инженерия программного обеспечения

Нгуен Ван Доан,
Сафонов Владимир Олегович

УДК 004.432.4

РЕАЛИЗАЦИЯ АСПЕКТОВ БЕЗОПАСНОСТИ В ASP.NET-ПРИЛОЖЕНИЯХ НА ОСНОВЕ СИСТЕМЫ ASPECT.NET

Аннотация

Предложены методы применения аспектно-ориентированного программирования для задач безопасности в ASP.NET-приложениях, таких как: аутентификация, авторизация, олицетворение, защита от Cross-Site Scripting атаки, инструментирование ASP.NET приложений для безопасности. Разработаны аспекты безопасности для ASP.NET-приложений в системе Aspect.NET. Проанализированы эффективность и производительность применения этих аспектов в ASP.NET-приложениях.

Ключевые слова: аспектно-ориентированное программирование, АОП, Aspect.NET, веб-приложение, Microsoft.NET, ASP.NET, безопасность.

ВВЕДЕНИЕ

Аспектно-ориентированное программирование (*АОП*) [1] – перспективный подход к инженерии программ, предназначенный для разработки сквозной функциональности (*cross-cutting concerns*) – идей, методов, функциональных возможностей, реализуемых и модифицируемых в ходе разработки программ, которые принципиально по своей природе не могут быть реализованы одной обобщенной процедурой (*generalized procedure*) – тесно взаимосвязанной совокупности модулей (например иерархией классов) и требуют для своей реализации совокупность рассредоточенных действий (*tangled actions*), которые должны быть добавлены в различные части существующего программного кода целевого приложения, для того чтобы новая сквозная функциональность заработала [3]. Иначе говоря, сквозная функциональность – это новая функциональность, реализация которой рассредоточена по коду приложения. Тем самым, АОП позволяет систематически добавлять и модифицировать новую функциональность, в том числе относящуюся к веб-программированию. Подробное описание АОП дано в работах [1–3].

В настоящее время веб-программирование играет важную роль в сфере разработки программного обеспечения (ПО). Веб-приложения (*Web applications*) быстро развиваются и постепенно заменяют приложения для настольных систем (*desktop applications*). Развитие Web-приложений является важной тенденцией эволюции компьютерных технологий, операционных систем, сетевых архитектур и прикладных программ [7]. Поэтому применение АОП в веб-программировании,

как показано в статье, имеет важное значение для разработки ПО. Оно позволяет разработчикам снизить время, стоимость и сложность разработки, упростить сопровождение веб-продуктов и внесение в них изменений, создавать надежные и безопасные веб-приложения.

В данной статье рассмотрено применение АОП для реализации безопасности в ASP.NET-приложениях на основе системы Aspect.NET, представляющей собой инструментарий АОП для платформы .NET, разработанный в лаборатории Java-технологий математико-механического факультета СПбГУ под руководством профессора В.О. Сафонова. Цель статьи – анализ методов применения АОП, реализованных в системе Aspect.NET, для разработки и модификации ASP.NET-приложений, а также для реализации следующих задач безопасности в ASP.NET-приложениях: *аутентификации* (*authentication*), *авторизации* (*authorization*), *олицетворения* (*impersonation*), *защиты от Cross-Site Scripting атаки*, *инструментирования ASP.NET-приложений для безопасности*; анализа эффективности и производительности применения АОП для разработки ASP.NET-приложений.

1. ЗАДАЧИ БЕЗОПАСНОСТИ В ASP.NET-ПРИЛОЖЕНИЯХ

Безопасность веб-приложений является важной задачей веб-программирования. Безопасность ASP.NET-приложений основана на трех основных операциях:

- *Аутентификация* – процесс идентификации пользователя для предоставления доступа к какому-либо ресурсу приложения (разделу сайта, странице, базе данных). Аутентификация основана на проверке сведений о пользователе (например, имени и пароля).
- *Авторизация* – процесс предоставления полномочий доступа пользователю на основе данных аутентификации.
- *Олицетворение* (*impersonation*) – механизм, предоставляющий серверному процессу ASP.NET права доступа аутентифицированного пользователя. Данный механизм работает только в режиме *Windows-аутентификации*.

ASP.NET предоставляет удобные механизмы для реализации этих операций [5, 6]. При разработке веб-приложений часто необходимо осуществлять программно действия аутентификации и авторизации пользователей или действия олицетворения некоторого фрагмента кода в веб-приложениях с Windows-аутентификацией. Эти действия могут повторяться в различных модулях приложения.

Кроме перечисленных задач, при разработке ASP.NET-приложений разработчики сталкиваются с другими задачами безопасности, в том числе и с *защитой от Cross-Site Scripting атаки* [9–11], с *инструментированием ASP.NET-приложений для безопасности* [14].

Cross-Site Scripting (*XSS*) – «*межсайтовый скриптинг*» – тип уязвимости интерактивных информационных систем в вебе. XSS возникает, когда в генерируемые сервером страницы по какой-то причине попадают пользовательские скрипты. Специфика подобных атак заключается в том, что, вместо непосредственной атаки сервера, они используют уязвимый сервер в качестве средства атаки на клиента. Сейчас XSS составляют около 33,64% всех обнаруженных уязвимостей (при автоматическом сканировании)[12]. Долгое время программисты не уделяли им должного внимания, считая их неопасными. Однако это мнение ошибочно: на странице или в HTTP-Cookie могут быть весьма уязвимые данные (например, идентификатор сессии администратора). На популярном сайте скрипт может организовать *DoS-атаку*, поэтому необходимо защитить веб-приложения от XSS-атак.

Термин *инструментирование* обозначает возможность отслеживания и измерения уровня производительности продукта и диагностики ошибок [15]. *Инструментирование ASP.NET-приложений для безопасности* заключается в использовании пользовательских веб-событий мониторинга (*Custom Health Monitoring Events*) для отслеживания связанных с безопасностью

событий и операций [14]. Оно включает в себя следующие шаги:

– использование встроенных или создание пользовательских веб-событий (*web events*);

– использование встроенных или создание пользовательских поставщиков (*providers*);

– настройка мониторинга работоспособности ASP.NET: раздела *healthMonitoring* файла конфигурации *Web.config* [13, 14];

– вызов (*raise*) объекта веб-событий в нужных местах веб-приложения.

Более подробно инструментирование ASP.NET-приложений описано в [13–15].

2. СУЩЕСТВУЮЩИЕ РЕШЕНИЯ ЗАДАЧ БЕЗОПАСНОСТИ ASP.NET-ПРИЛОЖЕНИЙ

В настоящее время существует значительное число реализаций АОП [2] для платформы Microsoft.NET, однако пока они не были использованы для решения задач веб-программирования. В данном разделе рассмотрим более традиционные основанные на ООП решения вышеперечисленных задач безопасности ASP.NET-приложений, а затем покажем, как они могут быть решены с помощью АОП и в чем преимущество использования АОП для их решения.

2.1. ОСНОВНЫЕ ОПЕРАЦИИ БЕЗОПАСНОСТИ ASP.NET

Основные операции безопасности ASP.NET (*аутентификация, авторизация, олицетворение*) реализуются двумя подходами: реконфигурируемая безопасность (*configurable security*) и программируемая безопасность (*programmatic security*). Реконфигурируемая безопасность связана с доступом к URL странице веб-сайта, то есть конфигурируется URL безопасность в файле *web.config* веб-приложения. При этом действия аутентификации, авторизации и олицетворения выполняются по странице. Более подробные сведения о реконфигурируемой безопасности можно найти в [5, 6]. Программируемая безопасность связана с проверкой права доступа в коде приложения. Возможные опции для программной безопасности представлены на рис. 1.

В зависимости от характеристики конкретного веб-приложения используется либо реконфигурируемая безопасность, либо программируемая безопасность, либо оба этих подхода.

2.2. ЗАЩИТА ОТ CROSS-SITE SCRIPTING АТАКИ

В работах [9–11] описаны методы защиты от XSS-атаки в ASP.NET. Они включают в себя следующие подходы:

- *Включение ASP.NET – валидации запросов (ASP.NET Request Validation)*. По умолчанию валидация запросов включена в файл *Machine.config*. Эту настройку также можно изменить через *Web.config* или на самой странице.
- *Фильтрация входящих данных*. Любые данные от клиента – это опасные данные, они должны быть проверены и обработаны. Это можно делать через *RegularExpressionValidator*, *RangeValidator* и *CustomValidator*, в более сложных ситуациях подойдут регулярные выражения на стороне сервера.
- *Кодирование гипертекста*. Например, код такого вида *Response.Write(Request.Form["name"]);* и *lbFeedbackMsg.Text = message;* передает клиенту не обработанные параметры из HTTP запроса и из переменной *message*, это не безопасно. Атакующий сможет передать в параметре, например, такой код: *<script>alert("hello");</script>*, который выполнится у клиента, поэтому необходимо кодировать передаваемую клиенту информацию с помощью *HttpUtility.HtmlEncode()*, который заменяет все специальные символы (<, > и т. д.) на их *Html-эквиваленты* (*"<"*, *">"* и т. д.):

```
String name = Request.Form["name"];
Response.Write(HttpUtility.HtmlEncode(name));
lbFeedbackMsg.Text =
        HttpUtility.HtmlEncode(message);
```

a) явная проверка безопасности с использованием *IPrinciple* интерфейса (объект *User* – объект класса, реализующего интерфейс *System.Security.Principal.IPrincipal* и является свойством объекта страницы *System.Web.UI.Page*):

Аутентификация:

```
if (User.Identity.IsAuthenticated) { ... }
```

Авторизация (проверка роли):

```
if (User.Identity.IsAuthenticated && User.IsInRole("Admin")){ ... }
```

b) использование *PrincipalPermission* класса для проверки безопасности:

Аутентификация:

```
PrincipalPermission pp = new PrincipalPermission(null, null, true);
     pp.Demand();
     //equivalent with the check: if (User.Identity.IsAuthenticated)
     //Do business action
```

Авторизация:

```
PrincipalPermission pp = new PrincipalPermission(null, "Admin", true);
     pp.Demand();
//Do business action
```

c) олицетворение кода с правами доступа аутентифицированного пользователя:

```
WindowsIdentity windowsID = User.Identity as WindowsIdentity;
   if (windowsID != null) {
       WindowsImpersonationContext wiContext =
           windowsID.Impersonate();
      //Do something here with permission of authenticated user
       wiContext.Undo();
   }
```

Рис. 1

• *Кодирование URL строк*. Аналогично HTML кодируются URL строки, полученные от пользователя методом *HttpUtility.UrlEncode()*.
Response.Write(HttpUtility.UrlEncode(urlString));

• *Фильтрация пользовательских данных*. Если полнофункциональная ASP.NET-страница принимает от пользователя данные с HTML, например, есть возможность использовать HTML-форматирование, приходится отключать *request validation*. В таком случае стоит фильтровать все HTML тэги, кроме пары базовых, например, типа <*b*> и <*i*>, на стороне сервера.

Более подробно защита от XSS-атаки в ASP.NET описана в [9–11]. Два первых подхода реализуются в коде представления, а три последних подхода реализуются в коде программной логики и могут потребоваться во многих модулях веб-приложения.

2.3. ИНСТРУМЕНТИРОВАНИЕ ASP.NET-ПРИЛОЖЕНИЙ ДЛЯ БЕЗОПАСНОСТИ

В [14] описан пример инструментирования ASP.NET-приложения для отслеживания вызовов методов, содержащих чувствительную бизнес-логику. После создания пользовательского веб-события *SensitiveFunctionEvent* и настройки мониторинга работоспособности (добавления элемента <*healthMonitoring*> внутри элемента <*system.web*> файла *Web.config*) необходимо добавить код вызова веб-события в код веб-приложения, где вызываются чувствительные методы. В [14] код вызова веб-события помещается внутри чувствительного метода (см. листинг 1).

С точки зрения ООП, для повторного использования и сопровождения приложения, лучше реализовать вызов веб-событий в отдельном модуле, например *SensitiveFunctionEventRaiser*. В нем реализуем вызов Веб-событий в методе, например, *Raise*. Потом вызовем метод *Raise* перед вызовом чувствительного метода.

3. АСПЕКТЫ БЕЗОПАСНОСТИ ДЛЯ ASP.NET-ПРИЛОЖЕНИЙ

Вышеперечисленный анализ показывает, что традиционные, основанные на ООП, подходы могут решать задачи безопасности ASP.NET-приложений: решение каждой задачи обычно реализуется в отдельном модуле или в наборе модулей; при необходимости разработчик вызывает эти модули в нужных ему точках выполнения кода. Проблема возникает, когда нам необходимо применить наши модули во многих точках выполнения приложения, например при вызове веб-событий или при вызове модуля кодирования гипертекста для защиты от XSS-атаки, или при использовании программируемой безопасности многократно в разных точках выполнения приложения. При этом размер кода увеличивается за счет повторения и рассредоточения кода вызова (реализации) задачи (или функциональности). Возможна также ситуация модификации разработанного веб-приложения для добавления новой сквозной функциональности, такая как рассмотренные нами задачи, например, кодирование гипертекста или фильтрация пользовательских данных для улучшения защиты от XSS-атаки. При этом для реализации новой функциональности приходится изменять код приложения вручную, реализуя эту функциональность в модулях и добавляя код вызова этих модулей вручную в нужные точки выполнения приложения.

Исходя из данных предпосылок, возникает идея применения АОП для решения указанных задач при разработке ASP.NET-приложений. С помощью АОП каждая задача (или функциональность) реализуется в аспекте в виде набора действий (*action*s), затем определяются условия внедрения для присоединения этих действий к нужным нам точкам выполнения веб-приложения, после этого запускается *подсистема внедрения* (*weaver*) аспектов системы Aspect.NET. Действия аспекта будут автоматически добавляться подсистемой внедрения в точки присоединения (то есть в нужные нам точки выполнения), определенные условиями внедрения аспекта. При этом изменения целевого веб-приложения выполняются на уровне *MSIL* кода.

Для практического подтверждения описанной идеи было принято решение разработать аспекты, поддерживающие вышеперечисленные задачи с использованием системы Aspect.NET, преимущества которой по сравнению с другими инструментами реализации АОП для .NET описаны в [2]. Были разработаны аспекты: аутентификации (*web authentication aspect*), авторизации (*web authorization aspect*), олицет-

Листинг 1

```
private void SomeFunctionContainingSensitiveLogic() {
        SensitiveFunctionEvent testEvent = new SensitiveFunctionEvent(
                                "Sensitive function has been accessed",
                                        this,
                                      WebEventCodes.WebExtendedBase + 3,
                                "SomeFunctionContainingSensitiveLogic");
        testEvent.Raise();
        // Some sensitive logic would appear below...
        // ...
       }
```

ворения (*impersonation aspect*), защиты от XSS-атак (*cross-site scripting protection aspect*), инструментирования (*instrumenting aspect*). Эти аспекты были разработаны на Microsoft Visual Studio 2008 с использованием новой версии Aspect.NET, которая будет выпущена в ближайшее время. Все разработанные аспекты поддержки веб-программирования на ASP.NET будут опубликованы на академическом сайте Microsoft вместе с новой версией Aspect.NET (для VS 2008).

Условия внедрения аспектов, которые описывают множество точек присоединений, будут разными для разных веб-приложений. Для каждого конкретного веб-приложения необходимо написать свои условия внедрения аспектов. Более подробно их форма описана в [3].

При поддержке или расширении приложения необходимо учитывать влияние разработанных аспектов, то есть учитывать условия внедрения аспектов, которые могут быть изменены. В свою очередь, Aspect.NET предоставляет возможность «вручную» отметить какую-либо потенциальную точку присоединения, либо отменить ее отметку, исключив тем самым ее из дальнейшего процесса внедрения. Это позволяет избежать «слепого», неконтролируемого внедрения аспектов – одной из самых серьезных опасностей, которые таит в себе АОП – мощное «оружие», управляющее групповыми модификациями кода. Описываемая возможность Aspect.NET – *внедрение, управляемое пользователем,* – позволяет сделать процесс внедрения аспектов более безопасным и разумным. Однако внедрение, управляемое пользователем, не обязательно: если какого-либо пользователя это затрудняет, он может внедрять аспекты автоматически нажатием кнопки Weave aspects. Отметим еще раз, что подобная возможность *отсутствует* во всех других известных нам, даже весьма популярных инструментах АОП.

4. ЭФФЕКТИВНОСТЬ И ПРОИЗВОДИТЕЛЬНОСТЬ ПРИМЕНЕНИЯ АОП

В этом разделе рассмотрим эффективность и производительность применения АОП для реализации задач веб-программирования в системе Aspect.NET. В [2] автор показал, что применение АОП с использованием Aspect.NET улучшает продуктивность разработчиков ПО, на основе оценки трудоемкости с использованием модели COCOMO [8]. Для оценки используется случай использования действий внедрения *%before* и *%after*. Пусть s – объем исходного кода оригинального приложения. Предположим, что мы должны реализовать новую функциональность добавлением некоторых фрагментов кода перед и после всех вызовов метода P в коде всего приложения. Пусть a – объем кода, добавленного после вызовов P, и b – объем кода, добавленного перед вызовами P, c – число вызовов P во всем коде приложения. Если мы реализуем требуемую функциональность вручную без использования Aspect.NET, то объем полученного кода будет вычислен по формуле:

$$s^* = s + (a + b) * c .$$

Так как в Aspect.NET реализуется внедрение аспекта на уровне бинарного кода (MSIL) после компиляции целевого приложения и аспекта, в случае применения Aspect.NET для реализации новой функциональности, объем полученного кода приложения равен сумме объема его оригинального кода и объема кода аспекта:

$$s^* = s + a + b .$$

Объем результирующего кода приложения в случае без применения Aspect.NET будет больше объема улучшенного кода приложения с применением Aspect.NET на следующую сумму:

$$\Delta_s = [s + (a + b) * c] - (s + a + b) = (a + b)(c - 1).$$

Согласно базовой модели COCOMO, суммарная трудоемкость E в человеко-месяцах в случае без применения Aspect.NET будет вычисляться по формуле:

$$E_{plain} = a_b \left[\frac{s + (a+b)c}{1000} \right]^{b_b}.$$

В случае применения Aspect.NET E будет вычисляться по другой формуле:

$$E_{Aspect.NET} = a_b \left(\frac{s + a + b}{1000} \right)^{b_b}.$$

Очевидно, что $E_{plain} > E_{Aspect.NET}$.

Время разработки будет вычисляться по формулам:

$$T_{plain} = c_b \left[E_{plain} \right]^{d_b} \text{ и}$$

$$T_{Aspect.NET} = c_b \left[E_{Aspect.NET} \right]^{d_b}.$$

Поскольку $E_{plain} > E_{Aspect.NET}$, то $T_{plain} > T_{Aspect.NET}$.

Число разработчиков будет вычисляться по формулам:

$$Ndev_{plain} = \frac{E_{plain}}{T_{plain}} \text{ и}$$

$$Ndev_{Aspect.NET} = \frac{E_{Aspect.NET}}{T_{Aspect.NET}}.$$

С точки зрения COCOMO, в худшем случае для *встроенных проектов* (*проекты*, которые разрабатываются с учетом множества жестких ограничений: по аппаратному, программному, операционному обеспечению и т. д.) формулы оценки будут выглядеть следующим образом:

$$E_{plain} = 3.6 \left[\frac{s + (a+b)c}{1000} \right]^{1.2} \text{ и}$$

$$E_{Aspect.NET} = 3.6 \left(\frac{s + a + b}{1000} \right)^{1.2};$$

$$T_{plain} = 2.5 \left[E_{plain} \right]^{0.32} \text{ и}$$

$$T_{Aspect.NET} = 2.5 \left[E_{Aspect.NET} \right]^{0.32}.$$

Для более конкретного примера предположим, что наш проект небольшой: s = 1000 строк кода, a = 1 и b = 1 (объем кода вставки является наименьшим возможным) и c = 100. В этом случае приблизительно, E_{plain} = 4.48 человеко-месяцев и $E_{Aspect.NET}$ = 3.6 человеко-месяцев (то есть применение Aspect.NET позволяет сократить срок разработки проекта на один человеко-месяц – четверть трудоемкости проекта), T_{plain} = 4.04 месяцев и $T_{Apsect.NET}$ = 3.77 месяцев (то есть применение Aspect.NET уменьшает время разработки примерно на 0.27 месяца – 8 дней).

Теперь рассмотрим производительность исполнения приложения с применением Aspect.NET для реализации основных задач безопасности ASP.NET (то есть для *аутентификации*, *авторизации*, *олицетворения*). Измерим время выполнения этих задач, вызываемых 10000 раз, для обоих случаев: без применения и с применением Aspect.NET. Измерения были реализованы на портативном компьютере *Intel Core 2 CPU 1.83GHz, 2Gb Ram*. Для каждой задачи измерение было произведено 100 раз. Были получены средние значения времени выполнения основных задач безопасности ASP.NET (в миллисекундах) и среднеквадратические погрешности, они приведены в табл. 1.

По полученным результатам измерения легко видеть, что время выполнения основных задач безопасности ASP.NET без применения и с применением Aspect.NET примерно одинаково. Поэтому можно заключить, что производительность исполнения веб-приложения после внедрения аспектов с помощью Aspect.NET не хуже производительности такого веб-приложения без применения АОП, никакого избыточного ин-

Табл. 1. Среднее время выполнения задач безопасности ASP.NET

		Average Time	Standard Deviation
Authentication	OOP	31.591807	2.046801
	AOP	**32.071834**	**2.129242**
Authorization	OOP	37.392139	3.006206
	AOP	**37.922111**	**2.921416**
Impersonation	OOP	312.807891	8.891007
	AOP	**309.227687**	**8.819479**

струментального кода не внедрено и не выполнено.

Наш подход – применение АОП для задач безопасности ASP.NET – имеет следующие преимущества по сравнению с реализацией задач без применения АОП:

- Использование синтаксиса определения условия внедрения аспекта [3] позволяет описать множество точек присоединения, в которые необходимо добавить функциональность наших задач.
- Действия аспекта будут автоматически добавляться на уровне MSIL-кода *подсистемой внедрения* (*weaver*) в точки присоединения (то есть в нужные нам точки выполнения), определенные условиями внедрения аспекта. Таким образом, не требуется «ручных» вставок кода реализации функциональности в этих точках выполнения. Благодаря этому, уменьшаются объем кода, вероятность программных ошибок, время и стоимость разработки.
- Никаких изменений исходного кода в целевом веб-приложении не требуется. Необходимо лишь реализовать наши задачи в виде аспектов, определить условия их внедрения и запустить подсистему внедрения аспектов системы Aspect.NET.
- Упрощается сопровождение, изменение и расширение веб-приложения. Новые требования (новая функциональность) реализуются в аспектах, затем внедряются в целевое веб-приложение. Изменение функциональности осуществляется также в коде реализации аспектов.
- Веб-приложения полностью работоспособны без применения аспектов. При этом функциональность, реализованная в аспектах, отсутствует в целевых веб-приложениях.
- Обеспечивается повторная используемость кода: разработанные аспекты могут внедряться в разные веб-приложения, требующие функциональности аспектов.

ЗАКЛЮЧЕНИЕ

В статье предложены методы применения аспектно-ориентированного программирования для задач безопасности в ASP.NET-приложениях, таких как: *аутентификация* (*authentication*), *авторизация* (*authorization*), *олицетворение* (*impersonation*), *защита от Cross-Site Scripting атаки*, *инструментирование ASP.NET-приложений для безопасности*. Разработаны аспекты безопасности в системе *Aspect.NET*: аутентификации (*web authentication aspect*), авторизации (*web authorization aspect*), олицетворения (*impersonation aspect*), защиты от XSS-атак (*cross-site scripting protection aspect*), инструментирования (*instrumenting aspect*). Проанализированы эффективность и производительность применения этих аспектов в ASP.NET-приложениях. Код разработанной библиотеки аспектов опубликован на сайте проекта Aspect.NET [4]. На основании рассмотренных и использованных идей выполняется дальнейшее исследование применения АОП и развитие библиотеки аспектов для разработки веб-приложений и веб-сервисов.

Литература

1. Web-сайт по аспектно-ориентированной разработке программ. [Электронный ресурс], режим доступа: http://aosd.net свободный, язык английский, последнее обращение 02/09/2010.

2. *Safonov V.O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008.

3. *Сафонов В.О.* Практическое руководство по системе аспектно-ориентированного программирования Aspect.NET // Компьютерные инструменты в образовании, 2008. № 2. С. 12–20.

4. Web-сайт проекта Aspect.NET. [Электронный ресурс], режим доступа: http://www.aspectdotnet.org свободный, язык английский, последнее обращение 02.09.2010.

5. *Matthew MacDonald, Mario Szpuszta.* Pro ASP.NET 3.5 in C# 2008. Apress. 2008.

6. Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication. [Электронный ресурс], режим доступа: http://msdn.microsoft.com/en-us/library/aa302388.aspx свободный, язык английский, последнее обращение 02.09.2010.

7. Rakesh Pai. Web Applications – The Wave Of The Future. [Электронный ресурс], режим доступа: http://piecesofrakesh.blogspot.com/2005/01/web-applications-wave-of-future.html свободный, язык английский, последнее обращение 02.09.2010.

8. Boehm B. Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ, 1981.

9. Prevent Cross-Site Scripting in ASP.NET. [Электронный ресурс], режим доступа: http://msdn.microsoft.com/en-us/library/ms998274.aspx свободный, язык английский, последнее обращение 02.09.2010.

10. Protect From Injection Attacks in ASP.NET. [Электронный ресурс], режим доступа: http://msdn.microsoft.com/en-us/library/bb355989.aspx свободный, язык английский, последнее обращение 02.09.2010.

11. Preventing JavaScript Injection Attacks. [Электронный ресурс], режим доступа: http://www.asp.net/LEARN/mvc/tutorial-06-cs.aspx свободный, язык английский, последнее обращение 02.09.2010.

12. Статистика уязвимостей веб-приложений за 2009 год. [Электронный ресурс], режим доступа: http://www.securitylab.ru/analytics/394205.php свободный, язык русский, последнее обращение 02.09.2010.

13. Общие сведения о мониторинге работоспособности системы ASP.NET. [Электронный ресурс], режим доступа: http://msdn.microsoft.com/ru-ru/library/bb398933%28v=VS.90%29.aspx свободный, язык русский, последнее обращение 02.09.2010.

14. Instrument ASP.NET 2.0 Applications for Security. [Электронный ресурс], режим доступа: http://msdn.microsoft.com/en-us/ms998325.aspx свободный, язык английский, последнее обращение 02.09.2010.

15. Введение. Подготовка к инструментированию и трассировка. [Электронный ресурс], режим доступа: http://msdn.microsoft.com/ru-ru/library/x5952w0c.aspx свободный, язык русский, последнее обращение 02.09.2010.

Abstract

Methods of application of aspect-oriented programming for security tasks in ASP.NET-applications, such as: authentication, authorization, impersonation, Cross-Site Scripting attacks protection, instrumenting ASP.NET-applications for security, are suggested. Security aspects for ASP.NET-applications in Aspect.NET system are developed. Efficiency and productivity of application of these aspects in ASP.NET-applications are analyzed.

Keywords: aspect-oriented programming, AOP, Aspect.NET, Web application, Microsoft.NET, ASP.NET, security.

Нгуен Ван Доан,
аспирант кафедры информатики
математико-механического
факультета СПбГУ,
ngvdoanbk@yahoo.co.uk,

Сафонов Владимир Олегович,
доктор технических наук,
профессор кафедры информатики
математико-механического
факультета СПбГУ,
vosafonov@gmail.com