УДК 004.056.53

СИГНАТУРНЫЙ АНАЛИЗ ПРОГРАММНОГО КОДА

С. С. Харченко, Е. М. Давыдова, С. В. Тимченко

В статье рассматривается проблема проведения сигнатурного анализа программного кода. Обосновывается возможность создания программного средства, обеспечивающего хранение типовых сигнатур и позволяющее проводить анализ в автоматическом и автоматизированном режиме.

Ключевые слова: анализ, программный код, информационная безопасность, обработка данных.

Введение

Согласно статистике, на 1000 строк исходного кода приходится примерно от 20 до 50 ошибок, которые, как правило, носят непреднамеренный характер. Это такие ошибки как: недостатки проектирования, разработки, программирования и в большей степени тестирования, а так же ошибки связанные с недостатками используемого языка программирования, которые создают основную категорию уязвимостей программного обеспечения, а именно: ошибки разработки и тестирования. Наибольшую опасность из них представляют недекларированные возможности программного обеспечения, в свою очередь к этой категории уязвимостей относятся, например, инженерные «бэкдоры», которые позволяют обходить процедуры контроля целостности и/или аутентификации[1-3].

Анализ и выявление уязвимостей программного кода может быть основным или добавочным методом тестирования программного обеспечения, называемый тестированием безопасности. Наиболее распространено применение анализаторов программного кода для выявления уязвимостей и недекларированных возможностей программного обеспечения (ПО). На сегодняшний день основными способами анализа являются статический и динамический анализ исходного код [4,5].

Так же стоит отметить, что наибольшую опасность представляют уязвимости, заложенные (чаще допущенные из-за невнимательности разработчиков) в приложении на самых ранних этапах его жизненного цикла (рисунок 1) на этапе разработки, а именно уязвимые сигнатуры в исходных текстах тестируемого программного обеспечения. На практике большая часть программного обеспечения на рынке информационных технологий проходит лишь этапы внутреннего тестирования (альфа-тестирование), основной целью которого является выявление ошибок в разработанном ПО. При этом о тестировании безопасности программного обеспечения чаще всего не задумываются. Основная проблема заключается в том, что пока нет никакого российского стандарта или руководящего документа, в котором описывался бы процесс тестирования по требованиям безопасности, когда в Руководящем документе «Защита от несанкционированного доступа к информации. Часть1» приводятся только основные положения и требования к уровням контроля при проведении испытаний на отсутствие недекларированных возможностей в программных изделиях [6-8].

По характеру появления, как правило, выделяют объективные и субъективные причины уязвимостей программного кода. Наиболее часто появление уязвимостей в программном коде вызвано невнимательностью разработчиков и чрезвычайно высокой структурной сложностью программных систем. Также достаточно часто встречается такая ситуация, что разработчик, переключаясь на другую задачу, часто забывает вернуться к прежним проблемам или откладывает в долгий ящик, а обнаружив, что задача так и не была закрыта спустя продолжительное время, так и не возвращается к ней, считая, что она уже не актуальна [9].

Объективные причины:

- чрезвычайно высокая структурная сложность программных систем;
- динамичность развития информационных технологий, версий и реализаций программных ресурсов;
- относительная легкость модификации программного кода;

- сложность идентификации программной закладки как преднамеренно внесенной.

Субъективные причины:

- отсутствие прикладных реализаций математического аппарата испытаний ПО;
- невнимательность разработчиков;
- отсутствие ответственности за внесение уязвимостей в программный код.

Большинство изданий в сети Internet приводят примерно схожую статистику по появлению уязвимостей и недекларированных возможностей в программном коде.

Практически во всех приложениях служащих для обеспечения безопасности программного обеспечения база данных сигнатур является ядром продукта, наиболее трудоемкой и ценной частью. Именно поэтому большинство поставщиков таких программных продуктов предпочитает держать свои сигнатуры закрытыми. Хотя и в этой области существует ряд открытого ПО, например ClamAV, а также исследования по обратной разработке закрытых сигнатур. Virus Bulletin регулярно публиковал сигнатуры новых вирусов вплоть до 2000 года [10].

Проблемы, связанные с разработкой сигнатур уязвимостей и слабого распределения приводят к тому, что разработанные и использующиеся в разных компаниях сигнатуры содержат определения для разных уязвимостей. Эту проблему можно решить, разработав стандарт на описание сигнатур, который будет содержать необходимую информацию для обнаружения, идентификации и устранения вредоносного ПО. Максимальный эффект от такого стандарта можно получить при реализации централизованной базы знаний уязвимостей программного кода, в которую все компании и исследовательские лаборатории будут иметь право записи новых сигнатур (или изменение существующих). Но в таком централизованном хранилище (базе данных) каждая запись должна иметь цифровую подпись автора (конкретной компании), которая будет защищать конечных пользователей от атаки на локальную базу сигнатур путем отравления её содержимого. Так же при добавлении новых записей в центральное хранилище сигнатур должен изменяться рейтинг фирм производителей. Этот рейтинг сделает фирмы заинтересованными, так как их работу пользователь может напрямую оценивать. А пользователем в данном случае являются разработчики программных продуктов разных уровней [11].

При правильном подходе сигнатурный анализ идеально подходит для решения многих проблем, связанных поиском НДВ, не хуже чем сигнатурно-эвристический анализ. Остается только создать базу сигнатур уязвимостей программного кода, и решить проблемы связанные с недостатками сигнатурного анализа.

Перенося данное решение на проблему выявления уязвимостей, было предложено создание web-ресурса, на котором каждый желающий мог бы зарегистрироваться и добавлять сигнатуры уязвимостей. Чтобы исключить наплыв «ботов», зарегистрированному пользователю будет необходимо пройти тестирование по соответствующей области знаний и регулярно подтверждать свои навыки и умения. Данное решение позволит собрать самую актуальную и полную базу сигнатур уязвимостей того или иного языка программирования.

Для проверки жизнеспособности данного подхода в рамках это концепции было создано приложение, реализующее сигнатурный анализ исходного кода тестируемого приложения. А так же web-ресурс для создания актуальной базы сигнатур.

Главная идея web-ресурса заключается в следующем: использование ресурса возможно только в том случае, если вы внесли вклад в его развитие; то есть добавили хотя бы одну актуальную сигнатуру, а добавление сигнатур возможно только в том случае, если вы являетесь специалистом в данной области или же имеете опыт проектирования и программирования систем на языке программирования высокого уровня.

Таким образом, следуя двум вышеупомянутым правилам можно определить алгоритм работы с web-ресурсом:

1. регистрация;
2. подтверждение регистрации;
3. прохождение тестирования;
4. добавление сигнатур;
5. использование ресурса.

Также было добавлено еще пару важных правил:

- свою квалификацию каждый их участников, зарегистрированных на web-ресурсе, должен будет периодически подтверждать;

- в зависимости от опыта пользователя добавленные им сигнатуры будут обрабатываться по-разному.

Окончательный «цикл жизни» пользователя на сайте будет иметь следующие этапы: Регистрация -> Подтверждение регистрации -> Прохождение тестирования -> Определение квалификации -> Возможность добавления сигнатур -> Использование ресурса -> Определение квалификации ->… ->… -> и т.д. (рисунок 1)

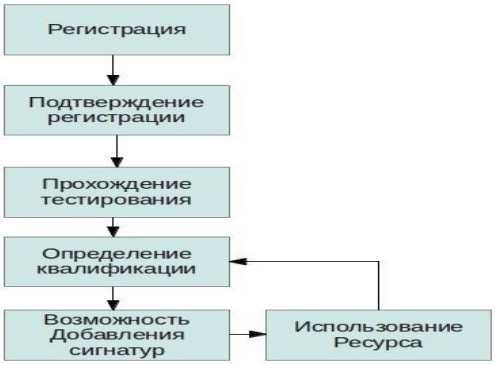

Рисунок 1 – Жизненный цикл пользователя web-ресурса

Определим ранжирование пользователей и правила обработки добавленных ими сигнатур в зависимости от их квалификации. Всех пользователей после прохождения тестирование планируется делить на 4 типа:

- профессиональные программисты (ПП), занимающиеся проектированием, составлением, отладкой, тестированием, документированием и модификацией программ, написанных на языках высокого уровня или Ассемблере;
- системные программисты (СП), работающие над операционными системами, компиляторами и утилитами, которые используются прикладными программами для решения пользовательских задач;
- программисты «от случая к случаю» (ПСС), которые пишут программы для научных исследований, инженерных разработок, обследований рынка, коммерческих приложений и т.д.;
- программисты-любители (ПЛ), пишущие небольшие программы коммерческого назначения для личных или домашних расчетов.

Роли и права пользователей в зависимости от их квалификации не будут отличаться кроме как способом обработки добавленных ими сигнатур. Также введем уровни доверия к пользователям:

- Р1: наивысший уровень доверия, пользователь, с уровнем доверия Р1, имеет возможность самостоятельно добавлять сигнатуры, его репутация в рамках web-ресурса неприкосновенна;
- Р2: достаточно высокий уровень доверия, достаточный для того, чтобы сигнатуры, добавленные пользователем с таким уровнем доверия, не проверялись, а лишь оставалась необходимость ознакомления с кратким описанием уязвимости. По умолчанию уровень доверия Р2 будет присваиваться пользователем с квалификацией ПП;
- Р3: уровень доверия, при котором достаточно поверхностное ознакомление с сигнатурами, добавленными пользователями с таким уровнем доверия. По умолчанию будет присвоен пользователем с квалификацией СП и ППС;
- Р4: низший уровень доверия, говорит не о том, что к пользователю нет доверия, а о том, что пользователю есть куда расти и к чему стремиться. Будет присвоен всем пользователям с квалификацией ПЛ.

Как было показано выше, в зависимости от уровня доверия к пользователю, добавленные им сигнатуры будут обрабатываться по-разному, то есть напрямую квалификация не будет влиять на обработку сигнатур. Более подробно в таблицах 1-2.

Таблица 1 - Обработка добавленных сигнатур в зависимости от уровня доверия

Способ обработки сигнатур	Уровень доверия
Автоматическое добавление в БД сигнатур	Р1
Ознакомление с кратким описанием уязвимости	Р2
Поверхностное ознакомление с сигнатурами	Р3
Голосование на добавление сигнатуры пользователями с уровнем доверия не ниже Р3	Р4

Таблица 2 – Зависимость квалификации и уровня доверия

Квалификация пользователя	Уровень доверия по умолчанию	Возможные уровни доверия
ПП	Р3	Р3, Р4
СП	Р3	Р3
ПСС	Р2	Р2, Р3
ПЛ	Р1	Р1, Р2

Стоит отметить, что при повторном тестировании пользователь получает такой же уровень доверия, какой у него был и ранее, если он подтверждает свою квалификацию. При результатах, соответствующих более высокой квалификации, присваивается наименьший уровень доверия для данной квалификации, но не меньше, чем был раньше. При понижении квалификации по результатам тестирования, также теряется уровень доверия до наименьшего из возможных (таблица 2). Теперь можно рассмотреть каждый их этапов «жизненного цикла» пользователей более подробно.

Во время работы приложения сигнатуры хранятся в оперативной памяти, после завершения работы приложения сигнатуры удаляются из оперативной памяти компьютера, на котором было запущено приложение. Каждая из сигнатур в себе несет следующую информацию:

- уникальный идентификатор;
- название (краткое описание);
- цвет, характеризующий степень опасности сигнатуры (всего 5 цветов);
- язык программирования, для которого характерна уязвимость описываемая сигнатурой;
- имя пользователя загрузившего сигнатуру в базу данных;
- дата добавления сигнатуры в базу данных;
- подробное описание последствий которые может повлечь собой неправильно написанный код;
- паттерн регулярного выражения, по которому идентифицируется сигнатура.

Для запуска приложения необходимо наличие самого исполняемого файла «TortoiseSHCA.exe», а также файла настроек «TortoiseSHCASettings.txt», расположенного в по пути «/Settings/» относительно исполняемого файла «Tortoise SHCA.exe». В файле настроек находиться следующая информация:

- proxyServer – прокси-сервер через который необходимо подключаться, если нет другой возможности;
- numberThreads – количество потоков, больше единицы если имеет смысл в нескольких потоках скачивать сигнатуры с удаленного узла;
- login – имя пользователя на сайте, можно оставить пустым и задать в ходе работы приложения;
- pass – пароль пользователя в виде хэш-суммы;
- Host – адрес ресурса к которому необходимо подключаться;
- Register_User – служебная информация, заполняемая программой, характеризующая зарегистрирован ли активный пользователь на сайте или нет.

Приложение тестировалось с целью определения эффективности разработанного метода анализа исходного кода. Цель тестирования состояла в том, чтобы определить, целесообразно ли использование подобных подходов проверки исходных текстов, а так же насколько эффективно происходит взаимодействие между компонентами программы (например, между серверной частью и модулем анализа клиентской части) и информационный обмен. В процессе тестирования фиксировалось число выявленных уязвимостей, количество строк исходного кода, а так же время потраченное на анализ исходного текста. Работа алгоритма считалась эффективной, если было найдено больше уязвимостей (по сравнению с «ручным» анализом) за наименьшее время (в зависимости от числа строк исходного текста).

Результаты тестирования зависят от «начальных» данных, а именно: полноты и актуальности базы данных уязвимостей. Поэтому, при актуальном состоянии базы данных, качество исполнения анализа на наличие и/или отсутствие уязвимостей зависит только от состояния исходного кода и его полноты, но не зависит от его объема.

После того как «начальные» сигнатуры были загружены в программу, запускается алгоритм анализа и проверки исходного кода, нажатием на кнопку «Анализ» (или нажатия клавиши F5). По окончании проверки, в итоговый отчет заносилось количество найденных уязвимостей. [12]

Результаты тестирования приведены в таблицах 3,4.

Таблица 3 – Количество найденных уязвимостей (в сравнении с «ручным» анализом)

Программный анализ	«Ручной» анализ	Количество строк исходного кода
2	0	447
5	5	334
6	5	177
2	2	45
2	1	450
14	8	86
2	0	298
3	1	1078
8	2	1797

Время, затрачиваемое на анализ исходного кода (в зависимости от количества строк исходного кода) приведено в таблице 4.

Таблица 4 – Время затрачиваемое на проверку исходного кода

№	Количество строк кода	Время анализа, сек.
1	45	0,5
2	86	0,6
3	177	0,7
4	298	1
5	334	1,2
6	450	1,9
7	1078	4,2
8	1797	7,4
9	2846	11,2

Выводы

В результате тестирования программы было установлено, что время исполнения анализа исходного кода зависит от количества строк исходного кода и увеличивается с увеличением объема исходного текста. Это подтверждают данные из таблицы 4. Таким образом (на основании таблиц 3,4), можно сказать, что реализованный метод анализа исходного кода полностью реализует функции по анализу и проверке и может считаться эффективным.

Так же следует отметить, что производительность системы на прямую зависит от аппаратно-программного обеспечения конечного пользователя.

СПИСОК ЛИТЕРАТУРЫ

1. Калайда И.А. Недекларированные возможности в программном обеспечении средств защиты информации. [Текст] /И.А. Клайда/ - Jet Info, 2000, № 8.
2. Макконелл С. – Совершенный код. Мастер класс. [Текст] / С. Макконел/ Пер. с англ. – М.: издательство «Русская редакция», 2010. – 896 стр.: ил.
3. Carl Hewitt. Towards Open Information Systems Semantics Journal of Artificial Intelligence. March 1997.
4. Carl Hewitt. Open Information Systems Semantics Journal of Artificial Intelligence. January 1991.
5. Christopher Alexander- Pattern language, 1977.
6. Дастин Э., Рэшке Д., Пол Д. Автоматизированное тестирование программного обеспечения: внедрение, управление, эксплуатация. [Текст] /Э.Дастин., Д.Рэшке, Д. Пол/ - М.: Лори, 2003. – 567 стр.
7. Методическое руководство по оценке качества функционирования информационных систем. М.: Изд-во 3 ЦНИИ МО РФ, 2003.
8. Хогланд Г., Мак-Гроу Г. Взлом программного обеспечения: анализ и использование кода. [Текст] / Г. Хогланд, Г. Мак-Гроу/ - М.: Вильямс, 2005. – 396 стр.:ил.
9. Ховард М., Лебланк Д., Виега Д. – Как писать безопасный код на C++, Java, Perl, PHP, ASP.NET. [Текст] / М. Хорвард, Д. Лебланк, Д. Виега/ – М.: ДМК-Пресс, 2009. – 288 стр.: ил.
10. «Testing Malware Detectors» [Электронный ресурс]. - Режим доступа: http://research.cs.wisc.edu/wisa/papers/issta04/issta.pdf
11. Обнаружение, основанное на сигнатурах, 2011 г. [Электронный ресурс] – Режим доступа: http://softarxiv.ru/signatura.html
12. Хабибулина Н.Ю., Мещеряков Р.В. Генератор индивидуальных заданий автоматизированной обучающей системы “Линейное программирование” [Текст] /Н.Ю. Хабибулина, Р.В. Мещеряков/ Промышленные АСУ и контроллеры. - N3. 2012г., - С.12-17.

***Харченко С.С.**, аспирант каф. КИБЭВС ФГБОУ ВПО ТУСУР. **Давыдова Е.М.**, к.т.н., доцент каф. КИБЭВС ФГБОУ ВПО ТУСУР. **Тимченко С.В.**, д.ф.-м.н., профессор, зав. каф. ПМиИ ФГБОУ ВПО ТУСУР.*