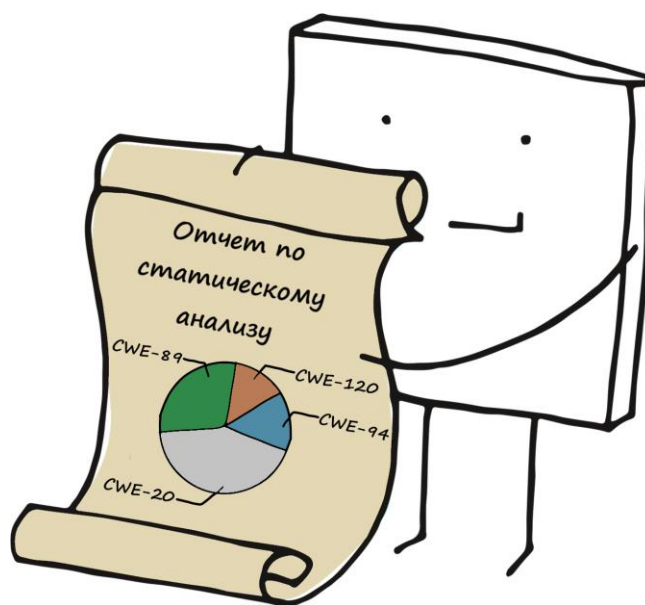


**А.Н. Бегаев, С.В. Кашин, Н.А. Маркевич,
А.А. Марченко**

**ВЫЯВЛЕНИЕ УЯЗВИМОСТЕЙ И
НЕДЕКЛАРИРОВАННЫХ ВОЗМОЖНОСТЕЙ В
ПРОГРАММНОМ ОБЕСПЕЧЕНИИ**



**Санкт-Петербург
2020**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**А.Н. Бегаев, С.В. Кашин, Н.А. Маркевич,
А.А. Марченко**

**ВЫЯВЛЕНИЕ УЯЗВИМОСТЕЙ И
НЕДЕКЛАРИРОВАННЫХ ВОЗМОЖНОСТЕЙ В
ПРОГРАММНОМ ОБЕСПЕЧЕНИИ**

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 10.03.01 Информационная безопасность
в качестве учебно-методического пособия для реализации основных
профессиональных образовательных программ высшего образования
бакалавриата

 УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2020

Бегаев А.Н., Кашин С.В., Маркевич Н.А., Марченко А.А., Выявление уязвимостей и недеklarированных возможностей в программном обеспечении – СПб: Университет ИТМО, 2020. – 38 с.

Рецензент(ы):

Заколдаев Данил Анатольевич, кандидат технических наук, доцент, декан факультета безопасности информационных технологий, Университета ИТМО.

Учебно-методическое пособие содержит теоретический материал, посвященный выявлению уязвимостей и недеklarированных возможностей в программном обеспечении. Пособие содержит общие знания о составе и содержании исследований по выявлению уязвимостей и недеklarированных возможностей в программном обеспечении при проведении сертификационных испытаний.



Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2020

© Бегаев А.Н., Кашин С.В., Маркевич Н.А., Марченко А.А., 2020

Содержание

1.	Введение.....	5
2.	Сертификация программного обеспечения.....	6
2.1.	Требования РД НДС	7
2.2.	Требования Методики АУ и НДС.....	9
3.	Испытания по выявлению уязвимостей и недекларируемых возможностей	12
3.1.	Подготовка к проведению исследований.....	13
3.2.	Анализ документации	13
3.3.	Подготовка исследовательского стенда	13
3.4.	Проведение исследований	14
3.5.	Экспертный анализ объекта оценки	14
3.6.	Статический анализ объекта оценки	19
3.7.	Динамический анализ объекта оценки	24
3.8.	Ручной анализ объекта оценки.....	26
3.9.	Оформление результатов исследований	26
4.	Средство для проведения фаззинг-тестирования American Fuzzy Loop	27
4.1.	Общая информация и основные функции American Fuzzy Loop ..	27
4.2.	Работа с American Fuzzy Loop	27
5.	Лабораторная работа.....	32
5.1.	Порядок выполнения работы.....	32
6.	Контрольные вопросы	33
	Заключение	34
	Список литературы	35

1. ВВЕДЕНИЕ

Данное учебно-методическое пособие содержит новые материалы, которые позволят студентам получить теоретические знания о порядке выявления уязвимостей и недекларированных возможностей в программном обеспечении в соответствии с Методикой выявления уязвимостей и недекларированных возможностей в программном обеспечении, утвержденной ФСТЭК России 11 февраля 2019, в ходе изучения дисциплины «Технология сертификации средств защиты информации» и получить практические навыки работы со средством «American Fuzzy Loop» для проведения фаззинг-тестирования программного обеспечения, проводимого в рамках сертификационных испытаний.

Структурно пособие состоит из шести разделов. Разделы содержат как теоретический, так и практический материал. Практический материал предназначен для подготовки к выполнению лабораторной работы и получения знаний в части прикладной реализации описанных в пособии методов испытаний. Пособие содержит лабораторную работу и контрольные вопросы, призванные помочь студентам в освоении результатов обучения по дисциплине «Технология сертификации средств защиты информации».

Ввиду неоспоримой сложности и узкой направленности отдельных видов испытаний по выявлению уязвимостей и недекларированных возможностей авторы привели подробное описание и методы практической реализации не для всех существующих видов испытаний.

Дополнительно приведен библиографический список (список литературы), который включает руководящие документы и литературу, рекомендуемую авторами для более глубокого освоения содержания дисциплины.

2. СЕРТИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В соответствии с Федеральным законом № 184 «О техническом регулировании» сертификация представляет собой форму осуществляемого органом по сертификации подтверждения соответствия объектов требованиям технических регламентов, документам по стандартизации или условиям договоров.

Сертификация может быть обязательной и добровольной. Обязательная сертификация осуществляется на основании законов и законодательных положений и проводится в системах обязательной сертификации, установленных Постановлением Правительства РФ от 26 июня 1995 г. № 608 «О сертификации средств защиты информации». Добровольная сертификация проводится только по инициативе заявителя (юридического или физического лица) в системах добровольной сертификации.

В соответствии с Постановлением Правительства № 608 обязательная сертификация проводится в рамках систем сертификации Федеральной службы по техническому и экспортному контролю (ФСТЭК России), Федеральной службы безопасности Российской Федерации (ФСБ России), Министерства обороны Российской Федерации (Минобороны России).

Система сертификации ФСТЭК России – это структура, состоящая из следующих участников сертификации:

- федерального органа по сертификации (ФСТЭК России);
- организаций, аккредитованных ФСТЭК России в качестве органов по сертификации (органы по сертификации);
- организаций, аккредитованных ФСТЭК России в качестве испытательных лабораторий (испытательные лаборатории);
- изготовители (производители) средств защиты информации.

Сертификация в системе ФСТЭК России проводится в соответствии с Положением о системе сертификации средств защиты информации, утвержденным приказом ФСТЭК России от 3 апреля 2018 года № 55.

Сертификация средств защиты информации осуществляется на соответствие нормативным правовым актам ФСТЭК России, техническим условиям (ГОСТ 2.114-2016), техническим заданиям (ГОСТ 19.201-78), заданиям по безопасности (ГОСТ Р ИСО/МЭК 15408).

До 1 июня 2019 года программные (программно-аппаратные) средства защиты информации проходили сертификацию на соответствие требованиям руководящего документа «Защита от несанкционированного доступа. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недеklarированных возможностей» (далее – РД НДВ). С 1 июня 2019 года вступили в силу новые нормативные правовые документы, утвержденные ФСТЭК России и устанавливающие новые требования

к средствам защиты информации и к проведению сертификационные испытаний, а именно: «Требования по безопасности информации, устанавливающие уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» и «Методика выявления уязвимостей и недекларированных возможностей в программном обеспечении» (Методика АУ и НДВ).

2.1. Требования РД НДВ

Руководящий документ Гостехкомиссии России «Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей», введенный в действие Приказом Председателя Гостехкомиссии России № 114 от 04.06.1999 г., устанавливает классификацию программного обеспечения средств защиты информации (ПО СЗИ) по уровню контроля отсутствия в нем недекларированных возможностей.

РД НДВ устанавливает четыре уровня контроля отсутствия недекларированных возможностей.

Самый высокий уровень контроля – первый. Первый уровень достаточен для ПО, используемого при защите информации, составляющей государственную тайну с грифом «особой важности».

Второй уровень контроля достаточен для ПО, используемого при защите информации с грифом «совершенно секретно».

Третий уровень контроля достаточен для ПО, используемого при защите информации с грифом «секретно».

Самый низкий уровень контроля – четвертый, достаточен для ПО, используемого при защите конфиденциальной информации.

Уровень контроля определяет состав и содержание программной документации на ПО, а также глубину и объем сертификационных испытаний.

Таблица 1 – Перечень требований

№	Наименование требования	Уровень контроля			
		4	3	2	1
	Требования к документации				
1	Контроль состава и содержания документации				
1.1.	Спецификация (ГОСТ 19.202-78)	+	=	=	=
1.2.	Описание программы (ГОСТ 19.402-78)	+	=	=	=
1.3.	Описание применения (ГОСТ 19.502-78)	+	=	=	=
1.4.	Пояснительная записка (ГОСТ 19.404-79)	-	+	=	=
1.5.	Тексты программ, входящих в состав ПО (ГОСТ 19.401-78)	+	=	=	=

	Требования к содержанию испытаний				
2.	Контроль исходного состояния ПО	+	=	=	=
3.	Статический анализ исходных текстов программ				
3.1.	Контроль полноты и отсутствия избыточности исходных текстов	+	+	+	=
3.2.	Контроль соответствия исходных текстов ПО его объектному (загрузочному) коду	+	=	=	+
3.3.	Контроль связей функциональных объектов по управлению	-	+	=	=
3.4.	Контроль связей функциональных объектов по информации	-	+	=	=
3.5.	Контроль информационных объектов	-	+	=	=
3.6.	Контроль наличия заданных конструкций в исходных текстах	-	-	+	+
3.7.	Формирование перечня маршрутов выполнения функциональных объектов	-	+	+	=
3.8.	Анализ критических маршрутов выполнения функциональных объектов	-	-	+	=
3.9.	Анализ алгоритма работы функциональных объектов на основе блок-схем, диаграмм и т. п., построенных по исходным текстам контролируемого ПО	-	-	+	=
4.	Динамический анализ исходных текстов программ				
4.1.	Контроль выполнения функциональных объектов	-	+	+	=
4.2.	Сопоставление фактических маршрутов выполнения функциональных объектов и маршрутов, построенных в процессе проведения статического анализа	-	+	+	=
5.	Отчетность	+	+	+	+

Обозначения

«-» – нет требований к данному уровню;

«+» – новые или дополнительные требования;

«=» – требования совпадают с требованиями предыдущего уровня.

Для подтверждения соответствия требованиям РД НДВ испытания СЗИ должны включать статический и динамический анализ исходных текстов программ.

2.2. Требования Методики АУ и НДС

С 1 июня 2019 г. обязательная сертификация средств защиты информации не проводится на соответствие требованиям руководящего документа «Защита от несанкционированного доступа. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недекларированных возможностей» в связи с утверждением «Методики выявления уязвимостей и недекларированных возможностей в программном обеспечении» (далее – Методика) от 11 февраля 2019 г. и приказа ФСТЭК России № 131 от 30 июля 2018 года «Об утверждении Требований по безопасности информации, устанавливающих уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (далее – Требования к уровням доверия).

Новая Методика определяет новый состав и содержание исследований по выявлению уязвимостей и недекларированных возможностей в программном обеспечении (в том числе программном обеспечении средств защиты информации) (далее – объекты оценки), а также применяемые при этом методы исследований и инструментальные средства.

Требования к глубине и объему исследований определяются в зависимости от уровня контроля объекта оценки, устанавливаемыми в соответствии с Требованиями к уровню контроля.

Уровень контроля объекта оценки соответствует уровню доверия объекта оценки (Таблица 2).

Таблица 2 – Соответствие уровня контроля с уровнем доверия

Уровень контроля	6	5	4	3	2	1
Уровень доверия	6	5	4	3	2	1

Выполнение Требований к уровню доверия является обязательным при проведении работ по сертификации средств защиты информации, организуемых ФСТЭК России в пределах своих полномочий.

Средства защиты информации, соответствующие 6 уровню доверия, подлежат применению в значимых объектах критической информационной инфраструктуры 3 категории, в государственных информационных системах 3 класса защищенности, в автоматизированных системах управления производственными и технологическими процессами 3 класса защищенности, в информационных системах персональных данных при необходимости обеспечения 3 и 4 уровня защищенности персональных данных.

Средства защиты информации, соответствующие 5 уровню доверия, подлежат применению в значимых объектах критической информационной инфраструктуры 2 категории, в государственных информационных системах 2

класса защищенности, в автоматизированных системах управления производственными и технологическими процессами 2 класса защищенности, в информационных системах персональных данных при необходимости обеспечения 2 уровня защищенности персональных данных.

Средства защиты информации, соответствующие 4 уровню доверия, подлежат применению в значимых объектах критической информационной инфраструктуры 1 категории, в государственных информационных системах 1 класса защищенности, в автоматизированных системах управления производственными и технологическими процессами 1 класса защищенности, в информационных системах персональных данных при необходимости обеспечения 1 уровня защищенности персональных данных, в информационных системах общего пользования II класса.

Средства защиты информации, соответствующие 1, 2 и 3 уровням доверия, применяются в информационных (автоматизированных) системах, в которых обрабатывается информация, содержащая сведения, составляющие государственную тайну.

Таблица 3 – Общие требования уровней доверия

	6 УД	5 УД	4 УД	3 УД	2 УД	1 УД
КИИ¹	3 категория	2 категория	1 категория	Применяются для защиты сведений, составляющих государственную тайну		
ГИС²	3 класс	2 класс	1 класс			
АСУ ТП³	3 класс	2 класс	1 класс			
ИСПДн⁴	3, 4 уровень	2 уровень	1 уровень			
ИСОП⁵			II класс			

Исследования по выявлению уязвимостей и НДВ объекта оценки включают:

- подготовку к проведению исследований по выявлению уязвимостей и НДВ объекта оценки;
- проведение исследований по выявлению уязвимостей и НДВ объекта оценки;

¹ КИИ (сокр. – критическая информационная инфраструктура). Категории устанавливаются в соответствии с Федеральным законом № 187-ФЗ от 26.07.2017 и Постановлением Правительства РФ № 127 от 08.02.2018.

² ГИС (сокр. – государственная информационная система). Классы устанавливаются в соответствии с Приказами ФСТЭК России № 17 от 11.02.2013 и № 27 от 15.02.2017.

³ АСУ ТП (сокр. – автоматизированная система управления технологическим процессом). Классы устанавливаются в соответствии с Приказами ФСТЭК России № 31 от 14.03.2014 и № 138 от 09.08.2018.

⁴ ИСПДн (сокр. – информационная система персональных данных). Уровни защищенности устанавливаются в соответствии с Постановлением Правительства № 1119 от 01.11.2012.

⁵ ИСОП (сокр. – информационная система общего пользования). Классы устанавливаются в соответствии с Приказом ФСБ России и ФСТЭК России № 416/489 от 31.08.2010.

– оформление результатов исследований по выявлению уязвимостей и НДВ объекта оценки.

В соответствии с Методикой испытания состоят из трех этапов (Таблица 4).

Таблица 4 – Порядок проведения исследований по выявлению уязвимостей и НДВ объекта оценки

№ п/п	Вид исследования
Этап 1. Подготовка к проведению исследований	
1.1	Анализ документации и других источников, содержащих сведения об объекте оценки
1.2	Подготовка исследовательского стенда
Этап 2. Проведение исследований	
2.1	Экспертный анализ объекта оценки
2.2	Статический анализ объекта оценки
2.3	Динамический анализ объекта оценки
2.4	Ручной анализ объекта оценки
Этап 3. Оформление результатов исследований	
3.1	Разработка протоколов проведения исследований объекта оценки

3. ИСПЫТАНИЯ ПО ВЫЯВЛЕНИЮ УЯЗВИМОСТЕЙ И НЕДЕКЛАРИРУЕМЫХ ВОЗМОЖНОСТЕЙ

Испытания по выявлению уязвимостей и НДВ объекта оценки должны предусматривать проведение экспертного, статического, динамического и ручного анализа.

Экспертный анализ предусматривает выявление уязвимостей и НДВ объекта оценки по результатам исследований документации на объект оценки и иных доступных информационных источников, результатов исследований, полученных другими исследователями, анализа архитектуры объекта оценки, а также проведения тестирования на проникновение. Основными методами, используемыми при проведении экспертного анализа, являются анализ документации, сканирование уязвимостей, моделирование атак, тестирование на проникновение, визуализация данных.

Статический анализ предусматривает выявление уязвимостей и НДВ по результатам исследования программы (анализа кода) в режиме, не предусматривающем реального выполнения программного кода объекта оценки. В ходе статического анализа на основе программного кода осуществляется построение модели, для которой выполняются операции статического анализа. Основными методами статического анализа являются синтаксический анализ, межпроцедурный контекстно-чувствительный анализ, формальная верификация. Для проведения статического анализа применяются статические анализаторы кода.

Динамический анализ предусматривает выявление уязвимостей и НДВ по результатам исследований программы (анализа кода) в режиме непосредственного исполнения (функционирования) кода. При проведении динамического анализа на вход программы подаются специально сформированные входные данные. В ходе выполнения программы осуществляется фиксация результатов её выполнения, в том числе записываются трассы выполнения программы, снимаются состояния памяти и регистров процессора, фиксируются выходные данные. Для использования методов динамического анализа в отношении встраиваемого программного обеспечения требуется воссоздание его среды функционирования, в ходе которого осуществляется эмулирование аппаратного и программного обеспечения. В этом случае динамический анализ, как правило, применяется к фрагментам встроенного программного обеспечения. Основными методами динамического анализа являются фаззинг, анализ активности легковесных процессов взаимодействия программ, отслеживание помеченных данных, а также другие методы. Проведение динамического анализа осуществляется с использованием фаззера, монитора активности приложения, отладчика, распаковщика, профилировщика.

Ручной анализ предусматривает выявление уязвимостей и НДВ по результатам экспертизы исходного/восстановленного кода объекта оценки. Основными методами, используемыми при проведении ручного анализа, являются отслеживание помеченных данных, направленный ручной анализ участков кода, визуализация данных.

3.1. Подготовка к проведению исследований

В ходе подготовки к проведению исследований по выявлению уязвимостей и НДВ объекта оценки должны выполняться:

- анализ документации и других источников, содержащих сведения об объекте оценки;
- подготовка исследовательского стенда.

3.2. Анализ документации

Для выявления потенциально опасных функциональных возможностей объекта оценки должен быть проведен анализ документации (спецификация, формуляр, руководство пользователя (программиста), руководство администратора (системного программиста), а также другая программная, конструкторская и эксплуатационная документация) и иных доступных источников, содержащих сведения об объекте оценки (официальные публикации об объекте оценки, официальный сайт разработчика, опубликованные результаты исследований об объекте оценки сторонних исследователей, форумы и отзывы исследователей и пользователей объекта оценки, базы данных уязвимостей).

В результате анализа полученных сведений об объекте оценки должны быть выявлены потенциально опасные функциональные возможности объекта оценки или подтверждено их отсутствие.

3.3. Подготовка исследовательского стенда

Для обеспечения возможности проведения всех видов исследований объекта оценки должен быть создан исследовательский стенд. В состав исследовательского стенда включается дистрибутив объекта оценки, программно-аппаратное средство (в случае если объектом оценки является встроенное программное обеспечение).

В ходе подготовки исследовательского стенда должны быть выполнены:

- настройка сред функционирования объекта оценки, в которых может эксплуатироваться объект оценки;
- установка и настройка объекта оценки в соответствии с эксплуатационной документацией (для прикладного или системного ПО – инсталляция, для встроенного ПО – подключение к стенду и конфигурирование программно-аппаратного средства);
- разработка и внедрение мер по обеспечению безопасности, направленных на обеспечение целостности или возможности восстановления

среды функционирования и самого объекта оценки, при проведении исследований объекта оценки.

- оценка настроек объекта оценки, предусматривающая контроль конфигурации и настроек с целью выявления уязвимостей конфигурации (например, с помощью средства контроля защищенности информации от несанкционированного доступа);

- антивирусный контроль объекта оценки (антивирусный контроль дистрибутива объекта оценки, а также всех файлов инсталлированного объекта).

3.4. Проведение исследований

В ходе проведения исследований по выявлению уязвимостей и НДВ объекта оценки должны выполняться:

- экспертный анализ объекта оценки;
- статический анализ объекта оценки;
- динамический анализ объекта оценки;
- ручной анализ участков исходного/восстановленного кода объекта оценки.

3.5. Экспертный анализ объекта оценки

В ходе проведения исследований должен быть выполнен экспертный анализ объекта оценки, который включает:

- анализ архитектуры объекта оценки, направленный на выявление потенциально опасных функциональных возможностей и архитектурных уязвимостей;

- идентификацию потенциальных уязвимостей объекта на основе анализа доступных источников, предусматривающую поиск всех возможных источников, содержащих сведения об объекте оценки и его потенциальных уязвимостях, и их анализ с целью формирования перечня потенциальных уязвимостей, которые актуальны для объекта оценки;

- подтверждение потенциальных уязвимостей объекта на основе результатов тестирования на проникновение объекта оценки в средах его функционирования.

В качестве доступных источников информации для поиска информации об уязвимостях можно использовать следующие источники: базу CAPEC (Common Attack Pattern Enumeration and Classification) (Рисунок 1), базу знаний MITRE ATT&CK (Adversarial Tactics, Techniques & Common Knowledge) (Рисунок 2), банк данных угроз и уязвимостей ФСТЭК России (Рисунок 3), база данных уязвимостей безопасности CVE Details (Рисунок 4).

1000 - Mechanisms of Attack

- Engage in Deceptive Interactions - (156)
 - Content Spoofing - (148)
 - Checksum Spoofing - (145)
 - Spoofing of UDDI/ebXML Messages - (218)
 - Intent Spoof - (502)
 - Counterfeit GPS Signals - (622)
 - Carry-Off GPS Attack - (628)
 - Identity Spoofing - (151)
 - Fake the Source of Data - (194)
 - DNS Rebinding - (275)
 - Counterfeit Websites - (543)
 - Counterfeit Organizations - (544)
 - DNS Spoofing - (598)
 - Token Impersonation - (632)
 - Principal Spoof - (195)
 - Cross Frame Scripting (XFS) - (582)
 - Terrestrial Jamming - (599)
 - Signature Spoof - (472)
 - Creating a Rogue Certification Authority Certificate - (459)
 - Signature Spoofing by Key Theft - (474)
 - Signature Spoofing by Improper Validation - (475)
 - Signature Spoofing by Misrepresentation - (476)
 - Signature Spoofing by Mixing Signed and Unsigned Content - (477)
 - Malicious Root Certificate - (479)
 - Signature Spoofing by Key Recreation - (485)
 - Pharming - (89)
 - Phishing - (98)

3000 - Domains of Attack

- Software - (512)
 - Exploitation of Trusted Credentials - (21)
 - Session Credential Falsification through Forging - (196)
 - Session Credential Falsification through Manipulation - (226)
 - Session Credential Falsification through Prediction - (89)
 - SaaS User Request Forgery - (510)
 - Use of Known Domain Credentials - (560)
 - Kerberoasting - (509)
 - Remote Services with Stolen Credentials - (555)
 - Windows Admin Shares with Stolen Credentials - (561)
 - Use of Captured Hashes (Pass The Hash) - (644)
 - Use of Captured Tickets (Pass The Ticket) - (645)
 - Session Hijacking - (593)
 - Session Sidejacking - (102)
 - Cross Site Tracing - (107)
 - Reusing Session IDs (aka Session Replay) - (60)
 - Session Fixation - (61)
 - Cross Site Request Forgery - (62)
 - Cross Site Identification - (467)
 - Exploiting Trust in Client - (22)
 - Create Malicious Client - (202)
 - Removing Important Client Functionality - (202)
 - Removal of filters: Input filters, output filters, data masking - (200)
 - Removing/short-circuiting 'Purse' logic: removing/mutating 'cash' decrements - (208)
 - Manipulating Opaque Client-based Data Tokens - (39)
 - Manipulating User-Controlled Variables - (77)
 - Subverting Environment Variable Values - (13)

Рисунок 1 – Классификация шаблонов компьютерных атак CAPEC

The screenshot shows the MITRE ATT&CK website interface. At the top is a navigation bar with 'MITRE ATT&CK' logo and various menu items like 'Matrices', 'Tactics', 'Techniques', 'Mitigations', 'Groups', 'Software', 'Resources', 'Blog', 'Contribute', and a search bar. Below the navigation bar is a secondary bar with the text 'Register to stream ATT&CKcon 2.0 October 29-30'. The main content area is titled 'Brute Force' and includes a description: 'Adversaries may use brute force techniques to attempt access to accounts when passwords are unknown or when password hashes are obtained.' It also mentions 'Credential Dumping' and 'password spraying'. A list of related techniques is provided, including SSH, Telnet, FTP, NetBIOS/SMB/Samba, LDAP, Kerberos, RDP/Terminal Services, HTTP/HTTPS Management Services, and MSSQL. On the right side, there is a metadata box for 'ID: T1110' with details on 'Tactic: Credential Access', 'Platform: Linux, macOS, Windows, Office 365, Azure AD, SaaS', 'Permissions Required: User', 'Data Sources: Office 365 account logs, Authentication logs', 'CAPEC ID: CAPEC-49', 'Contributors: Microsoft Threat Intelligence Center (MSTIC), John Strand, Ed Williams, Trustwave, SpiderLabs', and 'Version: 2.0'. A sidebar on the left shows the 'ENTERPRISE' matrix and a list of 'TECHNIQUES' with 'Brute Force' highlighted.

Рисунок 2 – База знаний АТТ&СК

Банк данных угроз безопасности информации

Федеральная служба по техническому и экспортному контролю
ФСТЭК России

Государственный научно-исследовательский испытательский центр проблем технической защиты информации
ФАУ «ГНИИИ ПТЗИ ФСТЭК России»

Угрозы | **Уязвимости** | Документы | Термины | Обратная связь | Обновления | Участники | ФСТЭК России

Главная / Список уязвимостей

ФИЛЬТРАЦИЯ

Контекстный поиск по названию уязвимости

Производитель ПО

Тип ПО

Программное обеспечение

Аппаратная платформа

Версия ПО

Статус уязвимости

Доп. параметры

Диапазон дат

Класс уязвимости

Уровень опасности

Выводить по: 10, 20, 50, 100 | Сортировка: | Элементы с 1 по 10 из 23367

BDU:2019-03845	Уязвимость обработчика сценариев VBScript браузера Internet Explorer, позволяющая нарушителю выполнить произвольный код Microsoft Corp. Internet Explorer 10	08.10.2019
BDU:2019-03844	Уязвимость виртуального программно-определяемого коммутатора Microsoft Hyper-V Network Switch операционных систем Windows, позволяющая нарушителю получить доступ к защищаемой информации Microsoft Corp. Windows 10 1703	08.10.2019
BDU:2019-03843	Уязвимость в реализациях протокола NTLM операционной системы Windows, связанная с недостаточной проверкой подлинности NTLM-пакетов при их обработке, позволяющая нарушителю понизить функции безопасности механизма защиты NTLM MIC (Message Integrity Check) Microsoft Corp. Windows Server 2008 R2 SP1	08.10.2019
BDU:2019-03842	Уязвимость программного обеспечения для электронного документооборота Microsoft SharePoint Foundation и пакета программ Microsoft SharePoint Enterprise Server, связанная с неприятием мер по защите структуры веб-страницы, позволяющая нарушителю осуществлять межсайтовые сценарные атаки и выполнить произвольный код в контексте текущего пользователя Microsoft Corp. Microsoft SharePoint Foundation 2013 SP1	08.10.2019
BDU:2019-03841	Уязвимость синтаксического анализатора набора служб Microsoft XML Core Services операционных систем Windows, позволяющая нарушителю выполнить произвольный код Microsoft Corp. Windows 8.1	08.10.2019
BDU:2019-03840	Уязвимость подкомпонента Loan Calculator компонента Oracle Banking Digital Experience банковской аналитической системы имитационного моделирования Oracle Financial Services Applications, позволяющая нарушителю получить доступ на изменение, добавление или удаление данных Oracle Corp. Financial Services Applications 18.1	15.10.2019

Рисунок 3 – Банк данных уязвимостей ФСТЭК России

CVE Details
The ultimate security vulnerability datasource

(e.g.: CVE-2009-1234 or 2010-1234 or 20101234)

[Log In](#) [Register](#) [Vulnerability Feeds & Widgets](#) [www.itscib.com](#)

Home

Browse:

- Vendors
- Products
- Vulnerabilities By Date
- Vulnerabilities By Type

Reports:

- CVSS Score Report
- CVSS Score Distribution

Search:

- Vendor Search
- Product Search
- Version Search
- Vulnerability Search
- By Microsoft References

Top 50:

- Vendors
- Vendor Cvs Scores
- Products
- Product Cvs Scores
- Versions

Other:

- Microsoft Bulletins
- BuTraG Entries
- CWE Definitions
- About & Contact
- Feedback
- CVE Help
- FAQ
- Articles

External Links:

- NVD Website
- CWE Web Site

Security Vulnerabilities Published In June 2019

2019: January February March April May June July August September October CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results by: CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

Total number of vulnerabilities: **1263** Page: 1 (This Page) 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

[Copy Results](#) [Download Results](#)

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2019-13118	20			2019-06-30	2019-07-22	5.0	None	Remote	Low	Not required	Partial	None	None
In numbers.c in libssl 1.1.33, a type holding grouping characters of an xsl:number instruction was too narrow and an invalid character/length combination could be passed to xsl:numberFormatDecimal, leading to a read of uninitialized stack data.														
2	CVE-2019-13117	20			2019-06-30	2019-07-22	5.0	None	Remote	Low	Not required	Partial	None	None
In numbers.c in libssl 1.1.33, an xsl:number with certain format strings could lead to a uninitialized read in xsl:numberFormatInsertNumbers. This could allow an attacker to discern whether a byte on the stack contains the characters A, a, 1, i, or 0, or any other character.														
3	CVE-2019-13114	476		DoS	2019-06-30	2019-07-15	4.3	None	Remote	Medium	Not required	None	None	Partial
http.c in Exiv2 through 0.27.1 allows a malicious http server to cause a denial of service (crash due to a NULL pointer dereference) by returning a crafted response that lacks a space character.														
4	CVE-2019-13113	20		DoS	2019-06-30	2019-07-15	4.3	None	Remote	Medium	Not required	None	None	Partial
Exiv2 through 0.27.1 allows an attacker to cause a denial of service (crash due to assertion failure) via an invalid data location in a CRW image file.														
5	CVE-2019-13112	400		DoS	2019-06-30	2019-07-15	4.3	None	Remote	Medium	Not required	None	None	Partial
A PngChunk::parseChunkContent uncontrolled memory allocation in Exiv2 through 0.27.1 allows an attacker to cause a denial of service (crash due to an std::bad_alloc exception) via a crafted PNG image file.														
6	CVE-2019-13111	190		DoS Overflow	2019-06-30	2019-08-08	4.3	None	Remote	Medium	Not required	None	None	Partial
A WebPImage::decodeChunks integer overflow in Exiv2 through 0.27.1 allows an attacker to cause a denial of service (large heap allocation followed by a very long running loop) via a crafted WEBP image file.														
7	CVE-2019-13110	190		DoS Overflow	2019-06-30	2019-07-15	4.3	None	Remote	Medium	Not required	None	None	Partial
A CiffDirectory::readDirectory integer overflow and out-of-bounds read in Exiv2 through 0.27.1 allows an attacker to cause a denial of service (SIGSEGV) via a crafted CRW image file.														
8	CVE-2019-13109	190		DoS Overflow	2019-06-30	2019-08-08	4.3	None	Remote	Medium	Not required	None	None	Partial
An integer overflow in Exiv2 through 0.27.1 allows an attacker to cause a denial of service (SIGSEGV) via a crafted PNG image file, because PngImage::readMetadata mishandles a chunkLength - iccOffset subtraction.														
9	CVE-2019-13108	190		DoS Overflow	2019-06-30	2019-08-08	4.3	None	Remote	Medium	Not required	None	None	Partial
An integer overflow in Exiv2 through 0.27.1 allows an attacker to cause a denial of service (SIGSEGV) via a crafted PNG image file, because PngImage::readMetadata mishandles a zero value for iccOffset.														
10	CVE-2019-13107	190		Overflow	2019-06-30	2019-07-18	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
Multiple integer overflows exist in MATIO before 1.5.16, related to mat.c, mat4.c, mats.c, mat73.c, and matvar_struct.c														
11	CVE-2019-13086	89		Sql CSRF	2019-06-30	2019-07-03	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
core/MY_Security.php in CSZ CMS 1.2.2 before 2019-06-20 has member/login/check SQL injection by sending a crafted HTTP User-Agent header and omitting the csrf_cs2 parameter.														
12	CVE-2019-13085	287			2019-06-30	2019-07-03	6.8	None	Remote	Medium	Not required	Partial	Partial	Partial
XnView Classic 2.48 has a User Mode Write AV starting at xnview+0x000000000030efc5.														
13	CVE-2019-13084	287			2019-06-30	2019-07-03	6.8	None	Remote	Medium	Not required	Partial	Partial	Partial
XnView Classic 2.48 has a User Mode Write AV starting at xnview+0x000000000026b739.														

Рисунок 4 – База данных CVE Details

Поиск по доступным источникам информации, содержащим данные об уязвимостях, проводится на основе данных о программном обеспечении (наименование, версия, язык программирования, среда функционирования, условия функционирования), являющемся объектом оценки, а также на основе данных об однотипных с объектом оценки изделиях.

Для идентификации потенциальных уязвимостей также должны применяться средства контроля защищенности информации от несанкционированного доступа, позволяющих обнаружить известные уязвимости в объекте оценки.

В качестве сканеров уязвимостей можно использовать:

- Nessus;
- Metasploit framework;
- OpenVAS;
- Сканер-ВС.

На рисунке 5 представлен результат сканирования уязвимостей сканером Nessus. Сканер Nessus, предназначенный для автоматического поиска известных уязвимостей в защите информационных систем, способен обнаружить следующие виды уязвимостей:

- наличие уязвимых версий служб или доменов;
- ошибки в конфигурации;
- наличие паролей по умолчанию, пустых или слабых паролей.

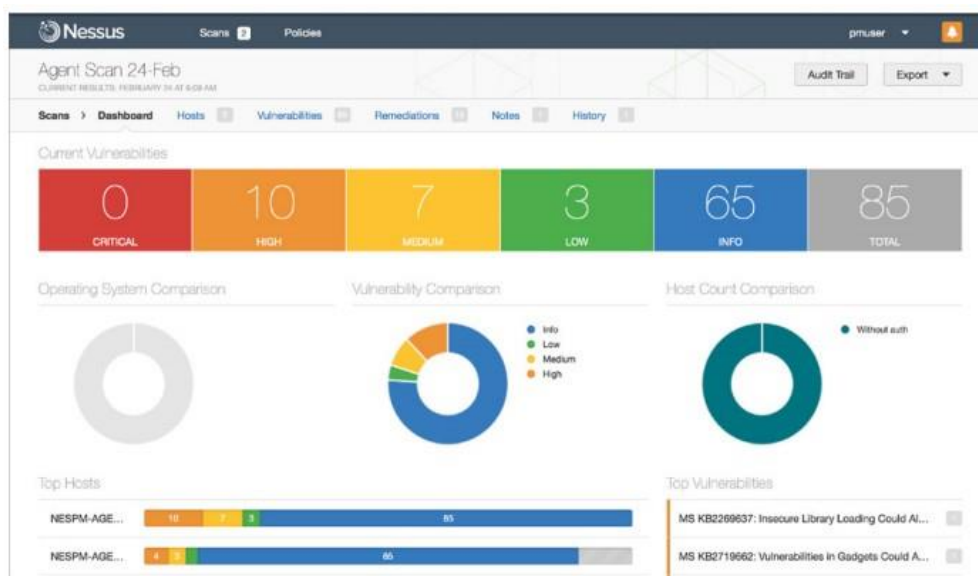


Рисунок 5 – Результат работы сканера Nessus

Metasploit Framework является одним из самых популярных инструментов сканирования уязвимостей в сетевых и веб-приложениях и инструментом для

тестирования на проникновение. Он имеет встроенные плагины для некоторых известных сканеров уязвимостей, таких как Nessus, Nexpose, OpenVAS и WMAP.

OpenVAS – это сканер уязвимостей и средство управления уязвимостями с открытым исходным кодом. OpenVAS предназначен для активного мониторинга узлов вычислительной сети на предмет наличия проблем, связанных с безопасностью, оценки серьезности этих проблем и для контроля их устранения. На рисунке 6 представлен результат работы сканера уязвимостей OpenVAS.

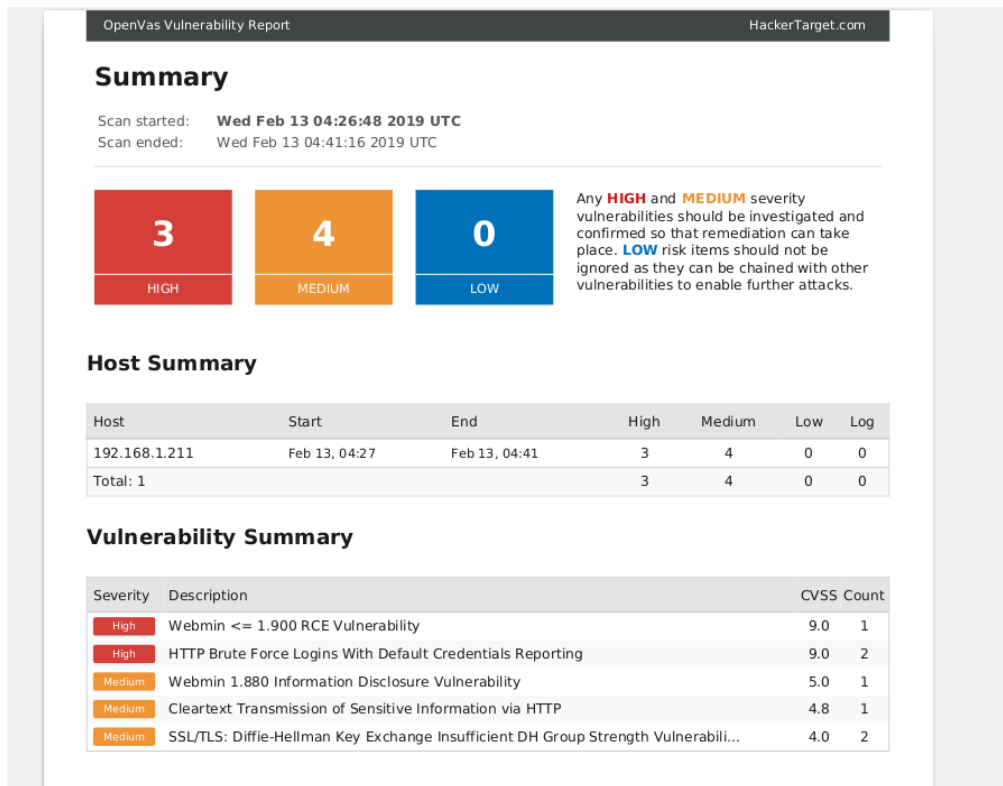


Рисунок 6 – Отчет по результатам поиска уязвимостей сканером OpenVAS

Сканер-ВС – сертифицированный инструмент для проведения тестирования и анализа защищенности информационных систем, а также для контроля эффективности средств защиты информации. В состав Сканер-ВС входит фреймворк Metasploit.

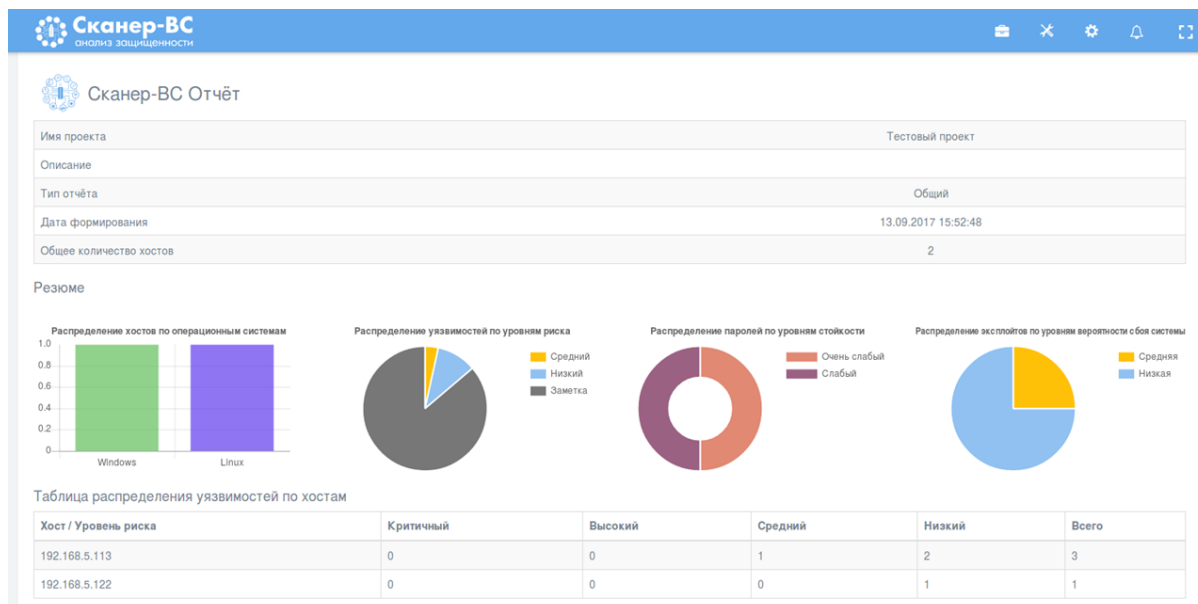


Рисунок 7 – Отчет Сканер-BC по результатам поиска уязвимостей

Представленный список сканеров уязвимостей не является исчерпывающим. Для идентификации потенциальных уязвимостей могут применяться любое из средств или их совокупность.

По результатам анализа архитектуры объекта оценки и идентификации потенциальных уязвимостей объекта путем анализа доступных источников и сканирования средствами контроля защищенности информации от несанкционированного доступа формируется перечень потенциальных уязвимостей, актуальных для объекта оценки.

В отношении всех идентифицированных уязвимостей, в том числе уязвимостей архитектуры, должны быть разработаны и проведены тесты на проникновение, в ходе которых должно подтвердиться отсутствие возможности эксплуатации этих уязвимостей нарушителями.

3.6. Статический анализ объекта оценки

В ходе проведения исследований должен быть проведен статический анализ объекта оценки, который включает:

- синтаксический анализ исходного кода объекта оценки, предусматривающий автоматическое сопоставление линейной последовательности лексем, в которую преобразуется исходный код, с неким формальным набором правил (сигнатур);

- анализ исходного кода объекта оценки, чувствительный к контексту и путям выполнения, предусматривающий автоматический анализ путей выполнения кода объекта оценки с учётом контекстов вызовов функций (фактических параметров функций, состояния глобальных переменных и т.п.);

– формальную верификацию исходного кода объекта оценки, предусматривающую строгое доказательство отсутствия в объекте оценки заданного класса программных ошибок.

В целом под статическим анализом чаще всего понимается синтаксический, который может быть дополнен различными методами, увеличивающими вероятность нахождения ошибок (например, межпроцедурный анализ) или снижающими вероятность ошибки второго рода (абстрактная интерпретация).

Статический анализ – анализ кода программы без её запуска. Статический анализ проводится с помощью специальных инструментов – статических анализаторов кода.

Статический анализ обладает преимуществами масштабируемости и скорости, может быть легко включен самим разработчиком в жизненный цикл разработки программного обеспечения – многие статические анализаторы имеют возможность проверять код прямо в IDE. Для работы со статическим анализатором чаще всего не требуется обладать особыми навыками.

Примеры статических анализаторов:

– AppChecker – статический анализатор, разработанный российской компанией АО «НПО «Эшелон», предназначен для программ, написанных с применением следующих языков программирования: C/C++/C#, PHP, Java.

– Svace – система статического анализа, разработанная в Институте системного программирования Российской Академии Наук, предназначена для обнаружения широкого спектра дефектов и уязвимостей в прикладных программах, написанных на языках C, C++, Java и C#.

– Coverity – статический анализатор, поддерживающий 20 языков программирования и более 70 платформ, включая Ruby on rails, Scala, PHP, Python, JavaScript, TypeScript, Java, Fortran, C, C++, C#, VB.NET.

– PVS-Studio – статический анализатор для кода на C, C++, C# и Java.

Статические анализаторы способны обнаружить много различных классов ошибок программирования, например:

- переполнение буфера на стеке и куче;
- целочисленные переполнения;
- двойные освобождения памяти;
- использование памяти после ее освобождения;
- разыменовывание нулевого указателя;
- использование неинициализированных переменных;
- недостижимый код;
- утечка информации через сообщения об ошибках;
- использование присваивания вместо сравнения.

На рисунке 8 представлен отчет по результатам статического анализа, проведенного анализатором AppChecker. Отчет статического анализа показывает

статус и категорию дефекта, его CWE-идентификатор и фрагмент кода, где дефект был обнаружен.

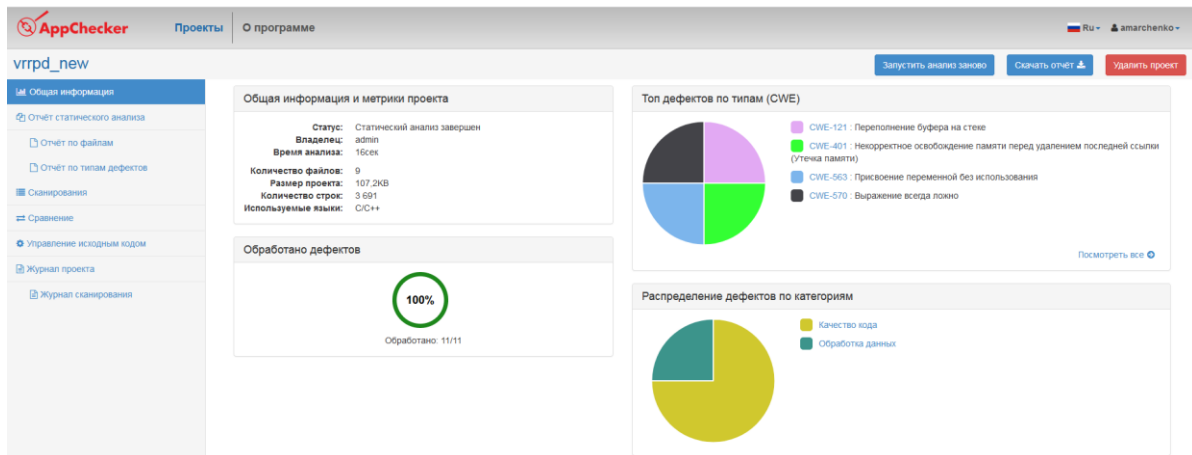


Рисунок 8 – Отчет статического анализа, сформированный AppChecker

На рисунке 9 представлен интерфейс пользователя по просмотру проекта в системе Coverity.

Преимущества статического анализа:

- скорость;
- хорошая масштабируемость на миллионы строк кода;
- большой выбор инструментов;
- возможность анализа на неполном наборе исходных файлов.

Недостатки статического анализа:

- алгоритмическая неразрешимость (теорема Райса);
- пропуск ошибок и ложные срабатывания.

Приведенный в настоящем учебно-методическом пособии список статических анализаторов не является исчерпывающим.

Не существует идеального анализатора, который бы находил все ошибки в любой программе и никогда бы не выдавал ни одного ложного предупреждения.

При проведении статического анализа исходного кода рекомендуется использовать несколько анализаторов от различных производителей. При этом анализ, направленный именно на поиск уязвимостей в коде (а именно это является целью сертификационных испытаний), должен проводиться исходя из стратегии получить как можно меньше пропусков реальных ошибок в программном обеспечении, в то время как при разработке программного обеспечения статический анализатор должен выдавать как можно меньше ложноположительных срабатываний, поскольку разбор таких ошибок программистом может затянуться, что приведет к трате драгоценного времени и увеличит стоимость разработки.

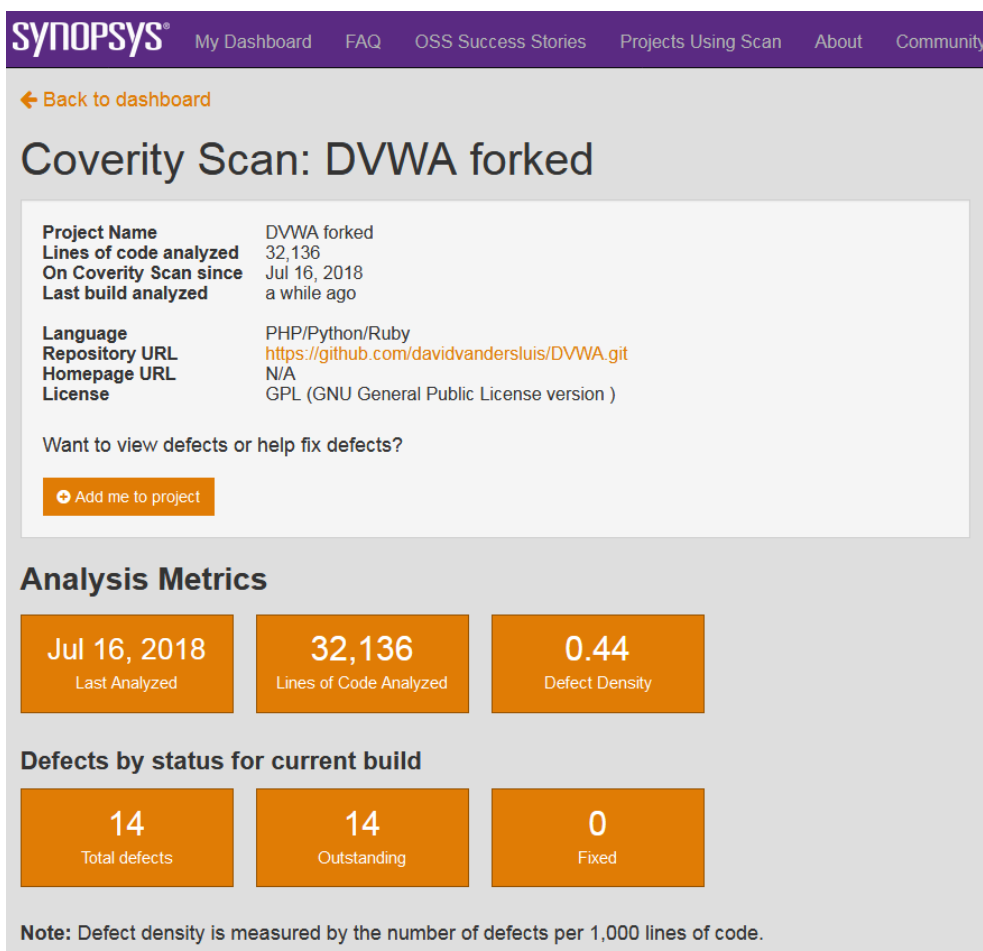


Рисунок 9 – Система Coverity

Даже самые передовые статические (синтаксические) анализаторы не могут охватить всех возможных путей выполнения программы. С этим не могут справиться также и средства динамического анализа (о которых будет рассказано в следующем разделе). В тех случаях, когда программное обеспечение выполняет критичные функции (например, в авионике) или используется для обработки и защиты информации, составляющей государственную тайну, на помощь приходят формальные методы.

Инструменты формального моделирования полагаются на математику, чтобы описать спецификацию программы и проверить ее. В отличие от многих методов тестирования, которые показывают наличие ошибок, формальные методы доказывают **отсутствие** ошибок. Инструменты можно разделить на три группы:

1. Инструменты проверки модели

Это автоматизированные инструменты для проверки состояния программы. Они генерируют абстракцию программы – модель состояний – и проверяют, что все состояния соответствуют некоторым спецификациям. Такие инструменты не

требуют взаимодействия с человеком для доказательства. Однако время выполнения и потребление памяти растут экспоненциально с увеличением числа состояний.

2. SMT-решатели

Эти инструменты позволяют определить, выполнима ли формула первого порядка. Уровень доказываемых свойств программы несколько снижен, однако такие инструменты полностью автоматические. Среди известные решателей выделяются Z3⁶, alt-Ergo⁷, Boolector⁸

3. Инструменты интерактивного доказательства теорем

Такие инструменты не являются полностью автоматическими, но они позволяют доказывать более выразительные свойства программы. При этом доказательства будут требовать взаимодействия с пользователем для уточнения модели программы.

Методы доказательства теорем можно разделить на две части. Это использование аннотаций и интерактивное доказательство.

Аннотирование используется для доказательства поведенческих свойств программы. В программе определяются пре- и постусловия, которые генерируют формулы для доказательств. Затем они подаются на вход SMT-решателю, инструменту проверки моделей или доказываются в интерактивном режиме. Ярким примером здесь является язык спецификации для C – ACSL⁹. Аннотирование с его помощью можно проводить прямо в исходном коде.

При интерактивном доказательстве чаще всего используется целостная модель всей системы. Модель вместе с описывающими ее свойствами пишется на соответствующем языке спецификаций, например, Coq¹⁰, Isabelle¹¹, Event-B¹². Средства, которые доказывают свойства модели, могут быть как встроены в инструмент, так и подключаться при необходимости.

Если математическая модель программы доказана (для определенного типа ошибок), это свидетельствует о том, что в самой программе таких ошибок нет.

Однако такой метод очень сложен, требует высокой квалификации, временных затрат и понимания деталей реализации программы.

⁶ de Moura L., Bjørner N. (2008) Z3: An Efficient SMT Solver. In: Ramakrishnan C.R., Rehof J. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008. Lecture Notes in Computer Science, vol 4963. Springer, Berlin, Heidelberg

⁷ Alt-Ergo by OCamlPro – <https://alt-ergo.ocamlpro.com/>

⁸ Robert Brummayer, Armin Biere. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09), Lecture Notes in Computer Science, volume 5505, Springer, 2009.

⁹ Frama-C – <http://frama-c.com/index.html>

¹⁰ The Coq Proof Assistant – <https://coq.inria.fr/>

¹¹ Isabelle – <https://isabelle.in.tum.de/index.html>

¹² Event-B and the Rodin Platform – <http://www.event-b.org/>

3.7. Динамический анализ объекта оценки

В ходе проведения исследований должен быть проведен динамический анализ объекта оценки, который включает:

- фаззинг-тестирование объекта оценки, направленное на выявление архитектурных уязвимостей и уязвимостей кода в объекте оценки путем передачи на вход объекта оценки неправильных, неожиданных или случайных данных и фиксации признаков нарушения штатного функционирования объекта оценки или его программ при обработке таких данных;
- анализ трасс выполнения объекта оценки, зафиксированных в ходе его функционирования, с целью выявления уязвимостей и опасных функциональных объектов в объекте оценки, а также контроля соответствия связей между информационными и функциональными объектами объекта оценки.

Фаззинг – это метод тестирования программного обеспечения, при котором на вход программы подаются случайные, неправильные или мусорные данные. Такой метод позволяет не только выявить ошибки логики работы программы, но и найти уязвимости. Чаще всего это ошибки работы с памятью, которые приводят к переполнению буфера на стеке и куче, раскрытию информации через неконтролируемую форматную строку, повреждение памяти через двойное освобождение или ее использование после освобождения.

Первая программа (и сам термин fuzz) была представлена осенью 1988 в Университете Висконсин как проект под названием «Operating System Utility Program Reliability – The Fuzz Generator»¹³. Цель проекта была в том, чтобы понять, когда и на каком типе данных происходит аварийное завершение тестируемой программы.

Существует несколько типов фаззинга, которые зависят от того, как фаззер работает с входными данными, которые в терминологии фаззинга правильно именовать *корпус*. Положиться на полностью случайные входные данные нельзя, поскольку программа ожидает на входе увидеть определенную структуру данных, например, конвертер png в jpeg проверяет, что первые 8 бит png-файла представляют его сигнатуру (89 50 4E 47 0D 0A 1A 0A), и если она не совпадает, то программа завершит свое выполнение с ошибкой и определенным кодом возврата. И это совершенно нормальный процесс, поскольку так было задумано программистом при разработке конвертера изображений. Найти уязвимость фаззером, который генерирует входные данные каждый раз случайно, очень тяжело, а в наше время практически невозможно. Поэтому были придуманы два основных подхода к фаззингу: мутационный и генерационный.

При мутационном фаззинге корпус изменяется случайным образом или на основании эвристик, например, к нему добавляется строка или в корпусе

¹³ <http://pages.cs.wisc.edu/~bart/fuzz/CS736-Projects-f1988.pdf>

происходит смещение бит. Такой процесс происходит при каждом новом проходе фаззера.

При генерационном фаззинге входных данных как таковых нет. Вместо этого есть описание их структуры, на основании которых фаззер входные данные генерирует. Один из ярких примеров генерационного фаззера – Peach Fuzzer¹⁴, который имеет в том числе community-версию и открытые исходные тексты. Peach использует XML в качестве инструмента описания структуры входных данных. Такие файлы носят название «Peach Pit» и обычно содержат всю необходимую информацию для проведения фаззинга – описывают модель данных и модель состояний, агентов, а также мониторов для отслеживания аварийных завершений программы.

Наиболее известным открытым мутационным фаззером является American Fuzzy Loop (AFL) фаззер, разработанный Michal Zalewski¹⁵. По большей части AFL работает с исходными текстами ПО, инструментируя их во время сборки. Это позволяет установить связь между входными данными и ветвлениями в программе, что позволяет увеличить покрытие фаззинга.

Для проведения сертификационных испытаний разработчик передает исходные тексты своего ПО в испытательную лабораторию. Однако в ряде случаев исходных текстов у разработчика может не быть. В этом случае описанный выше вид фаззинга не может быть применен.

Фаззинг приложения, при котором недоступны его исходные тексты, называется black-box фаззингом или фаззингом черного ящика. В этом случае происходит инструментация бинарного файла на лету за счет эмуляции пространства пользователя с помощью QEMU¹⁶. Подробнее про это можно узнать на странице проекта в сети Интернет.

Фаззинг – одна из самых мощных и проверенных стратегий для выявления проблем безопасности в программном обеспечении. В наши дни большинство обнаруженных в ПО ошибок, связанных с удаленным выполнением кода и повышением привилегий, найдены при помощи фаззинга. Помимо AFL и Peach фаззера, хочется отметить и другие разработки, такие как:

- libFuzzer¹⁷ – является частью проекта LLVM и осуществляет In Memory Fuzzing – мутации напрямую в памяти без необходимости перезапуска исследуемого ПО;
- driller¹⁸ – совмещает в себе функции фаззера и инструмента символьного исполнения;
- Fuzzilli¹⁹ – фаззер с обратной связью для JavaScript;

¹⁴ <https://www.peach.tech/resources/peachcommunity/>

¹⁵ <http://lcamtuf.coredump.cx/afl/>

¹⁶ https://github.com/google/AFL/blob/master/qemu_mode/README.qemu

¹⁷ <http://llvm.org/docs/LibFuzzer.html>

¹⁸ https://sites.cs.ucsb.edu/~vigna/publications/2016_NDSS_Driller.pdf

¹⁹ <https://github.com/googleprojectzero/fuzzilli>

– Eclipse²⁰ – фаззер бинарных приложений, который использует технику символьного исполнения серого ящика.

3.8. Ручной анализ объекта оценки

В ходе проведения исследований должен быть проведен ручной анализ, направленный на выявление архитектурных уязвимостей, уязвимостей кода и опасных функциональных объектов в объекте оценки.

Ручной анализ предусматривает экспертизу участков исходного/восстановленного кода, недействующего в ходе статического анализа кода, фаззинг-тестирования, анализа трасс выполнения, а также участков исходного/восстановленного кода, написанного с использованием языков программирования отличных от заявленного кода и самомодифицирующегося кода.

3.9. Оформление результатов исследований

По результатам проведения исследований по выявлению уязвимостей и НДВ должен быть разработан протокол проведения исследований. В протоколе должны быть отражены условия и результаты каждого исследования.

²⁰ <https://github.com/SoftSec-KAIST/Eclipser>

4. СРЕДСТВО ДЛЯ ПРОВЕДЕНИЯ ФАЗЗИНГ-ТЕСТИРОВАНИЯ AMERICAN FUZZY LOOP

4.1. Общая информация и основные функции American Fuzzy Loop

Алгоритм работы AFL-фаззера сводится к следующему:

1. Данные из корпуса загружаются в очередь.
2. Из очереди извлекается следующий вход и обрезается до наименьшего размера, который не изменяет поведения программы.
3. Вход мутируется с использованием различных методов.
4. Вход подается на программу.
5. Если вход привел к открытию нового состояния программ (о котором нам известно благодаря инструментации), то этот вход добавляется в очередь.
6. Переход на шаг 2.

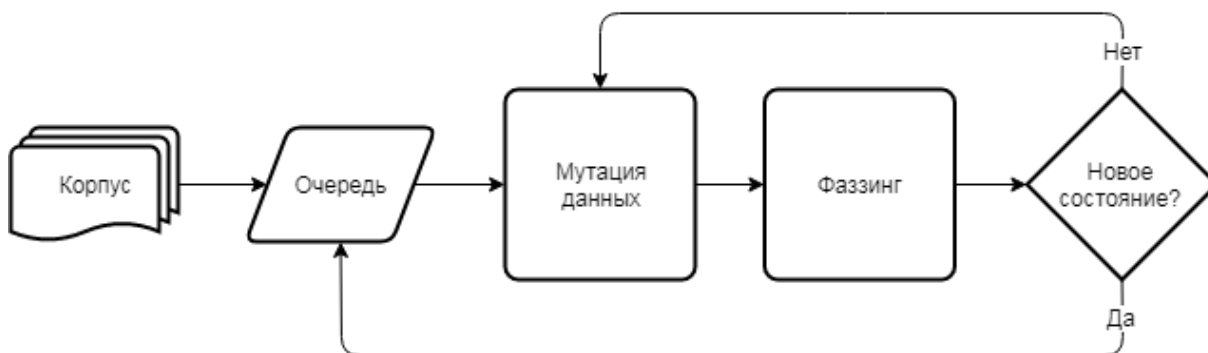


Рисунок 10 – Работа AFL-фаззера

4.2. Работа с American Fuzzy Loop

Для работы потребуется ОС Ubuntu Server 18.04.3 LTS.

Примечание: «\$» в начале строки говорит о том, что команда должна быть выполнена от имени пользователя, в то время как «#» говорит о том, что команда должна быть выполнена от имени суперпользователя.

До того, как вы начнете работу с фаззером, совершите следующие действия:

1. Установите дополнительное ПО:

```
$ sudo apt install gcc  
$ sudo apt install make  
$ sudo apt install libclang-dev  
$ sudo apt install clang
```

2. Клонировать репозиторий с последней версией AFL-фаззера:

```
$ git clone https://github.com/google/AFL.git
```

3. Перейдите в каталог с исходными текстами AFL-фаззера, соберите его и установите:

```
$ cd AFL
$ make
$ sudo make install
```

Первое, что необходимо сделать, это провести инструментацию исходного текста программы. Универсальным подходом считается запуск скрипта `configure` с переменной среды `CC/CXX`, которая указывает путь к `afl-gcc/afl-g++` и последующая сборка при помощи `make`:

```
$ CC=/path/to/afl/afl-gcc ./configure
$ make clean all
```

Затем выполните следующую команду для получения доступа к дампу памяти AFL-фаззеру:

```
# echo core > /proc/sys/kernel/core_pattern
```

Сам процесс фаззинга выполняется утилитой `afl-fuzz`. Для этой программы требуются доступный для чтения каталог с корпусом, отдельное место для хранения результатов и путь к бинарному файлу для тестирования.

Для программ, которые принимают входные данные непосредственно из стандартного потока ввода, синтаксис запуска выглядит следующим образом:

```
$ afl-fuzz -i input_dir -o output_dir /path/to/program
[... params ...]
```

Для программ, которые принимают входные данные из файла, используйте «@@», чтобы отметить место в строке команды, где должно быть размещено имя входного файла:

```
$ afl-fuzz -i input_dir -o output_dir /path/to/program @@
```

При выборе входных данных рекомендуется использовать файлы небольшого размера, хотя это не строго необходимо, но позволяет увеличить производительность фаззера.

Если у вас в распоряжении имеется несколько ядер или центральных процессоров, то рекомендуется запустить несколько фаззеров, при этом один из них должен быть запущен в режиме `master`, а остальные в режиме `slave`:

```
$ afl-fuzz -i input_dir -o shared_dir -M master /path/to/program @@
$ afl-fuzz -i input_dir -o shared_dir -S slave01 /path/to/program @@
$ afl-fuzz -i input_dir -o shared_dir -S slave02 /path/to/program @@
$ ...
```

Запущенные экземпляры время от времени будут синхронизировать свою работу, обмениваясь информацией о найденных путях выполнения исследуемой программы.

После запуска AFL-фаззера отобразится окно состояния (Рисунок 11).

Секция `findings in depth` содержит информацию о наиболее удачных путях выполнения программы, которые привели к наибольшему покрытию, а также дополнительные счетчики – общее количество аварийных завершений и таймаутов ПО.

Секция `fuzzing strategy yields` позволяет отслеживать отношение количества найденных путей к общему числу запусков по каждому из методов мутации.

Секция `path geometry` отражает глубину мутаций (`levels 1` – данные от пользователя, `levels 2` – единожды мутированные данные от пользователя, `levels 3` – данные, с предыдущего шага, которые были подвергнуты мутации еще раз и т.д.), `pending` показывает количество входов, которые еще не подверглись мутации, `pend fav` – входы, которые могут стать избранными для фаззера, `own finds` и `imported` показывают количество путей, найденных фаззером и количество путей, которые фаззер импортировал из других запущенных экземпляров фаззера при параллельном выполнении. Поле `stability` показывает консистентность наблюдаемых трасс выполнения исследуемой программы. Если значение меньше 100%, то могут возникнуть проблемы с результатами.

На рисунках 13 и 14 показано состояние двух экземпляров AFL-фаззера после 15 часов работы над утилитой `readelf` из пакета `binutils-2.25`. Как видно, `slave`-экземпляру удалось найти 40 уникальных аварийных завершений программы, в то время как `master` не нашел и одного. Это может быть связано с тем, что `master` экземпляр выполняет мутации пакетами, т.е. один тип мутации применяется к набору входных данных тысячи раз, в то время как `slave` по умолчанию выполняет мутации случайным образом, что в этом случае позволило ему быстрее найти новые пути выполнения исследуемой программы и, как следствие, аварийные завершения.

Попытки запуска программы с входными данными, обнаруженными фаззером, приведут к аварийному завершению (Рисунок 12).

```
$ ./binutils-2.25/binutils/readelf -a binutils-2.25/out/slave01/crashes/id\:\:0000...
```

```
readelf: readelf.c:7586: decode_tic6x_unwind_bytecode: Assertion `i < sizeof (buf)' failed.
Aborted (core dumped)
```

Рисунок 12 – Аварийное завершение программы

```

american fuzzy lop 2.56b (master)

lq process timing qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq overall results qqqqqk
x   run time : 0 days, 15 hrs, 20 min, 11 sec      x cycles done : 0      x
x last new path : 0 days, 0 hrs, 8 min, 38 sec    x total paths : 24.8k  x
x last uniq crash : none seen yet                x uniq crashes : 0    x
x last uniq hang : 0 days, 8 hrs, 11 min, 47 sec x   uniq hangs : 4    x
tq cycle progress qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq map coverage qvqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
x now processing : 18.4k* (74.00%)  x map density : 1.28% / 20.04%  x
x paths timed out : 6 (0.02%)      x count coverage : 5.05 bits/tuple  x
tq stage progress qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq findings in depth qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
x now trying : auto extras (over)  x favored paths : 2941 (11.84%)  x
x stage execs : 12.8k/35.9k (35.73%) x new edges on : 4813 (19.37%)  x
x total execs : 91.3M              x total crashes : 0 (0 unique)  x
x exec speed : 1281/sec            x total tmouts : 5410 (48 unique) x
tq fuzzing strategy yields qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq path geometry qqqqqqqqqq
x bit flips : 68/4.51M, 39/4.51M, 15/4.51M      x levels : 4              x
x byte flips : 3/564k, 0/361k, 0/370k          x pending : 22.5k        x
x arithmetics : 79/20.0M, 3/13.3M, 2/6.73M      x pend fav : 964         x
x known ints : 11/1.49M, 11/7.21M, 20/13.4M     x own finds : 479        x
x dictionary : 0/0, 0/0, 32/11.5M              x imported : 24.4k       x
x   havoc : 196/2.37M, 0/0                    x stability : 100.00%    x
x   trim : 42.78%/237k, 36.47%                tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq [cpu000:102%]

```

Рисунок 13 – Состояние master-фаззера после 15 часов работы

```

american fuzzy lop 2.56b (slave01)

lq process timing qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq overall results qqqqqk
x   run time : 0 days, 15 hrs, 20 min, 57 sec      x cycles done : 74      x
x last new path : 0 days, 0 hrs, 7 min, 23 sec    x total paths : 26.2k  x
x last uniq crash : 0 days, 1 hrs, 5 min, 24 sec  x uniq crashes : 40    x
x last uniq hang : 0 days, 0 hrs, 3 min, 20 sec   x   uniq hangs : 100   x
tq cycle progress qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq map coverage qvqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
x now processing : 12.8k* (48.76%)  x map density : 0.70% / 20.04%  x
x paths timed out : 0 (0.00%)      x count coverage : 5.05 bits/tuple  x
tq stage progress qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq findings in depth qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
x now trying : splice 7            x favored paths : 2948 (11.27%)  x
x stage execs : 33/64 (51.56%)      x new edges on : 4958 (18.95%)  x
x total execs : 97.7M              x total crashes : 65 (40 unique) x
x exec speed : 1679/sec            x total tmouts : 748 (155 unique) x
tq fuzzing strategy yields qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq path geometry qqqqqqqqqq
x bit flips : n/a, n/a, n/a          x levels : 24              x
x byte flips : n/a, n/a, n/a          x pending : 7618          x
x arithmetics : n/a, n/a, n/a        x pend fav : 0            x
x known ints : n/a, n/a, n/a        x own finds : 14.3k       x
x dictionary : n/a, n/a, n/a        x imported : 11.8k       x
x   havoc : 7858/34.7M, 6519/54.6M     x stability : 100.00%    x
x   trim : 59.39%/8.16M, n/a         tqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq [cpu001:100%]

```

Рисунок 14 – Состояние slave-фаззера после 15 часов работы

5. ЛАБОРАТОРНАЯ РАБОТА

При помощи AFL-фаззера проверьте одну из следующих утилит:

- `tiff2pdf` из состава `tiff-4.0.3`;
- `convert` из состава `ImageMagick-6.9.0-0`.

В утилите `tiff2pdf` содержится ошибка (идентификатор уязвимости CVE-2014-8128), которая позволяет записывать за границы выделенной памяти при конвертации TIFF изображения в PDF.

В утилите `convert` содержится ошибка (идентификатор уязвимости CVE-2014-9847), которая приводит программу в состоянии отказ в обслуживании при попытке конвертации изображения.

5.1. Порядок выполнения работы

Скачайте исходные тексты одной из указанной выше утилиты. Скомпилируйте их с использованием `afl-gcc`. Установите, используя `make install`. Запустите фаззер, предоставив действительные входные данные. Дождитесь нахождения аварийного завершения. Если вам кажется, что процесс фаззинга растянулся во времени, запустите несколько экземпляров фаззера параллельно. Запустите исследуемую программу с входными данными, которые расположены в папке `./out/crashes`

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое сертификация?
2. Сколько уровней контроля на отсутствие недекларируемых возможностей устанавливает РД НДС? Охарактеризуйте их.
3. Назовите основные этапы выявления уязвимостей и НДС в соответствии с Методикой.
4. Какие испытания включает в себя статический анализ?
5. Какие испытания включает в себя динамический анализ?
6. Опишите цель проведения тестирования на проникновение?
7. Какие виды испытаний программного обеспечения впервые появились в Методике?
8. Приведите примеры программ для проведения статического, динамического анализа.
9. Дайте определение термину «Фаззинг».
10. Ошибки какого типа могут быть выявлены с помощью фаззинга?
11. Какие типы фаззинга существуют?
12. Какой метод позволяет увеличить покрытие тестируемой программы при наличии ее исходных текстов?
13. Что такое black-box фаззинг?

ЗАКЛЮЧЕНИЕ

Данное учебно-методическое пособие призвано помочь студентам, изучающим в рамках своей образовательной программы дисциплину «Технология сертификации средств защиты информации», эффективнее освоить изучаемый материал – законодательство в области сертификации средств защиты информации, виды сертификации, участников сертификации, состав и содержание сертификационных испытаний, отличия и порядок проведения экспертного, статистического, динамического и ручного анализа исходных кодов.

Лабораторная работа, включенная в состав данного учебно-методического пособия, призвана помочь понять, как на практике выполняется один из видов динамического анализа – фаззинг.

СПИСОК ЛИТЕРАТУРЫ

1. О техническом регулировании [Электронный ресурс] : федеральный закон Российской Федерации от 27 дек. 2002 г. № 184-ФЗ, ред. от 29.07.2017 – Режим доступа: http://www.consultant.ru/document/cons_doc_LAW_40241/ (20.10.2019).
2. О сертификации средств защиты информации [Электронный ресурс] : Постановление Правительства РФ от 26 июня 1995 г. № 608, ред. от 21.04.2010 г. – Режим доступа: <http://base.garant.ru/102670/> (20.10.2019).
3. Защита от несанкционированного доступа к информации. Часть 1. Программное обеспечение средств защиты информации. Классификация по уровню контроля отсутствия недеklarированных возможностей [Текст] : руководящий документ от 4 июня 1999 г. № 114.
4. О системе сертификации средств защиты информации [Электронный ресурс] : Положение, утверждено приказом ФСТЭК России от 3 апреля 2018 г. № 55 – Режим доступа: <https://fstec.ru/component/attachments/download/1883> (20.10.2019).
5. Бегаев А.Н., Тарасюк М.В. Контроль безопасности программного кода в составе объекта информатизации [Текст] / Защита информации. Инсайд. 2013. № 5 (53). С. 63-67.
6. Марков А., Никулин М., Цирлов В. Сертификация средств защиты персональных данных: революция или эволюция [Текст] / Защита информации. Инсайд. 2008. № 5 (23). С. 20-25.
7. Марков А., Цирлов В. Сертификация программ: мифы и реальность [Текст] / Открытые системы. СУБД. 2011. № 6. С. 26.
8. Марков А.С., Цирлов В.Л., Барабанов А.В. Методы оценки несоответствия средств защиты информации [Текст]. Под. ред. А.С.Маркова. -М.: Радио и связь, 2012. 192 с.
9. Барабанов А., Марков А., Цирлов В. Сертификация систем обнаружения вторжений [Текст] / Открытые системы. СУБД. 2012. № 3. С. 31-33.
10. Барабанов А.В., Марков А.С., Цирлов В.Л. Испытания межсетевых экранов по требованиям безопасности информации [Текст]: Учебное издание. М.: НЦПИ при Минюсте России, 2017. 44 с.
11. Барабанов А.В., Марков А.С., Цирлов В.Л. Международная сертификация в области информационной безопасности [Текст] / Стандарты и качество. 2016. № 7. С. 30-33.
12. Барабанов А.В., Федичев А.В. Разработка типовой методики анализа уязвимостей в веб-приложениях при проведении сертификационных испытаний по требованиям безопасности информации [Текст] / Вопросы кибербезопасности. 2016. № 2 (15). С. 2-8.

13. Горюнов М.Н., Юдичев Р.М., Фадин А.А. Внедрение сертификации в жизненный цикл программного обеспечения [Текст] / Защита информации. Инсайд. 2016. № 3 (69). С. 28-35.
14. Марков А.С., Шеремет И.А. Теоретические аспекты сертификации средств защиты информации [Текст] / Оборонный комплекс - научно-техническому прогрессу России. 2015. № 4 (128). С. 7-15.
15. Александров Я.А., Сафин Л.К., Чернов А.В., Трошина К.Н. Определение границ подпрограмм при статическом анализе бинарных образов [Текст] / Вопросы кибербезопасности. 2016. № 1 (14). С. 53-60.
16. Барабанов А.В. Инструментальные средства проведения испытаний систем по требованиям безопасности информации [Текст] / Защита информации. Инсайд. 2011. № 1 (37). С. 49-51.
17. Барабанов А.В., Марков А.С., Фадин А.А., Цирлов В.Л. Статистика выявления уязвимостей программного обеспечения при проведении сертификационных испытаний [Текст] / Вопросы кибербезопасности. 2017. № 2 (20). С. 2-8.
18. Марков А.С., Барабанов А.В., Фадин А.А. Выявление недеklarированных возможностей в декомпилированных текстах программного обеспечения [Текст] / Известия Института инженерной физики. 2010. Т. 4. № 18. С. 24-26.
19. Марков А.С., Матвеев В.А., Фадин А.А., Цирлов В.Л. Эвристический анализ безопасности программного кода [Текст] / Вестник Московского государственного технического университета им. Н.Э. Баумана. Серия: Приборостроение. 2016. № 1 (106). С. 98-111.
20. Марков А.С., Фадин А.А. Статический сигнатурный анализ безопасности программ [Текст] / Программная инженерия и информационная безопасность. 2013. № 1. С. 50-56.
21. Марков А.С., Фадин А.А., Цирлов В.Л. Концептуальные основы построения анализатора безопасности программного кода [Текст] / Программные продукты и системы. 2013. № 1. С. 10.
22. Мельников П.В., Горюнов М.Н., Анисимов Д.В. Подход к проведению динамического анализа исходных текстов программ [Текст] / Вопросы кибербезопасности. 2016. № 3 (16). С. 33-39.
23. Осовецкий Л.Г. Технология выявления недеklarированных возможностей при сертификации промышленного программного обеспечения по требованиям безопасности информации [Текст] / Вопросы кибербезопасности. 2015. № 1 (9). С. 60-64.
24. Осовецкий Л.Г., Ефимова А.В. Гарантии выявления недеklarированных возможностей в промышленном программном обеспечении [Текст] / Известия Института инженерной физики. 2016. Т. 2. № 40. С. 59-63.

Бегаев Алексей Николаевич
Кашин Семен Владимирович
Маркевич Никита Алексеевич
Марченко Алина Андреевна

**Выявление уязвимостей и недеklarированных
возможностей в программном обеспечении**

Учебно-методическое пособие

В авторской редакции

Редакционно-издательский отдел Университета ИТМО

Зав. РИО

Н.Ф. Гусарова

Подписано к печати

Заказ №

Тираж

Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверский пр., 49