K. A. Samosadniy, A. A. Petukhov

Detection of Malicious Flash Banner Advertisements

Keywords: flash advertisements, malicious, detection

The paper addresses the problem of detecting malicious flash advertisements. As a result, detection method based on dynamic analysis that modify flash application and execute it in Adobe Flash player is proposed and evaluated on synthetic and real world examples.

К. А. Самосадный, А. А. Петухов

ОБНАРУЖЕНИЕ ВРЕДОНОСНЫХ FLASH-БАННЕРОВ

Введение

Flash-баннер — приложение, написанное для браузерного плагина Adobe Flash Player и встроенное в некоторую HTML-страницу. Популярность вредоносных Flash-приложений обуславливается следующими факторами:

- большим количеством уязвимостей в Adobe Flash Player;
- использованием устаревших версий ПО [1], содержащих хорошо известные уязвимости;
- поддержкой данной технологии подавляющим большинством компьютеров, подключенных к Сети [1].

Популярность данной технологии обусловила спектр различных угроз:

- эксплуатация уязвимостей в самом Adobe Flash Player;
- эксплуатация уязвимостей в приложениях, написанных на ActionScript[1];
- эксплуатация уязвимостей в системах, использующих Adobe Flash (баннерная реклама и т. п.).

Первые возникают обычно в том случае, когда отсутствует корректная фильтрация пользовательских данных и когда допущены ошибки в реализации Flash-приложений.

Adobe Flash позволяет гибко взаимодействовать с серверами в Сети, предоставляя для этого широкие возможности. Например, механизм сокетов, классы для загрузки данных Сети и для открытия произвольной страницы в браузере. При отсутствии фильтрации пользовательского ввода, попадающего в подобные методы, возможны уязвимости классов XSS[2] и CSRF[3] [2, 3]. Если SWF-файл проигрывается встроенным в браузер плагином, то во всех HTTP-запросах, происходящих при подобных действиях, будут проставлены заголовки, как будто запрос происходит от имени браузера. В частности, запрос также будет содержать заголовок Cookie, который широко применяется для аутентификации на различных сайтах. Единственным отличием будет наличие HTTP-заголовка “X-Requested-With”, в котором будет указано, что данные были запрошены плагином Flash Player.

Помимо легитимных приложений, Adobe Flash в силу своей распространенности также используется и злоумышленниками. Использование старых версий Adobe Flash Player и большое количество уязвимостей [4] позволяют злоумышленникам эксплуатировать их, таким образом обходя механизмы защиты как Adobe Flash Player, так и операционной системы [5]. Также злоумышленники пользуются Adobe Flash для того, чтобы скрыть вредоносный JavaScript-код, который может:

- перенаправить пользователя на вредоносный сайт, содержащий код, эксплуатирующий уязвимости, которые подходят для установленной у пользователя версии Adobe Flash Player, операционной системы и т. д.;

[1] ActionScript — язык программирования, который используется в Adobe Flash.

[2] XSS (Cross-Site Scripting) — внедрение вредоносного кода на HTML-страницу.

[3] CSRF (Cross-Site Request Forgery) — уязвимость в приложении, при наличии которой злоумышленник может вынудить пользователя сделать некоторые действия без его ведома.

• эксплуатировать уязвимости классов XSS или CSRF.

Многие сайты позволяют использовать Adobe Flash для демонстрации рекламы на странице. Подобный баннер может появиться на странице двумя путями:

• рекламодатель непосредственно договаривается с владельцем сайта о размещении своей рекламы;

• сайт становится участником некоей сети, в которой права на договор с рекламодателем делегируются третьему лицу.

В первом случае из-за недостатка знаний или по невнимательности в качестве рекламодателя могут выступать злоумышленники, которые договариваются о размещении вредоносного Flash-приложения под видом баннера. При этом, если баннер был вставлен на страницу без учета некоторых особенностей технологии Adobe Flash[4], злоумышленник получает возможность эксплуатации уязвимостей, описанных выше. Владелец сайта в таком случае рискует потерять репутацию и посетителей, так как при обнаружении вредоносной активности о ней будут предупреждать поисковые системы и браузеры, например: Google Safe Browsing или Yandex Safe Browsing. Во втором случае администрация ресурса, как правило, не может контролировать код, с помощью которого вставляется баннер. К тому же это обычно запрещено правилами сети. В таком случае при неграмотной настройке становятся уязвимыми все участники подобной сети.

1. Механизмы безопасности в Adobe Flash Player

1.1. Модель контролируемых сред выполнения

Основным механизмом безопасности в Flash Player является контролируемая среда выполнения (КСВ; англ. Sandbox — песочница). КСВ — логическое разбиение на группы безопасности, используемое Flash Player для ограничения ресурсов. Каждая КСВ изолирована от операционной системы, файловой системы, сети, приложений и от других КСВ.

Flash Player помещает SWF-файлы, загруженные из Сети (например, с веб-сайтов) в КСВ, соответствующие домену сайта, с которого они были загружены. Каждому домену выделяется отдельная КСВ, которая выбирается на основе принципа одинакового источника. Два URL-адреса считаются адресами из одного источника, если у них полностью совпадает доменное имя, протокол и номер порта.

Любые два SWF-файла, запущенные в одной КСВ, могут взаимодействовать друг с другом без ограничений: считывать «чужие» данные и вызывать «чужие» функции. SWF-файлы могут взаимодействовать и с другими КСВ (и обращаться к серверам с другим DNS-именем), но только в соответствии со специальными правилами безопасности и настройками.

По умолчанию:

• Любые два SWF-файла в одной КСВ могут свободно взаимодействовать друг с другом.

• SWF-файл может читать файлы, загруженные с домена, соответствующего КСВ, к которой этот файл относится.

• SWF-файл может отправлять данные в Сеть.

• SWF-файл не может читать данные из другой КСВ.

Для того чтобы SWF-файл мог прочитать данные другой КСВ, он должен получить явное разрешение владельцев другой КСВ.

1.2. Байт-код и изоляция кода

Байт-код — это внутренний код, который выполняется на виртуальной машине и не может быть непосредственно выполнен на процессоре. Машинные инструкции, которые выполняются во время работы Flash-приложения, являются частью предопределенного в Flash Player набора

[4] См. раздел 1.3.

инструкций и получаются в ходе трансляции байт-кода SWF-файла (и это только после проверки соответствия данных инструкций типам данных, полученных от предыдущих инструкций, и применения политик безопасности). Эти ограничения должны гарантировать, что Flash-приложение не может повлиять на другие приложения или данные.

Flash-приложение компилируется в байт-код на машине разработчика в процессе публикации этого приложения. Верификация байт-кода происходит на компьютере пользователя с помощью специальной компоненты Adobe Flash Player. Верификация состоит из определения типов, которые каждый байт-код получает на вход и выдает в качестве результата[5]. Это происходит посредством построения карты типов для значений в стеке значений для каждого байт-кода. Таким образом, байт-кодовая программа, которая использует неправильные байт-коды (с учетом их типов операндов), может быть обнаружена и отклонена.

Flash-приложение может читать файлы, создавать сетевые соединения и взаимодействовать с другими SWF-файлами. Каждая из вышеперечисленных операций, в конечном счете, выполняется подпрограммами, которые являются частью и контролируются внутренним кодом Flash Player, и только после проверки соблюдения всех используемых политик доступа, установленных моделью безопасности.

1.3. Доступ к контейнеру

SWF-файл при воспроизведении может получить доступ к содержащему его приложению. Например, для доступа к странице, содержащей Flash-приложение, у объекта, с помощью которого оно встроено в страницу, должен быть выставлен атрибут allowScriptAccess [6].

Этот атрибут может принимать три значения:

• sameDomain (значение по умолчанию) — возможно взаимодействие только со страницами, происходящими из того же источника (согласно принципу одинакового источника), с которого был загружен SWF-файл;

• never — невозможно получить доступ для любых SWF-файлов;

• always — полный доступ для всех SWF-файлов вне зависимости от их происхождения.

Получив доступ к странице, его содержащей, SWF-файл может исполнять любой JavaScript-код на этой странице посредством методов класса ExternallInterface:

• call — вызов произвольной функции, содержащейся в данном документе (в частности, все встроенные функции языка). Например, чтобы выполнить произвольный код “somecode” на языке JavaScript, нужно выполнить следующую функцию

ExternallInterface.call (“eval”, “somecode”);

• addCallback — позволяет описывать функции, которые можно будет вызывать непосредственно из JavaScript-кода.

Таким образом, эти методы позволяют организовывать взаимодействие между HTML-страницей и содержащимся на ней SWF-файлом, что может быть использовано для того, чтобы полностью изменить страницу или обойти ограничения на взаимодействие с серверами, предписываемые политиками безопасности Adobe Flash Player.

Например, владелец домена может явно запретить SWF-файлам, загруженным с его домена, отправлять данные на любые сервера, кроме своего, и это ограничение будет выполняться для Flash-приложения. Однако это ограничение не действует для содержащей приложение страницы, и SWF-файл может обойти его, исполняя JavaScript-код в контексте страницы с помощью класса ExternallInterface.

Атрибут allowNetworking помимо ограничения сетевого взаимодействия также может ограничивать и взаимодействие Flash-приложения со страницей:

[5] Здесь под байт-кодом понимается не только то представление, в которое преобразуется программа, но и каждая инструкция виртуальной машины.

• all (значение по умолчанию) — в SWF-файле разрешены все API сетевых подключений;

• internal — нельзя обращаться к контейнеру, однако можно вызывать API сетевого подключения;

• none — все взаимодействие запрещено.

2. Обзор методов и средств обнаружения вредоносных Flash-приложений

Антивирус — собирательное название систем, установленных на компьютер пользователя и предназначенных для обнаружения и предотвращения выполнения вредоносного программного обеспечения, попадающего на машину пользователя.

Основные методы, применяемые подобными системами:

1. статический сигнатурный анализ, когда в программе ищется заранее известная подстрока, характерная только для вредоносных программ;

2. динамическое обнаружение эксплуатации уязвимостей, когда антивирус обнаруживает подозрительную активность во время выполнения программы.

Методическими недостатками антивирусов является то, что:

1. статический сигнатурный анализ неустойчив к полиморфизму кода, то есть к изменению кода программы при сохранении ее функциональности;

2. статический сигнатурный анализ не способен выявить ранее неизвестные атаки;

3. методы обнаружения эксплуатации уязвимостей не могут обнаруживать атаки, которые не направлены на эксплуатацию уязвимостей на компьютере пользователя, например, атаки класса Cross-Site Scripting.

Таким образом, антивирусы фактически не могут обнаружить рассматриваемые в данной работе классы вредоносных Flash-баннеров за исключением двух случаев:

• антивирусы могут обнаружить конкретный известный им вредоносный Flash-баннер в том случае, если у них есть для него специальная сигнатура;

• антивирусы могут обнаружить последствие перенаправления на вредоносный сайт в результате обнаружения попытки атаки на компьютер пользователя.

OdoSwiff — часть системы Wepawet, разработанная в Калифорнийском университете для обнаружения вредоносных SWF-файлов. Это средство одновременно использует статический и динамический анализ для определения вредоносности Flash-приложения [8].

При выполнении SWF-файла в модифицированном плагине строится и анализируется на предмет наличия аномалий и известных последовательностей вредоносных вызовов трасса, которая состоит из следующих данных:

1. Вызываемые методы. Например, частое использование таких методов для работы со строками, как charCodeAt, fromCharCode, parseInt и slice, может свидетельствовать о наличии обфускации. Если в некотором Flash-приложении подобных вызовов больше 95 % от общего числа, то средство считает его обфусцированным.

2. Сетевая активность. Анализируются все методы, которые вызывают сетевую активность, и их параметры. В том случае, когда вызов некоторого метода может привести к перенаправлению в браузере, в параметрах этого метода будет целевой URL-адрес.

3. URL-адреса. Помимо URL-адресов, которые используются в методах, вызывающих сетевую активность, анализируются все URL-адреса, которые встречаются в SWF-файле или создаются на стеке в процессе деобфускации. Все обнаруженные URL-адреса проверяются на наличие в черных списках доменов, с которых распространяется вредоносное ПО.

4. Методы, которые взаимодействуют с окружением.

Динамический анализ сделан на основе проекта с открытым кодом Gnash, который частично поддерживает версии 8 и 9 технологии Adobe Flash и не поддерживает более поздние версии. Поэтому в OdoSwiff реализован динамический анализ только для ActionScript 2.

Ключевым ограничением (хоть и техническим) используемого в данной системе метода динамического анализа является то, что модифицированные среды легко обнаруживаются злоумышленниками. Использование нестандартного плагина, в котором реализован неполный набор инструкций по сравнению с Adobe Flash Player [9], может быть обнаружено с помощью проверки поддержки нереализованных команд. А поскольку Gnash не получил широкого распространения, то вредоносный Flash-баннер, обнаружив, что он выполняется в этом плагине, может не проявлять вредоносную активность, тем самым избежав обнаружения при динамическом анализе. Стандартный плагин нельзя модифицировать в силу его проприетарности.

3. Предлагаемое решение

С целью преодоления подобного недостатка в данной работе предлагается динамический анализ с помощью модификации Flash-приложения и последующего выполнения в стандартном плагине. Подобный анализ возможен в силу того, что байт-код Flash-приложения не может быть изменен при выполнении, и того, что Flash-приложение не может получить доступ к своему коду. Следовательно, при модификации байт-кода перед выполнением покрывается весь потенциально исполняемый код Flash-приложения, которое при выполнении не сможет обнаружить изменение своего кода. Все обращения к необходимым для анализа системным классам модифицируются таким образом, что в ходе выполнения Flash-баннера становится возможным анализ вызовов методов этих классов и их параметров. На основе трассы из подобных вызовов и статического анализа бинарного кода Flash-баннеры классифицируются на вредоносные и легитимные.

Статический анализ используется для первичного анализа SWF-файлов. Это позволяет до динамического анализа отсеивать большую часть легитимных Flash-приложений, если в них нет никаких потенциально опасных инструкций. В отличие от OdoSwiff, где и статический, и динамический анализ обязательны.

Динамический анализ применяется для более точной классификации SWF-файлов, при этом в отличие от антивирусов предлагаемый динамический анализ может обнаруживать угрозы, направленные не только на компьютер пользователя, но и на сервисы, используемые пользователем.

Статический анализ на основе подсчета характеристик и обнаружения аномалий (по подходу аналогичный статическому анализу в средстве OdoSwiff) используется для:

1. подсчета характеристик, которые могут свидетельствовать о вредоносности или обфусцированности SWF-файла и которые могут быть вычислены статически;

2. поиска аномалий, которые могут свидетельствовать о вредоносности или обфусцированности SWF-файла и которые могут быть обнаружены статически;

3. проверки наличия потенциально вредоносных команд и определения необходимости динамического анализа.

Подобный анализ имеет следующие преимущества:

1. быстрота работы;

2. возможность не отправлять на динамический анализ SWF-файлы, в которых нет потенциально вредоносной активности.

Однако у него есть определенные недостатки:

1. велика вероятность ошибки первого рода (например, обфусцированное легитимное приложение);

2. возможность пропустить ранее неизвестную атаку, если она использует инструкции из набора, которые не считаются потенциально вредоносными.

В качестве динамического анализа предлагается использовать динамический анализ на основе выполнения модифицированного Flash-баннера в стандартном плагине и стандартном окружении.

Преимуществами подобного подхода являются:

1. использование стандартного окружения, что дает:

a. полную поддержку средством, предлагаемым в данной работе, всех существующих и появляющихся особенностей Flash-приложений с помощью стандартного обновления плагина для браузера;

b. затруднение обнаружения эмуляции окружения и наличия анализа;

c. обнаружение атак, направленных на конкретное окружение, то есть на данную машину и данного пользователя.

2. возможность исследования вызовов функций и их параметров после процесса деобфускации (если такой есть).

Недостатками такого подхода являются:

1. Необходимость перехвата и модификации SWF-файла перед выполнением. Для анализа с помощью подобного метода требуется инструментировать Flash-приложение перед тем, как оно попадет в браузер и будет выполнено. Традиционным средством для подобных модификаций являются прокси-серверы, однако существует возможность динамического создания и выполнения другого Flash-приложения изнутри Flash-баннера. В случае использования только прокси-сервера такое приложение не будет инструментировано и, как следствие, не будет проанализировано.

2. Для замены обращений к системным классам необходимы следующие особенности технологии Adobe Flash:

a. обращение к системным классам с помощью имен этих классов;

b. создание исполняемого кода при создании приложения и невозможность его изменения в процессе выполнения;

c. отсутствие возможности для приложения верифицировать свой код.

При отсутствии хотя бы одного из этих свойств предложенный динамический анализ невозможен. Например, при возможности для приложения верифицировать свой код оно сможет обнаруживать модификацию своего исходного кода и соответственно попытку анализа, а следовательно, не будет проявлять вредоносной активности.

1. Анализ основывается только на возможностях, предоставляемых анализируемой технологией. Например, анализ не может занимать больше 15 секунд, потому что в противном случае такой процесс может быть закончен как зациклившийся [7].

4. Реализация

Согласно предложенному решению, система должна использовать статический и динамический анализ и по результатам их работы делать вывод о вредоносности Flash-баннера.

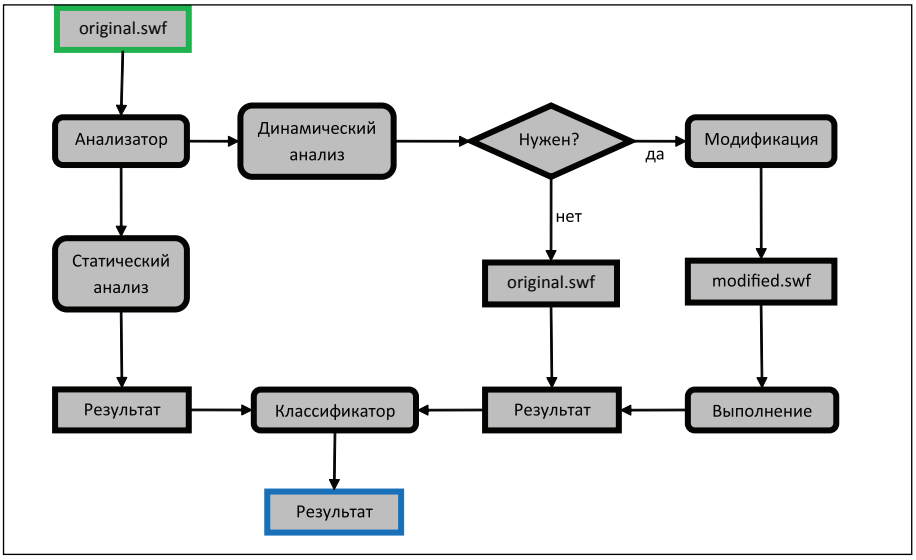

Рис. 1. Схема работы системы

Для того чтобы анализировать Flash-баннер в реальном окружении, необходимо перехватить его и модифицировать перед выполнением. Для решения этой задачи используется прокси-сервер, который перехватывает трафик пользователя, ищет в нем Flash-приложения, отдает их на статический анализ в систему и заменяет исходное приложение на модифицированное в случае необходимости динамического анализа.

Затем при выполнении модифицированного Flash-баннера в стандартном окружении данные собираются с помощью модификаторов, которые являются частью этого баннера, а затем собранные данные отправляются классификатору.

В такой реализации используется прокси-сервер Fiddler2, который работает под операционными системами семейства Windows, и плагин для него, написанный на языке C# и реализующий требуемую функциональность по перехвату и модификации Flash-баннеров.

Анализатор — точка входа в систему. Принимает на вход SWF-файл, запускает для этого класса статический и динамический анализ и классификатор по результатам, а также возвращает результаты исследования файла. Как и вся система, анализатор написан на языке программирования Python.

Динамический анализ состоит из следующих этапов:

1. проверка необходимости динамического анализа;
2. дизассемблирование Flash-приложения;
3. модификация ассемблерного кода;
4. ассемблирование модифицированного приложения;
5. выполнение модифицированного приложения в стандартном окружении и наблюдение за ним.

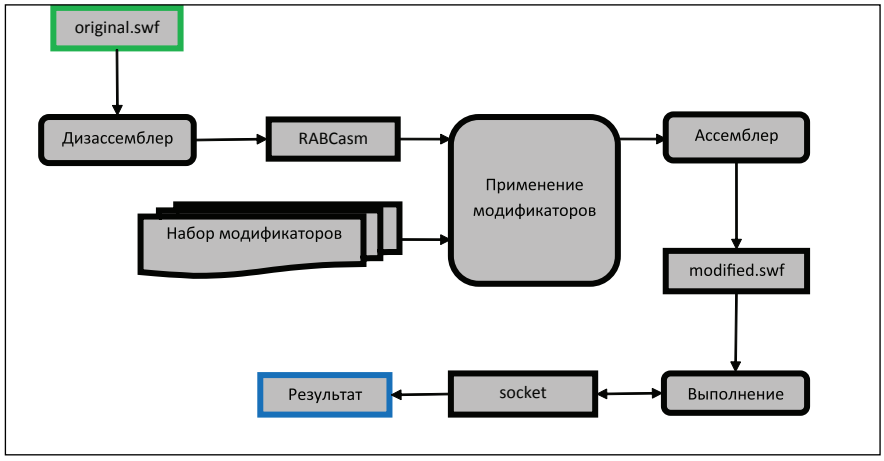

Рис. 2. Схема работы динамического анализатора

Проверка необходимости динамического анализа проводится статически с помощью средства SWFDump. Динамический анализ считается необходимым, если:

1. В коде присутствуют статические обращения к инструкциям или методам, для которых есть модификатор. В таком случае динамический анализ проводится только для тех модификаторов, обращения к которым есть в дизассемблированном коде.

2. В коде есть динамические обращения к методам. В этом случае используются все модификаторы.

В случае необходимости динамического анализа дизассемблирование и последующее ассемблирование выполняются соответственно средствами RABCDasm и RABCasm, а извлечение бинарных данных и их последующая замена — abcexport и abcreplace соответственно.

Модификатор — пара, состоящая из инструкции, которая должна быть изменена, и кода, на который заменяется эта инструкция. В случаях, когда требуется заменить определенный класс и обращения к нему:

1. Если обращение статическое, то исходное пространство имен изменяется на специально созданное, в котором существует класс с таким же именем. Например, обращение к классу ExternalInterface из пакета flash.external:

QName(PackageNamespace("flash.external"), "ExternalInterface")

заменяется на обращение к классу ExternalInterface из test1.test2:

QName(PackageNamespace("test1.test2"), "ExternalInterface")

Таким образом, вместо обращения к стандартному классу будет обращение к специально созданному классу, в котором можно анализировать исходное обращение в процессе выполнения Flash-приложения.

1. В случае динамического обращения, если это возможно, статически выясняется необходимость динамической проверки, которая добавляется, если требуется, то есть

a. в случае RTQName (когда пространство имен определяется при выполнении, а имя класса известно статически) проверяется совпадение этого имени с именами классов в модификаторах, и при совпадении добавляется код, который выполнит проверку пространства имен перед вызовом и при необходимости заменит его;

b. в случае MultinameL (когда, наоборот, известно множество пространств имен, а имя класса неизвестно) проверяется наличие интересующего пространства имен в этом множестве и при необходимости перед вызовом добавляется проверка имени класса, к которому произойдет обращение;

с. в случае RTQNameL и имя класса, и пространство имен создаются динамически, поэтому статически провести проверку нельзя, и она выполняется динамически.

5. Экспериментальное исследование

Для тестирования в качестве легитимных образцов были взяты Flash-баннеры с 500 самых популярных сайтов в разных регионах и мире в целом, согласно рейтингу alexa top 500. Популярные сайты регулярно исследуются на наличие на них вредоносных файлов, а также посещаются большим количеством пользователей с различными установленными антивирусами. Поэтому шанс того, что на них присутствуют нелимитные Flash-баннеры, крайне мал, а в том случае, если вредоносные Flash-приложения все-таки попадут на популярный ресурс, он практически сразу окажется во всех черных списках и браузер, используемый для анализа, не перейдет по ссылке на этот сайт.

Всего было собрано 28369 уникальных DNS-имени, которые впоследствии обошел бот, использовавший обычный браузер Google Chrome, стандартный плагин Adobe Flash Player и реальное окружение. В результате работы бота было исследовано 3640 различных Flash-баннеров и 2 картинки (.jpg и .png), переданные как Flash-приложения.

Динамический анализ потребовался в 697 случаях. При этом случаев использования динамически генерируемых имен (RTQName, RTQNameL или MultinameL) не зафиксировано.

На основе собранных легитимных баннеров были высчитаны граничные значения характеристик. В качестве известного вредоносного поведения системой рассматривалось перенаправление на вредоносный сайт.

Система была протестирована на множестве вредоносных Flash-баннеров, полученных от команды «Безопасного поиска “Яндекса”».

С помощью реализованной системы были обнаружены следующие образцы:

1. CVE-2013-0634 за счет:

a. обращений к свойствам окружения, которые необходимы для генерации полезной нагрузки под конкретное окружение;

b. большого количества числовых констант (которые используются для генерации полезной нагрузки), их большого среднего значения и относительно небольшой разницы с максимальным значением.

2. Все Flash-баннеры, производящие перенаправление на вредоносный сайт и использующие ActionScript 3.

3. Часть Flash-баннеров, производящих перенаправление на вредоносный сайт и использующих ActionScript 2, в которых ссылка на этот сайт хранилась в явном виде (2 из 3 подобных образцов).

СПИСОК ЛИТЕРАТУРЫ:

1. Adobe Flash runtimes Statistics. URL: http://www.adobe.com/products/flashplatformruntimes/statistics.html (дата обращения: 17.06.2013).
2. *Fukami*. Testing and exploiting Flash applications. URL: http://events.ccc.de/camp/2007/Fahrplan/attachments/1320-FlashSec.pdf (дата обращения: 17.06.2013).
3. *Jagdale P.* Blinded by Flash: Widespread Security Risks Flash Developers Don’t See. URL: http://www.blackhat.com/presentations/bh-dc-09/Jagdale/BlackHat-DC-09-Jagdale-Blinded-by-Flash.pdf (дата обращения: 17.06.2013).
4. Fuzzing at scale. URL: http://googleonlinesecurity.blogspot.com/2011/08/fuzzing-at-scale.html (дата обращения: 17.06.2013).
5. *Матросов А.* JIT spraying или новые техники обхода ASLR и DEP для IE8. URL: http://www.securitylab.ru/blog/personal/amatrosov/18955.php (дата обращения: 17.06.2013).
6. Программирование на Adobe ActionScript 3.0 в Adobe Flash. URL: http://help.adobe.com/ru_RU/ActionScript/3.0_ProgrammingAS3/ (дата обращения: 17.06.2013).

7. Adobe Flash Player 10 Security. URL: http://www.adobe.com/content/dam/Adobe/en/devnet/flashplayer/pdfs/flash_player_10_security.pdf (дата обращения: 17.06.2013).
8. *Ford S., Cova M., Kruegel C., Vigna G.* Analyzing and Detecting Malicious Flash Advertisements. URL: http://www.cs.bham.ac.uk/~covam/data/papers/acsac09_flash.pdf (дата обращения: 17.06.2013).
9. Gnash Reference Manual. URL: http://www.gnu.org/software/gnash/manual/gnashref.html (дата обращения: 17.06.2013).

REFERENCES:

1. Adobe Flash runtimes Statistics. URL: http://www.adobe.com/products/flashplatformruntimes/statistics.html (data obrasheniay: 17.06.2013).
2. *Fukami* Testing and exploiting Flash applications. URL: http://events.ccc.de/camp/2007/Fahrplan/attachments/1320-FlashSec.pdf (data obrasheniay: 17.06.2013).
3. *Prajakta Jagdale* Blinded by Flash: Widespread Security Risks Flash Developers Don't See. URL: http://www.blackhat.com/presentations/bh-dc-09/Jagdale/BlackHat-DC-09-Jagdale-Blinded-by-Flash.pdf (data obrasheniay: 17.06.2013).
4. Fuzzing at scale. URL: http://googleonlinesecurity.blogspot.com/2011/08/fuzzing-at-scale.html
5. *Matrosov A.* LT spraying bkb novie tekhniki obkhoda ASLR и DEP dlay IE8. URL: http://www.securitylab.ru/blog/personal/amatrosov/18955.php (data obrasheniay: 17.06.2013).
6. Programmirovanie na Adobe ActionScript 3.0 в Adobe Flash. URL: http://help.adobe.com/ru_RU/ActionScript/3.0_ProgrammingAS3/ (data obrasheniay: 17.06.2013).
7. Adobe Flash Player 10 Security. URL: http://www.adobe.com/content/dam/Adobe/en/devnet/flashplayer/pdfs/flash_player_10_security.pdf (дата обращения: 17.06.2013).
8. *Ford S., Cova M., Kruegel C., Vigna G.* Analyzing and Detecting Malicious Flash Advertisements. URL: http://www.cs.bham.ac.uk/~covam/data/papers/acsac09_flash.pdf (data obrasheniay: 17.06.2013).
9. Gnash Reference Manual. URL: http://www.gnu.org/software/gnash/manual/gnashref.html (data obrasheniay: 17.06.2013).