МАТЕМАТИЧНЕ ТА КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ

MATHEMATICAL AND COMPUTER MODELLING

УДК 004.056.53

Коломыцев М. В.[1], Носок С. А.[1]

[1]Канд. техн. наук, доцент Национального технического университета Украины «Киевский политехнический институт»

УЯЗВИМОСТИ ПРИЛОЖЕНИЙ К НЕКОРРЕКТНЫМ ВХОДНЫМ ДАННЫМ

В данной статье рассматриваются различного рода уязвимости, присущие приложениям без достаточного контроля входных данных. Приводятся характерные признаки таких уязвимостей и даются рекомендации по их устранению.

Ключевые слова: уязвимость, приложения, корректные данные, некорректные данные.

ВВЕДЕНИЕ

Одной из основных причин уязвимостей программного обеспечения является неполная проверка входных данных либо ее полное отсутствие. Входные данные могут поступать от пользователей, из хранилища данных, сетевых сокетов и других источников. Любые данные, поступающие в приложение извне, перед их использованием должны проверяться для определения их корректности. Под корректностью данных в этом случае понимается их соответствие требованиям приложения – к формату, диапазону допустимых значений и т. п.

В процессе контроля приложение должно проверять, относятся ли данные к заведомо корректным, и отвергать их, если убедится, что это не так. Нельзя строить контроль входных данных на проверке того, относятся ли полученные данные к заведомо некорректным или потенциально опасным, поскольку атакующий может использовать различные приемы для того, чтобы обойти процедуру такой проверки. Он может по-иному представить фрагменты входных данных, либо использовать структуры данных, отсутствующие в образцах потенциально опасных.

Тот факт, что обрабатываться будут только заведомо корректные данные, может означать, что часть поступившей корректной информации будет отвергнута, как не прошедшая проверку, однако только так можно существенно снизить вероятность обработки некорректных данных.

Таким образом, разработчик обязан заложить в приложение проверку всех поступивших данных, как пользовательского ввода, так и данных, поступивших из других источников.

Анализу уязвимостей приложений вообще и рассматриваемого класса уязвимостей в частности посвящено много работ и информационных источников [1, 2].

Рассмотрим основные виды уязвимостей и способы их устранения.

ПОСТАНОВКА ЗАДАЧИ

Безопасность приложения предполагает, что данные, обрабатываемые, передаваемые и хранимые с помощью приложения, защищены от неавторизованного доступа. Угроза неавторизованного доступа возникает, если в приложении присутствуют определенные уязвимости. В данной статье ставится задача определения перечня уязвимостей приложений, обусловленных недостаточно полной проверкой данных, поступивших в приложение извне. Определяются характерные признаки, свидетельствующие о наличии таких уязвимостей, и предлагаются меры, направленные на их устранение.

ВНЕДРЕНИЕ КОДА В SQL ЗАПРОСЫ (SQL INJECTION)

Данная уязвимость позволяет модифицировать запросы к базе данных, с тем, чтобы получить несанкционированный доступ к данным. Данная уязвимость эксплуатируется через недостаточную проверку входных данных.

Следующие особенности кода приложения свидетельствуют о потенциальной возможности SQL injection:

– вводимые пользователем данные без проверки используются для построения запросов или как параметры хранимых процедур;

– при построении запросов используется конкатенация строк;

– при построении запросов используется замена строк;

– для выполнения запроса используется функция exec (в случае использования СУБД Microsoft SQL Server).

Основным методом обнаружения данной уязвимости является анализ (ревизия) исходных текстов программного обеспечения. Принципиально важно, чтобы все приложения, работающие с базами данных, были проанализированы на предмет наличия данной уязвимости.

Чтобы минимизировать вероятность возникновения данной уязвимости, следует придерживаться следующих правил:

– для проверки текстов программ использовать инструментальные средства анализа, обладающие низкой вероятностью ошибок 2-го рода для SQL injection (т. е. низкой вероятностью пропуска уязвимости в исходных кодах приложения). Существует широкий набор инструментальных средств для анализа исходного кода [3]. Например, Ounce Labs, Parasoft's Jtest & C++test, Klocwork Insight, Fortify Source Code Analyzer (SCA), GrammaTech CodeSonar, Coverity Prevent;

– проверять все данные, введенные пользователем, разрешая только заведомо корректные данные;

– использовать параметризированные запросы;

– не использовать конкатенацию или замену строк при построении кода запроса в тексте приложения;

– не использовать учетную запись администратора для подключения к БД;

– для доступа к данным использовать представления, непосредственный доступ к таблицам должен быть запрещен;

– для выполнения динамически созданного запроса использовать команду sp_executesql вместо exec (для СУБД Microsoft SQL Server);

– для проверки параметров хранимых процедур использовать функцию quotename (для СУБД Microsoft SQL Server).

УЯЗВИМОСТЬ ЦЕЛОЧИСЛЕННОГО ПЕРЕПОЛНЕНИЯ

Уязвимость целочисленного переполнения возникает как результат некорректных операций над данными целого типа [4]. Последствия целочисленного переполнения могут быть самыми разными – от некорректного результата до краха приложения. Данная уязвимость может быть использована для изменения значения критически важных данных – изменения размера буфера, значения индекса массива. При этом возникает возможность переполнения буфера. К возникновению данной уязвимости могут привести следующие операции:

– совместное оперирование знаковыми и беззнаковыми целыми (например, сравнение знакового целого и беззнакового);

– усечение целых (например, усечение 32-битового целого до 16-битового);

– потеря значимости и переполнение (например, в результате суммирования двух целых может быть получено число большее, чем максимально возможное для целого типа данных).

Следующие особенности кода приложения свидетельствуют о потенциальной возможности уязвимости:

– смешивание знаковых и беззнаковых целых в операциях вычисления и сравнения;

– смешивание данных различных типов в операциях вычисления и сравнения;

– сравнение переменных и литералов;

– отсутствие проверки входных данных;

– использование результата вычисления без его проверки.

Основным методом обнаружения данной уязвимости является ревизия исходных текстов программного обеспечения. Важно, чтобы все случаи динамически выделяемой памяти и индексных массивов, использующих целочисленную арифметику, были проанализированы на предмет корректности. В процессе такой проверки необходимо тестировать следующие ситуации:

– ввод отрицательных значений при запросе на ввод целых чисел;

– ввод целых, соответствующих граничным значениям хранения данных в одном байте, двух байтах и т. д. – т. е. чисел 0, 7, 8, 254, 255, 16353, 16354;

– ввод очень длинных строк (более 64 К);

– ввод строк, длина которых равна типичным граничным значениям (32К, 32К-1, 64К-1, 64К);

– ввод случайных, непредусмотренных или неверных данных – так называемый Fuzz testing. Fuzz testing – это техника тестирования, состоящая в подаче на вход приложения случайных и направленно сформированных наборов данных с целью генерации ошибок в приложении или его аварийного завершения. Процентный показатель сбоев и крахов приложения является показателем уязвимости.

Чтобы минимизировать вероятность возникновения данной уязвимости, следует придерживаться следующих правил:

– для проверки текстов программ использовать инструментальные средства анализа, обладающие низкой вероятностью ошибок 2-го рода для уязвимостей данного типа;

– везде, где возможно, использовать беззнаковые целые;

– при выделении памяти использовать только беззнаковые целые;

– при построении индексированных массивов использовать только беззнаковые целые;

– проверять введенные пользователем числовые данные, разрешая только заведомо корректные данные;

– при компиляции приложения устанавливать максимально подробный уровень сообщений компилятора.

УЯЗВИМОСТИ СТРОК ФОРМАТИРОВАНИЯ (FORMAT STRING)

Наличие данной уязвимости позволяет передать в качестве входного параметра функции ввода/вывода специальным образом сконструированную строку, что позволяет атакующему получить информацию об управлении приложением и даже изменить ход выполнения приложения [4]. Многие функции ввода/вывода позволяют нужным образом отформатировать строку, переданную им в качестве входного параметра. Это означает, что входные параметры функции могут содержать специальные символы, определяющие формат преобразования. Специальным образом подобранные символы форматирования позволяют прочесть содержимое областей памяти, организовать переполнение буфера. В худшем случае эксплуатация уязвимости позволяет атакующему выполнить произвольный код в системе. Первопричиной такой уязвимости является недостаточная проверка входных данных.

Основным методом обнаружения уязвимости строк форматирования является анализ исходных текстов программ. Следует очень осторожно использовать такой спецификатор форматирования, как %p (указатель) и избегать спецификатора %n (количество символов, записанных по адресу, указанному в качестве второго аргумента).

При тестировании приложения следует вставлять спецификаторы форматирования во вводимые данные во всех точках ввода строковых данных и анализировать полученный результат. Количество и вид спецификаторов существенно варьируется в зависимости от используемого языка программирования. При тестировании, кроме спецификаторов, определенных в используемом языке, следует обязательно проверять спецификаторы, определенные в языках C/C++.

Чтобы минимизировать вероятность появления данной уязвимости, необходимо придерживаться следующих правил:

– для проверки текстов программ использовать инструментальные средства анализа, обладающие низкой вероятностью ошибок 2-го рода для уязвимостей данного типа;

– проверять все полученные извне данные перед обработкой, разрешая только заведомо корректные данные;

– функции форматирования строк, используемые в приложении, должны быть доступны только привилегированным пользователям;

– если исходный код приложения написан на языке C++, нужно использовать операторы управления потоком (stream operators) вместо функций семейства printf;

– если используется компилятор GCC, необходимо устанавливать режимы -Wformat, -Wformat-security для обнаружения ошибок использования функций форматирования строк;

– предпочтение следует отдавать компилятору Microsoft Visual C++ 2005 или более позднему для обнаружения ошибок использования функций форматирования строк в процессе выполнения.

УЯЗВИМОСТЬ ВНЕДРЕНИЯ КОМАНД (COMMAND INJECTION)

При эксплуатации данной уязвимости в приложение могут быть переданы данные, приводящие к тому, что у атакующего появляется возможность манипулировать командной оболочкой (shell) операционной системы. В результате могут быть выполнены команды с административными полномочиями.

Потенциально, уязвимость возникает, если в приложении используются команды порождения процессов, параметры которых не проверяются должным образом. Список таких команд для разных языков программирования приведен в табл. 1.

Таблица 1. Команды порождения процессов для разных языков программирования

Язык программирования	Потенциально опасные функции
C/C++	system (), popen (), execlp (), execvp (), ShellExecute (), ShellExecuteEx (), _wsystem ()
Perl	system, exec, `, open, |, eval, /e
Python	exec, eval, os.system, os.popen, execfile, input, compile
Java	Class.forName (), Class.newInstance (), Runtime.exec ()

В дополнение к ревизии исходных текстов программ необходимо провести тестирование приложения на предмет обнаружения уязвимости внедрения команд. В процессе тестирования необходимо:

– идентифицировать все интерпретаторы и компиляторы, используемые для создания и функционирования приложения;

– идентифицировать все потенциально опасные символьные последовательности, которые могут изменить характер действий интерпретатора;

– сформировать входные данные, содержащие такие потенциально опасные наборы символов и проанализировать видимые последствия обработки таких данных.

УЯЗВИМОСТЬ К МЕЖСАЙТОВОМУ СКРИПТИНГУ (CROSS SITE SCRIPTING – XSS)

Уязвимость XSS возникает, когда входные данные, получаемые Web-сервером от одного клиента, могут быть пересланы другому клиентскому браузеру через Web страницу [5]. Эти данные могут содержать программный код, например на JavaScript, который будет выполнен браузером клиента. Поскольку система безопасности клиента считает, что код получен от Web-сервера, то злоумышленный код может получить доступ к данным, доступ к которым имеет Web-сервер (например, файлам cookie), либо изменить Web страницу, манипулируя ссылками на ней и т. п. Эксплуатируя данную уязвимость, атакующий может получить доступ к персональным данным пользователя либо перенаправить пользователя на нужный атакующему сайт.

Если пользовательский ввод возвращается в браузер, то приложение потенциально уязвимо для XSS атак.

В дополнение к ревизии исходных текстов программ необходимо провести тестирование приложения на предмет обнаружения уязвимости к XSS атакам. В процессе тестирования необходимо:

– выполнять запрос к приложению, помещая в пользовательский ввод заведомо неправильные данные;

– проанализировать ответный HTML код с целью обнаружения в нем отправленных данных. Причем необходимо учитывать, что отправленные данные могут быть возвращены не сразу, а в ответ на последующие запросы.

Для минимизации вероятности возникновения XSS уязвимости следует:

– для проверки текстов программ использовать инструментальные средства анализа, обладающие низкой вероятностью ошибок 2-го рода для уязвимостей данного типа;

– контролировать все входные данные, разрешая только заведомо правильные данные;

– если во входных данных необходимо использовать специальные символы HTML-кода, необходимо удалять все HTML-теги, кроме разрешенных;

– устанавливать для Web страниц определенную кодовую страницу, чтобы избежать ввода непредусмотренных данных.

УЯЗВИМОСТЬ К ПЕРЕПОЛНЕНИЮ БУФЕРА (BUFFER OVERFLOW)

Данная уязвимость возникает, если существует возможность записи в буфер информации больше, чем позволяет область памяти, выделенная под буфер. Существует несколько вариантов переполнения буфера [4], все они потенциально опасны для приложения и могут привести к аварийному завершению приложения либо к выполнению злоумышленного

кода в атакуемой системе. Если приложение выполняется от имени привилегированной учетной записи, такой как system или root, последствия для атакуемой системы могут быть катастрофическими. Эксплуатация данной уязвимости в основном осуществляется при недостаточной проверке входных данных.

Следующие особенности приложения могут свидетельствовать о наличии уязвимости переполнения буфера:

– входные данные не проверяются перед тем, как будут скопированы в буфер;

– некорректное использование небезопасных функций;

– некорректное определение необходимых размеров буфера;

– некорректное вычисление размера индексных массивов.

Основным методом обнаружения уязвимости является анализ исходных текстов программ и fuzz testing. Чтобы минимизировать вероятность возникновения данной уязвимости, следует придерживаться следующих правил:

– для проверки текстов программ использовать инструментальные средства анализа, обладающие низкой вероятностью ошибок 2-го рода для уязвимостей данного типа;

– проверять все полученные извне данные перед обработкой, разрешая только заведомо корректные данные;

– не использовать заведомо небезопасные функции (такие как strcpy, strcat, lstrcat, sprintf, fprintf и другие), заменяя их безопасными;

– тщательно проверить все алгоритмы вычисления размера буфера;

– при компиляции использовать опции компилятора для контроля переполнения буфера. Например, в компиляторе Visual C++ 2005 Service Pack 1 и более поздних следует использовать опции /GS, /SAFESEH, /NXCOMPAT и /DYNAMICBASE. В компиляторе gcc 4.1.2-25 и более поздних используйте опцию -stack-protector.

ВЫВОДЫ

Широкое распространение распределенных информационных систем, Internet-технологий существенно увеличило множество точек доступа к приложениям и, как следствие, расширило возможности злоумышленников атаковать информационные системы. Большинство таких атак становится возможным в силу наличия уязвимостей в приложениях, обрабатывающих данные из внешних источников. К сожалению, традиционные методы защиты периметра (firewalls, email filters и пр.) не способны полностью решить задачу защиты от внешних атак. Разработчики приложений должны знать характерные признаки уязвимостей и уметь таким образом строить код приложения, чтобы исключить возможность атак, эксплуатирующих такие уязвимости. В статье рассматриваются популярные виды уязвимостей, описываются их признаки и даются рекомендации по устранению уязвимостей.

СПИСОК ЛИТЕРАТУРЫ

1. National Vulnerability Database [Электронный ресурс] / National Institute of Standards and Technology. – Режим доступа: http://nvd.nist.gov/, свободный. – Загл. с экрана.
2. List of Common Vulnerabilities and Exposures [Электронный ресурс] / The MITRE Corporation. – Режим доступа: http://cve.mitre.org/, свободный. – Загл. с экрана.
3. Поиск уязвимостей в программах с помощью анализаторов кода [Электронный ресурс] / Елена Харитонова. – Режим доступа: http://www.codenet.ru/progr/other/code-analysers.php, свободный. – Загл. с экрана.
4. *Thompson, H. H.* The Software Vulnerability Guide (Programming Series) / Herbert H. Thompson, Scott G. Chase. – Pontypridd, UK, United Kingdom : Charles River Media, 2005. – 354 p.
5. *Хогланд, Г.* Взлом программного обеспечения: анализ и использование кода / Грег Хогланд, Гари Мак-Гроу. – М. : Вильямс, 2005. – 389 с. : ил.

Надійшла 4.11.2010

Коломицев М. В., Носок С. О.

ВРАЗЛИВОСТІ ДОДАТКІВ ДО НЕКОРЕКТНИХ ВХІДНИХ ДАНИХ

У даній статті розглядаються різного роду вразливості, властиві додаткам без достатнього контролю вхідних даних. Наводяться характерні ознаки таких вразливостей і даються рекомендації з їхнього усунення.

Ключові слова: вразливість, додатки, коректні дані, некоректні дані.

Kolomytsev M., Nosok S.

APPLICATIONS VULNERABILITIES CAUSED BY INCORRECT INPUT DATA

Different kinds of vulnerabilities typical for applications without sufficient input data control are discussed in this article. Typical characteristics of such vulnerabilities are given and elimination recommendations are proposed.

Key words: vulnerability, applications, correct data, incorrect data.