Шаг 2. Открыть процесс на отладку, используя системный вызов `ptrace`.

Шаг 3. Изменить код библиотеки `ld-linux.so`, находящийся в адресном пространстве процесса, так, чтобы можно было загрузить произвольную библиотеку в адресное пространство процесса.

Шаг 4. Записать код, подгружающий библиотеку из шага 1, в любую область памяти, имеющую право на выполнение кода процессом.

Шаг 5. Изменить указатель следующей выполняемой команды на адрес кода, подгружающего библиотеку.

Шаг 6. Закрыть процесс, используя системный вызов `ptrace`. При этом процесс начинает выполнять код, подгружающий библиотеку.

Эти методы реализованы и опробованы в дистрибутиве `Gentoo`.

ЛИТЕРАТУРА

1. Using Process Infection to Bypass Windows Software Firewalls [Электронный ресурс]. — Режим доступа : `http://www.phrack.org/issues.php?issue=62&id=13`
2. `http://www.phrack.com/issues.html?issue=65&id=10` phook — The PEB Hooker [Электронный ресурс].
3. `http://www.phrack.com/issues.html?issue=59&id=12` Building ptrace injecting shellcodes [Электронный ресурс].

УДК 004.42

РАЗРАБОТКА И РЕАЛИЗАЦИЯ СЕРВЕРА ИГРЫ CTF[1]

Н. О. Ткаченко, Д. В. Чернов

Соревнования по защите информации Capture the Flag (CTF) [1] традиционно проводятся по следующим правилам. Каждой команде перед началом игры выдается одинаковый образ виртуальной машины с какой-либо операционной системой, на которой установлен определенный набор сервисов, содержащих уязвимости. На эти сервисы в процессе игры жюри высылает некоторую информацию, называемую флагами. Цель игры — захватить флаги противника, используя найденные в сервисах уязвимости, и при этом защитить свои флаги, устраняя уязвимости в собственных сервисах. Командам начисляются баллы как за защиту собственного флага, так и за успешное обнаружение чужого. Лучшая и самая простая защита флагов — их удаление, но, по очевидным причинам, такая защита запрещена правилами. Жюри постоянно проверяет сервисы на наличие флагов и снимает очки с команды, если не может получить к ним доступ. В процессе игры командам предоставляется возможность решать дополнительные задания, называемые *квестами*, или давать советы другим командам по увеличению надежности их сервисов, называемые *эдвайзори*. За это командам также начисляются очки.

В классических правилах первый час игры команды изолированы друг от друга межсетевым экраном. Данное время отводится для настройки оборудования и программ, а также для начального ознакомления с сервисами. Затем жюри предоставляет доступ командам к подсетям друг друга и начинает отправлять на сервисы флаги. Во время игры каждая команда имеет доступ к любому сервису противника.

[1]Работа выполнена в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг. (гос. контракт № П1010).

Команда Томского государственного университета SiBears [2] предложила внести изменения в правила, при которых осуществляется бóльшая связь между всеми аспектами игры, чем в классических правилах CTF.

Целью работы является создание игрового сервера CTF, функционирующего по измененным правилам: команды большую часть времени изолированы друг от друга межсетевым экраном, которым управляет жюри. Для атаки на сервис X команды B команда A должна отправить соответствующий запрос игровому серверу. Запрос может быть либо удовлетворен, либо отклонен. Разрешение на атаку производится в следующих случаях: атакуемая команда B разрешает атаку; у команды A есть в наличии решенный квест или опубликованный эдвайзори; команда B отказала в атаке на сервис X больше определенного числа раз. Во всех остальных случаях запрос на атаку не удовлетворяется. После проведения атаки жюри начисляет очки за атаку и защиту.

В качестве архитектуры игрового сервера был использован шаблон Model—View—Controller (MVC, «модель — представление — контроллер») [3]. Приложение разделено на три компоненты — модель данных приложения, пользовательский интерфейс и управляющая логика:

— *модель* (model) предоставляет данные (обычно для представления), а также реагирует на запросы (обычно от контроллера), изменяя свое состояние;
— *представление* (view) отвечает за отображение информации, то есть это пользовательский интерфейс;
— *контроллер* (controller) интерпретирует данные, введенные пользователем, и информирует модель и представление о необходимости соответствующей реакции.

Модификация одной из компонент оказывает минимальное воздействие на другие. Взаимосвязь компонент представлена на рис. 1.

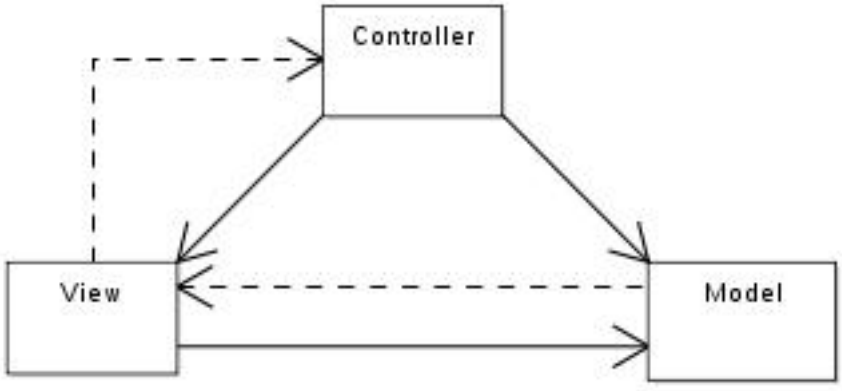

Рис. 1. Компоненты архитектуры Model—View—Controller

Модель реализована в виде реляционной базы данных, методом работы с которой является Object Relational Mapping (ORM) [4]. Суть проблемы, которая решается с помощью ORM-слоя, заключается в необходимости преобразования объектных структур в памяти приложения в форму, удобную для сохранения в реляционных базах данных, а также для решения обратной задачи — представления реляционной модели в объектной с сохранением свойств объектов и отношений между ними. В данной работе построена база данных для хранения и обработки результатов игры; для данной базы данных реализован ORM-слой. Для каждой её таблицы реализован свой класс, член-данными которого являются поля этой таблицы. Член-функциями класса являются функции, необходимые для связи класса с соответствующей таблицей. На рис. 2

представлен пример соответствия классов в объектно-ориентированной программе и таблиц в реляционной базе данных.

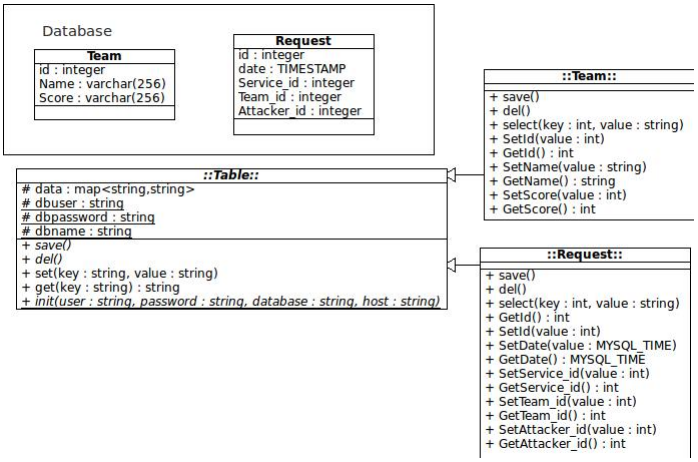

Рис. 2. Пример соответствия классов и таблиц в ORM

Для представления пользователю информации о текущем состоянии игрового сервера, а также для приема его запросов используется клиент-серверная архитектура веб-приложения. Пользователи (жюри и команды) с помощью веб-браузера отправляют запросы веб-серверу, который, в свою очередь, передаёт их обработчикам, возвращающим результат в формате XHTML.

Обработчики запросов реализуют *контроллер* игрового сервера. Они написаны с использованием стандарта CGI (Common Gateway Interface — «общий интерфейс шлюза»), что позволяет избавиться от необходимости делать игровой сервер многопоточным — задача по одновременной обработке запросов от пользователей возлагается на веб-сервер. Так как CGI-скрипт в основном работает только с вводом/выводом, он осуществляет достаточно быструю работу.

Представление реализовано преобразованием XML-данных, получаемых на выходе CGI-скриптов, в XHTML — формат данных для отображения с помощью веб-браузеров.

Для реализации игрового сервера был выбран язык программирования C++. Это обусловлено тем, что C++ является компилируемым объектно-ориентированным языком программирования с поддержкой множества дополнительных библиотек.

ЛИТЕРАТУРА

1. *Колегов Д. Н., Чернушенко Ю. Н.* О соревнованиях CTF по компьютерной безопасности // Прикладная дискретная математика. 2008. №2(2). С. 81–83.
2. Команда SiBears / Томский государственный университет. Электрон. дан., 2010. Режим доступа: `http://sibears.ru`, свободный.
3. The Model-View-Controller (MVC) in past and present / Reenskaug T., University of Oslo. Электрон. дан., 2003. Режим доступа: `http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf`, свободный.
4. *Ambler S.* Agile database techniques: effective strategies for the agile software developer. Wiley, 2003. 480 p.