Козлов Д. Д., Петухов А. А.

Методы обнаружения уязвимостей в web-приложениях

Введение

В современных информационных системах широкое распространение получило использование web-приложений. Web-приложения обладают такими важными достоинствами, как простота и привычность интерфейса, возможность удаленной работы через Интернет, быстрота разработки приложения. Вместе с этим web-приложения создают большое число проблем, связанных с обеспечением информационной безопасности, ведь их разработка часто выполняется в сжатые сроки, а приложение становится доступным через Интернет и для пользователей, и для злоумышленников. Уязвимости позволяют злоумышленникам похищать конфиденциальную информацию, проводить несанкционированные изменения данных, нарушать доступность приложения. В настоящее время проблема обеспечения безопасности web-приложений весьма актуальна: согласно [1] более 60% от всех обнаруживаемых уязвимостей относятся к web-приложениям.

Одним из широко распространенных методов обеспечения безопасности web-приложений является обнаружение уязвимостей web-приложения с целью последующего их устранения. В данной работе рассмотрены современные методы обнаружения уязвимостей в web-приложениях и проведен анализ их возможностей.

1. Методы обнаружения уязвимостей web-приложений

На сегодняшний день согласно исследованиям OWASP [8] наиболее эффективным способом обнаружения уязвимостей web-приложений является экспертный анализ исходных кодов web-приложения (code review). Этот способ весьма трудоемок, требует высокой квалификации эксперта и не защищен от ошибок эксперта. Поэтому активно развиваются методы автоматического обнаружения уязвимостей web-приложений.

Методы автоматического обнаружения уязвимостей web-приложений можно разделить на две основные группы: 1) методы, анализирующие работу развернутого на стенде web-приложения без обращения к исходным кодам web-приложения; 2) методы, анализирующие исходные коды web-приложения и конфигурационные настройки.

Первая группа методов рассматривает web-приложение с точки зрения внешнего пользователя, то есть потенциального злоумышленника. В эту группу входят следующие методы:

- Метод получения идентифицирующей информации о web-приложении и выявления его уязвимостей с помощью бюллетеней безопасности (security advisory).
- Метод тестирования на проникновение.

Вторая группа состоит из следующих методов:

- Метод статического анализа исходных кодов web-приложения.
- Метод динамического анализа исходных кодов web-приложения.

Для разработки web-приложений применяются разнообразные инструментальные средства и языки программирования. Наиболее популярными среди них являются PHP, Perl, Python, Ruby, Java/JSP, .Net/ASP. Возможности, предоставляемые этими средствами, существенно различаются, ограничивая применение тех или иных методов анализа. В данной работе рассмотрены методы автоматического обнаружения уязвимостей и описаны области применимости и ограничения каждого из методов.

1.1 Метод получения идентифицирующей информации о web-приложении

Метод получения идентифицирующей информации о web-приложении основан на посылке от имени обычного пользователя web-приложению набора HTTP-запросов, ответы на которые позволят сделать вывод о том, на каком web-сервере работает приложение, с помощью какой технологии оно разработано, какие версии программного обеспечения использует, какие стандартные компоненты

оно использует и т.д. Для определения типа и версии web-сервера используется техника fingerprint [2], которая основана на том, что каждый web-сервер по-своему обрабатывает HTTP-протокол, в результате чего можно с высокой степенью вероятности определить тип и даже версию web-сервера путем посылки серверу набора корректных и некорректных запросов и анализа соответствующих ответов. Для определения остальных параметров используется анализ HTTP-ответов и HTML-страниц средствами поиска регулярных выражений. Шаблоны для поиска задаются экспертом. Например, использование расширения .jsp в URI может свидетельствовать о том, что web-приложение было разработано с использованием технологии JSP.

Возможность получения идентифицирующей информации пользователем приложения является потенциальной уязвимостью ввиду следующих причин: в сети Интернет накоплены огромные объёмы данных по уязвимостям программных продуктов с указанием версий программ, методов реализации атак, бюллетени безопасности Bugtraq [9] и Security Focus [10] ежедневно публикуют отчёты о новых найденных уязвимостях. Эта информация предназначена для администраторов и специалистов в области информационной безопасности, в тоже время она доступна через сеть Интернет всем желающим. Администраторы информационных систем часто не устанавливают обновления безопасности вовремя. В результате становится возможным взломать информационную систему путем использования всем известной уязвимости, которую администратор не закрыл установкой соответствующего обновления. В последние годы на хорошо известные и незакрытые уязвимости операционных систем было проведено большое количество атак, вызвавших широко известные эпидемии Интернет-червей.

Метод получения идентифицирующей информации о web-приложении и выявления его уязвимостей с помощью бюллетеней безопасности получил широкое практическое применение при проведении атак ввиду своей простоты и доступности. При этом сам метод не позволяет найти новые уязвимости web-приложения, а его полезность для обнаружения уязвимостей новых приложений состоит в том, что он позволяет указать на саму возможность утечки информации о web-приложении.

Данный метод требует от эксперта настройки шаблонов, по которым производится поиск идентифицирующей информации. Эти шаблоны в большинстве случаев специфичны для конкретной технологии разработки web-приложений.

1.2 Метод тестирования на проникновение

Метод тестирования на проникновение (penetration testing) рассматривает web-приложение с точки зрения внешнего пользователя, то есть потенциального злоумышленника. При этом считается, что злоумышленник обладает такими же возможностями, как и обычный пользователь, т.е. не имеет доступа к исходным кодам web-приложения, конфигурационным настройкам и т.п. Метод предусматривает тестирование работающего на стенде web-приложения путем посылки запросов которые эмулируют пользовательскую активность, включающую в том числе и некорректные запросы, соответствующие действиям злоумышленника.

При поиске уязвимостей в web-приложении методом тестирования на проникновение возникают три основные задачи [3]:

- Получение и анализ структуры web-приложения.
- Построение набора тестовых HTTP-запросов на основе построенной структуры web-приложения.
- Прогон тестового набора с анализом ответов web-приложения для выявления уязвимостей.

Задача получения и анализа структуры web-приложения состоит в том, чтобы построить полный список URI web-приложения, методов доступа к ним и списков их параметров, выделить URI, защищённые аутентификацией. Данная информация необходима для построения набора тестовых запросов к web-приложению. Для автоматического получения структуры web-приложения используются сетевые роботы (web crawlers) [5]. В случае статических HTML-страниц работа робота сводится к обходу всех доступных ссылок web-приложения, а в случае наличия форм и скриптов – к заполнению форм и извлечению гиперссылок из скриптов. Получение полной структуры web-приложения возможно не во всех случаях – основные проблемы возникают при заполнении web-форм и интерпретации скриптов [4, 5].

Web-формы делятся на две категории: формы, содержащие только элементы с ограниченными областями значений (select, option и т.д.) и формы, содержащие элементы с неограниченными областями значений (например, textarea). Обход страниц, связанных с посылкой форм первого типа осуществляется последовательным перебором всех комбинаций значений полей формы. Для переходов, связанных с заполнением текстовых полей, возможны два способа обхода: ручной или автоматизированный. При ручном способе управление процессом передаётся человеку, а при автоматизированном способе применяются эвристические методы заполнения текстовых полей, основанные на словарях, например, VeriWeb [4], HiWE [5]. Для форм, содержащих элементы с неограниченной областью значений, автоматизированный обход не гарантирует нахождения всех URI приложения.

Для автоматического обхода страниц, содержащих ссылки, формируемые средствами скриптовых языков, необходима возможность не только интерпретации скриптового языка, но и эмуляции действий пользователей для формирования тех или иных событий, например, onClick, onMove. Существующие средства интерпретации скриптовых языков не гарантируют выявления всех гиперссылок.

Задача построения тестового набора запросов к web-приложению состоит в том, чтобы по исходным данным (список URI приложения, методы доступа, принимаемые параметры) подобрать запросы так, чтобы было обнаружено как можно больше уязвимостей [7]. Существующие способы построения таких запросов рассмотрены ниже.

Построение запросов по базе ресурсов подразумевает, что существует база ресурсов, которые потенциально могут встретиться в структуре web-приложения. Наличие в web-приложении того или иного ресурса из базы свидетельствует об уязвимости, связанной с возможным доступом к этому ресурсу. Например, в web-приложении присутствуют имена известных уязвимых CGI-сценариев и имена конфигурационных файлов web-серверов. Поиск ресурсов происходит по всей структуре web-приложения. В каждом каталоге (из структуры web-приложения) по очереди запрашиваются все имена, представленные в базе ресурсов. Способ построение запросов по базе ресурсов полностью автоматический и опирается на накопленную информацию о

характерных уязвимостях web-приложений. Таким образом, данный способ имеет ограничения, аналогичные рассмотренным выше в методе получения идентифицирующей информации, но в тоже время позволяет выявить новые уязвимости web-приложения, основанные на типовых ошибках разработчиков и администратора.

Генерация запросов по шаблону с типизированными параметрами состоит в том, что для каждого URI задаётся шаблон, параметры которого типизированы, после чего происходит автоматическая генерация запросов по заданному шаблону со случайным выбором значений конкретных параметров. Значения параметров могут задаваться регулярными выражениями. Данный способ используется для обнаружения ошибок проверки корректности введенных пользователем данных. Критерий наличия уязвимости вводит пользователь – если ответ web-сервера соответствует некоторому заданному регулярному выражению, то делается вывод об уязвимости приложения. Описанным способом также реализуются переборы паролей. Данный способ требует привлечения эксперта и тонкой настройки, так как необходимо для каждого ресурса web-приложения составить наборы значений параметров. При этом данный способ не зависит от технологии, на которой разработано web-приложение, так как работает только в терминах протокола HTTP.

Анализ настроек каталогов web-приложения заключается в том, что по структуре web-приложения проверяются типовые уязвимости, связанные с неправильным конфигурированием web-приложения и web-сервера. Сюда относятся проверка возможности автоматического построения индекса каталога, выполнения HTTP-методов PUT и DELETE, возможность обращения к ресурсам из областей аутентификации напрямую, возможность получения исходных кодов web-приложения.

Задача прогона тестового набора и анализа ответов сервера состоит с том, чтобы сделать правильный вывод о том, демонстрирует ли данный HTTP-запрос наличие уязвимости в web-приложении или нет. Данная задача тесно связана с задачей построения тестового набора, а основная проблема состоит в определении критериев наличия уязвимости. В настоящее время для решения этой задачи применяется метод, в котором распознавание осуществляется регулярными

выражениями, задаваемыми экспертом. Таким образом, данный метод также требует тонкой настройки экспертом, при этом метод работает в терминах протокола HTTP и не зависит от технологии, на которой разработано web-приложение.

Метод тестирования на проникновение существенно меньше зависит от технологии разработки web-приложения, чем другие методы. Данный метод позволяет накапливать знания эксперта в виде набора правил построения запросов и набора шаблонов для анализа HTTP-ответов. Этот метод получил широкое распространение для выявления уязвимостей разработанных web-приложений, когда необходимо оценить наличие хотя бы типовых ошибок при разработке и настройке web-приложения. Достоинством данного метода является то, что метод позволяет оценивать развернутое и настроенное web-приложение, выявляя не только ошибки кодирования, но и ошибки конфигурирования web-сервера и web-приложения.

1.3 Метод статического анализа

Метод статического анализа исходных кодов web-приложения не предусматривает реального выполнения web-приложения. Вместо этого производится построение графов управления и зависимостей по данным и обнаружение уязвимостей посредством анализа этих графов. Для обнаружения уязвимостей используется два основных подхода: анализ типов безопасности [7] и анализ потоков данных [6]. В каждом из подходов уязвимость определяется, как нарушение в программе свойства noninterference [11].

При анализе типов безопасности вводится набор типов безопасности *(t1, t2, …, tn)*, над которыми вводится отношение частичного порядка ≤. Каждой переменной программы ставится в соответствие её тип безопасности. Существуют два способа определения типов безопасности всех переменных – ручной и автоматический. Ручной способ предполагает, что при каждом объявлении переменной программист должен явно задать её тип безопасности. Автоматический способ предполагает, что даны: разметка типами безопасности переменных, содержащих входные данные, разметка функций, осуществляющих пользовательский ввод правила вывода типов безопасности ещё неразмеченных переменных. Таким

образом, при анализе программы последовательно от начала до конца все неразмеченные переменные получают свой тип безопасности автоматически. Тип безопасности переменной постоянен на всё время жизни переменной.

Для выявления уязвимостей необходимо специфицировать, требуемые уровни безопасности параметров функций. В реализациях метода данная спецификация часто находится в отдельном конфигурационном файле и не зависит от конкретной программы (аннотируются стандартные библиотечные функции). Следующий пример иллюстрирует данный подход:

```
Типы безопасности: {untainted, tainted}
Начальная разметка: int main (int argc, tainted char *argv[])
Сама программа:
void LogMessage(char *fmt, …){
        …
        fprintf(fd, fmt, arg);
}

int main (int argc, tainted char *argv[]){
        char *arg2 = argv[2];
        …
        char *Mess = new [strlen(arg2) + strlen(“arg2 = ”) + 1];
        strcat(strcpy(Mess, “arg2 =”), arg2);
        LogMessage(Mess);
}
```

Программа имеет уязвимость форматной строки. Методом анализа уровней безопасности данная уязвимость будет обнаружена, если функция *fprintf* будет аннотирована, как принимающая второй аргумент только уровня безопасности *untainted*.

В работе [12] было доказано, что в языках со строгой типизацией данный подход обнаруживает все уязвимости, связанные с некорректными потоками информации.

Анализ типов безопасности имеет существенный недостаток – постоянную привязку типа безопасности к переменной. В результате тип безопасности определяется без учета того, из какого источника данные попали в переменную, что приводит к большому числу ошибок

первого рода. Для того чтобы избежать большого числа ошибок первого рода, было предложено привязывать тип безопасности не к переменной, а к её значению. В результате каждая переменная теоретически может получать любой тип безопасности на протяжении своей жизни. Этот подход получил название анализа потоков данных.

В рамках анализа потоков данных каждой конструкции языка программирования сопоставляются формальные правила вывода результирующего типа безопасности переменных, участвующих в этой конструкции. А для библиотечных функций создается разметка, описывающая типы безопасности параметров и результата функции. На основе разметки и формальных правил вывода анализатор определяет в каждом участке кода тип безопасности конкретных данных. Сопоставляя аннотации библиотечных функций и типы безопасности их параметров, можно выявить несоответствия типа безопасности требованиям разметки, что указывает на наличие уязвимости в web-приложении.

В рамках данного подхода для части конструкций программы программист сам должен указывать тип безопасности переменных, например, при необходимости повысить тип безопасности после проверки корректности введенных данных.

Обнаружение уязвимостей web-приложений методом статического анализа предусматривает решение следующих основных задач:

- Определение конструкций языка и библиотечных функций, которые возвращают данные, потенциально контролируемые злоумышленником. Например, это параметры HTTP запросов GET и POST, пользовательские cookie и т.д. Вся информация, полученная из таких конструкций, помечается меткой tainted.
- Разработка аксиом распространения метки tainted между переменными приложения. Данные аксиомы должны поддерживать не только тривиальные присваивания, но и более сложные конструкции языка, такие, например, как присваивания через массивы, работу через ссылки и т.д. Данная система аксиом должна содержать правила, по которым информация, помеченная как tainted, может вновь стать безопасной, untainted. Полнота и точность метода обнаружения

уязвимостей преимущественно зависит от качества модели распространения метки tainted по приложению.

- Определение конструкций языка, которые не должны принимать данные с типом безопасности tainted. В этот список должны входить функции для работы с системным окружением, с СУБД, с почтовыми сервисами и т.д.

Метод статического анализа исходных кодов web-приложения позволяет обнаруживать уязвимости, связанные с неустойчивостью web-приложения к некорректным входным данным. Другие классы уязвимостей данный метод обнаруживать не позволяет. Метод специфичен для каждой технологии создания web-приложений и требует от программиста аннотирования функций проверки корректности входных данных.

1.4 Метод динамического анализа

Метод динамического анализа исходных кодов web-приложения по своей сути аналогичен методу статического анализа исходных кодов за исключением того, что анализ производится в процессе выполнения web-приложения без построения графов программ. Вместо этого в процессе выполнения программы (т.е. неявно сформирован один из путей в графе управления) для каждой переменной производится вычисление типа безопасности, а для каждой размеченной функции сравниваются типы безопасности параметров со спецификацией, указанной в разметке.

Метод динамического анализа позволяет осуществлять обнаружение уязвимостей для широко применяемых при создании web-приложений динамических языков, как, например, Perl, PHP, Python, Ruby. В настоящее время для интерпретируемых языков web-приложений метод динамического анализа часто интегрируется в интерпретатор языка, например, Perl Tainted Mode для языка Perl, PHPrevent для языка PHP и Ruby's Tainted Mode для языка Ruby.

Основной проблемой метода динамического анализа является проблема снятия метки tainted с данных. Существует несколько способов решения данной проблемы: дополнительная разметка исходного кода, применение стандартных библиотеки «очищающих» функций из состава технологии, на которой разработано web-

приложение, применение различного рода эвристик. Так, в технологии Perl любое регулярное выражение над tainted данными снимает с них метку tainted. Таким образом, Perl полностью доверяет программисту в проведённых очистках входной информации. В технологии Ruby программист должен явно указать после своих проверок изменение типа безопасности на untainted.

Метод динамического анализа, в отличие от статического анализа, позволяет осуществлять обнаружение уязвимостей в генерируемом на лету коде. В тоже время, метод динамического анализа анализирует лишь один из возможных графов выполнения программы и не позволяет сделать вывод об отсутствии уязвимостей во всей программе. Метод динамического анализа, аналогично статическому анализу специфичен для каждой технологии создания web-приложений.

Заключение

Для разработки web-приложений применяются разнообразные инструментальные средства и языки программирования. В частности широко распространены динамические языки программирования, такие как PHP, Perl, Python. Применение метода статического анализа к динамическим языкам имеет существенный недостаток: в приложениях, где исходный код может генерироваться во время исполнения, метод статического анализа не может гарантировать обнаружения всех уязвимостей, связанных с нарушением свойства noninterference. Метод динамического анализа также не может гарантировать обнаружения всех уязвимостей, связанных с нарушением свойства noninterference, если тестовые наборы данных не покрывают все возможные пути выполнения программы. Метод тестирования на проникновение также не может гарантировать обнаружения всех уязвимостей. Таким образом, все рассмотренные методы обнаружения уязвимостей web-приложений не могут гарантировать обнаружения всех уязвимостей.

Как было указано выше, методы статического и динамического анализа позволяют обнаруживать только уязвимости, связанные с передачей web-приложению некорректных входных данных. В то время как метод тестирования на проникновение позволяет обнаруживать и другие уязвимости.

На основании проведенного анализа можно сделать вывод, что дальнейшее развитие методов автоматического обнаружения

уязвимостей web-приложений пойдет по пути интеграции возможностей различных методов с тем, чтобы было возможно

- охватить максимально широкий класс уязвимостей;
- контролировать полноту обнаружения уязвимостей, чтобы по итогам автоматического анализа можно было бы (в идеале) дать гарантию отсутствия уязвимостей заданных классов.

Литература

1. Andrews M., The State of Web Security. IEEE Security & Privacy, vol. 4, no. 4, pp. 14-15, 2006.
2. Shah S., “An Introduction to HTTP fingerprinting”, http://net-square.com/httprint/httprint_paper.html, 2004.
3. Auronen L., Tool-Based Approach to Assessing Web Application Security. Seminar on Network Security, 2002.
4. Benedikt M., Freire J., Godefroid P., VeriWeb: Automatically Testing Dynamic Web Sites. Proceesings of 11-th WWW Conference, 2002.
5. Raghavan S., Garcia-Molina H., Crawling the Hidden Web. Stanford University Technical Report, 2000.
6. Sabelfeld, A. Myers. Language-Based Information-Flow Security.
7. Yao-Wen Huang, Shih-Kun Huang Tsung-Po Lin Chung-Hung Tsai Web Application Security Assessment by Fault Injection and Behavior Monitoring. Proceedings of 12-th WWW Conference, 2003.
8. Curphey M., Wiesman A., Van der Stock A., Stirbei R. A Guide to Building Secure Web Applications and Web Services. OWASP, 2005.
9. Bugtraq. http://en.wikipedia.org/wiki/Bugtraq.
10. Security Focus. http://securityfocus.com.
11. J. A. Goguen and J. Meseguer Security Policies and Security Models. In Proc. 1982 IEEE Symposium on Security and Privacy, 1982.
12. D. Volpano, G. Smith A type-based approach to program security. In Proc. TAPSOFT'97, LNCS 1214, 1997.