A Method for Collecting Security-Specific Architectural Information for Microservice-Based Systems for Design Security Assessment

Alexander V. Barabanov

Abstract

Objective. The microservice architecture is being increasingly used for designing and implementing application systems in both cloud-based and on premise infrastructures. There are many security challenges need to be addressed in the application design and implementation phases. In order to address some security challenges it is necessity to collect security-specific information on application architecture. The goal of this article is to provide a concrete proposal of approach on how to collect microservice-based architecture information to securing application that can be applicable in immature processes and agile development.

Method. In this paper, we conduct a systematic review of major electronic databases and libraries and analysis of several practical use cases related with security architecture reviews.

Results and practical relevance. In this work based on research papers and several practical use cases analysis, we presented method for collecting architecture security-specific information for microservice-based applications and recommendations for applications security architect on how to use collected information to provide application verification against OWASP ASVS standard.

Keywords

Microservices, microservice architectures, security, architecture artifacts, architecture documentation

1. Introduction

Imagine that you are an application security engineer that was recently hired by a small young company or startup without any matured development processes. The application that company developed is microservice-based application system and you are the only one person responsible for application security. You probably use GOST R 56939, OWASP SAMM or other secure development lifecycle framework [1, 2] in order to establish application security processes. You provided self-assessment, created a roadmap and even started to implement some application security activities like security testing or static code analysis. And at one day you understand that you have to implement some activities from “Security Design” portion, like “Threat assessment” or “Security architecture”. But in order to do that you need some input information about architecture of the application you want to secure like low-level design or data flow diagram. During application development based on microservices architecture security architects/engineers usually face with the questions related with attack surface analysis, data leakage analysis and application components business/security functions verification [3, 4] to build secure application and minimize number of vulnerabilities and threats [5, 6, 7]. But if company is young firm without any matured development processes there are probably no such artifacts at all or several artifacts are in place but they are not suitable for such application security activities. Moreover, usually in agile development practices source code and presentation slides are the only artifacts available for application security architect. On the other hand, microservice-based system tends to change their architecture approximately every sprint that is every sprint security engineer faces
with a new microservices or a new storage or a new connection between microservices. Thereby to have an up-to-date security documentation for microservice system is a challengeable task because you need somehow to collect that information, update it probably every sprint (and that operation should not take too much time) and, more importantly, use collected information on a daily basis to make your application secure [8, 9, 10, 11].

The goal of this article is to offer some concrete proposal of approach on how to collect microservice-based architecture information and collected information to secure application. In summary, this paper makes the following contributions:

- method for collecting architecture security-specific information for microservice-based applications (Section 2);
- recommendations for applications security architect on how to use collected information to provide application verification against OWASP ASVS (Section 3).

2. Method for collecting architecture security-specific information

Microservice architectures has emerged as a new architectural style allowing building application systems by composing lightweight services that perform very cohesive business functions [12]. Security threats and countermeasures for microservice-based system is a very important theme nowadays. Recent studies [13, 14] shows that unauthorized access, sensitive data exposure and compromising individual microservices are the most treated and addressed threats by contemporary studies and auditing, enforcing access control, and prevention based solutions are the most proposed security mechanisms for microservice-based systems.

This chapter contains information about proposed method on how collect architecture security-specific information for microservice based-system and prepare graphical representation of modeling application system.

2.1. Collect information on the building blocks

Step 1. First steps to identify and describe application-functionality services that are the services that implement business-related functions like storing customer details, storing and displaying product catalog. It is advisable to collect the following information related to each microservices:

- unique service name or ID;
- short (one or two sentences) description of business process or functionality implemented by the microservice;
- link to source code repository;
- development team which develops the microservice because it is usual to have several development teams are working on product;
- API definition (e.g., OpenAPI specification) that describes interfaces exposed by microservice.

It is also advisable to collect some additional information like link to the microservice runbook or microservice internal architecture description, it is not necessary but it may help you. You should not collect too much information, e.g. information about 3rd-party components or libraries used in microservice can be extracted from source code repository via analyzing pom-files or similar artifacts. For application example provide on the Fig.1 during step 1 we capture information about following services: “User info”, “Booking”, “Frontend app” and “Admin”.
Step 2. Second step is to identify and describe infrastructure services. Infrastructure service is
general-purpose service including remote services that does not implement any business-related
functionality. Examples are: service registration and discovery, API gateway, security token service or
OAuth authorization service or logging service. It is advisable to collect the following information
related to each infrastructure service:

- unique service name or ID;
- short description of functionality implemented by the service (e.g., authentication,
  authorization, service registration and discovery, logging, security monitoring, API gateway);
- link to source code repository;
- link to the service documentation that includes service API definition, operational
guidance/runbook, etc.

To collect this information you can investigate project repositories and collaborate with System
architect/Development Lead. For application example provide on the Fig.1 during step 2 we capture
information about following services: “Security Token Service”, “API Gateway” and “Service
Registry”.

Step 3. The next step is to identify data storages. It could be database management systems or caches.
You should collect a following information’s related to each storage:

- unique storage name or ID;
- software that implements the data storage (e.g., “PostgreSQL”, “Redis”, “Apache Cassandra”).

For application example provide on the Fig.1 during step 3 we capture information about
PostgreSQL database instance (“PG”).

Step 4: Identify and describe message queues
Because event-driven architecture is widely adopted pattern there should be messaging systems in
application architecture like Apache Kafka or RabbitMQ. So, step number four is to identify those
messaging systems and collect information on:

- unique message queue name or ID;
- software type, i.e. software that implements the message queue (e.g., RabbitMQ, Apache
  Kafka).

For application example provide on the Fig.1 during step 3 we capture information about “Apache
Kafka” instance.

Step 5. And the final step dealing with collecting information on the building blocks is to identify
data assets. That step is more intelligent that other because you cannot just inspect you repositories or
execute a command to list data assets. To identify data assets you should actively collaborate with other
member of you team like system analyst, product owner or business architect. Obviously, you could
not identify all data assets at once – so it is advisable firstly to identify assets, which are valuable from
a security perspective (e.g., “User information”, “Payment”). Collect information on the parameters
listed below related to each asset
2.2. **Collect information on relations between building blocks**

Next steps after identification of application building blocks is to collect information on relations between those building blocks. The typical relation types are:

- “service-to-storage” relations;
- “service-to-service” synchronous communications;
- “service-to-service” asynchronous communications;
- “asset-to-storage” relations.

**Step 6.** Identify “service-to-storage” relations. Collect information on the parameters listed below related to each “service-to-storage” relation:

- service name (ID);
- storage name (ID);
- access type, i.e. specify access type, e.g. “Read” or “Read/Write”.

For application example provide on the Fig.1 during step 6 we may e.g., capture information about “User info” to “PG” relation.

**Step 7.** Identify “service-to-service” synchronous communications. Collect information on the parameters listed below related to each “service-to-service” synchronous communication:

- caller service name (ID);
- called service name (ID);
- protocol/framework used, i.e. specify protocol/framework used for communication, e.g. HTTP (REST, SOAP), Apache Thrift, gRPC;
- shortly describe the purpose of communication (requests for query of information or request/commands for a state-changing business function) and data passed between services (if possible, in terms of assets defined above).

For application example provide on the Fig.1 during step 7 we may capture information about “Admin” and “Security Token Service” synchronous communications.

**Step 8.** Identify “service-to-service” asynchronous communications. Collect information on the parameters listed below related to each “service-to-service” asynchronous communication.

- publisher service name (ID);
- subscriber service name (ID);
- message queue (ID);
- shortly describe the purpose of communication (receiving of information or commands for a state-changing business function) and data passed between services (if possible, in terms of assets defined above).

For application example provide on the Fig.1 during step 3 we may capture information about “Booking” and “Admin” asynchronous communications.

**Step 9.** Identify “asset-to-storage” relations. Collect information on the parameters listed below related to each “asset-to-storage” relation:

- Asset name (ID);
- Storage name (ID);
- Specify storage type for the asset, e.g. “golden source” or “cache”.

2.3. **Create a graphical presentation of application architecture**

It is advisable to follow “architecture-as-a-code” [10] practice and create graphical presentation of application architecture (building blocks and relations defined above) in form of services call graph or data flow diagram. In order to do that one can use special software tools (e.g. Enterprise Architect) or DOT language. An example of using DOT language to describe a simple microservice-based application architecture is the following:
digraph architecture {
  rankdir=LR;

  subgraph client_side_app {
    front_end -> {API_GW} [label = "HTTPS"]
  }

  subgraph api_gateways {
    API_GW -> {AuthN, ms_1, ms_2, ms_3} [label = "HTTP"]
  }

  subgraph microservices {
    ms_1 -> {DB} [label="JDBC"]
    ms_2 -> {Queue} [label="gRPC"]
    ms_3 -> {Queue} [label="gRPC"]
  }
}

That code can be transformed to the following graphical presentation (Figure 2).

![Figure 2: Example of using DOT language to describe microservice-based system architecture](image)

3. Using collected information in secure software development practices

Collected information may be useful for doing application security practices, e.g. during defining security requirements, threat modeling or security testing [15]. This section contains examples of activities related to securing application architecture as well as its mapping to OWASP ASVS [16] requirements and tips for their implementation using information collected above. This implementation tips and recommendations were extracted and collected during multiple security architecture reviews.
<table>
<thead>
<tr>
<th>OWASP ASVS ID</th>
<th>OWASP ASVS Description</th>
<th>Implementation tips</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.2</td>
<td>Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.</td>
<td>Collected information can be used for threat modeling purpose. Please see example listed after the tables.</td>
</tr>
<tr>
<td>1.1.4</td>
<td>Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.</td>
<td>To verify documentation and justification of all the application's trust boundaries, components, and significant data flows analyze data collected during following steps:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 1 “Identify and describe application-functionality services”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 2 “Identify and describe infrastructure services”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 3 “Identify and describe data storages”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 4 “Identify and describe message queues”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 6 “Identify ‘service-to-storage’ relations”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 7 “Identify ‘service-to-service’ synchronous communications”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 8 “Identify ‘service-to-service’ asynchronous communications”.</td>
</tr>
<tr>
<td>1.1.5</td>
<td>Verify definition and security analysis of the application's high-level architecture and all connected remote services.</td>
<td>To verify that analyze data on application architecture collected during following steps:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 1 “Identify and describe application-functionality services”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 2 “Identify and describe infrastructure services”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 7 “Identify ‘service-to-service’ synchronous communications”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 8 “Identify ‘service-to-service’ asynchronous communications”.</td>
</tr>
<tr>
<td>1.1.6</td>
<td>Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls</td>
<td>To verify that analyze data on application architecture collected during following step in order to derive what component provides authentication, authorization and logging:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Step 2 “Identify and describe infrastructure services”.</td>
</tr>
</tbody>
</table>
### Table 2
Collected data usage for OWASP ASVS “V1.2 Authentication Architecture”

<table>
<thead>
<tr>
<th>OWASP ASVS ID</th>
<th>OWASP ASVS Description</th>
<th>Implementation tips</th>
</tr>
</thead>
</table>
| 1.2.2         | Verify that communications between application components, including APIs, middleware and data layers, are authenticated and use individual user accounts | To enumerate microservices endpoints that need to be tested during security testing and analyzed during threat modeling analyze data collected under the following sections:  
- Step 1 “Identify and describe application-functionality services” (parameter "API definition");  
- Step 2 “Identify and describe infrastructure services” (parameter "Link to the service documentation") |
| 1.2.3         | Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches | To verify that analyze data on application architecture collected during following step in order to derive what component provides authentication:  
- Step 2 “Identify and describe infrastructure services” |
| 1.2.4         | Verify that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application | To verify that analyze data on application architecture collected during following step in order to derive what component provides authentication:  
- Step 2 “Identify and describe infrastructure services” |

### Table 3
Collected data usage for OWASP ASVS “V1.4 Access Control Architecture”

<table>
<thead>
<tr>
<th>OWASP ASVS ID</th>
<th>OWASP ASVS Description</th>
<th>Implementation tips</th>
</tr>
</thead>
</table>
| 1.4.1         | Verify that trusted enforcement points, such as access control gateways, servers, and serverless functions, enforce access controls. Never enforce access controls on the client. | To verify that analyze data on application architecture collected during following step in order to derive what component provides authorization:  
- Step 1 “Identify and describe application-functionality services”  
- Step 2 “Identify and describe infrastructure services”. |
| 1.4.4         | Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths | To verify that analyze data on application architecture collected during following step in order to derive what component provides authorization:  
- Step 2 “Identify and describe infrastructure services”. |
### Table 4
**Collected data usage for OWASP ASVS “V1.7 Errors, Logging and Auditing Architecture”**

<table>
<thead>
<tr>
<th>OWASP ASVS ID</th>
<th>OWASP ASVS Description</th>
<th>Implementation tips</th>
</tr>
</thead>
</table>
| 1.7.2         | Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation | To verify that analyze data on application architecture collected during following steps in order to understand how logging implemented:  
- Step 1 “Identify and describe application-functionality services”;  
- Step 2 “Identify and describe infrastructure services”;  
- Step 7 “Identify “service-to-service” synchronous communications”;  
- Step 8 “Identify “service-to-service” asynchronous communications”. |

### Table 5
**Collected data usage for OWASP ASVS “V1.8 Data Protection and Privacy Architecture”**

<table>
<thead>
<tr>
<th>OWASP ASVS ID</th>
<th>OWASP ASVS Description</th>
<th>Implementation tips</th>
</tr>
</thead>
</table>
| 1.8.1         | Verify that all sensitive data is identified and classified into protection levels. | To verify that analyze data on application architecture collected during following steps:  
- Step 5 “Identify data assets”  
- Step 9. Identify “asset-to-storage” relations. |

### Table 6
**Collected data usage for OWASP ASVS “V1.9 Communications Architecture”**

<table>
<thead>
<tr>
<th>OWASP ASVS ID</th>
<th>OWASP ASVS Description</th>
<th>Implementation tips</th>
</tr>
</thead>
</table>
| 1.9.1         | Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers. | To verify that analyze data on application architecture collected during following steps in order to understand how communication protection is implemented:  
- Step 1 “Identify and describe application-functionality services”;  
- Step 2 “Identify and describe infrastructure services”;  
- Step 7 “Identify “service-to-service” synchronous communications”;  
- Step 8 “Identify “service-to-service” asynchronous communications”. |

Here is an example on how to use collected information for threat modeling purpose. OWASP ASVS requires to use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing. During sprint planning an application security engineer can easily modify architecture diagram (see fig.3) or its part.
to reflect proposed architectural changes. Then, having updated diagram and description of services and data assets, you can make some threat modeling activities based on STRIDE methodology or similar, e.g.:

- Does “Frontend App” really need connection to “Security Token Service”? If yes, what minimal privileges does that service need?
- Does “Booking” service really need an access to “Admin” service? If yes, what minimal privileges does that service need?

![Diagram of microservice-based system architecture](image)

Figure 3: Example of using DOT language to describe microservice-based system architecture

### 4. Related work

Various techniques have been developed and applied to document microservice-based system architecture and use that information for security purposes but in most cases it can be applied in mature software development lifecycle processes and do not focus on application security aspects.

B. Mayer and R. Weinreich [17] presented an approach to extract and analyze the architecture of a microservice-based software system based on a combination of static service information with infrastructure-related and aggregated runtime (logged outgoing and incoming requests) information.

G. Granchelli et al. [18, 19] presented an approach for semi-automatically recovering design of microservice-based systems. Their approach is based on model driven engineering techniques usage and domain-specific language for representing the key aspects of the architecture of a microservice-based system.

S. Ma et al. [20, 21, 22] proposed an approach to the development of microservice-based systems that enables the automatic generation of a service dependency graph by which to visualize and analyze dependency relationships between microservices as well as between services and scenarios. It also enables the automatic retrieval of test cases required for system changes to reduce the time and costs associated with regression testing.

N. Riopelle et al. [23] proposed the use of dependency graph based modeling to streamline the failure analysis process for private cloud and microservice-based applications.

Y. Lan et al. [24] proposed and designed a dependency model of microservices and dependency mining method based on call chain logs to extract local dependencies and the discontinuous dependency relationship.

N. Chondamrongkul et al. [25] presented an automated security analysis approach for microservice architecture that can automatically identify security threats according to a collection of formally defined security characteristics and provide a result that demonstrates how the attack scenarios may happen.

Compared with the related works our study is more narrow and concentrated on security of microservice-based system, whereas most of the above-mentioned work considers some quality characteristics. In contrast with above-mentioned works we are not only take into account microservice-to-microservice dependency but also dependencies on general purpose services (like API Gateway and service Discovery), data storages and messages queues. Moreover, to design our graph we use object called “asset” (in contrast with [25]) that allows us to cover more security checks from OWASP ASVS.
5. Conclusion and further work

In this paper, we presented a method for collecting architecture security-specific information for microservice-based systems that can be used even in immature software development processes to secure application. Collected information may be useful for doing application security practices during defining security architecture: attack surface analysis, data leakage analysis, analysis of the application’s high-level architecture, enforcement of the principle of least privilege and sensitive data identification and classification in secure software development lifecycle. We tested our approach during development of microservice-based system in Oil & Gas automation and received a positive feedback. We also contributed that approach to OWASP Community in Cheat sheet series. Practical usage of proposed light-weight method for collecting security-specific architectural information for microservice-based systems allowed to decrease the time for collecting application design information and to focus more on practical security aspects (like threat modeling) during sprint planning.

Further research is intended for combining graph algorithms with proposed approach in order to automate steps needed to design microservice application design and make threat modeling for large-scale distributed microservice-based applications.

6. References


