7. Johnson R., Hoeller J., Donald K. *Spring Framework Reference Documentation. WebSocket Support.* Available at: https://docs.spring.io/spring/docs/5.0.0.BUILD-SNAPSHOT/spring-framework-reference/html/websocket.html. (accessed 11.10.2018).
8. Kane C., Carroll A., Brener A. *Spring in Action, 4th Edition. Covers Spring 4.* NY, Manning Publications Co., pp. 485-510.
9. DB-Engines Ranking. Available at: https://db-engines.com/en/ranking (accessed 11.10.2018).
10. Sharma S. An Extended Classification and Comparison of NoSQL Big Data Models. *IJBDI*, 2015, no. 2, pp. 201-221.
11. Berners-Lee T., Fielding R., Frystyk H. *Hypertext Transfer Protocol -- HTTP/1.0*. Available at: https://tools.ietf.org/html/rfc1945 (accessed 11.10.2018).
12. Krishnamurthy B., Mogul J.C., Kristol D.M. Key differences between HTTP/1.0 and HTTP/1.1. *WWW '99 Proceedings of the eighth international conference on World Wide Web*. Elsevier North-Holland, Inc. New York, NY, USA 1999, pp. 1737-1751

Received 01.10.2018

УДК 005.62 + 004.05

ОСНОВЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ ПРИ РАЗРАБОТКЕ САЙТОВ

Юлдашев Т.З., Шептунов С.А.
Институт конструкторско-технологической информатики Российской академии наук, Москва, РФ
E-mail: info@dlay.ru

В условиях широкого внедрения автоматизированной обработки данных особая важность придается вопросу защиты информации от неправомерного доступа к ней злоумышленников. Оценку защищенности на этапе веб-разработки следует проводить методом анализа угроз по разным критериям (например, программный код, разметка страницы, пароли и т.д.). В статье приведены основные ошибки разработчиков сайтов при создании приложений, проведен анализ методов их устранения и повышения уровня безопасности для пользователя. На основе проведенного анализа сделан вывод о необходимости уделять значительное внимание сетевой безопасности при разработке сайтов и их использовании. Знание основ информационной безопасности пользователями может помочь защитить данные пользователя и само приложение от самых популярных современных угроз.

Ключевые слова: *сетевая безопасность, разработка сайтов, информационная безопасность, создание приложений*

Введение

В современном мире процессы работы с информацией являются одними из наиболее значимых составляющих существования любой системы окружающего мира. Потеря информации, ее искажение, а также невозможность своевременного получения необходимых данных может иногда привести даже к катастрофическим последствиям. В деловой и частной жизни люди достаточно часто получают необходимую им информацию из веб-источников, поэтому при разработке сайтов следует уделять особое внимание информационной безопасности.

К сожалению, современный уровень веб-разработки не всегда находится на достаточном уровне безопасности, так как на практике при разработке проекта сайта на обеспечение его защищенности часто выделяется недостаточное количество финансовых и квалифицированных человеческих ресурсов.

Помимо того, что разработчик должен быть компетентен, ему необходимо соблюдать основные правила обеспечения безопасности при написании кода в разрабатываемом проекте.

Код должен быть понятным и масштабируемым, а также сбалансированным по следующим критериям: достаточно гибким, и, одновременно, стабильным и надежным. Необходимо создавать интуитивно-понятный интерфейс, оптимизировать базу данных и совершенствовать проект, используя новый инструментарий веб-разработки и современное оборудование.

Обеспечение информационной безопасности разрабатываемого проекта сайта для заказчика, с одной стороны, является очень важной составляющей, а с другой стороны, возникают вопросы о том, какой уровень безопасности им необходим. В отличие от производительности веб-сайта невозможно дать количественную оценку «достаточной безопасности» разрабатываемого проекта. Посто-

янное возникновение новых киберугроз, а также разработка методов кражи, подмены и искажения информации злоумышленниками намного усложняет жизнь разработчика современных программных продуктов с точки зрения определения требований информационной безопасности, так как для этого требуется трата времени и соответствующих средств.

Безопасность внешних данных

Прежде чем перейти к основам безопасности ввода и вывода, стоит упомянуть один из основополагающих принципов безопасности: доверие. Нельзя доверять целостности запроса, поступающего из браузера пользователя. Нельзя доверять службам, которые проверяют поступающие данные и фильтруют их. Не следует доверять браузеру пользователя, так как браузер может быть подделан.

HTML-формы могут создавать иллюзию управления вводом. Автор разметки формы может полагать, что, поскольку они ограничивают типы значений, которые пользователь может ввести в форму, данные будут соответствовать этим ограничениям. Даже проверка на основе JavaScript на стороне клиента с точки зрения безопасности не обеспечивает необходимой защиты.

Безопасность поступающих из браузера пользователя данных, независимо от того, установлена ли фильтрация формы и является ли соединение защищенным HTTPS, фактически равна нулю. Пользователь может легко модифицировать разметку перед ее отправкой или использовать приложение командной строки, например, *curl*, для отправки подделанных данных. Или совершенно обычный пользователь может непроизвольно отправить модифицированную версию формы с враждебного веб-сайта. Как итог, чтобы обеспечить целостность входящих данных, проверка должна выполняться на сервере.

Значительной угрозой безопасности является умышленное искажение данных. В зависимости от прикладной логики и использования кодировки появляется возможность неожиданного поведения приложения, утечки данных и даже предоставления злоумышленнику возможности нарушить границы входных данных в исполняемый код.

Для сведения к минимуму опасности поступления ненадежных данных разработчику рекомендуется применять серверную фильтрацию и обработку данных. Проверка входных данных – это процесс обработки поступающей информации, экранирование спецсимволов и их фильтрация. Данные, выходящие за пределы ожидаемого набора значений, могут привести к тому, что приложение даст неожиданные результаты, например, нарушит бизнес-логику, вызовет ошибки и даже позволит злоумышленнику взять под контроль ресурсы или само приложение. Ввод, который обрабатывается на сервере как исполняемый код, например, запрос базы данных, выполненный на клиенте, как HTML JavaScript, особенно опасен. Валидация ввода – первая важная линия защиты информации.

Разработчики веб-приложений и веб-сервисов часто создают программный продукт с некоторой базовой проверкой входных данных, например, чтобы гарантировать, что значение не пусто или целое положительно. Следующий шаг к снижению риска атаки злоумышленниками заключается в ограничении ввода только логически приемлемыми значениями.

Валидация входа более эффективна для входов, которые могут быть ограничены небольшим набором. Обычно числовые типы могут быть ограничены значениями в определенном диапазоне. Например, для пользователя не имеет смысла запрашивать передачу отрицательной суммы денег или добавлять несколько тысяч предметов в свою корзину покупок. Эта стратегия ограничения ввода известных приемлемых типов известна как положительная проверка или «белый список значений». Белый список может ограничивать строку определенной формы, такой как URL-адрес или дату формы «уууу / mm / dd». Он может ограничивать длину ввода, одно допустимое кодирование символов или, в приведенном выше примере, только значения, доступные в вашей форме.

Необходимо решить, что именно делать, если ввод не прошел проверку. Наиболее популярным и, возможно, желательным является отказ от данных полностью, без обратной связи, и инцидент должен быть отмечен посредством регистрации или мониторинга. Пользователям не всегда необходимо давать информацию о том, почему данные недействительны, так как злоумышленник может воспользоваться такой информацией.

Такая атака называется рефлексивной атакой XSS, которая использует файлы сеансов чужих пользователей при работе с сайтом. Предоставлять обратную связь с пользователем лучше всего придерживаясь универсального ответа, который не отправляет ненадежные пользовательские данные (пример обратной связи: «Вы должны ввести электронную почту»). Часто разработчики пробуют отдельно фильтровать нежелательные теги, чтобы помешать атаке. Ввод отклоняется, если содержит известные опасные значения, такая стратегия называется отрицательной проверкой или «черным списком» данных.

Таблица 1. Фреймворки и методы обработки данных

Фреймфорк	Методы обработки данных
Java	Hibernate (Bean Validation)
	ESAPI
Spring	Встроенные типы параметров в контроллере
	Встроенный валидатор
Ruby on Rails	Встроенные средства проверки
ASP.NET	Встроенная проверка
Play	Встроенный валидатор
Generic JavaScript	XSS-фильтры
NodeJS	Валидатор js
General	Проверка на основе Regex

Проблема такого подхода заключается в том, что количество возможных нежелательных данных чрезвычайно велико. Актуализация полного списка потенциально опасных материалов будет дорогостоящей и трудоемкой. Функционал проверки ввода встроен в большинство современных фреймворков и, если он отсутствует, также может быть найден во внешних библиотеках, которые позволяют разработчику применять правила для каждого поля ввода (см. таблицу 1).

Кодирование вывода HTML

Помимо ограничения доступа к приложению, разработчикам веб-приложений необходимо уделять пристальное внимание данным, присылаемым с сервера. Современные веб-приложения обычно имеет базовую разметку HTML для структуры документа, CSS для стиля документа, JavaScript для логики приложения и пользовательский контент.

HTML имеет свободный формат. Браузеры стремятся не замечать множество ошибок при оформлении контента. Это может быть полезным для разработчика, поскольку некоторые ошибки не просто найти и исправить, однако рендеринг плохо сформированной разметки является основным источником уязвимостей. У атакующих есть возможность вводить контент на свои страницы, чтобы прорваться через контексты выполнения.

Приложения, получающие данные из таких источников как базы, должны обеспечивать верную фильтрацию контента, чтобы данные не нарушали работу приложения, но риск ошибок становится особенно высоким при рендеринге контента из ненадежного источника. Разработчики должны отфильтровывать входные данные, выходящие за рамки правил приложения. Как обработать контент, который содержит одну кавычку или фигурную скобку? Необходимо применять кодирование принимаемых данных.

Кодирование вывода – преобразование входящих данных в необходимый для работы приложения формат. Без соответствующего кодирования приложение может предоставить клиенту неверные данные, что может нарушить работу приложения или сделать опасным. Злоумышленник, который натыкается на недостаточное или ненадлежащее кодирование, знает, что это потенциальная уязвимость, которая может позволить кардинально изменить работу приложения.

Например, одним из первых клиентов системы является человек с именем Сандра Дей О’Коннор. Такой контент будет выведен на странице HTML:

<p>Уважаемая, Сандра Дей О'Коннор</ p>, отображается как: Уважаемый судья Сандра Дей О'Коннор.

Все работает в штатном режиме, страница создается, как и следовало ожидать. Но это может быть динамический интерфейс с архитектурой model / view / controller. Данные строки также появятся в JavaScript. Но вот что происходит, когда страница выводит это в браузер:

```
document.getElementById ('name').
innerText = 'Sandra Day O'Connor'
//
```

и в строке присутствует неэкранированная кавычка в имени человека.

Результатом является искаженный JavaScript. Это то, что хакеры ищут, чтобы прорваться че-

рез контекст выполнения и превратить невинные данные в опасный исполняемый код, например:

```
Sandra Day
O';window.location='http://evil.m
artinfowler.com/';
```

наш пользователь будет перенесен на враждебный сайт. Если, однако, мы правильно кодируем вывод для контекста JavaScript, текст будет выглядеть следующим образом:

```
'Sandra Day
O\';window.location=\'http://evil
.martinfowler.com/\';'
```

Это совершенно безобидная экранированная строка.

Существует несколько стратегий для кодирования JavaScript. Эта конкретная кодировка использует escape-последовательности для представления апострофа (« \'»), но также может быть безопасно представлена с помощью Unicode escape seqeence в кодировке – («'»).

Многие современные веб-фреймворки имеют механизмы безопасного хранения контента и защиты от зарезервированных символов. Однако большинство этих фреймворков включают механизм обхода этой защиты, и разработчики часто используют их либо из-за незнания, либо потому, что они полагаются на них, чтобы отображать исполняемый код, который, по их мнению, является безопасным.

Сегодня существует так много инструментов и фреймворков и так много контекстов кодирования (например, HTML, XML, JavaScript, PDF, CSS, SQL и т.д.), что создание их полного списка просто невозможно, однако ниже приведена таблица с контентом, которого следует избегать при кодировании на HTML (см. таблицу 2).

Если используется фреймворк, необходимо проверить его документацию на безопасные функции кодирования вывода. Если структура их не имеет, следует рассмотреть возможность смены фреймворка на другой, или же возникает задача создания кода вывода самостоятельно. Также стоит обратить внимание на то, что если фреймворк безопасно отображает HTML, это еще не означает, что он будет безопасно отображать JavaScript или PDF-файлы. Необходимо знать о кодировке конкретного контекста, на который написан инструмент кодирования.

Нужно отметить, что вложенные контексты рендеринга усложняют работу и их следует избегать, когда это возможно. Достаточно сложно получить одну строку вывода правильно, но, когда вы показываете URL-адрес, в HTML в JavaScript имеется три контекста, которые могут быть опасны лишь в одной строке. Если абсолютно невозможно избежать вложенных контекстов, обязательно нужно декомпозировать проблему на отдельные этапы, тщательно протестировать их, обращая особое внимание на порядок рендеринга. OWASP предоставляет некоторые рекомендации для кодирования контента в DOM, основанной на XSS, Cheat Sheet.

Защита передачи данных

При использовании обычного HTTP-соединения пользователи подвергаются рискам, связанным с фактом передачи данных в открытом виде. Злоумышленник, перехватывающий сетевой трафик между браузером пользователя и сервером, может «подслушивать» или даже изменять данные, полностью не обнаруженные при ата-

Таблица 2. Проверка документации фреймворка на безопасность функций кодирования вывода

Фреймворк	Закодированный контент	Опасный контент
Generic JS	innerText	innerHTML
JQuery	text()	html()
HandleBars	{{variable}}	{{{variable}}}
ERB	<%= variable %>	raw(variable)
JSP	<c:out value="${variable}"> or${fn:escapeXml(variable)}	${variable}
Thymeleaf	th:text="${variable}"	th:utext="${variable}"
Freemarker	${variable} (in escape directive)	<#noescape> or ${variable} without an escape directive
Angular	ng-bind	ng-bind-html (pre 1.2 and when sceProvider is disabled)

ке «человек посередине». Нет ограничений на то, что может сделать злоумышленник, включая кражу пользовательской сессии или его личной информации, инъекцию вредоносного кода, который будет выполняться браузером в контексте веб-сайта, или изменение данных, отправляемых пользователем на сервер.

Обычно разработчик не может контролировать сеть, в которой работают пользователи. Данные и пароли доступа к сети клиента могут быть перехвачены злоумышленниками (например, при работе в открытой беспроводной сети в кафе или в самолете). Если злоумышленник может подслушивать пользователя или вмешиваться в веб-трафик, то пользователю грозит опасность. Обмен данными не может быть доверен ни одной из сторон. Однако возможно защитить сеть и снизить риски с помощью HTTPS.

Первоначально HTTPS использовался главным образом для защиты конфиденциального веб-трафика, такого как финансовые транзакции, но теперь распространено мнение, что оно используется по умолчанию на многих сайтах, которые мы используем в повседневной жизни – таких как социальные сети и поисковые системы. Протокол HTTPS использует протокол безопасности транспортного уровня (TLS), являющийся преемником протокола Secure Sockets Layer (SSL) для защиты сообщений. Когда протокол настроен и используется правильно, он обеспечивает защиту от подслушивания и подделки, а также дает определенную гарантию того, что веб-сайт является тем самым, который пользователь намеревается использовать. Он обеспечивает конфиденциальность и целостность данных, а также аутентификацию веб-сайта.

С учетом многочисленных рисков, рекомендуется шифровать весь сетевой трафик. Компания Google – разработчик браузера Chrome объявила о своем намерении отказаться от ненадежного HTTP и будет предупреждать пользователей, когда сайт не использует HTTPS. Большинство реализаций HTTP/2 в браузерах будут поддерживать только связь через TLS.

Долгое время на применение HTTPS существовал ряд ограничений – это воспринималось как слишком дорогостоящее использование вычислительной техники для всего трафика. Протокол SSL и ранние версии протокола TLS поддерживают только один сертификат веб-сайта для каждого IP-адреса, но это ограничение было отменено в TLS с введением расширения протокола, называемого SNI (имя сервера), которое теперь поддерживается в большинство браузеров.

Стоимость получения сертификата от центра сертификации заметно уменьшилась, появились бесплатные сертификаты, такие как Let's Encrypt. Сегодня проблем для использования HTTPS практически нет.

Хеширование паролей пользователя

При разработке приложений нужно защитить приложение от злоумышленников, защитить пользователей от злоумышленников и от самих себя. Самый очевидный способ написать аутентификацию по паролю – это сохранить имя пользователя и пароль в таблице и сравнивать их при вводе из формы, показанной на рисунке 1. Данный способ аутентификации работоспособен, однако не лишен важных недостатков. Небезопасное хранение паролей создает риски как у разработчиков, так и у пользователей. В первом случае разработчик приложения или администратор базы данных, который может читать указанную выше таблицу application_user, имеет доступ к учетным данным всей базы пользователей. Часто упускается из виду риск того, что сами разработчики теперь могут использовать профили пользователей в приложении.

```
-- SQL
CREATE TABLE application_user (
 email_address VARCHAR(100) NOT NULL PRIMARY KEY,
 password VARCHAR(100) NOT NULL
)

# python
def login(conn, email, password):
 result = conn.cursor().execute(
 "SELECT * FROM application_user WHERE email_address = ? AND password =
?",
 [email, password])
 return result.fetchone() is not None
```

Рисунок 1. Программный код аутентификации пользователя по паролю

Сохранение учетных данных пользователей без соответствующей криптографической защиты представляет совершенно новый класс векторов атаки для пользователя. Дело в том, что пользователи используют одни и те же учетные данные на разных сайтах. Пользователь регистрируется на сайте с безобидным медиаконтентом, используя адрес электронной почты и пароль, который используется для своего банка. При этом создается такая ситуация, при которой база сайта с медиаконтентом становится средой хранения доступа к финансовым данным зарегистрировавшегося пользователя. Если злоумышленник похищает учетные данные из базы вышеуказанного сайта, то он может использовать их для попыток входа на банковские сайты.

Чтобы избежать такого сценария, следует хранить в базе данных не сам пароль, а его хеш. Криптографический алгоритм хэширования представляет собой одностороннее преобразование от ввода к выходному сигналу, из которого исходный вход практически невозможен. Например, пароль может быть «littlegreenjedi». Применяя Argon2 с «солью» «12345678» и параметрами командной строки по умолчанию, вы получите результат hex9b83665561e7ddf91b7fd0d4873894 bbd5afd4ac58ca397826e11d5fb02082a1. Теперь в базе не сохраняется пароль вообще, а только этот хеш.

«Соль» – это некоторые дополнительные данные, которые добавляются к паролю до его хэширования, так что два экземпляра заданного пароля не имеют одинакового значения хэш-функции. Чтобы проверить пароль пользователя, просто применяется тот же алгоритм хеширования к текстовому паролю, который отправляется, и, если они совпадают, то пароль действителен. Уровень безопасности стал выше, однако проблема в том, что не изменяется «соль»: каждый пользователь с паролем «littlegreenjedi» будет иметь тот же хеш в базе данных. Многие люди повторно используют один и тот же старый пароль. Таблицы поиска, сгенерированные с использованием наиболее часто встречающихся паролей и их вариаций, могут быть использованы для эффективного преобразования хешированных паролей. Если злоумышленник завладеет вашим хранилищем паролей, он может просто перекрестно ссылаться на таблицу поиска с помощью хэшей паролей и, по статистике, может извлечь много учетных данных за довольно короткий промежуток времени. Следует использовать динамическую «соль», которая будет формироваться при записи паролей приложения независимо от внешних факторов.

Настоящая выгода заключается в том, что она увеличивает диапазон возможных хэшей заданного пароля. Если мы снова используем «соль», то получим строку «BNY0LGUZWWIZ3BVP», а затем хеш с Argon2 67ddb83d85dc6f91b2e7 0878f333528d86674ecba1ae1c7aa5a94c7b4c6b 2c52. С другой стороны, если мы используем «M3WIBNKBYVSJW4ZJ», то получаем 64e7d4 2fb1a19bcf0dc8a3533dd3766ba2d87fd7ab75eb7ac b6c737593cef14e. Теперь, если злоумышленник получает доступ к хранилищу хэша паролей, ему гораздо сложней будет расшифровать эти пароли.

«Соль» должна быть абсолютно уникальной для каждого пользователя. OWASP рекомендует 32 или 64-битную «соль», если вы можете управлять им, а NIST требует как минимум 128 бит.

Заключение

С развитием IT-сферы, где появляются новые уязвимости и новые методы защиты от них, требования безопасности к сетям и сетевым приложениям возрастают. Чтобы поддерживать высокий уровень безопасности, требуются разные средства, однако существуют некие базовые правила, соблюдая которые, можно обезопасить себя от большинства типовых атак.

Следует фильтровать и экранировать любые данные, которые веб-приложение получает из разных источников. Каждое поле нужно разделять по типу данных и проводить соответствующие проверки на тип. Любые данные, которые приложение отдает клиенту, необходимо кодировать и обрабатывать. Иначе возникает риск для пользователя или самого сайта из-за неверно распознанного контента, который может принести вред.

Для шифрования данных, передаваемых по сетям, следует использовать протокол HTTPS. Это поможет уберечь пользователя от перехвата трафика или прослушивания сети со стороны злоумышленников. Важные данные пользователей лучше не хранить в базах данных, а создавать и хранить хеши этих данных. Для этого следует применять современные методы шифрования данных с использования своей уникальной «соли».

Литература

1. Umesh Hodeghatta Rao, Umesha Nayak The InfoSec Handbook: An Introduction to Information Security. Apress Media, 2014. – 392 p.
2. Бармен Скотт Разработка правил информационной безопасности. Пер. с англ. М.: ИД «Вильямс», 2002. – 208 с.

3. Douglas J. Landoll. Information Security Policies, Procedures, and Standards: A Practitioner's Reference. Auerbach Publications, 2016. – 254 p.
4. Karlova T.V., Bekmeshov A.Y., Sheptunov S.A., Kuznetsova N.M. Methods Dedicated to Fight Against Complex Information Security Theats on Automated Factories Systems // 2016 IEEE Conference on Quality Management, Transport and Information Security, Information Technologies (IT&MQ&IS). Proceedings // Institute of Electrical and Electronics Engineers Inc., 2016. Pp. 72-76.
5. Карлова Т.В., Кузнецова Н.М., Бекмешов А.Ю. Оптимизация доступа к информационным ресурсам // Вестник Брянского ГТУ. 2015. №3 (47). С. 135-138.
6. Sheptunov S.A., Larionov M.V., Suhanova N.V. e.a. Optimization of the complex software reliability of control systems // 2016 IEEE Conference on Quality Management, Transport and Information Security, Information Technologies, 2016. P. 189-192.
7. Грибунин В.Г. Комплексная система защиты информации на предприятии. М.: «Академия», 2009. – 416 с.
8. ГОСТ Р 54593-2011 Информационные технологии. Свободное программное обеспечение. Общие положения М.: Стандартинформ, 2012. – 12 с. // URL: http://protect. gost.ru/v.aspx?control=7&id (д.о. 17.10.2018).
9. ГОСТ Р ИСО/МЭК 12207-2010 Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств М.: Стандартинформ, 2011. – 106 с. // URL: http:// pro-tect.gost.ru/document1.aspx?control=31&baseC=6&page=0&month= (д.о. 17.10.2018).
10. ГОСТ Р ИСО/МЭК 29362-2013 Информационная технология. Функциональная совместимость веб-служб. Профиль вложений WS-1. Версия 1.0 М.: Стандартинформ, 2014. – 32 с. // URL: http://protect. gost.ru/v.aspx?control=7&id (д.о. 17.10.2018).

Получено 01.10.2018

Юлдашев Тимур Захарович, аспирант ФГАУН Институт конструкторско-технологической информатики Российской академии наук (ИКТИ РАН). Тел. +7-929-916-41-51. E-mail: info@dlay.ru

Шептунов Сергей Александрович, д.т.н., директор ИКТИ РАН. Тел. (8-499) 978-57-15. E-mail: ship@ikti.ru

BASES OF INFORMATION SECURITY IN DEVELOPING SITES

Yuldashev T.Z., Sheptunov S.A.
Institute for Design-Technological Informatics RAS, Moscow, Russian Federation
E-mail: info@dlay.ru

In the context of the widespread introduction of automated data processing, special importance is attached to the issue of protecting information from unauthorized access to it by intruders. Security assessment at the web development stage should be carried out by analyzing threats using different criteria (for example, program code, page markup, passwords, etc.). The article presents the main mistakes of site developers during creating applications, analyzes the methods of eliminating them and increasing the level of security for the user. Based on the analysis, a conclusion was drawn on the need to pay considerable attention to network security during developing websites and using them. Knowledge of the basics of information security by users can help to protect user data and the application itself from the most popular modern threats.

Keywords: *network security, site development, information security, application creation*

DOI: 10.18469/ikt.2018.16.4.10

Yuldashev Timur Zakharovich, Institute for Design and Technological Informatics of RAS; Vadkovskiy pereulok, 18, str. 1A, Moscow, 127055, Russian Federation; PhD Student. Tel. +79299164151. E-mail: info@dlay.ru

Sheptunov Sergey Aleksandrovich, Institute for Design and Technological Informatics of RAS; Vadkovskiy pereulok, 18, str. 1A, Moscow, 127055, Russian Federation; Director, PhD in Technical Sciences. Tel. +74999785715. E-mail: ship@ikti.org.ru

References

1. Umesh Hodeghatta Rao, Umesha Nayak. *The InfoSec Handbook: An Introduction to Information Security. Apress Media*, 2014. 392 p.
2. Barman S. *Writing Information Security Policies*. Boston, Indianapolis, London, Munich, New York, San Francisco, Pearson Education, 2002. (Russ. ed. Barmen S. *Razrabotka pravil informatsionnoy bezopasnosti*. Moscow, ID «Vil'yams», 2002. 208 p.
3. Douglas J. Landoll. *Information Security Policies, Procedures, and Standards: A Practitioner's Reference.* Auerbach Publications, 2016. 254 p.
4. Karlova T.V., Bekmeshov A.Yu., Sheptunov S.A., Kuznetsova N.M. Methods Dedicated to Fight Against Complex Information Security Theats on Automated Factories Systems. *2016 IEEE Conference on Quality Management, Transport and Information Security, Information Technologies (IT&MQ&IS),* 2016, pp. 72-76.
5. Karlova T.V., Kuznetsova N.M., Bekmeshov A.YU. Optimizatsiya dostupa k informatsionnym resursam [Optimization of access to information resources]. *Vestnik Bryanskogo gosudarstvennogo tekhnicheskogo universiteta,* 2015, no. 3, pp. 135-138 pp.
6. Sheptunov S.A., Larionov M.V., Suhanova N.V., Kabak I.S., Alshynbaeva D.A. Optimization of the complex software reliability of control systems. *2016 IEEE Conference on Quality Management, Transport and Information Security, Information Technologies*, 2016, pp. 189-192.
7. Gribunin V.G. *Kompleksnaya sistema zashchity informatsii na predpriyatii* [Complex information security system at the enterprise]. Moscow, Akademiya Publ., 2009. 416 p.
8. GOST R 54593-2011 Informatsionnyye tekhnologii. Svobodnoye programmnoye obespecheniye. Obshchiye polozheniya. Moscow, Standartinform Publ., 2012. 12 p. Available at: http://protect.gost.ru/v.aspx?control=7&id=179084 (accessed 17.10.2018).
9. GOST R ISO/MEK 12207-2010 Informatsionnaya tekhnologiya. Sistemnaya i programmnaya inzheneriya. Protsessy zhiznennogo tsikla programmnykh sredstv. Moscow, Standartinform Publ., 2011. 106 p. Available at: http://protect.gost.ru/document1.aspx?control= 31&baseC=6&page=0&month=2&year=2018&search=12207&id=176990 (accessed 17.10.2018).
10. GOST R ISO/MEK 29362-2013 Informatsionnaya tekhnologiya. Funktsional'naya sovmestimost' veb-sluzhb. Profil' vlozheniy WS-1. Versiya 1.0. Moscow, Standartinform Publ., 2014. 32 p. Available at: http://protect.gost.ru/v.aspx?control=7&id=186336 (accessed. 17.10.2018).

Received 01.10.2018

УДК 005.62+658.562.44

СИСТЕМА МОНИТОРИНГА КАК ОСНОВА СОВЕРШЕНСТВОВАНИЯ КАЧЕСТВА ПРОИЗВОДСТВЕННЫХ ПРОЦЕССОВ

Фомина А.Э.[1], Карлова Т.В.[2]

[1]Московский государственный технологический университет «СТАНКИН», Москва, РФ

[2]Институт конструкторско-технологической информатики Российской академии наук, Москва, РФ

E-mail: karlova-t@yandex.ru

Рассмотрены вопросы внедрения и необходимости применения автоматизированной системы мониторинга, позволяющей в реальном времени не только контролировать сам процесс, но и разрабатывать инновационные процессы производства. Мониторинг представляет собой внутреннюю процедуру, основанную на индикаторах и результатах, целью которой является выявление факторов, нарушающих функционирование системы. Одновременно с данным положением мониторинг является инструментом сбора информации и отчетности. Системы мониторинга, в зависимости от поставленных перед ними задач, условно можно разделить на использующие пассивный и активный методы сбора информации. Автоматизация системы мониторинга в космической промышленности позволит повысить качество работы спутников, обеспечивая тем самым высокий уровень эффективности обработки информации, который позволит адекватно, с увеличенной степенью оперативности оценить и проанализировать реально происходящую ситуацию в конкретный момент времени. Все вышеперечисленное позволит в итоге принять взвешенное и целенаправленное управленческое решение.

Ключевые слова: *система мониторинга, автоматизация процессов, космическая связь, спутниковая связь, качество, активный мониторинг, пассивный мониторинг*