УДК 621.391 (075.8)

А. В. Бєгун, канд. екон. наук,
ДВНЗ «КНЕУ імені Вадима Гетьмана»

АНАЛІЗ ЗАГРОЗ ІНФОРМАЦІЇ WEB-ПОРТАЛУ ЧЕРЕЗ АТАКИ НА WEB-ДОДАТКИ

АНОТАЦІЯ. В статті розглянуто підходи до аналізу можливих загроз Web-порталу через атаки на Web-додатки, визначено, що контроль введення й перевірки достовірності даних є ключовими факторами для забезпечення їхнього захисту. Одним з перших кроків тестування має бути визначення оточення, у якому виконується Web-додаток, що включає в собі інформацію про використовувану скриптову мову, Web-сервер та операційну систему.

ANNOTATION. The article examines approaches to the analysis of potential threats to Web-portal through attacks on Web-based applications, is that the control input and data validation are key factors to ensure their protection. One of the first steps of testing should determine the environment in which the Web-application that contains information about used scripting language, Web-server and operating system.

КЛЮЧОВІ СЛОВА. Web-портал, Web-додатки, тестування, захист, атака, програмний контент, хост.

В Інтернет почав приходити бізнес — здійснюється купівля і продаж через мережу, широкого вжитку набувають системи «електронних грошей». Довіряючи мережі свою конфіденційну інформацію, свої фінанси суб'єкти хочуть бути впевненими в їх захисті від шахраїв, тому дуже гостро стоїть питання захисту інформації Web-порталів [1].

Сучасний Web-портал неможливий без використання Web-додатків, на які часто спрямовують свої атаки зловмисники.

Web-додатки стають усе більш розповсюдженими й усе більш складними, відіграючи в такий спосіб основну роль у більшості онлайнових проектів. Як і в усіх системах, заснованих на взаємодії між клієнтом і сервером, вразливості Web-додатків зазвичай виникають через некоректну обробку запитів клієнта й/або недостатню перевірку вхідної інформації з боку розробника.

Сама природа Web-додатків — їхня здатність зіставляти, обробляти та поширювати інформацію через Internet — робить їх подвійно вразливими. По-перше, що саме очевидне, вони повністю відкриті через необхідність забезпечення публічного доступу. Це унеможливлює застосування техніки «security through obscurity» (безпека за рахунок приховування інформації) і підвищує вимогу до стійкості коду. По-друге, що важливе з точки зору тестування на проникнення, вони обробляють дані, отримані із запитів HTTP-

протоколу — протоколу, який може використовувати безліч способів кодування та інкапсуляції інформації.

Більшість середовищ, у яких виконуються додатки, передають ці дані розробнику таким чином, що він не в змозі визначити, звідки вони були отримані й, отже, якій перевірці мають бути піддані. Оскільки Web-середовище настільки різноманітне й містить багато форм програмного контенту, контроль введення й перевірки достовірності даних є ключовими факторами для забезпечення безпеки Web-додатків. Це включає як виявлення областей можливих значень для всіх елементів даних, що вводяться користувачем (і примусове завдання коректних значень), так і достатнє розуміння того, звідки надходить інформація, з метою визначення тих даних, які можуть бути задані користувачем.

Помилки, що пов'язані з перевіркою коректності введення, часто досить складно виявити у великому обсязі коду, який взаємодіє з користувачем. Це є основною причиною того, що розробники використовують методологію тестування додатків на проникнення для їхнього виявлення. Web-додатки, однак, не мають імунітету й до більш традиційних способів атаки. Досить поширені погані механізми аутентифікації, логічні помилки, ненавмисне розкриття інформації, а також такі традиційні помилки для звичайних додатків, як переповнення буфера. Приступаючи до тестування Web-додатків, необхідно брати до уваги все перераховане і застосовувати методичний підхід тестування вводу/виводу за схемою «чорної скриньки» у сполученні (якщо можливо) з аудитом вихідного коду.

Web-додаток являє собою систему, що звичайно включає набір скриптів, розташованих на Web-сервері й взаємодіючих з базами даних та іншими джерелами динамічного контенту. Вони швидко стають всюдисущими, оскільки дозволяють постачальникам послуг і їхнім клієнтам обмінюватися й маніпулювати інформацією незалежним від платформи способом за допомогою інфраструктури Internet. Прикладами Web-додатків є пошукові системи, Web-пошти, онлайнові магазини і т.п.

Web-додатки звичайно взаємодіють з користувачем за допомогою елементів форм і змінних GET або POST. У випадку використання методу GET всі значення, передані додатку, можуть бути показані безпосередньо в URL, у той час як у випадку POST-запитів для визначення того, що вводить користувач, часто доводиться вивчити вихідний текст сторінки, що містить форму.

HTTP-запит, переданий типовим Web-додатком, виглядає таким чином:

GET /sample.php?var=val&var2=val2 HTTP/1.1 | метод REQUEST-URL протокол/версія Session-ID: 361873127da673c | заголовок Session-ID Host: www.webserver.com | заголовок Host <CR><LF><CR><LF> |

Кожен елемент цього запиту потенційно може бути використаний Web-додатком, який його обробляє. В REQUEST-URL перебуває адреса програмної одиниці, що буде запущена для обробки запиту, а також рядок зі списком параметрів — список пар &variable=value, що визначають вхідні параметри. Це основна частина вхідних даних Web-додатків. Заголовок Session-ID містить ідентифікатор сесії, встановленої клієнтом як примітивна форма аутентифікації. Заголовок Host призначений для вибору одного з декількох віртуальних хостів, що використовують ту саму IP-адресу, і звичайно обробляється Web-сервером, але може бути доступний і Web-додатку.

Тестуючий повинен використати всі доступні методи введення даних для досягнення виняткових умов усередині додатку. HTTP-запити дуже легко генеруються за допомогою таких утиліт, як curl, або скриптів оболонки, що використовують netcat. Процес вичерпного тестування по методу чорної скриньки містить у собі дослідження кожного елемента даних, визначення очікуваного уведення, перекручування його, і аналіз результатів роботи додатка на предмет виявлення ознак несподіваного поводження.

Одним з перших кроків тестування має бути визначення оточення, у якому виконується Web-додаток. Це включає інформацію про використовувану скриптову мову, Web-сервер та операційну систему. Всі ці критичні деталі легко вивідати у типового Web-сервера в такий спосіб:

проаналізувати відгук на HTTP-запити HEAD й OPTIONS .

Заголовок і будь-яка сторінка, повернута запиту HEAD й OPTIONS, зазвичай містить рядок SERVER: (або щось аналогічне), у якому вміщується інформація про версії Web-сервера й, можливо, інформація про оточення або використовувану операційну систему:

OPTIONS / HTTP/1.0 HTTP/1.1 200 OK Server: Microsoft-IIS/5.0 Date: Wed, 04 Jun 2009 11:02:45 GMT MS-Author-Via: DAV Content-Length: 0 Accept-Ranges: none DASL: <DAV:sql> DAV: 1, 2 Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK Cache-Control: private

Деякі системи мають специфічні й, отже, легко пізнавані повідомлення про помилки, а також часто дозволяють визначити версії використовуваних скриптових мов. Тестуючий повинен навмисно запросити сторінки, що приводять до подібних помилок, а також використати альтернативні методи запиту (POST, PUT і т.п.) для добування цієї информації з сервера.

Багато Web-серверів по-різному реагують на запити файлів з підтримуваними й невідомими розширеннями. Тестуючому варто спробувати запросити файли зі стандартними розширеннями, такими як .ASP, .HTM, .PHP, .EXE, і стежити за появою якого-небудь незвичайного результату або кодів помилок [2].

GET /blah.idq HTTP/1.0 HTTP/1.1 200 OK Server: Microsoft-IIS/5.0 Date: Wed, 04 Jun 2009 11:12:24 GMT Content-Type: text/html <HTML>The IDQ file blah.idq could not be found.

Вихідний текст сторінки, згенерованої Web-додатком, може дати натяк на використовуване програмне оточення.

<title>Home Page</title> <meta content=«Microsoft Visual Studio 7.0» name=«GENERATOR»> <meta content=«C#» name=«CODE_LANGUAGE»> <meta content=«JavaScript» name=«vs_defaultClient Script»>

У цьому випадку, схоже, розробник використав MS Visual Studio 7, а додаток, швидше за все, виконується на Microsoft IIS 5.0 з .NET framework.

За допомогою традиційних засобів сканування, таких як Nmap й Queso, або більше нових аналізаторів додатків Amap і webServerFP, тестуючий може одержати більше точну інформацію про використану операційну систему й використане програмне оточення, ніж за допомогою багатьох інших методів. Nmap і Queso використають перевірку особливостей реалізації TCP/IP на досліджуваному хості для визначення операційної системи й, у деяких випадках, версій ядра й встановлених патчів. Аналізатори додатків використовують таку інформацію, як серверні HTTP-заголовки для визначення запущених на хості додатків.

У багатьох випадках розробники вимагають від клієнтів вхідні дані, які повинні бути захищені від зміни користувачем — такі як ідентифікатор користувача, що генерується динамічно, передається клієнтові, після чого використається у всіх наступних запитах. Для того, щоб перешкодити користувачам побачити й спотворити цю інформацію, розробники звичайно використають елементи форм із тегом HIDDEN. На жаль, ця інформація не видна не лише у відображеній сторінці, але й у її вихідному коді.

Відомо багато прикладів безграмотно спроектованих систем, що дозволяють, приміром, зберегти локальну копію сторінки з підтвердженням замовлення, відредагувати заховані змінні, у яких зберігається ціна/вартість доставки, після чого відправити виправлене замовлення. Web-додаток не перевіряє дані, і замовлення обслуговується з великою знижкою :

<FORM METHOD=«LINK» ACTION=«/shop/checkout.htm»> <INPUT TYPE=«HIDDEN» name=«quoteprice» value=«4.25»> Quantity: <INPUT TYPE=«text» NAME=«totalnum»> <INPUT TYPE=«submit» VALUE=«Checkout»> </FORM>

Такий підхід дотепер можна зустріти на багатьох сайтах, хоча вже й у меншому обсязі. Звичайно в схованих полях передається некритична або попередньо зашифрована інформація. Однак, поза залежністю від значимості цих полів, вони є ще одним обов'язковим місцем додатку зусиль тестуючого.

Всі вихідні тексти сторінок повинні бути перевірені на предмет виявлення будь-якої корисної інформації, ненавмисно залишеної розробником — це можуть бути фрагменти скриптів, розташованих усередині HTML-коду, посилання на які підключають або зв'язані скрипти, неакуратні права доступу до критичних файлів з вихідним текстом. Має бути перевірена наявність кожної програми й скрипта, посилання на які були знайдені. У випадку виявлення вони теж повинні бути протестованими.

Код на Javascript й інших аналогічних клієнтських скриптових мовах також може надати багато відомостей про внутрішню логіку роботи Web-додатку, що є досить важливим при тестуванні за принципом чорної скриньки [3]. На відміну від тестування з аудитом коду, коли є доступ до відомостей про логіку роботи додатку, для закритого тестування подібна інформація є розкішшю, що може забезпечити засоби для майбутніх атак. Наприклад, розглянемо наступний фрагмент коду:

<INPUT TYPE=«SUBMIT» onClick=« if (document.forms['product'].elements['quantity'].value >= 255) { document.forms['product'].elements['quantity'].value=''; alert('Invalid quantity'); return false; } else { return true; } «>

Це означає, що додаток намагається захиститися в оброблювачі форми від передачі у змінну quantity значень, більше або рівних 255 — максимальне значення для поля у більшості баз даних. Ця перевірка з боку клієнта обходиться елементарно, після чого записуємо в поле quantite більше значення, відправляємо запит і дивимося, чи не викличе це якогось нестандартного поводження додатку.

Одніє з головних вад середовища Web-додатків є неможливість забезпечення стійкого механізму аутентифікації. Ще більшою проблемою є нездатність ефективно використати доступні механізми. HTTP забезпечує два види аутентифікації: Basic й Digest. Обидва вони реалізовані як серія HTTP-запитів і відповідей, у рамках якої клієнт запитує ресурс, сервер вимагає аутентифікацію, і клієнт повторює запит, забезпечивши його аутентифікаційним посвідченням (логін + пароль). Вся різниця між ними полягає в тому, що у випадку Basic-аутентифікації інформація передається у відкритому вигляді, а у випадку Digest — шифрується за допомогою ключа, згенерованого сервером.

Крім очевидної проблеми використання відкритого тексту у випадку Basic-аутентифікації, тут немає нічого принципово поганого, та й ця проблема відкритого тексту пом'якшується при використанні HTTPS. Реальна ж проблема подвійна. По-перше, оскільки ця аутентифікація здійснюється Web-сервером, її не так легко контролювати з Web-додатку у випадку відсутності інтерфейсу взаємодії з аутентифікаційною базою даних Web-сервера. Отже, часто використовуються саморобні механізми аутентифікації, тим самим насправді відкриваючи «ящик Пандори» [1]. По-друге, розробникам часто просто не вдається коректно застосувати механізм аутентифікації для обмеження доступу до всіх ресурсів.

Враховуючи це, тестуючий повинен спробувати встановити як використовуваний механізм аутентифікації, так і те, яким чином цей механізм застосований по відношенню до кожного ресурсу, використовуваному Web-додатком. Багато Web-систем пропонують засобу підтримки сесій, засновані на збереженні в cookie або Session-ID псевдоунікального рядка, що характеризує їхній статус. Такий підхід може бути уразливий до таких атак, як прямий перебір, повторне відправлення, або спроба відновлення — у тому випадку, якщо даний рядок є простим хешем або рядком, складеним з відомих елементів.

Варто спробувати одержати доступ до всіх ресурсів через кожне місце входу. Це може виявити наступну помилку — у той час як головне меню вимагає аутентифікації, ресурси, до яких воно надає доступ, її вже не вимагають. Прикладом може послужити Web-додаток, що забезпечує доступ до різних документів у такий спосіб. Додаток вимагає пароль, після чого авторизований користувач одержує меню з документами, де кожний документ представлений у ньому як посилання на ресурс виду ***http://www.server.com/showdoc.asp?docid=10.***

Незважаючи на те, що вивід меню вимагає авторизації, скрипт showdoc.asp її не вимагає й сліпо дає доступ усім бажаючим до запитаного документа, дозволяючи атакуючому просто вставити бажаний ідентифікатор та одержати документ.

Таким чином, ефективний захіст Web-додатоку від атак можливий тільки на стадії розробки шляхом моделювання дій зловмисника.

Література

1. *Низамутдинов М. Ф.* Тактика защиты и нападения на Web-приложения. — СПб.: БХВ — Петербург, 2005. — 432 с.
2. *I. Kiselev*. Aspect-Oriented Programming with AspectJ. Indianapolis, IN, USA: SAMS Publishing, 2002.
3. *J. Hannemann, G. Kiczales*. Design pattern implementations in Java and AspectJ OOPSLA 02, New York, USA, November 2002. P. 161—173.

УДК 338.43 : 519.246.8

П. М. Грицюк, канд. фіз.-мат. наук,
доцент к-ри прикладної математики,
Національний університет водного господарства
та природокористування

СТАТИСТИЧНЕ МОДЕЛЮВАННЯ РЕНТАБЕЛЬНОСТІ ВИРОЩУВАННЯ ЗЕРНОВИХ В УКРАЇНІ

АНОТАЦІЯ. В статті узагальнено досвід автора в галузі прогнозування часових рядів урожайності озимої пшениці для областей України. Виявлено сильний кореляційний зв'язок між рентабельністю вирощування зернових і валовим збором за два останніх роки. Побудовано регресійні моделі рентабельності вирощування зернових і озимої пшениці. Виконано прогнозну оцінку рентабельності для регіонів України.

ANNOTATION. In the paper the experience of author in the area of time series winter wheat yield forecasting for the regions of Ukraine is generalized. The correlation dependence between total grain crops for two last years and grain-production profitability is disclosed. The profitability regression models of grain production and winter wheat production are built. The projected profitability for the regions of Ukraine was carried out.

КЛЮЧОВІ СЛОВА. Часові ряди врожайності, полігармонічна модель врожайності, статистична модель рентабельності, регіональний коефіцієнт затрат.

1. Проблема стійкості сільськогосподарського виробництва. Одним із найважливіших досягнень так званої зеленої революції є стабілізація аграрного виробництва. Наприклад, за 25-річ-