*}, $j = \overline{0, K_t - 1}$ – множество, состоящее из множеств переходов между состояниями каждого потока;

– множество меток во множестве наборов инструкций, исполняемых i-м потоком, $L^i \subset L$, где *L* = {*Lj*}, $j = \overline{0, K_t - 1}$ – множество, состоящее из множеств меток во множестве наборов инструкций, исполняемых каждым потоком.

В общем виде M^i_t описывается кортежем

$$M^i_t = (S^i, R^i, L^i). \qquad (1)$$

Множество меток Li во множестве l^i в (1) имеет вид $L^i = \{l^i_j\}$, $j = \overline{0, K^i_L - 1}$, где l^i_j – некоторая метка во множестве L^i, $K^i_L = |L^i|$ – количество меток во множестве l_i, причем для $\forall$ j, p выполняется $l_j \neq l_p$ при $j \neq p$, $l_j \in L$, $l_p \in L$.

Множество состояний S^i в (1) i-го потока программы имеет вид $S^i = \{S^i_p\}$, $p = \overline{0, K^i_s - 1}$, где S^i_p – некоторое состояние *i*-го потока программы, $K^i_s = |S^i|$.

Состояние Sip описывается набором меток:

$$S^i_p = \bigwedge_{r=0}^{K_L - 1} l^i_r,\ l^i_r \in L^i,$$

где K_L – количество меток. Здесь символ $\wedge$ означает одновременное наличие элементов в заданном порядке.

Начальное состояние i-го потока, которое обозначается Si0, соответствует пустому стеку вызовов функций, то есть не содержит ни одной метки из Li.

При этом множество состояний Si обладает следующим свойством наличия бесконечных путей: для $\forall p, r\ \exists \pi(S^i_p, S^i_r)$ при $p \neq r$, где $\pi(S^i_p, S^i_r)$ – путь из состояния S^i_p в состояние S^i_r. Под путем из S^i_p в S^i_r понимается последовательность состояний, начинающаяся в S^i_p и заканчивающаяся в S^i_r: $\pi(S^i_p, Sir) = (S^i_p, \ldots, S^i_r)$, причем на протяжении всего пути существует переход из S^i_t в $S^i_t + 1$ для $\forall t = \overline{p, r-1}$. Выполнение данного свойства позволяет использовать модель функциональных переходов для верификации программы методом проверки на модели [4, 5, 6].

Множество переходов между состояниями R^i в (1) i-го потока имеет вид $R^i = \{R^i_j\}$, $j = \overline{0, K^i_R - 1}$, где $R^i_j = (S^i_p, S^i_r)$ – переход из состояния S^i_p в состояние S^i_r, $p \neq r$, $K^i_R = |R^i|$ – количество переходов между состояниями *i*-го потока. Множество R^i в начальный момент времени объявляется пустым.

Множество метаданных имеет вид *Tb* = {Tb^i}, $i = \overline{0, K_t - 1}$, где Tb^i – множество метаданных о множестве I^i.

Множество метаданных о множестве Ii имеет вид $Tb^i = \left\{Tb^{f^i_j}\right\}$, $j = \overline{0, K^i_f - 1}$, где $Tb^{f^i_j}$ – кортеж метаданных о функции f^i_j.

Подробное описание модели на основе функциональных переходов и алгоритмов ее построения представлены в статье [7].

Разработанная модель многопоточной программы содержит много меньше состояний, в сравнении с моделью Крипке, а также позволяет определять пути исполнения до уязвимостей тестирования программы, в отличие от модели с динамической семантикой [8].

Процедура обнаружения уязвимостей, приводящих к ситуациям «гонок», позволяет выявить количество E^0 уязвимостей посредством обработки множества *Tb* метаданных о множестве *I* наборов инструкций исходного кода многопоточной программы. Определяются уязвимости, возникающие при организации обращения к критическим секциям как однородных потоков, так и потоков, исполняющих различные наборы инструкция исходного кода.

Процедура обнаружения уязвимостей, приводящих к ситуациям «гонок», представляет собой следующую последовательность действий:

1) преобразовать множество *Tb* в *Tb′* за счет объединения элементов, у которых различается только компонента *th*;

2) найти количество E^0_0 уязвимостей, возникающих при организации доступа разнородных потоков к критическим секциям по множеству *Tb′* – поиск элементов множества *Tb′*, используемых несколькими потоками при отсутствии механизмов обеспечения монопольного доступа;

3) найти количество E^0_1 уязвимостей, возникающих при организации доступа однородных потоков к критическим секциям по множеству *Tb'* – поиск элементов множества Tb', используемых одним потоком, не являющимся главным или родительским для текущего элемента, при отсутствии механизмов обеспечения монопольного доступа.

Общее количество уязвимостей в проверяемой программе, способных привести к ситуациям «гонок», $E^0 = E^0_0 + E^0_1$.

Процедура обнаружения уязвимостей, приводящих к взаимоблокировкам, позволяет определить количество E^1 уязвимостей посредством обработки построенной модели M_p многопоточной программы на основе функциональных переходов. Определяются блокировки потоком самого себя, взаимоблокировки потоков, исполняющих как одинаковые, так и различные наборы инструкций, а также ложные обнаружения уязвимостей.

Процедура обнаружения уязвимостей, приводящих к взаимоблокировкам, представляет собой следующую последовательность действий:

– сформировать множества $Tb^i \subset Tb$ метаданных о функциях, исполняемых каждым из потоков;

– сформировать множества данных об используемых средствах обеспечения монопольного доступа и соответствующих им метках для каждого из потоков;

– обнаружить взаимоблокировки:

1) построить и проверить спецификации, позволяющие определить каждом из потоков наличие путей исполнения, на которых начинает использоваться средство обеспечения монопольного доступа $prot_2$ во время использования другого средства обеспечения монопольного доступа $prot_1$ ($prot_1 \longrightarrow prot_2$);

2) для каждого потока, соответствующего хотя бы одной из ранее построенных спецификаций, построить и проверить новые спецификации, позволяющие определить наличие пути, содержащего ($prot_2 \longrightarrow prot_1$);

3) для каждого потока, соответствующего хотя бы одной спецификации из шага 2, выполнить проверку ложного обнаружения взаимоблокировки, то есть построить и проверить спецификации, позволяющие определить отсутствие путей ($prot_3 \longrightarrow prot_1 \longrightarrow prot_2$) и ($prot_3 \longrightarrow prot_2 \longrightarrow prot_1$); в случае одновременного существования таких путей обнаруженная взаимоблокировка является ложной.

Количество состояний по модели на основе функциональных переходов N и модели Крипке N' было найдено для ряда многопоточных программ. Результаты представлены в таблице 1. При нахождении количества состояний в модели Крипке изменение значений параметров цикла с неизвестным числом итераций было ограничено до 103 изменений вместо всего диапазона изменения значений.

Таблица 1

Количество состояний по модели на основе функциональных переходов и модели Крипке

Кол-во строк	*N*	*N'*
6921	1724	1,4 * 107
9417	2520	1,9 * 107
12014	3207	2,4 * 107
21569	4832	4,3 * 107
35720	9418	7,1 * 107
42548	10159	8,5 * 107
50071	12367	108
61593	14216	1,2 * 108
72553	18925	1,5 * 108
91326	22311	1,8 * 108

На рисунке 1 представлена динамика изменения количества состояний от числа строк кода модели Крипке (верхний график) и модели на основе функциональных переходов (нижний график) в одной из проверяемых программ с ограничением до 104 значений параметров цикла с неизвестным числом итераций. Из рисунка 1 видно, что для одного и того же исходного кода программы количество состояний в модели на основе функциональных переходов много меньше, чем в модели Крипке.

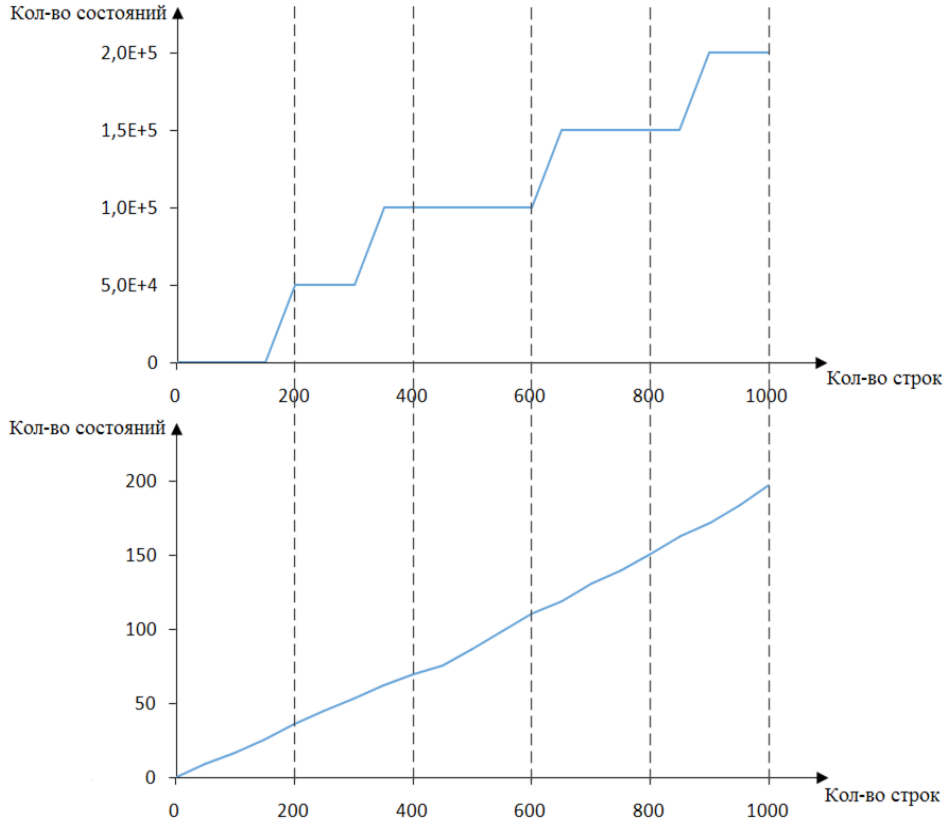

Рис. 1. Динамика изменения количества состояний от числа строк кода в модели Крипке (верхний график) и модели на основе функциональных переходов (нижний график)

Применение представленных в статье процедур позволяет более чем в 110 раз снизить временные затраты на выявление скрытых уязвимостей в сравнении с верификацией по модели Крипке.

Разработанные процедуры снижают временные затраты на обнаружение скрытых уязвимостей, поскольку: модель на основе функциональных переходов содержит меньшее число состояний, чем модель Крипке; процедуры исключают решение задачи поиска инварианта модели [2, 3, 4]; позволяют находить пути исполнения программы до уязвимости без тестирования, в отличие от верификации модели на основе динамической семантики.

Представленные в статье модель и процедуры предоставляют возможность разработки средства автоматического обнаружения скрытых уязвимостей многопоточных компьютерных программ для некоторого языка программирования высокого уровня, что имеет практическую ценность при отладке, а также оценке качества программ во время проведения сертификации.

Литература

1. ГОСТ 27.002-2015 Надежность в технике (ССНТ). Термины и определения. Дата введения 2017-03-01.

2. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. Пер. с англ. Слинкин А.А. – М.: ДМК Пресс, 2014. – 672 с.: ил.

3. Шлее М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 560 с. : ил.

4. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программы: Model Checking. Пер. с англ. / Под ред. Р. Смелянского. – М.: МЦНМО, 2002. – 416 с. : ил.

5. Кулямин В.В. Методы верификации программного обеспечения. М.: Институт Системного Программирования РАН, 2008. – 117 с.

6. Карпов Ю.А. MODEL CHECKING. Верификация параллельных и распределенных программных систем. – СПб.: БВХ-Петербург, 2010. – 560 с.: ил. + CD-ROM.

7. Моргунов Д.А., Букин А.Г. Математическая модель многопоточной программы ЭВМ, построенная на основе переходов между функциями исходного кода // Известия Института инженерной физики, 2018, №2 (48), – С. 50-55.

8. Handbook of Model Checking. / Clarke, Edmund M. (Editor); Henzinger, Thomas A.; Veith, Helmut; Bloem, Roderick (Editor). Cham : Springer, 2018. 1210 p. ISBN 978-3-319-10574-1.

References

1. GOST 27.002-2015 Nadezhnost' v tekhnike (SSNT). Terminy i opredeleniya. Data vvedeniya 2017-03-01.

2. Uil'yams E. Parallel'noe programmirovanie na S++ v deystvii. Praktika razrabotki mnogopotochnykh programm. Per. s angl. Slinkin A.A. – M.: DMK Press, 2014. – 672 s.: il.

3. Shlee M. Qt. Professional'noe programmirovanie. Razrabotka krossplatformennykh prilozheniy na C++. – Per. s angl. – SPb.: Simvol-Plyus, 2011. – 560 s., il.

4. Klark E.M., Gramberg O., Peled D. Verifikatsiya modeley programmy: Model Checking. Per. s angl. / Pod red. R. Smelyanskogo. – M.: MTsNMO, 2002. – 416 s.: il.

5. Kulyamin V.V. Metody verifikatsii programmnogo obespecheniya. M.: Institut Sistemnogo Programmirovaniya RAN, 2008. – 117 s.

6. Karpov Yu.A. MODEL CHECKING. Verifikatsiya parallel'nykh i raspredelennykh programmnykh sistem. – SPb.: BVKh-Peterburg, 2010. – 560 s.: il. + CD-ROM.

7. Morgunov D.A., Bukin A.G. Matematicheskaya model' mnogopotochnoy programmy EVM, postroennaya na osnove perekhodov mezhdu funktsiyami iskhodnogo koda // Izvestiya Instituta inzhenernoy fiziki, 2018, №2 (48), – S. 50-55.

8. Handbook of Model Checking. / Clarke, Edmund M. (Editor); Henzinger, Thomas A.; Veith, Helmut; Bloem, Roderick (Editor). Cham : Springer, 2018. 1210 p. ISBN 978-3-319-10574-1.

МОРГУНОВ Дмитрий Андреевич, начальник отдела, Межрегиональное общественное учреждение «Институт инженерной физики». 142210, Российская Федерация, Московская область, г. Серпухов, Большой Ударный пер., д. 1а. E-mail: mor.dmitrij@gmail.com

MORGUNOV Dmitriy, Head of Department, Interregional Public Institution "Institute of Engineering Physics". 142210, Russian Federation, Moscow region, Serpukhov, Bolshoy Udarny per., 1a. E-mail: mor.dmitrij@gmail.com