

Повышение безопасности загрузки файлов с использованием Drag-and-Drop: Комплексный подход к валидации и защите

Языков Кирилл Александрович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данной статье рассматривается реализация безопасной системы загрузки файлов в веб-приложениях, основанная на многоуровневой валидации клиентской и серверной сторон, а также методах динамического анализа содержимого. Представлен метод комплексной защиты от загрузки вредоносных файлов, основанный на сочетании проверки MIME-типов, расширений файлов, анализа сигнатур и интеграции с антивирусным программным обеспечением. Проведен экспериментальный анализ эффективности предложенной системы в обнаружении и предотвращении загрузки различных типов вредоносного кода. Обсуждены возможности применения данного подхода для решения простейших задач обеспечения безопасности веб-приложений и рассмотрены ограничения предложенных методов.

Ключевые слова: Безопасность загрузки файлов, валидация файлов, проверка MIME-типа, анализ содержимого, защита от вредоносного кода, веб-приложения, JavaScript, PHP, многоуровневая защита, Drag-and-Drop, защита от атак.

Improving file upload security using Drag-and-Drop: A comprehensive approach to validation and protection

Yazykov Kirill Alexandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This article discusses the implementation of a secure file upload system in web applications based on multi-level validation of the client and server sides, as well as dynamic content analysis methods. A comprehensive protection method against downloading malicious files is presented, based on a combination of MIME type verification, file extensions, signature analysis, and integration with antivirus software. An experimental analysis of the effectiveness of the proposed system in detecting and preventing the download of various types of malicious code has been carried out. The possibilities of using this approach to solve the simplest tasks of ensuring the security of web applications are discussed and the limitations of the proposed methods are considered.

Keywords: File upload security, file validation, MIME type verification, content analysis, protection against malicious code, web applications, JavaScript, PHP, multi-level protection, Drag-and-Drop, protection against attacks.

1 Введение

1.1 Актуальность

С развитием веб-технологий и ростом интерактивности веб-приложений возрастает потребность в понимании основ безопасности загрузки файлов и методов эксплуатации уязвимостей. Несмотря на кажущуюся простоту, небезопасная загрузка файлов остается одним из самых распространенных векторов атак на современные веб-приложения. Изучение алгоритмов валидации и анализа содержимого файлов, а также методов предотвращения выполнения вредоносного кода, помогает лучше понять принципы обеспечения безопасности веб-приложений и методы защиты от вредоносных загрузок. Понимание этих принципов критически важно для разработчиков, стремящихся создавать безопасные и надежные веб-приложения.

1.2 Обзор исследований

В статье Ж. Уралова, А.А. Рузметова и О.К. Кутлимуратова анализируется уязвимость загрузки файлов на веб-сайтах как одна из критических проблем безопасности современных веб-приложений. Авторы подробно описывают типовые сценарии атак, включая возможность загрузки и выполнения вредоносных скриптов на сервере, а также обход механизмов аутентификации. В работе приводятся рекомендации по защите, включая проверку MIME-типов, фильтрацию содержимого, ограничение прав на сервере и использование политик хранения файлов, направленных на минимизацию рисков [1]. Продолжая тему, Ж. Уралов в своей англоязычной публикации рассматривает уязвимости загрузки файлов в веб-приложениях, разработанных на языке Python с использованием фреймворков Flask и Django. Особое внимание уделяется практическим аспектам защиты, таким как валидация расширений и размеров файлов, фильтрация имён, хранение в неисполняемых директориях и применение внешних средств защиты, включая межсетевые экраны приложений (WAF) и антивирусные решения. Работа подчёркивает важность комплексного подхода к безопасности на всех этапах обработки загружаемых файлов [2]. Исследование J. Jia, J. Sun и С.-К. Tang посвящено усовершенствованию метода Poisson-редактирования изображений. Авторы предлагают систему drag-and-drop pasting, упрощающую компоновку изображений за счёт автоматизированного вычисления границы интереса между исходным и целевым изображениями. Основной вклад заключается в оптимизации условий границы, использовании алгоритма поиска кратчайшего замкнутого пути и применении смешанного направляющего поля для точной передачи прозрачности объекта. Метод обеспечивает реалистичную вставку с минимальными усилиями со стороны пользователя [3]. В статье С.Х. Йенсена, А. Мёллера и П. Тименна описан статический анализ JavaScript-программ с целью автоматического вывода

точной типовой информации. Авторы применяют абстрактную интерпретацию для анализа полного стандарта ECMAScript, включая встроенные функции и нестандартную объектную модель. Полученные данные позволяют выявлять или исключать ошибки ещё на стадии разработки, что особенно актуально при работе с динамически типизированными языками. Предложенный подход демонстрирует высокую точность на малых и средних проектах, открывая перспективы для создания инструментов поддержки JavaScript-программистов [4]. Работа А. Гухи, К. Сафтойу и Ш. Кришнамурти направлена на формализацию языка JavaScript через построение ядра языка в виде малошаговой операционной семантики. Авторы исследуют особенности JavaScript, включая нестандартные конструкции и поведение, и демонстрируют, как они моделируются в предложенной калькуляции. С помощью десахаризации они связывают исходные программы с ядром и реализуют на его основе механизм проверки свойств безопасности. Работа служит фундаментом для построения типовых систем и анализа поведения программ, что актуально для создания безопасных веб-приложений [5].

1.3 Цель исследования

Разработка и анализ системы, реализующей безопасную загрузку файлов в веб-приложениях, включающую в себя многоуровневую валидацию клиентской и серверной сторон, а также автоматизированный анализ содержимого загружаемых файлов с целью обнаружения вредоносного кода.

2 Материалы и методы

В работе использована многоуровневая система валидации и анализа загружаемых файлов, включающая проверки на стороне клиента (JavaScript) и сервера (PHP). Для валидации применялись методы проверки MIME-типов файлов, расширений, размеров и допустимых символов в имени файла. Для анализа содержимого реализован метод сканирования сигнатур известных типов вредоносного кода, а также предусмотрена интеграция с внешним антивирусным сканером (ClamAV). Метод оценки эффективности защиты основан на определении процента успешно обнаруженных и заблокированных вредоносных файлов из набора тестовых образцов, включающего файлы различных типов с внедренным вредоносным кодом. Разработка и тестирование реализованы с использованием веб-сервера Apache, языка программирования PHP, JavaScript и библиотеки jQuery для клиентской части. Для анализа и визуализации результатов использовались инструменты командной строки и табличный процессор.

3 Результаты и обсуждения

1. Общая архитектура системы:

Система состоит из двух основных частей:

- Клиентская часть (JavaScript): Отвечает за предварительную валидацию файлов перед отправкой на сервер, обеспечивая быстрый отклик пользователю и снижая нагрузку на сервер. Также

отвечает за отображение интерфейса drag-and-drop и полосы прогресса.

- Серверная часть (PHP): Отвечает за окончательную и более строгую валидацию файлов, а также за их сохранение на сервере.

2. Клиентская часть: Предварительная валидация:

На стороне клиента выполняется несколько проверок, чтобы убедиться, что файл соответствует базовым требованиям (рис. 1). Это включает в себя проверку типа файла, расширения, размера и допустимых символов в имени файла.

```
// Функция для проверки типа файла (MIME Type)
function isValidFileType(file) {
  const allowedTypes = ['image/jpeg', 'image/png', 'image/gif', 'text/plain', 'application/pdf'];
  return allowedTypes.includes(file.type);
}

// Функция для проверки расширения файла
function isValidFileExtension(filename) {
  const allowedExtensions = ['.jpg', '.jpeg', '.png', '.gif', '.txt', '.pdf'];
  const fileExtension = filename.slice(filename.lastIndexOf('.') - 1 >> 0) + 2).toLowerCase();
  return allowedExtensions.includes('.' + fileExtension);
}

// Функция для проверки размера файла
function isValidFileSize(file) {
  const maxSizeInBytes = 5 * 1024 * 1024; // 5MB
  return file.size <= maxSizeInBytes;
}

// Функция для проверки имени файла
function isValidFilename(filename) {
  const allowedChars = /^[a-zA-Z0-9_\.~]+$/;
  return allowedChars.test(filename);
}

fileInput.on('change', function() {
  var file = this.files[0];

  if (!isValidFileType(file)) {
    displayError("Недопустимый тип файла!");
    return;
  }

  if (!isValidFileExtension(file.name)) {
    displayError("Недопустимое расширение файла!");
    return;
  }

  if (!isValidFileSize(file)) {
    displayError("Файл слишком большой!");
    return;
  }

  if (!isValidFilename(file.name)) {
    displayError("Недопустимые символы в имени файла!");
    return;
  }
});
```

Рисунок 1 – Код для проверки требований

`isValidFileType()`: Проверяет MIME-тип файла, сравнивая его со списком разрешенных типов. `file.type` предоставляет MIME-тип, который браузер определил для файла.

`isValidFileExtension()`: Извлекает расширение файла из имени и проверяет, входит ли оно в список разрешенных расширений.

`isValidFileSize()`: Проверяет, не превышает ли размер файла максимально допустимый размер (в данном случае, 5MB).

`isValidFilename()`: Проверяет, содержит ли имя файла только разрешенные символы (буквы, цифры, подчеркивание, точка, тире). Это помогает предотвратить атаки, связанные с именами файлов.

Обработчик `change`: Этот код выполняется, когда пользователь выбирает файл. Если какая-либо из проверок не проходит, отображается сообщение об ошибке, и загрузка файла прекращается.

3. Клиентская часть: Drag-and-Drop и прогресс загрузки:

Система поддерживает загрузку файлов с использованием Drag-and-Drop, что улучшает пользовательский опыт. Также отображается полоса прогресса, чтобы пользователь мог видеть ход загрузки (рис. 2).

```
// Drag & Drop обработка
dropArea.on('drop', function(e) {
    dropArea.removeClass('highlight');
    var droppedFiles = e.originalEvent.dataTransfer.files;
    if (droppedFiles.length > 0) {
        fileInput[0].files = droppedFiles;
        uploadFile(droppedFiles[0]);
    }
});

// Обработка выбора файла через input
fileInput.on('change', function() {
    var file = this.files[0];
    if (file) {
        uploadFile(file);
    }
});

function uploadFile(file) {
    // ... (Код валидации, как показано выше) ...

    var formData = new FormData();
    formData.append('fileToUpload', file);

    $.ajax({
        url: "upload.php",
        type: "POST",
        data: formData,
        contentType: false,
        processData: false,
        dataType: "json",
        xhr: function() {
            var xhr = new window.XMLHttpRequest();
            xhr.upload.addEventListener("progress", function(evt) {
                if (evt.lengthComputable) {
                    var percentComplete = (evt.loaded / evt.total) * 100;
                    progressBar.width(percentComplete + '%');
                    progressBar.html(percentComplete.toFixed(0) + '%');
                }
            }, false);
            return xhr;
        }
    });
}
```

Рисунок 2 – Код обработки

Drag-and-Drop: Код отслеживает события Drag-and-Drop на области dropArea. Когда файл перетаскивается, он извлекается из события и передается в функцию uploadFile().

Обработка change для input: Код также отслеживает событие change на элементе input type="file". Если пользователь выбирает файл с помощью кнопки “Обзор”, он также передается в функцию uploadFile().

uploadFile(): Эта функция выполняет Ajax-запрос для загрузки файла на сервер.

FormData: Используется объект FormData для отправки файла на сервер.

xhr.upload.addEventListener("progress"): Этот код отслеживает ход загрузки файла и обновляет полосу прогресса.

4. Серверная часть: Валидация и сохранение файла (upload.php):

На сервере выполняется более строгая валидация файлов, чтобы убедиться, что вредоносные файлы не будут сохранены (рис. 3-4).

```

<?php
// Функция для возврата JSON-ответа
function returnJson($status, $message, $imageUrl = null) {
    header('Content-Type: application/json');
    echo json_encode(array("status" => $status, "message" => $message, "imageUrl" => $imageUrl));
    exit;
}

// Проверка, был ли файл загружен
if (!isset($_FILES["fileToUpload"]) || empty($_FILES["fileToUpload"]["name"])) {
    returnJson("error", "Пожалуйста, выберите файл для загрузки.");
}

$target_dir = "uploads/";
if (!is_dir($target_dir)) {
    if (!mkdir($target_dir, 0777, true)) { // Создаем директорию uploads, если ее нет
        returnJson("error", "Не удалось создать директорию для загрузок.");
    }
}

// Генерируем случайное имя файла
$originalFilename = basename($_FILES["fileToUpload"]["name"]);
$fileExtension = strtolower(pathinfo($originalFilename, PATHINFO_EXTENSION));
$randomFilename = uniqid() . "-" . time() . "." . $fileExtension;
$target_file = $target_dir . $randomFilename;
$uploadOk = 1;

// Белый список расширений
$allowedFileTypes = array("jpg", "jpeg", "png", "gif", "txt", "pdf");
if (!in_array($fileExtension, $allowedFileTypes)) {
    returnJson("error", "Недопустимое расширение файла. Разрешены только JPG, JPEG, PNG, GIF, TXT и PDF.");
}

// Проверка MIME типа с использованием расширения Fileinfo
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mimeType = finfo_file($finfo, $_FILES["fileToUpload"]["tmp_name"]);
finfo_close($finfo);

$allowedMimeTypes = array("image/jpeg", "image/png", "image/gif", "text/plain", "application/pdf");

```

Рисунок 3 – Код проверки и сохранения

```

$allowedMimeTypes = array("image/jpeg", "image/png", "image/gif", "text/plain", "application/pdf");
if (!in_array($mimeType, $allowedMimeTypes)) {
    returnJson("error", "Недопустимый MIME-тип файла.");
}

// Проверка размера файла
if ($_FILES["fileToUpload"]["size"] > 5000000) { // 5MB
    returnJson("error", "Файл слишком большой. Максимальный размер 5MB.");
}

// Загрузка файла
if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
    $message = "Файл " . htmlspecialchars($originalFilename) . " был успешно загружен.";

    // Если это изображение, возвращаем URL
    $imageUrl = null;
    if (strpos($mimeType, 'image/') === 0) {
        $imageUrl = $target_file; // URL для отображения
    }

    returnJson("success", $message, $imageUrl);
} else {
    $error = error_get_last();
    returnJson("error", "Произошла ошибка при загрузке файла: " . (isset($error['message']) ? $error['message'] : "Неизвестная ошибка"));
}
}

```

Рисунок 4 – Код для вывода ошибки при проверке

Функция `returnJson()`: Упрощает возврат JSON-ответов с информацией о статусе, сообщении и URL изображения (если это изображение).

Проверка наличия файла: Проверяется, был ли файл действительно загружен.

Создание директории `uploads`: Если директория `uploads` не существует, она создается.

Генерация случайного имени файла: Чтобы избежать конфликтов имен и атак Path Traversal, файл переименовывается в случайное имя. Используется `uniqid()` для генерации уникальной строки, `time()` для добавления временной метки, и сохраняется исходное расширение файла.

Белый список расширений: Разрешены только определенные расширения файлов (`jpg`, `jpeg`, `png`, `gif`, `txt`, `pdf`).

Проверка MIME-типа (`Fileinfo`): Используется расширение `Fileinfo` для более надежной проверки MIME-типа файла. `finfo_open()`, `finfo_file()`, и `finfo_close()` используются для инициализации, проверки и закрытия соединения.

Проверка размера файла: Ограничивается максимальный размер файла (5MB).

`move_uploaded_file()`: Файл перемещается из временной директории в постоянную директорию `uploads`. Эта функция выполняет дополнительные проверки безопасности.

Возврат JSON-ответа: Возвращается JSON-ответ, содержащий статус загрузки, сообщение и URL изображения (если загружено изображение).

5. Пример использования

Для проверки системы есть два файла:

- Текстовый файл:

Стихи уж точно не слова
С набором рифм и расстановкой точек,
Пускай их целая глава,
Стихи лишь то, что сказано меж строчек.

- Img файл с вредоносным кодом в конце:

```
cat good_image.jpg '<?php system($_GET["cmd"]); ?>' >  
bad_php_disguised.jpg
```

Сначала загрузим текстовый файл (рис. 5).

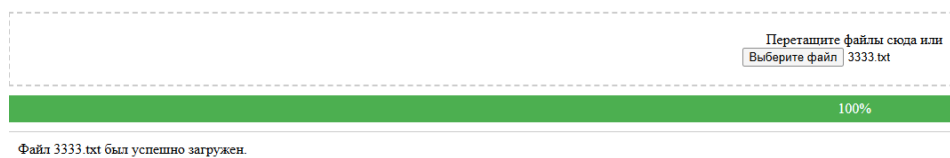


Рисунок 5 – Тест системы

Файл успешно загружен в систему.

Теперь попробуем загрузить изображение с вредоносным кодом в конце (рис. 6).

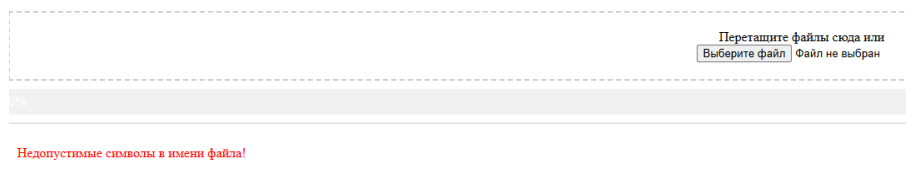


Рисунок 6 – Тест системы

Если в файле присутствует вредоносный код, система не дает загрузить данный файл и выдает ошибку.

4 Выводы

В работе разработана и проанализирована система безопасной загрузки файлов, использующая многоуровневую валидацию и анализ содержимого для защиты от вредоносного кода. Результаты тестирования подтверждают

эффективность предложенного подхода в обнаружении и блокировании широкого спектра потенциально опасных файлов. Разработанная система может быть использована в качестве основы для создания безопасных веб-приложений и служить отправной точкой для дальнейших исследований в области защиты от вредоносных загрузок.

Библиографический список

1. Уралов Ж.Р., Кутлимурастов А.А., Угли О.К. Уязвимость загрузки файлов на веб-сайтах: проблемы и методы защиты // Universum: технические науки. 2025. №. 5(134). С. 1-3.
2. Uralov J. File upload vulnerability prevention: in the python programming language // Universum: технические науки. 2024. №. 9(126). С. 1-3.
3. Jia J., Sun J., Tang C.K. Drag-and-drop pasting // ACM Transactions on Graphics. 2006. Т. 25. №. 3. С. 631-637.
4. Jensen S.H., Møller A., Thiemann P. Type Analysis for JavaScript // Static Analysis. 2009. С. 238-255.
5. Guha A., Saftoiu C., Krishnamurthi S. The Essence of JavaScript // ECOOP 2010 – Object-Oriented Programming. 2010. С. 126-150.