Business logic vulnerabilities in web applications

1 Introduction

The rapid growth of Internet has influenced many companies in the way their services are delivered. More and more companies deploy online services allowing their clients and partners to perform their activities faster and more convenient. As the number of delivered services increase so does the complexity of the underlying information systems. The public interface exposed by these systems is commonly implemented via web applications. The lack of client software requirements and the common HTML interface are the major benefits of web applications.

As these information systems often deal with private data (credit card numbers, insurance numbers, home addresses, license numbers, etc.) and are exposed to public, the security requirements should be strictly held: the information system should enforce data confidentiality (privacy), data integrity and availability of services [1]. However, due to complexity of systems, time-to-market pressure, security unawareness of web programmers, the most of the systems (90%) [2] suffer from errors that make possible breaking of confidentiality, integrity or availability of delivered services. These kinds of errors are widely known as security vulnerabilities, and activities performed to exploit them are called web attacks.

This paper discusses business logic vulnerabilities and reasons about possibility of automated detection thereof.

2 What are business logic vulnerabilities?

This section is intended to give an informal notion of what ‘business logic’ is and what vulnerabilities are business logic flaws. First, three definitions of term ‘business logic’ is given and then the approach to analysis of business logic is proposed.

Wikipedia [3]:

Business logic is a non-technical term generally used to describe the functional algorithms which handle information exchange between a database and a user interface. It is distinguished from input/output data validation and product logic. Business logic models real life business objects such as accounts, loans, itineraries, and inventories. It also prescribes how business objects interact with one another. Finally, business logic enforces the routes and the methods by which business objects are accessed and updated.

Business logic comprises [4]:

- business rules that express business policy (such as channels, location, logistics, prices, and products); and

- workflows that are the ordered tasks of passing documents or data from one participant (a person or a software system) to another.

PCMag Encyclopedia [5]:

The part of an application program that performs the required data processing of the business. It refers to the routines that perform the data entry, update, query and report processing, and more specifically to the processing that takes place behind the scenes rather than the presentation logic required to display the data on the screen (GUI processing).

Client applications are made up of user interface and business logic. Server applications are mostly business logic. Both client and server...
applications also require communications links, but the network infrastructure, like the user interface, is not part of the business logic.

Discussion at Simon Brown’s blog [6]:

Business logic is the set of rules, processes, and algorithms that operate on and with information (data) in the business domain, as implemented by the application. Business logic can, and should, be modeled and built with little or no regard as to user interface presentation or request/response communication methods.

The first observation is that business logic is defined as an abstract, nontechnical notion. Another observation is the presence of the requirement for software architecture models: it states that business logic, GUI and communication services should be included in the architecture model as separate blocks. This is the first step in defining the boundaries of ‘business logic’.

According to [7], architecture model of a specific system is a collection of computational components together with a description of the interactions between these components—the connectors. Thus architecture model is a certain decomposition of an entity under consideration.

This means that in order to find clear boundaries for business logic in web applications it is necessary to:

1. Review the existing approaches to model software architecture;
2. Consider how do the aforementioned business logic definitions map into web application decomposition induced by particular model;
3. Chose the model, which provides the clearest boundaries (or points of application) in the scope of the ‘business logic’ definitions.

The resulting decomposition will serve as the starting point to classification of web application vulnerabilities in general and business logic vulnerabilities in particular.

There are widely known architectural styles [7]: Pipes and Filters, Data Abstraction and Object-Oriented Organization, Event-based Implicit Invocation, Layered Systems, Repositories, Table Driven Interpreters, Distributed processes, Main program/subroutine organizations, State transition systems, Process control systems.

Of course, particular software architecture could be implemented as a superposition of the listed styles. To be more specific, web application developers widely adopted three styles: Distributed processes for multi-tier models, Event-based Implicit Invocation for MVC models and Layered Systems for multi-layered models. The concepts of layer and tier are often used interchangeably. However, one fairly common point of view is that there is indeed a difference, and that a layer is a logical structuring mechanism for the elements that make up software solution, while a tier is a physical structuring mechanism for the system infrastructure.

The following sections contain reasoning about what software styles (i.e. ways of decomposition) are applicable to web applications in the scope of business logic. The following architectural styles were not considered since they were developed for the different kind of software than web applications (see [7]): Repositories, Table Driven Interpreters and Process control systems. Furthermore, two other styles cannot be used for refinement of ‘business logic’ either: Data Abstraction and Object-Oriented Organization, Main program/subroutine organizations. These styles deal only with software implementation issues but the ‘business logic’ is implementation independent.

So, the following model will be considered: Pipes and Filters, Event-based Implicit Invocation (i.e. MVC), Layered Systems, Distributed processes (i.e. decomposition on the physical component basis) and State transition systems.

### 2.1 Distributed processes style

The model of common components running server side during HTTP request processing is presented in the Figure 1.
First, the incoming HTTP request is served by the web server. Web server parses it and determines the requested resource. If the resource is a registered executable (i.e. .jsp, .asp, .php, etc.) the web server invokes (we do not focus how: CGI, web server API, etc.) the execution of the requested executable and passes the request to it. The web application component is executed within the language runtime environment. The component uses the standard library (e.g. .NET CLR, Java Runtime Environment, PHP or Python standard library, etc.) and surrounding web framework (which is built upon standard library as well) to obtain parsed data from HTTP request and to manipulate it according to its business logic. At times web component utilizes services like OS system calls, DBMS, Mail, LDAP and so on. Eventually, web application by means of web framework and/or standard library generates HTTP response, which is finally delivered to the client via web server.

Apparently, the model of physical components resembles a sandwich, where subsequent component is wrapped into preceding one. To make things worse, often Language Runtime is wrapped into Web Server component. The examples include IIS with ASP, Geronimo, Apache with mod_php or mod_perl or mod_python, etc. This wrapping is the root cause of inconvenience for further refinement to taxonomy. The model does not provide clear and separate boundaries as it does not provide abstraction over technology. For example, in Java 5 EE authentication and authorization is basically implemented by web server, in ASP.NET – by web server (IIS) and Language Runtime (ASP library), in PHP and Perl – by web server and web frameworks.

2.2 **Event-based Implicit Invocation and Layered Systems styles**

As was already said, web application developers widely adopted three styles: Distributed processes for multi-tier models, Event-based Implicit Invocation for MVC models and Layered Systems for multi-layered models.

MVC and three-layer (presentation layer, business logic layer and data access layer) models are similar at the first glance, but not the same. These concepts are topologically different. A fundamental rule in three-layer architecture is that the presentation layer never communicates directly with the data access layer; in a three-layer model all communication must pass through the business logic layer. Conceptually the three layer architecture is linear. However, the MVC architecture is triangular: the View sends updates to the Controller, the Controller updates the Model, and the View gets updated directly from the Model. The MVC concept was introduced by Smalltalk developers in early 1980s. The three layer concept appeared only in the 1990s from observations of distributed systems.
Both of these models incorporate business logic completely into separate element, specifically designed to represent it. Two models just reflect the stated previously property of business logic (“business logic, GUI and communication services should be included in the architecture model as separate blocks”) and do not provided more insight into business logic structure.

2.3 Pipes and Filters and State transition systems styles

In a pipe and filter style each component has a set of inputs and a set of outputs. A component reads streams of data on its inputs and produces streams of data on its outputs. The State transition systems are defined in terms of states and transitions that move a system from one state to another.

The proposed view of web application is as follows:

- Web application is built from a number of filters. Filters include both common items like authentication, encryption, authorization, input validation, and custom ones, which correspond to processing domain specific data.
- Web application is viewed as a State transition system. States represent logical points in web application workflow process each client is currently in. Transitions are built from the noted filters and are used to change web application states (i.e. advance users throughout web application workflow). Thus, working session of particular user maps to a path consisting of certain filters, the sequence of which is controlled by state transition rules.

According to Church-Turing thesis, Turing machines capture the informal notion of effective method in logic and mathematics, and provide a precise definition of an algorithm or 'mechanical procedure’. The Turing machine is a finite state machine. This is the motivation to utilize this architectural style. The pipes and filters style is used for convenient abstraction of state transitions in particular and web application workflow in general.

The basic workflow of web application is shown in the Figure 2.

![Figure 2. Basic workflow for web application.](attachment:image)

This decomposition implies an important property on filters: the closer common filters are located to the custom filters (in web application workflows) the more abstract output structure they generate. Custom filters operate with most abstract data from the web application domain. For example:

- The first filter is the parser of incoming HTTP-request. The result of it is a set of named values.
- The next filter is syntax check that incoming parameter ‘price’ is an integer.
- The next filter is semantic check that the incoming prices contain positive values.
- Finally, custom filter performing useful processing receives only abstract data ‘price’ representing certain entity in web application domain.
2.3.1 Basic workflow stages

The underlying basis for every network service is the Communication subsystem. This subsystem implements transport protocols, accepts and serves incoming connections. The next step in processing the connection is to pre-process any incoming data according to syntax checking rules. The following activities are performed during this step:

- Canonicalization of input data into certain format;
- Parse and validation of HTTP request according to RFC;
- Forgery of request object and its environment (i.e. connection information, session data, etc.) according to the rules of the certain API (e.g. PHP SAPI, CGI, FastCGI, ASP.NET);
- Application of syntax policy (check and/or enforce syntax) to input data.

The goal of the step is to insure that particular input parameters comply with the certain syntax rules implied by the APIs used in web application. The examples of these rules are: “input parameter A should not contain SQL special characters” or “input parameter B should be valid email address” or “input parameter C should be valid integer representation”, etc. No semantic checks occur at this stage: the syntax checking routines do not care whether parameter A represents a price, or an age, or even an Id of particular internal object.

The next step is to authenticate the HTTP request: that is to ensure that the request was indeed submitted by the user, whose credentials it contains. Authentication is performed by the special authentication protocols, the most popular of which are: HTTP basic, HTTP digest and Cookie authentication. It’s important to note, that this step is optional: there are lots of public services that do not require knowing user identities. These services treat any user as general “unknown” person.

The next step after authentication is authorization. Authorization is the mechanism by which a system determines what level of access a particular authenticated user should have to secure resources controlled by the system. Authorization systems provide answers to the questions:

- Is user X authorized to access resource R?
- Is user X authorized to perform operation P?
- Is user X authorized to perform operation P on resource R?

Since every part of the HTTP request is controlled by the client, the following activities are performed during this step:

- Detection of every input parameter’s origin: whether it was set by user or by application (and was not changed);
- Application of access control rules, that is, the answer to the stated above questions.

It’s important to note, that this step is paired with the previous one. The both steps could be omitted, but when any of them is present, the lack of the other is a severe vulnerability.

The next step is to perform useful processing of input data according to the client’s expectations. The input data already has correct syntax and the access to the requested operation is granted by authorization step. The following activities are performed during this step:

- Application of semantic checking policy to input data. Examples of semantic checks are: “ensure that square root function argument is non-negative value”, “ensure that minimum funds withdraw is 5$”, “ensure that message post does not contain foul words”, etc. These checks heavily rely on the intended data domain for business operations.
- Forgery of object references, queries to external services. Object reference is a link in the notation of particular API, which is used to access data in certain environment. The examples of references are filenames, XPath expressions, URIs, etc. Queries are commands to external services such as DBMS, LDAP, OS command interpreter, and so on.
- Sequencing of data manipulation routines according to algorithm implementing use cases;
- Output generation.
The goal of this step is the same as of the whole web application: to perform certain useful data processing.

The final step in the request handling life cycle is Post-processing. The goal of this step is to apply custom transformation policies to the generated HTTP response. The examples of the policies are:

- **URL rewriting.** This processing occurs when the authentication protocol is implemented via cookieless session identifiers.
- **Comments stripping.** Comments sometimes contain sensitive data that could be exploited by malicious users. A good idea is to strip all comments from the resulting HTML pages.
- **Server header replacement.** The easiest way to fingerprint web server is to check the Server header of HTTP response. A good idea to mess things up is to replace real Server header value with different version.
- **JavaScript obfuscation.** JavaScript obfuscation is a good option to conceal logic of client side processing, which is practical in AJAX enabled applications.
- **Error suppression (custom error page presentation).**

### 2.3.2 Examples of common and custom filters

After defining the stages comprising basic workflow of web application it is time to give examples to some common and custom filters.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Filter/Type</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Receive request</td>
<td>Network packets</td>
<td>Byte stream</td>
<td>The class of filters that implement HTTP/HTTPS/SHTTPS protocols</td>
</tr>
<tr>
<td>Communication</td>
<td>Send response</td>
<td>HTTP-response</td>
<td>Network packets</td>
<td>The class of filters that implement HTTP/HTTPS/SHTTPS protocols</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>Canonicalize</td>
<td>String Grammar</td>
<td>String</td>
<td>The class of filters that perform canonicalization of data according to certain rules: URL, file system names, HTTP structure, etc.</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>Ensure-syntax</td>
<td>String Grammar</td>
<td>True/False</td>
<td>The class of filters that check whether input conforms to certain syntax (i.e. the grammar accepts the input). Examples: valid email addresses, valid integer/float number, etc.</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>Enforce-syntax</td>
<td>String Substitution rules</td>
<td>String</td>
<td>The class of filters that enforce syntax according to the specified rules. Examples: escaping, encoding, replacing, etc.</td>
</tr>
<tr>
<td>Authentication</td>
<td>Authenticate</td>
<td>Credentials</td>
<td>User identity</td>
<td>The class of filters that implement authentication protocols. Examples: HTTP basic, form authentication, request authentication using cookies.</td>
</tr>
<tr>
<td>Authorization</td>
<td>Can access</td>
<td>User identity</td>
<td>True/False</td>
<td>The class of filters that implement authorization models. Can be both common (implement standard RBAC) and custom, based on the rules of web application domain.</td>
</tr>
<tr>
<td>Main</td>
<td>Enforce-domain</td>
<td>Parameter</td>
<td>Parameter</td>
<td>The class of filters that implement certain rules that enforces domain of the parameters used in useful processing. Examples: check that</td>
</tr>
<tr>
<td>Filter</td>
<td>Type</td>
<td>Def (filter)</td>
<td>Image(filter)</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------</td>
<td>---------------------------------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>Calculate integral</td>
<td>Custom</td>
<td>Integrable function ×</td>
<td>Real numbers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Finite interval</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Square root</td>
<td>Common/Custom</td>
<td>Nonnegative numbers</td>
<td>Nonnegative numbers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set of users ×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Examples of common and custom filters in web application.

### 2.3.3 Business logic

As was already mentioned in the beginning of the section, state transition system models the behavior of web application and consists of state transition rules, common and custom filters. It defines concrete rules for data manipulation and presentation.

To be more specific, the filter is defined as a map from certain input domain to certain output domain:

\[
\text{Filter: Input } \rightarrow \text{ Output}
\]

Every filter has its definitional domain and images. These will be referred to as Def(filter) and Image(filter) respectively. The examples of definitional domains and images for different filters are given in Table 2.
Can access | Common/Custom | Set of access types × Set of resources × Set of access matrices | {0, 1}
---|---|---|---
Forge SQL query | Common | SQL grammar with parameterized terminals × parametrized terminal values | SQL language (in formal grammar sense)
Ensure valid age | Custom | Integers | {0, 1}
Enforce-syntax “no javascript” | Common | String × \{“>” → “&gt;”, “…”, “”→ “&quot;\} | String/\{“>”, “…”, “”\}

Table 2. Examples of definitional domains and images for different filters.

Web application workflow consists of the sequence of data transformations (filters). We need to identify the part of the workflow, which comprises business logic. The criterion for this separation is as follows:

- Business logic comprises of the workflow between the first and the last custom filters. This part of workflow will be called henceforth a use case.

Web application business logic consists of these use cases. This criterion is consistent with intuitive definition of use cases and business logic: for example when a user requests a picture (e.g. footer background image) the request is served by common filters only and according to our expectations it is not business logic.

This criterion for business logic separation is formalized as follows:

- Let web application consist of the set of workflows \{workflow_1, ..., workflow_N\};
- Let workflow_k consist of the sequence of filters \{filter^k_1, ..., filter^k_M\};
- The use case for workflow_k is the subsequence \{filter^k_1, ..., filter^k_{i+p}\}:
  - i+p < M(k);
  - filter^k_1 and filter^k_{i+p} are custom;
  - \(\forall j: j \in [1, i) \cup (i+p, M(k)] \Rightarrow \) filter^k_i is common.
- This use case performs data transformation Def(filter^k_i) → Image(filter^k_{i+p}):
  - It is result of transitively applied mappings;
  - The input and output sets are domain specific, hence it does not conflict with informal business logic definition.
- Business logic is a set of domain specific data transformations performed in its use cases: \{Def(filter^k_i) → Image(filter^k_{i+p}), 1 ≤ k ≤ N\}.

The next section discusses business logic vulnerabilities.

### 2.3.4 Business logic vulnerabilities

Basically, software is based on the certain requirements, which specify boundary conditions for input data, expected behavior and output for valid data, rules for handling error conditions, etc. Let us call the notion of how web application should perform ‘an intended web application’. The implementation will be called ‘an observed web application’.

Software verification provides means how to formally specify software requirements, how to express data processing algorithms and prove their compliance with the initial requirements. Furthermore, there are means to check whether initial requirements are held in the implemented software (see model checking for example). However, web applications are rarely implemented in this way (lack of skills, lack of time, etc.). In this case the observed web application cannot be compared with the intended one.
– the intended web application exists only in minds of developers, customers and those are even not equal.

In the scope of the aforesaid there is no way to get the following properties from a custom filter:

- Intended definitional domain;
- Intended data transformation algorithm.

Note, those properties are known for common filters.

According to the used model, there are only three possibilities for errors:

1. An observed state transition system has unintended transition (i.e. a backdoor);
2. An observed state transition system lacks a transition (there is a filter, which definitional domain is not enforced);
3. An observed filter performs wrong data transformation.

With the lack of formal specification the only objective knowledge for automated analysis is the definitional domains and images for common filters. In this scope, there are two possible kinds of checks:

1. Check that definitional domain is enforced;
2. Check that the filter performs correct transformation for every element from the definitional domain.

This checks based on the noted knowledge are implemented in current vulnerability analysis approaches: taint analysis and string analysis. Indeed, taint analysis approach checks for the enforcement of definitional domains for some common filters (i.e. check for existence of certain filters (sanitization) before other filters (sinks)). String analysis checks that string transformations along execution hold certain properties.

Of course, current approaches could be extended to detect unenforced definitional domains for custom filters: the operator would have to provide a formal specification of these domains and rules how to identify corresponding custom filters. The same with the second kind of check: there is need to specify intended definitional domain the intended image.

The goal of the AcCoRuTe project is to detect access control vulnerabilities. With the assumption that web applications implement RBAC access control model, we can analyze the common filter “Can access” defined in Table 1:

- definitional domain is \(\text{users} \times \text{resources} \times \text{access} \times \text{access matrix}\);
- image is \(\{0, 1\}\).

It is not difficult to get the set of web application users. The resources are obtained through web application navigation. Access denotes HTTP methods. The problem is that the intended access control matrix is unknown. Web application provides us only with the observed access control matrix – it is reflected in the HTML user interface. The observed access control matrix is union of resources accessible by web application users at current time. At this point we make another assumption: the observed matrix is the intended one. Now, if a user does not have right to access certain resource according to access matrix but was able to do so – the filter has an error.

More details on this method are provided in the document called “Testing Access Control Policy” document.

### 3 References


