TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Science

Cyber Security

Tiit Hallas

Logging Requirement Analysis and Specification for Development Based on Governmental Institutions of Estonia

Master's Thesis

Supervisors:

Mait Peekma, MSc

Toomas Lepik, MSc

TALLINN 2014

Declaration

I declare that this master's thesis is the result of my own research except as cited in the references. The thesis has not been submitted before any other degree or examination.

…….. ……

(Date) (Author's Signature)

Abstract

Logging is an important part of any information system. If done properly, logging gives vital input to several IT processes such as information security and system maintenance. Only a limited set of high-level requirements for logging are defined for the governmental institutions of Estonia. There are no consolidated requirements defined for the content of the logs.

The aim of this study is to analyse the existing requirements for logging, synthesize additional requirements to cover the areas that are not covered by the existing requirements and to explain the rationale of each requirement. As a result, a consolidated list of log content requirements is created. The thesis will bring out the most common mistakes done in logging implementations and offer solutions to eliminate them. The outcome of this thesis is a detailed requirement document for logging in English and Estonian language that can be applied as a part of non-functional requirements in the development process.

Annotatsioon

Logimine on iga infosüsteemi oluline osa. Kui logimine on korrektselt tehtud, annavad logid olulist informatsiooni mitmetele IT protsessidele alates infoturbest lõpetades süsteemihaldusega. Eesti riigiasutustele on seatud logimise osas ainult üldised nõuded ja eesmärgid. Puudub konsolideeritud dokument, mis seaks nõudeid logi sisu kvaliteedile või mida saaks kasutada juhendina logimise planeerimisel tarkvara arendusprotsessis.

Magistritöö eesmärgiks on analüüsida olemasolevaid nõudeid logimisele, sünteesida täiendavaid nõudeid, katmaks ära olemasolevate nõuete poolt katmata logimise funktsionaalsust ning selgitada analüüsi käigus iga uue nõude otstarvet. Magistritöö toob välja põhilised vead, mida logimise puhul tehakse ning pakub välja toodud vigadele lahendusi. Magistritöö tulemuseks on eraldiseisev eesti- ja inglisekeelne detailne loginõuete dokument, mida on võimalik kasutada arendusprotsessis mitte-funktsionaalsete nõuete osana.

Index

1 Introduction

Logging is an important part of the information system. If done properly, logging gives vital information about the user activities and the information system to the people dealing with the information system. Logs give valuable input for several IT processes from information security to system maintenance and development. The quality of such input depends on the quality of the log.

Currently there are no consolidated requirements for the content of the log that can be used as guidelines during the development process. Although ISKE[10] has set some high-level requirements and goals for logging, there are no detailed requirements or checklists to follow when planning the logging in the design phase of a new service or application. ISKE states that these requirements should be an outcome of collaborative work of the Security Manager and Head of IT department. Since there are no guidelines to take as a reference, important details might not be logged. This might lead to a situation where there is not enough evidence in case of a security incident or might not give as much information about client behavior as it could.

The problem has been recognized by Renee Trisberg - one of the most experienced information security experts in Estonia, who has dealt with log analysis for over 16 years; Andres Kütt – adviser in RIA[1], who is responsible for Estonia's IT architecture and strategic planning and Mehis Hakkaja, owner of penetration testing company Clarified Security OÜ that also tests and analyses the weaknesses of logging.

Log lifecycle can be divided into the following steps:

1. Policy definition and requirements;
2. Logging configuration and content;
3. Collection of logs;
4. Log event normalization;

[1] RIA – Riigi Infosüsteemide Amet (Information Security Authority in Estonia)

5. Log event indexing;
6. Storing log events;
7. Log correlation;
8. Baselining of events;
9. Alerting;
10. Reporting.

There have been publications that cover Log management aspects (points 3-6)[22] and Log analysis (points 7-9)[31] aspects. By authors knowledge no thorough study has been done on the log requirements and logging content.

This thesis will cover the first two steps of the log lifecycle, giving additional input for the requirements and defining the actual data that must be logged in such cases. The thesis will focus on the log content requirements for a log event and will explain during an analysis where these requirements come from. The thesis will bring out the most common mistakes done in logging implementations and offer solutions to eliminate them.

Author of this thesis has been dealing with log creation, correlation and analysis for the past 14 years from development, system maintenance and information security perspective. The need to gather data and investigate IT- and fraud-related incidents has given the author a thorough understanding about the information that should be gathered about the events. Author has participated in several international real-time cyber defence exercises as a part of an attacking team and practiced evasion techniques in simulated real world situations, which has given the author an understanding about how an attacker might think in such situations.

Although the examples brought in thesis are based on a web server, the solutions can be applied on any client-server application. Logs in the client device are under clients control and cannot be trusted thus thesis concentrates on the server side logging. Thesis covers the logging requirements for middleware (application) layer of a service, which is influenced the most from logging

perspective by the development of the system. Other layers (front-end, database and network) require different approach and are not solved during a development process, thus are not a subject for this thesis.

The outcome of this thesis is a separate detailed requirement document for logging in English and Estonian language (Appendixes 3 and 4 respectively) that can be applied as a part of non-functional requirements in the development process. By authors' knowledge no such document has previously existed. Although the study has been done based on a governmental institution, the document can be applied in both public and private sector development processes.

The target audience of this thesis is a person who has knowledge about information system design and basic understanding about client-server system architecture, web services and session handling.

2 Background

2.1 Scope of the thesis

Since the output of this thesis is a set of requirements that can be set for the application developers during a new application or service development, thesis concentrates on the issues that can be handled within the development process. The thesis focuses on application level, because it usually contains the majority of the business logic implementation of the system. Logging that is related to existing infrastructure (e.g. database logging, networking and server component logging) and must be handled with different approach is not covered within this thesis. Since the client can easily manipulate logs that originate from client devices, they cannot be considered trustworthy and must be handled in a different manner and are out of scope.

Although most of the given examples are based on a web service and HTTP requests, the requirements can be applied to any client-server application. Based on the Authors' opinion, web-based applications are the most popular types of applications being developed today and HTTP protocol is the most common protocol used in client-server communication, thus there is a great probability, that the reader is already familiar with the protocol. Thus it is easier for the reader to follow the examples without having to get familiar with another client-server protocol.

There are several other aspects that are related to logging and are mandatory when providing integrity to the log events, but these aspects create much more additional attack vectors and problems that need to be addressed and are left out of this thesis' scope. These aspects that are not covered include (but are not limited to):

- Log centralization;
- Log event shipping tools and protocols;
- SOA-type logging and alternatives to file-based logging;
- Log signing and crypto chaining.

2.2 Terms and roles

The roles that are used in this thesis are:

- Security Specialist – Role in the company that is responsible for preventing, detecting and protecting the company against the security incidents, misuse of data, data leakages and attacks against the infrastructure;
- Session ID – Unique session identificator that ise used to identify the message sender in case of stateless protocol like HTTP;
- Event ID – Unique event identificator;
- Alice, Bob – Two common application users that use the application or exchange information with each other;
- Eve – Malicious user that tries to evesdrop a sent message or tries to attack the system.

2.3 History

The logging history is tightly connected to the naval navigation and logbooks that were kept on vessels in order to manage, operate and navigate them. It was used to measure the distance ship had travelled within a certain amount of time through recording the readings of the speed log[24] – a wooden board attached to a string called a "log line". Later on, captains often marked down all other events that were occurring on the ship: ports the ship had visited, crew behavior, both routine events and significant incidents, the weather conditions and other ships they had encountered during the voyage. Aside from the navigational part, it gave a good overview about what went on during the voyage. Todays "black box"[6] on aircrafts has the same purpose.

At the late 1830's, Matthew Fontaine Maury's[28] started to bring out the additional value from the collected and archived log files: he was the first person to analyze and correlate thousands of old ship logs and charts in order to collect information about the winds, calms and currents for all seas in all seasons and mapped the migration of whales through the information stored in the logs. He was able to prove his theory, that an area in the ocean near the North Pole is

occasionally free of ice – whales, being mammals, had to come on surface and breathe. Through his findings from the logs he published his "Wind and Current Chart of the North Atlantic"[2], which showed how to use the ocean's currents and winds to drastically reduce the length of ocean voyages.

The work of Maury can be considered as a first occurrence of an actual log analysis and correlation to gain benefit from historical data and also – he can be considered as an open-data project founder at that time.

2.4 Logging modes

The usual situation in today's information systems is similar to the situation as described before about the ship logs. Some logs are collected and saved but only limited amount of beneficial information is gained from them. When looking into the custom-made applications, there usually are logs present, but the log quality is too low to provide any usable information. Developers do not have usable requirements related to logging set for them, thus they usually log only debug information to discover development issues. System administrators are usually interested in performance-related information of the system and metrics about the system behavior.

As described in chapter 3, logs can be used for much more if they contain relevant data and there is an interested party to correlate and analyze the information in there. Unfortunately no one has put any effort or invested energy into creating better requirements for logging, thus - actual benefit is not gained from the logging. At best, there are some best-effort logging done that will be described further in this chapter.

Based on the authors' knowledge, there are four different modes of application logging that can usually be found depending on the system and other non-functional requirements for the system.

2.4.1 Default logging

Everything that is set to log by the default configuration of the application, service, server or the operating system, will be logged. No additional logging is

being implemented, i.e. if a component, that logs by default (e.g. Apache web server), is included into the system, then system would be considered as a system that logs, despite of the fact that the main functionality lacks logging.

2.4.2 Development-driven logging

It is not always possible to debug everything in production environment. Also – it is sometimes impossible to repeat the error in development environment, so developers have created additional logging capabilities for debugging purpose: In order to debug errors that have occurred in the production environment in an efficient way, the following (but not limited to) is usually written down:

- Application states;
- Stack traces;
- Memory dumps;
- Database queries (and responses);
- Other information for debugging.

2.4.3 Maintenance-driven logging

In these cases there have been some requirements written into the development plan by the system administrators. This usually occurs in case of a larger-scale company, where development and maintenance roles are separated. The system administrators have to analyze the performance of the application, so some metrics have to be reported in the logs. Information that is usually logged is:

- Resource generating time;
- Resource execution time;
- Resource memory consumption;
- Database connection strings and parameters;
- Result: error and timeout messages;
- Other monitoring- and maintenance-related information.

2.4.4 Compliance

There are some laws, regulations and other policies that address the logging issue and pose requirements for systems to log down some events: logins, executions, information queries, retaining deadlines and other security-related information. Usually the requirements are not specific enough to give an actual

input to either the developers or the administrators about the content of the event being logged. That leads to logging “something”, which will never be used. The existing compliance requirements are covered in Chapter 4.

3 Roles and logging needs

There are roughly four roles that have interests in logging and which all have different approaches or needs from the logs: developers, system administrators, security specialists and management. Developers and system administrators were already covered in Chapters 2.4.2 and 2.4.3 respectively. Security specialists and management will be covered in chapters 3.1 and 3.2.

3.1 For security specialists

A security specialist is responsible for preventing, detecting and protecting the company against the security incidents, misuse of data, data leakages and attacks against the infrastructure. Unfortunately it is often difficult to detect a security incident from the system by its symptoms. An attack against an information system can have symptoms that would express in an unexpected way: humidity level-, temperature- or power fluctuation, higher load or power consumptions of a server or other symptom or incident that would seem unrelated to security at first sight. To prevent security incidents from left unhandled, a security specialist must handle every anomaly in the information system as a security incident or an attack until proven otherwise. A security specialist cannot rule out a possibility of a malicious user behavior behind an unexpected behavior in the system. Even if a home appliance acts unexpectedly, one cannot be sure that it is not an intended behavior caused by somebody and the appliance is under somebody else's control[13].

To be sure that the anomaly was not an attack, logs need to be analyzed to properly evaluate the situation. For that, the logs have to contain enough evidence to investigate such incidents. Also the logs have to provide information to detect possible fraudulent activities and to profile users to detect activities that are out of the ordinary. Such detections are useful to prevent data leakage and misuse of user privileges: if the logs give enough information to generate user profiles based on their activities and roles, security specialist can start looking for behavioral anomalies, e.g. if a system analyst was recently employed but the action profile matches a system administrator who has been working

with the system for the past five years, then this might be a subject for deeper analysis and background check of the analyst.

All the previous aspects are useful only if the security officer is familiar with the logs – it is important to know, what is normal in the logs and what are the anomalies. If the security specialist does not have experience with the logs from the system under investigation, then many anomalies in the log may seem related to the ongoing incident, even though the anomaly is present in the log on daily basis. I.e. security specialist should expect the known exceptions and must be able to differ them from the real anomalies that are connected to the incident at hand.

3.2 For management

Management is the main stakeholder of all logs. Management benefits from all the activities mentioned in the previous chapters and should be interested in the productivity of all the respective counterparts: developing high-quality systems, quick and efficient bug fixing, monitoring the infrastructure and detecting anomalies. However, there is another view that should not be neglected: direct business benefit.

Logs can be used to analyze customer behavior in the application. Based on client usage, management can make decisions on how to improve the applications usability: if all customers are frequently using one component of the application but have to make an extra effort to use it (e.g. have to make two mouse clicks to get there from the main page), then management can make an executive decision and move the component that is used more frequently to the main page to optimize the application usage and clients user experience.

Through such application usage analysis, management can make decisions either to invest more to the application and plan additional resources for the application or close it down due to lack of usage. These decisions need a solid and trustworthy input, which could be extracted from the logs.

4 Existing requirements

There are application security requirements in the public sector that have to be followed. These requirements originate from the legislation (laws and regulations published by the government) and from the official Estonian Three-level IT Baseline Security System called ISKE[10].

4.1 Requirements from legislation

There are some laws and regulations that can be found from the official website "Riigi Teataja"[20], which publishes all legislation-related documents in Estonia. The laws that give most direct input to the log quality (or what could be considered as requirements for logging) originate from *Personal Data Protection Act*[18], *Requirements for PC and LAN protection*[15] that is an addition to the *State Secret and Foreign Classified Information Law*[19], *Data Exchange Layer of the Information Systems*[16] and *Regulation for saving, passing on and destroying data, inquiries, log files and applications*[14].

The most detailed requirements are present in the *Personal Data Protection Act*, which describes in §25 (*Organisational, physical and information technology security measures for protection of personal data*) that the processor of personal data has to prevent unauthorized saving, modifying and deletion of personal data and guarantee that it is possible to prove afterwards, when, by whom and which data was modified and which personal data was accessed. The same paragraph also describes that same kind of information should be saved when forwarding personal data.

Other laws and regulations do not provide instructions which data should be written in the log, but give guidelines or high-level ideas about what kind of problems the logs should solve. The fore mentioned *Requirements for PC and LAN Protection* give requirements for log information preserving period, necessary information for analyzing the system usage (e.g. execution and closing, user access, changes in user privileges, changes in log settings, date and time settings and unsuccessful login attempts), accessing restricted data and details about the

event like date, type and notification about success or failure of the event. Other two laws state, that logging should be applied and logs should be kept, but no actual requirements for the content is mentioned.

4.2 Estonian Three-level IT Baseline Security System – ISKE

The *Estonian Three-level Baseline System (ISKE)*[10] is based on the German information security standard - *IT Baseline Protection Manual* (*BSI: IT-Grundschutz in German*[3]) and is meant to ensure the sufficient security level for the data processed in IT systems. Implementing the organizational, infrastructural, physical and technical security measures from the standard achieve the necessary security level.

It is an information security standard that is developed for the Estonian public sector. According to Government Regulation no. 273 of 12 August 2004[17], ISKE is compulsory for state and local government organizations, which handle databases or registers. The first version of the ISKE implementation manual was completed by October 2003.

ISKE looks at the three aspects of information security: Confidentiality, Integrity and Availability[30]. Of these three, most relevant to logging is Integrity. ISKE has four different integrity levels that are mapped to the application, based on the nature of the information that is handled in the system. The levels as follows[9; page 18]:

- **T0** – the possibility to detect the source of creation, changing or destroying the information is not relevant; no controls for checking the accuracy, integrity or opportune of the data is not needed;
- **T1** – the possibility to detect the source of creation, changing or destroying the information must be established; controls for checking the accuracy, integrity or opportune are done on special occasions and based on necessity;
- **T2** – the possibility to detect the source of creation, changing or destroying the information must be established; controls for checking the accuracy, integrity or opportune are needed to be done regularly;

- **T3** – the fact of information creation, changes and deletion must have legal proof; controls for checking the accuracy, integrity or opportune are done in real time;

ISKE provides methods[8] that are mandatory in order to achieve the respective integrity level. The method B 5.22 (*Logging*)[8; page 383] has some baseline suggestions regarding logging. A log should contain:

- Information regarding the event time (when the event happened);
- End result of the event (what did the event cause);
- Which instrument was used for the event;
- State of the system, e.g. who had access rights and for which timeframe were they valid.

The method also states that all such events should be logged which has relevance to information security, but it does not give any additional suggestions regarding what events should they be and which information should be logged.

There are also other methods, which cover logging in one-way or another: M 4.172 (*Logging access to archives*) states, which user activities should be logged (e.g. loosing confidentiality or integrity of the data due to user error, wrong user privileges, shutting down the server during data analysis, defective removable media, manipulating the system components etc.).

M 4.431 (*Choosing important data for logging and processing*) states, that aside from all other things that need to be logged, one should not forget to log events that include inserting a wrong password, blocking an account, unauthorized access, excessive network loads and alerts from IDS systems.

One of the most comprehensive methods from the log content perspective is M 4.47 (*Logging firewall operations*), which defines specific values that need to be logged in different cases. It defines log requirements for all packets that are logged and defines specific requirements for five protocols: DNS, FTP, HTTP NNTP and SMTP. For example: the method states, that for every event log file should contain source IP, destination IP, source and destination port (or ICMP

type), date and time and a rule that it matched in the packet filter. If a service is known, then service and duration of the connection should be also marked down. In case of HTTP, log file should additionally contain information about the URL, transferred data amount, request method (GET, POST, CONNECT), filters that applied and status.

There are many methods which mention logging and pose requirements for what kind of events should be logged (e.g. M 2.110 – *Data privacy guidelines in logging procedures*, M 2.133 – *Controlling database log files*, M 4.205 - *Router logging*, M 4.292 – *IP phone logging*, M 4.302 – *Logging of printers, copiers and multifunctional devices*, M 4.81 – *Logging network activities*, M 5.9 – *Server log*, etc.). Also there about four methods (HT.14, HT.16, HT.17, HS.55) on how to crypto-chain the log events in chorographical order and timestamp it with a local signature in order to preserve the log files and protect its integrity.

Although ISKE has set some high-level requirements and goals for logging, there are no detailed requirements or checklists to follow when planning the logging while designing a new service or application. There is only one method (HG.25 – *Mandatory logging for remote working*) that states how exactly the composition and data that is written to the log file should be decided: it should be an outcome of collaborative work of the Security Manager and Head of IT department[8; page 3895]. But unfortunately it is not realistic, that anyone would analyze the whole ISKE methods catalog to get requirements for developers when creating a new system.

5 Analysis

5.1 Consolidated requirements

Although the laws, regulations and ISKE bring out the fore mentioned requirements, there is no consolidated view or list of logging requirements that could be given as an input to the developers. The exact items what must be logged, what is beneficial to log or what to consider when developing a new application cannot be found in one consolidated document. There is no baseline, example scenarios or defined requirements to help the customer to understand, what has to be logged. Result would be that developers write logs as they see fit. This might be useless for the customer at the end of the day, especially in the case where developers are not involved in the maintenance or log analysis in a later phase of the application.

To solve this problem, this thesis goes through a list of problems that usually occur in an information system and provides solutions, which will be gathered into one logical requirement document (Appendix 3). The following chapters will address different problems in an average information system from logging point of view.

5.2 Quality of logs

Based on authors' experience, when a system is set for logging, the events are usually written to the same physical file irrespectively to the event types. The events might have different log severity levels (e.g. *DEBUG*, *INFO*, *WARNING* etc.) but usually other distinction can only be found from the message content itself. This makes the log analysis difficult and does not allow the logs to be analyzed based on the event type: if a developer needs access to the debug log and security specialist need to access the user activities, then in case of one file, all parties have to use the same file, thus see information that is either unimportant or not meant for them to be seen.

To optimize the log usage and to separate different types of events from each other, events must be divided into different logs based on their characteristics (Appendix 3, p. 2). The proposed divisibility could be (but is not limited to) as follows:

a) **Session log**: all relevant information for user authentication to the application (or sub-part of the application that has a higher security), authorization, session timeouts and invalidations with respective information. An event must include at least the following information:
    a. Date and time;
    b. Instance, which served the event;
    c. Unique identifier of the event;
    d. Action that was performed (e.g. login, logout, timeout);
    e. IP address;
    f. Hash of a session ID;
    g. Method that was used (e.g. Password, ID-card, Token etc.);
    h. Result of the event (success, failure, attempt);
    i. Payload (additional relevant parameters that were sent and processed).

    Sample events from the session log are given in Appendix 2.

b) **Request log**: all requests done by the users, which includes all the required input parameters, activity types, session-related information (to correlate the event to the session log) and outcome. The event must contain at least the following information:
    a. Date and time;
    b. Instance, which served the event;
    c. Unique identifier of the event;
    d. Action or a type of request (e.g. search, request, query etc.)
    e. IP address;
    f. Username;
    g. Hash of a session ID;
    h. Result;
    i. Payload (Additional parameters that are relevant to the request).

    Sample events from the request log are given in Appendix 2.

c) **Debug log**: the debug log with detailed system information for developers in order to debug a problem in a production environment in case of an incident. By default, the debug mode must be disabled in production environment;
d) **Security log**: security-related events and problems (IP changes during a session, user privilege escalations, authentication avoidance attempts, incorrect user certificates etc. events that require security specialists attention);
e) **Error log**: both user errors and technical errors must be divided into separate logs.
    a. User errors must include errors that were invoked by the user and are related to use cases where users gets an error message, i.e. handled exceptions (tried to view a file without proper privileges, make a payment that has larger amount then your account balance etc.);
    b. Technical errors are system-related errors that were caused by other system components, the system itself or unexpected user behavior, i.e. unhandled exceptions. Message can contain technical information such as thrown exceptions, stack traces and other error-related information.

To provide better analyzability of the logs, dividing must be done by functionality, not by module or by application components. If an application has several parts or components (e.g. by Security levels), then the logging must still contain only certain types of events (Appendix 3, p. 9). I.e. all session-related events (regardless of the module) must be in the same session log.

To avoid wasteful system usage from both analyzing processing power and storage point of view, duplicating information in the logs must be avoided whenever possible (Appendix 3, p. 3). To able to tie security events and error messages with the requests done by the user, information must be linked with unique event ID's between different log files (Appendix 3, p. 4).

Logging must be considered as a part of the functionality. If logging is a mandatory feature (e.g. in case of information systems with ISKE T1 or higher), the request must not be executed if logging fails. If the integrity of the system is more important than availability, then fail-close solution is suggested: If logging fails then system must be shut down (Appendix 3, p.10).

5.3 Level of detail in logging

Appropriate amount of logging detail is a challenge for the developers. Since the requirements for the logs do not specify what kind of information and in which detail should be logged, developers have to figure without a proper set of data what to store about the event. Unfortunately it is often seen, that not all events are logged. It is usual, that only successful events are logged.

If Bob logs into a system, the time and fact that Bob has entered the system might be logged, however other relevant information (e.g. IP address, Bobs' device-related information etc.) is left unlogged and if for some reason the login fails, the failed attempt is also usually left unlogged.

From the information security perspective, most crucial information that has to be logged is related to user activities, authentication and authorization. However, to provide evidence about the activities done in the information system, all user activities have to be logged (Appendix 3, p. 11). The log files must contain enough information to make it possible for a log analyst to fully understand (and reproduce) all the users activities to get the same end result that the user did. This does not apply to only the cases where the result is a failure: even successful information viewing (or attempt) must be logged.

From the data protection point of view logs must reveal, who accessed (or tried to access) which information at what time. Search criteria and the size of the response are usually enough: the full response must not be logged. This also includes the administrative users: when an administrator changes the system settings, user privileges or does other system-related activities, there has to be log trace for administrative activities as well (Appendix 3, p.11.2).

Administrators must not be able to hide their activities nor be able to change or delete the logs from the system (Appendix 3, p.11.3). These kinds of attempts must be logged in a separate file (e.g. security logs).

One of the most crucial aspects that must be logged in detail is authentication and authorization of the user. Logs must reveal, what kind of relevant privileges the user had at the time (i.e. what was the user able to do in the system). It is important to log all the events related to authentication: successful logins, login attempts [2] and failed logins must be logged with the appropriate details (Appendix 3, p. 11.1).

In order to provide the full ability to reproduce all the users activities, the log entry has to provide at least the information, that answers to the following questions: who, what, where, whence and when, also the result of the action must be logged (Appendix 3, p.21). Based on the Authors' experience, answering the fore mentioned questions would provide viable information to conduct a security incident analysis or to monitor user activities and to prove what happened in the system during that time.

5.3.1 Who

The answer to the question '*who*' must define, which user triggered the event. The parameters that can reveal this kind of information might be (depending on the system or service):

- Username, that is unique (at least) within the service (Appendix 3, p.21.1.1) and what is linked with only one responsible person (Appendix 3, p.21.1.2);
- Hash or other one-way derivation of the *'session_ID',* within the user performed the activity;

[2] Attempt - in this context is to try to achieve something without the needed prerequisites and is bound to fail. E.g. a login without a password is an attempt not failure because it is not possible to login without a password. The difference is needed to detect anomalies in case of an incident or problems with the service.

Automatic processes must be clearly distinguished from all other activities (Appendix 3, p.21.1.3) in order to understand, which processes are invoked by the user at the exact time and which changes or activities were done automatically (usually pre-planned by administrators). All the automatic activities must have a responsible person (Appendix 3, p.21.1.4), e.g. the system's responsible administrator who created or executed the script, which resulted in the logged event.

5.3.2 What

The answer must describe the actions that the user performed, e.g. user authorization, administrative actions, data reading or -modifying events (Appendix 3, p. 21.2). These actions must include the respective parameters that manipulated the system in as much detail as possible: request parameters, search criteria and values and other relevant input data (Appendix 3, p.21.2.2) like uploaded file names (and hash values of the files) and selected objects (Appendix 3, p.21.2.3). The software component or an object that received the request must be also defined in the logged event (Appendix 3, p.21.2.1). The fore-mentioned data is important in order to understand, what happened in the information system: as much data is saved in the log files for later analysis, the better from the forensic point of view.

5.3.3 Where

The answer to the question '*where*' must contain the relevant information about the information system to be able to determine the service, application, and its instance (Appendix 3, p. 21.3). If the request or query used some vulnerability on one certain host, then this kind of information can help the administrators to pinpoint the problem and fix the vulnerability. It is also valuable information when debugging performance issues. If the event just states that the server experienced a memory corruption problem when responding to the request, then without the instance name it is hard to find the instance with the problem.

5.3.4 Whence

It is important to log the initial device, where the request originated (Appendix 3, p. 21.4). The origin can be defined with:

a. IPv4 address of the device (in case of an internal application);
b. Public IPv4 address of the network, where the device is located (in case of NAT);
c. IPv6 address of the device;
d. Hostname of the sender;
e. Information about the device certificate.

The information in the log file must provide information to determine the exact device that was used to send the request (Appendix 3, p. 21.4.1).

In some cases, services might have various proxy devices in front of the actual application. This could be either because of load balancing or caching reasons. System administrators must be certain that the log files would not contain just the proxy IP as the source address in the logs, but would have the client source information in a more precise manner. There are solutions to prevent such situations (avoid source NAT, use HTTP headers like X-Forwarded-For[23], etc.).

The most common solution to forward the originating IP address of the user to the application is to configure the proxy device to add an additional HTTP header to the request. The list of HTTP headers for this purpose includes (but is not limited to):

- Real-IP;
- Client-IP;
- X-Forwarded-For;
- X-Real-IP.

This is used to add relevant information to the backend logs. The common mistake is that only the frontend IP will be written to the log files instead of the originating IP. E.g. the Apache HTTP Server configuration that is usually applied to log the forwarded IP address is given in Figure 1.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-
Agent}i\"" combined
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b
\"%{Referer}i\" \"%{User-Agent}i\"" proxy
SetEnvIf X-Forwarded-For "^.*\..*\..*\..*" forwarded
CustomLog "logs/access_log" combined env=!forwarded
CustomLog "logs/access_log" proxy env=forwarded
```

Figure 1

Apache HTTP Server checks the presence of the *X-Forwarded-For* header and if the header is set, it replaces the IP address with the respective address in the header. In most cases this is sufficient, but there are exceptions.

If the service that is being logged is not behind a proxy server, it introduces a risk of logging incorrect information about the source IP. If Eve sends a request with the crafted header, which is given in Figure 2, then Apache will get the header and based on the previous configuration example will overwrite the IP address with the one, given by Eve in the header. This would result in log poisoning: the activity is logged, but with a wrong source IP address.

```
GET / HTTP/1.1
Host: www.service.int
X-Forwarded-For: 1.22.55.22
Connection: keep-alive
```

Figure 2

To avoid such attacks, the logging of such headers should be decided based on the respective system architecture: if the proxy device doesn't add *X-Forwarded-For* or other headers, then the logging system must NOT overwrite the original IP address. Both addresses (the original source and IP in the header) must be logged distinctively.

5.3.5 When

Every log event has to have a corresponding timestamp (date and time) to be able correlate the event to other (IT and real-life) events in other systems. To be able to correlate different events, the server must have an accurate time

configured to it (Appendix 3, p. 21.5.1; p.5). This can be done through an NTP service. If global NTP services cannot be used, then at least local servers must have synchronized times among themselves to enable correlation within the organization. The time must be defined at least with the detail of a second (Appendix 3, p.21.5.2). Depending on the amount of events in the system, it could be needed to detail down to a millisecond. To avoid time multiplication within the log file due to daylight savings and to enable correlating logs between different time zones, UTC time zone must be used in the logs (Appendix 3, p.21.5.3).

When considering the date format, usability of the format must be taken into account in case of automated log analysis and the tools that could be used for the initial information gathering (statistics, measuring etc.). All the systems must log in the same date format in order to simplify the correlation and analysis.

To use simple command-line tools (*awk*, *grep*, *sort*, *uniq* etc.) for analyzing the logs, ISO8601[29] should be considered as a suitable standard, suggested format is `yyyy-MM-DD`**`T`**`HH:mm:ss.SSSZ` (Appendix 3, p. 21.5.4). There are several benefits when using this format:

- The format is human-readable. When comparing to UNIX-timestamps, analyst can easily understand the content without date conversion;
- Number-based format enables simple analytic activities. If the format would include String values (e.g. month names as short textual values *Jan – Dec*), then sorting would be more difficult than using numbers;
- The "from larger to smaller" approach gives the possibility to collect information and gather statistics in an easier way (analyst does not have to parse the date but can just cut a needed portion of it using simple command-line tools).

5.3.6 Result

In order to find the errors from the system, security incidents, attacks on the system and to discover possible data leaks, the logs must contain information about the result (Appendix 3, p. 21.6). It is not always required to log all the data

that server responded: this might duplicate the critical information from the database to logs and might cause additional security issues rather than mitigating them.

It must be decided case-by-case how much information should be in the logs about the response: response code, response size and outcome type (success, attempt, failure, error etc.) must be mentioned within the log event. For example: if the user searches for something in the UI, logged event must contain not only the response size in bytes but also the number of returned rows (Appendix 3, p. 21.6.1).

5.4 Usability of logs

Since the requirements for logging usually do not define the method for logging, developers tend to use their own preferred mechanisms. One of the common ways to log the events in the system is to log them into the same database that the service uses. The benefits of this kind of approach (but not limited to) are:

- Existing connection to the database can be reused;
- Foreign keys can be used in the log event for referring to objects in the database;
- Guaranteed logging can be achieved simply due to committing the event.

However this kind of approach might be harmful for the system:

- Exporting logs from the database might create issues or cause performance problems (specially exporting them online);
- Logs in the database can be altered if no proper database audit is in place;
- In case of database corruption, there is no good way to get the events from the database to investigate the source of the corruptive behavior;
- Security specialists and log analysts must access the production environment to analyze the log events, thus having access to production environment and creating additional load on the production servers;
- If the log analysis takes place on different servers, the internal foreign key references might not be usable;

- If an object is removed from the database that the log event references to (with a foreign key) the logs could be unusable;
- If service is obsolete and taken down but there is a requirement to keep the logs for 10 years and the log is stored in a database with foreign keys, the whole service has to be archived for the required amount of time instead of just logs, later analysis of the logs require the whole service to be restored;
- Log table must be append-only;
- Etc.

To guarantee the availability of and replication of those logs to a central log repository, events must be written to a file on the operating system (Appendix 3, p. 6) or sent directly to centralized log server if there is no way to get the flat files from the server (Appendix 3, p.28).

There are some approaches where in order to save space on the disk proprietary binary log formats are being used for logging. To simplify the log analysis and finding the events from the logs (and to be able to use standard tools for quick analysis), the flat files must be in ASCII format (Appendix 3, p. 7). The ASCII format also prevents certain problems with log transportation and analysis, which this thesis will cover later.

The logs might contain data, which is difficult to parse through automatically or creates additional overhead that is not needed. Human-readable events in the log like "*User Bob has logged in from a remote IP '1.2.3.4' yesterday morning at 10*" could be suitable event description in cases where there are not many events per second. When analyzing thousands of events at the same time, the "story-book" approach creates additional administrative overhead.

To simplify the log analysis, the log file must have a well-defined structure that is easy to analyze with command-line tools or import them to analyzing software. Columns must be separated with a symbol that has a low probability in

appearing in the actual events, e.g. ';' – semicolon, 0x09 – TAB etc. (Appendix 3, p.16.1).

One of the common approaches is that the logger does not distinguish the parameter-value pairs that are created on the server side and the pairs that are sent by the user with a request. This could be harmful for the log quality due to log poisoning and evading logging. For example:
The service has to log the timestamp, IP address, username, derivate of the session ID and input parameters with their respective values. It has been decided, that JSON format will be used for the log. An example of an ordinary log entry is given in Figure 3.

```
{"timestamp":"2014-11-
15T14:59:29.000Z","host":"192.168.1.13","session_ID":"1a5d0525543a
559686dcedb2a3d585d1ef582240","username":"Alice","action":"make_pa
yment","account":"22051102","beneficiary":"1100102","amount":"100"
,"details":"Car leasing payment"}
```

Figure 3

The user sends the parameters and values that are marked with red. If a malicious user has some insights of the logging format, it might be possible to evade logging or send falsified information to the logs. An example of the sent malicious request is given in Figure 4. Falsified information is highlighted.

```
GET
/action.php?action=make_payment&account=22051102&beneficiary=11001
02&amount=100&details=Car%20leasing%20payment&host=172.28.1.2&user
name=Eve HTTP/1.1
Host: www.service.int
Cookie: PHPSESSID=86888qoi8gsjn3cdap6q74fju2
Connection: keep-alive
```

Figure 4

In case of such request, false information will be written in the log. The outcome that will be logged with such request is given in Figure 5.

```
{"timestamp":"2014-11-
15T15:05:16.000Z","host":"172.28.1.2","session_ID":"09b24a8413c746
2d5df69ef8422ca9b58accf410","username":"Eve","action":"make_paymen
t","account":"22051102","beneficiary":"1100102","amount":"100","de
tails":"Car leasing payment"}
```

Figure 5

To prevent log injections and -evasion, log files must distinguish the parameters that are server-side and the parameters that are sent with the user request (Appendix 3, p. 18). An example log format is given in figure 6.

```
2014-11-15T15:14:03.000Z;192.168.1.1;
65b7e8ed48d82bce0e749af5f1a889884ff1223;Alice;{"action":"make_paym
ent","account":"22051102","beneficiary":"1100102","amount":"100.-
","details":"Car leasing
payment","host":"172.28.1.2","username":"Eve"}
```

Figure 6

One common approach for simple logging is to use several lines for one event (i.e. multi-line events). An example of the log that would be created in such manner by the previous payment example is given in Figure 7.

```
timestamp=2014-11-15T15:25:42.000Z
host=217.71.44.6
session_ID=a12dcddd0ee5054b240d7d17eb16e87c5baffd52
username=Alice
action=make_payment
account=22051102
beneficiary=1100102
amount=100.-
details=Car leasing payment
host=172.28.1.2
username=Eve
```

Figure 7

This can be an acceptable approach in cases where all the requests have always the same amount of fields that are either empty or have respective values. However, if the amount of fields depends on the amount of parameters the request has and it can vary depending on the action (e.g. payment has six

parameters but balance overview has only three) then it is a huge overload during log analysis.

In order to prevent such problems, multi-line events must be avoided and event must be written into one line, e.g. using JSON format (Appendix 3, p.16.2). Log files must be homogeneous: if there is no other way and multi-line events have to be used (e.g. stack traces, debug information etc.) then separate log file must be used for multi-line events to simplify log analysis (Appendix 3, p.17.1). The beginning and ending of the multi-line event must be identifiable (e.g. event always starts with a timestamp, ends with a symbol or a blank line (CRLF) (Appendix 3, 17.2).

5.5 Logging is vulnerable

The attackers who are targeting information systems have usually two common goals:

- Extract or modify confidential data;
- Use the system to attack other systems.

System has to keep an audit trail of all activities in order to provide evidence about what was done and how. This gives relevant information in order to improve security and prevent such attacks from happening in the future. To reach the goals of an attack, attackers may:

- Try to evade logging or destroy the evidence about their activities;
- Use vulnerabilities in the logging system to attack other systems.

There are several ways to use logging vulnerabilities to inject falsified data, evade detection or attack other systems. To prevent such incidents, countermeasures must be taken.

5.5.1 Log evasion

Information from the server can be requested in different ways. Client can send request parameters via *GET* or *POST* method or via Cookies. In case of PHP, the documentation adds two additional sources for variables: Environment and Server[26]. The prioritization order can be defined in a respective configuration

file (in case of PHP, a parameter called "*variables_order*"[23] defines the priority order, by default it is EGPCS, defining the order from the least important (Environment) to the one that cannot be over-written (Server)). This is relevant because it might cause a situation, where the values that are logged are not the ones that are used by the service, thus resulting in log evasion.

```
GET /login.php?user_id=Alice&password=Secret&user_id=Bob HTTP/1.1
Host: www.service.int
Connection: keep-alive
```

Figure 8

In Figure 8 a request is done via *GET* method but it contains two variables with the same name *'user_id'*. Which parameter is used by the web server, depends on the used technology and its version[25]. To be sure that all the evidence is saved in a proper way, logging system must save the same value as is used by the system (Appendix 3, p. 11.5). If all the variables are logged, then it should be noted, that the order of the variables is also important in the log (Appendix 3, p.11.7). To provide more usability of the log and to provide better analyzability, the value must always be visible in the log. If the value of a required parameter is missing or empty, the placeholder (e.g. "null") must be in the log (Appendix 3, p.11.10). This helps to distinguish the parameters that were not present, were empty or filled with 0x32 (space) characters at the time of writing the event to the log file. If the placeholder is being used, it should be distinguished from the values that user inserts, i.e. if user insers "null" as a value, then it must be possible to understand if the value is from a placeholder or user-inserted value (Appendix 3, p. 11.10.1).

```
POST /login.php?user_id=Alice&password=Secret&user_id=Bob HTTP/1.1
Host: www.service.net
Connection: keep-alive
Content-Length: 35

user_id=John&user_id=James
```

Figure 9

In Figure 9 a request is done via *POST* method, but the variable *'user_id'* is set twice in both *GET* and *POST* requests. If a service does not distinguish *POST* and

GET (e.g. uses *$_REQUEST['user_id']* in PHP or *request.getParameter('user_id')* in Java) then *GET* variables could be used instead of *POST* and *vice et versa*. If the default priority is set, then *POST* overrides *GET* variables, which means that in this case *'user_id'*(s) from the *GET* request are written to the log file but the *POST* variable will be used by the service. In cases, where *POST* is not logged, the real username who tried to authenticate will not be logged.

To avoid such cases, *POST* request body parameters should be also logged wherever needed (Appendix 3, p. 11.6). In case the values exceeds a reasonable length (e.g. 256 bytes), only a hash value of the payload and the payload length in bytes must be written to the log file. The exact payload can be saved to a separate file (e.g. in base64 encoded format) with the hash of the payload, date and the unique identifier of the event for detailed analysis (Appendix 3, p. 20). This gives the possibility to link the exact posted value to the user request in the request log.

If GPC[3] is used, then Cookie values can interfere as well: if a cookie parameter *'user_id'* is set, then Cookies will be used, leaving no traces of their values. However, saving all the cookies with all the requests might not be reasonable, so developers should keep in mind that such possibility exists and log the correct *'user_id'* that would be used by the service itself to the authentication log event (Appendix 3, p. 11.5).

To detect such incidents where a malicious user tries to send multiple *'user_id'* values, the service should write an event with all the relevant data (including cookie values) to the security log for later analysis.

Some logging systems have limited the maximum event length. While this might be a good idea to save space and have some boundaries for the event, it might create a situation where some information will not be logged at all. For example, Syslog default message size is by default 8192 bytes, PHP function

[3] GPC – GET-POST-Cookie priority in variable ordering

trigger_error()[27] limits the message size to 1024 bytes and older versions of IIS logged only 4097 bytes of the GET request[33]. This might result in a situation where a request given in Figure 10 would log only the first part of the request and the relevant part (in this case – *'user_id'*) will not fit in to the log event.

```
GET /login.php?dummy=<4097 x A>&user_id=Bob HTTP/1.1
Host: www.service.int
Connection: keep-alive
```

Figure 10

Such scenario can be prevented with the mitigation mentioned in the previous example (Appendix 3, p. 20), but the length issue remains: if an attacker would generate 4096 variables (e.g. *'dummy_1'* – *'dummy_4096'*) with the value 'A', then logging large payloads to a separate file would not mitigate the attack. To prevent such situations, all default log message limits must be extended to be longer then a possible attacker expects (Appendix 3, p. 19). To detect such attacks or to extend the maximum limit, length measuring of the request must be done (Appendix 3, p.19.1).

Also there is a possibility to use separate files for used variables and raw variables (Appendix 3, p. 2). For example: a system could include a request log that logs only these parameters and values that are required by the system to work (and are actually used by the service), all other parameters (or the full request with generated attributes) will be logged to a so-called raw request log file for investigation purposes.

There are other ways to evade logging of malicious attacks against an application user, for example: there is a possibility to use XSS attacks against victims via URL anchors. An example is given in Figure 11.

```
http://www.service.net/xss.php?inject=<script>eval(location.hash.s
lice(1));</script>#document.write('<img
src=http://hack.er/img.jpg'+document.cookie+'>')
```

Figure 11

Since the anchors are executed on the client side to refer to an object or a place on the loaded page, then it is not forwarded to the server (thus they will not be

shown in the log). So unfortunately there is nothing that can be done on the server side from logging perspective to mitigate this issue. This kind of attack vector must be taken into account from the system administration perspective – Content-Security Policy can be implemented to mitigate this kind of attack. From logging perspective, Content-security policy reporting[32] must be in place to detect such XSS attacks (Chrome 16+, Safari 6+, Firefox 4+, IE 10).

5.5.2 Attacks against logging

There are three types of attacks against logging: attacking the log files, attacking the log viewer and attacking other systems through the logging systems. The goal of the attack is to hide malicious activities in the log files or generate new events to misguide the analyst who looks for such events.

One of the attack vectors is inserting a new-line symbol to the log: if an attacker knows the log structure, it is easy to inject new events to the log file. Example log file structure is given in figure 12.

```
2013-04-03 12:43:32     Login for user Alice succeeded.
2013-04-03 12:43:32     Login for user Bob failed.
```

Figure 12

If the username and event is written directly to the log file, then Eve can enter the value "`Eve succeeded.\n2012-04-02 10:12:53  Login for user Bob`" as her username. This would result in an entry marked in Figure 13, leaving the log analyst wondering, how Eve managed to log in to the service without the proper privileges.

```
2013-04-03 12:43:32     Login for user Alice succeeded.
2013-04-03 12:43:36     Login for user Bob failed.
2013-04-03 12:44:02     Login for user Eve succeeded.
2012-04-02 10:12:53     Login for user Bob failed.
```

Figure 13

Most of the solutions[4][11][21] found online suggest replacing *new-line* character with an alternative character such as an underscore (_), but unfortunately this is not sufficient and gives falsified information to the log analyzers. The solution

proposed by the author for such log poisoning will be given later on in this chapter.

Another attack works, if the logging viewer is a fixed-width application. Injecting whitespaces might result in the same way as in the previous example: instead of a new-line insertion, a proper amount of whitespaces will be inserted, which will pad the entered information until the next line. Even if the character replacement might work in case of the new lines, replacing spaces might not be possible due to the application peculiarity.

Many SIEM systems and custom log viewers are developed as web-based systems to provide multi-platform support and usability from different operating systems. Due to not sanitized log entries, web based log viewers might be affected by the same kind of vulnerabilities as ordinary web applications (e.g. XSS, CSRF). Through these kinds of vulnerabilities, an attacker can gain access to the log analysis server or attack the log analyst through a vulnerability log analysis tool.

A malicious user could enter some bogus information just to confuse the log analyzers or to evade logging. An attacker might use homoglyphs[4] as values to confuse and miss-lead the security team or to get people to open specially crafted links. An analysis[5] was done on the use of punycode and homoglyph attacks to obfuscate URLs for phishing. There are several letters, which look the same but their actual ASCII value is different (e.g. the lower-case "o" can be represented with 0x006f, 0x03bf and 0x043e). So if the log files state, that user "admin" logged in, there are actually 287 other combinations[5] that could look like "admin" but for the computer is with a different meaning.

---

[4] Homoglyph - one of two or more graphemes, characters, or glyphs with shapes that either appear identical or cannot be differentiated by quick visual inspection. Two characters look alike but are with different ASCII codes.

[5] Symbols: **a** – 0x0061, 0x0430, 0xff41; **d** – 0x0064, 0x0501, 0x217e, 0xff44; **m** – 0x006d, 0x217f, 0xff4d; **i** – 0x0069, 0x0456, 0x2170, 0xff49; **n** – 0x6e, 0xff4e.
Calculation: 3 x 4 x 3 x 4 x 2 – 1 = 287

Another possibility to evade logging and miss-lead the analyst is to use right-to-left override[7]. The non-printable sequence of symbols is meant to aid languages that must be read from right to left (e.g. Persian, Arabic, Syrian and Hebrew). This functionality has been used in the cyber criminal world for e-mail attacks[12]. If the non-printable symbols that execute the over-ride (0xe2 0x80 0xae) are entered, then starting from that point all the following text will be printed in a reversed mode. Since the characters that activate this are "non-printable" and cannot be seen in the web-based log viewer, it is hard to determine what really happened. I.e. if a user would log in with a username "nimda" with the surrounding characters, then html-based log viewers might show that there were some login events with the user "admin".

There are other problems with logging and non-printable symbols[1]. When looking at the common command line tools (e.g. *cat*, *more*, *less*, *tail*), administrator cannot be safe from various attacks. If Eve sends the backspace code (0x08) with the request, she can hide events from sight. A sample request is given in Figure 14. If the last *'user_id'* value that is being sent will be used by the system (in this example – *'user_id=Bob'*), the log entry will be shown in the terminal as given in Figure 15.

```
GET /login.jsp?user_id=Alice&user_id=Bob&<08><08><08><08><08><08>
<08><08><08><08><08><08>password=Secret HTTP/1.1
Host: www.service.net
Connection: keep-alive
```

Figure 14

```
192.168.1.13 - - [09/Nov/2014:22:04:12 +0200] "GET
/login.jsp?user_id=Alice&password=Secret HTTP/1.1" 200 1887
```

Figure 15

The events are actually written down to the file in the right way. Due to the non-printable symbols, the programs mentioned earlier cannot show the log events properly. If an administrator would *grep* the log event, he would get the response given in Figure 16. *Grep* can find the line by the pattern *'Bob'*, but unfortunately was not able to display the event properly, thus still hiding Bob from the events.

```
$ grep Bob localhost_access_log.2014-11-09.txt
192.168.1.13 - - [09/Nov/2014:22:16:24 +0200] "GET
/login.jsp?user_id=Alice&a=b&password=Secret HTTP/1.1" 200 1887
```

Figure 16

The escape code usage is not limited to the current line. This is due to many computer terminals and terminal emulators (e.g. *vt100* and *xterm*-compatible terminals) supporting color and cursor control through a system of escape codes.

Some simpler attacks include color changes: if Eve would want to change the background- and foreground colors of the terminal, she just has to send the right CSI byte sequence (\e[<3n>;<4n>m, where n defines the color for foreground and background respectively) as brought out in Figure 17.

```
GET /login.jsp?<1b>[33;41ma=b&user_id=Bob&password=Secret HTTP/1.1
Host: www.service.int
Connection: keep-alive
```

Figure 17

The example given in Figure 17 will turn the text color to yellow and background to red staring from the first parameter.

Playing with colors is not the only trick that can be done with escape sequences. An attacker can also move the cursor to another location inside the terminal, which would result with overwriting the previous log entries on the screen. The escape code '*s*' saves the cursor position; '*u*' restores the cursor position. For example, the request given in Figure 18 would be visible in the vulnerable terminal as given in Figure 19. The text in Figure 18 that is marked with red will not be visible in the terminal.

```
GET
/index.html?<1b>[sa=b&user_id=Bob&password=Secret&<1b>[u;some_tota
lly_bogus_information HTTP/1.1
Host: www.service.int
Connection: keep-alive
```

Figure 18

```
192.168.1.13 - - [10/Nov/2014:00:03:34 +0200] "GET
/index.html?some_totally_bogus_information HTTP/1.1" 200 1887
```

Figure 19

5.5.2.1 Mitigation against attacks

To overcome the aforementioned vulnerabilities and attack vectors against the web- and terminal-based log analyzing tools, the log entries must be sanitized. Replacing characters with an underscore or other dummy characters is not sufficient: blacklisting all the possible string would require updating the blacklist based on the currently known threats. Also – such replacement would remove important information from the log files: log analyzers would not know that some malicious user is trying to attack the system with crafted queries or know which symbols were entered by an attacker.

The solution would be coding (Appendix 3, p. 13). All non-printable symbols (0x0000...0x001f, 0x007f...0x00ff) must be saved in a safe manner in order not to disrupt the analyzing tools but at the same time give an indication about the exact characters entered by the user. Log file must contain only ASCII printable symbols (Appendix 3, p.13.1).

5.6 User privacy within log file

Log files are used by different roles with different needs: developers, system administrators, security specialists, forensic experts, business analysts and other people with necessity might get a hold of some portions of logs to perform their tasks. Also there is a chance that log files might get misplaced. If too much confidential information about events is logged, it creates new problems from the user privacy point of view. To protect the system, application and the end users, some restrictions have to be set for the content that must be logged.

When Alice logs into a web service, she uses username and password to authenticate herself. Although all relevant information must be logged to reproduce the Alice's activities, service must not log such information that might give the log reader the ability to miss-use Alice's account (Appendix 3, p 12).

For the same reason, any kind of passwords must never be logged (Appendix 3, p. 12.1). Instead - the service must log the entered password length in bytes (Appendix 3, p.12.1.1). Any kind of private keys and other alternative repeatable authentication values must not be logged (Appendix 3, p.12.2). To prevent session hijacking, real *session ID* must not be logged (Appendix 3, p.12.3), instead the service can log the hash or other irreversible derivation of the *session ID* (Appendix 3, p.12.3.1).

To prevent duplicating data into logs from the databases and other data sources in order to:

- Prevent data leakage;
- Unauthorized data access through logs;
- Save up disk space;
- Provide data security

Responses from the data sources must not be logged (Appendix 3, p.12.4). Results (including error codes) and response size (returned rows, size in bytes) must be logged (Appendix 3, p.12.4.1).

5.7 Documentation requirements

Security and logging is not a component in a service that can be easily attached to an existing system. They must be considered already in the design phase and they must be taken into account every time when functionality is created or added to the system. Even the log samples must be introduced with the use cases at the end of the software analysis phase (Appendix 3, p. 22, p. 23) to guarantee proper logging and to avoid situations where logging is not done or is done improperly.

It is common, that developers plan to implement logging in the final phase of the development, after the functionality has already been developed. Since budget and deadlines pressure the scope of the project, there is a possibility that the final phase where logging should have been implemented will not be executed. To prevent such situations, logging and other security measures must be implemented in parallel with the functionality. Every time a new functionality is

introduced (and verified by the client), logging of the functionality must be also delivered and verified (Appendix 3, p. 24).

Documentation of logs including descriptions for the separate log files and events they contain, fields and constants used in the log files and other relevant data must be included in the service documentation and kept up to date based on developed logging functionality not the initial requirement documents (Appendix 3, p. 23). This will guarantee the documents being up to date, not based on the initial task that might not be opportune.

Conclusion

Logging is an important part of the information system. By authors knowledge no thorough study was done prior to this thesis on the log requirements and logging content. This thesis covered the first two steps of the log lifecycle, giving additional input for the requirements and defining the actual data that must be logged in such cases. The thesis covered the log content requirements for a log event and analysed where these requirements evolved. The thesis brought out the most common mistakes done in logging implementations and offered solutions to eliminate them.

The outcome of this thesis is a separate detailed requirement document for logging in English and Estonian language (Appendixes 3 and 4 respectively) that can be applied as a part of non-functional requirements in the development process. Although the study has been done based on a governmental institution, the document can be applied in both public and private sector developments.

The outcome of this thesis has been applied in the administrative field of The Ministry of Interior and has proven to be a valid, understandable and applicable set of requirements for developments. The requirement document has been added to the non-functional requirements of several software development procurements.

Author of this thesis thanks his supervisors Mait Peekma and Toomas Lepik and Jaan Priisalu, Andres Kütt and Renee Trisberg for their contribution and input to this research.

6 References

1. ASCIITable homepage - ASCII Table and Description [WWW] http://www.asciitable.com (11.11.2014)
2. Bowditch, Nathaniel. (1966). "U.S. Hydrographic Office". American Practical Navigator: an Epitome of Navigation. Washington, DC: U.S. Government Printing Office. p. 31.
3. Bundesampt für Sicherheit in der Informationstechnik - IT-Grundschutz [WWW] https://www.bsi.bund.de/DE/Themen/ITGrundschutz/itgrundschutz_node.html (11.11.2014)
4. CERT: Carnegie Mellon University, Software Engineering Institute - IDS03-J. Do not log unsanitized user input [WWW] https://www.securecoding.cert.org/confluence/display/java/IDS03-J.+Do+not+log+unsanitized+user+input (11.11.2014)
5. Crenshaw, Adrian - Use of Punycode and Homoglyph Attacks to Obfuscate URLs for Phishing [WWW] http://www.irongeek.com/i.php?page=security/out-of-character-use-of-punycode-and-homoglyph-attacks-to-obfuscate-urls-for-phishing (11.11.2014)
6. Federal Aviation Administration - Final Rule [WWW] http://www.faa.gov/regulations_policies/rulemaking/recently_published/media/23532.DOC (11.11.2014)
7. FileFormat homepage - Unicode Character 'RIGHT-TO-LEFT OVERRIDE' [WWW] http://www.fileformat.info/info/unicode/char/202e/index.htm (11.11.2014)
8. Information System Authority - ISKE catalogues v. 7.00 [WWW] http://www.ria.ee/public/ISKE/ISKE_kataloogid/ISKE_kataloogid_7.pdf (11.11.2014)
9. Information System Authority - ISKE Implementation Guide [WWW] https://www.ria.ee/public/ISKE/ISKE_kataloogid/ISKE_rakendusjuhend_7.pdf (11.11.2014)

10. Information System Authority – Three-level IT baseline security system ISKE [WWW] https://www.ria.ee/iske-introduction/ (11.11.2014)
11. John Melton's Weblog - Preventing Log Forging in Java [WWW] http://www.jtmelton.com/2010/09/21/preventing-log-forging-in-java/ (11.11.2014)
12. Krebs on Security - 'Right-to-Left Override' Aids Email Attacks [WWW] http://krebsonsecurity.com/2011/09/right-to-left-override-aids-email-attacks/ (11.11.2014)
13. Proofprint homepage - Proofpoint Uncovers Internet of Things (IoT) Cyberattack [WWW] http://www.proofpoint.com./about-us/press-releases/01162014.php (11.11.2014)
14. Rainers Blog - On the (un)reliability of plain tcp syslog [WWW] http://blog.gerhards.net/2008/04/on-unreliability-of-plain-tcp-syslog.html (11.11.2014)
15. Riigi Teataja - Andmete, järelepärimiste, logifailide ja taotluste säilitamise, Tehnilise Järelevalve Ametile üleandmise ning kustutamise ja hävitamise kord [WWW] https://www.riigiteataja.ee/akt/13100712 (11.11.2014)
16. Riigi Teataja - Arvutite ja kohtvõrkude kaitse nõuded [WWW] https://www.riigiteataja.ee/akt/12905091 (11.11.2014)
17. Riigi Teataja - Infosüsteemide andmevahetuskiht [WWW] https://www.riigiteataja.ee/akt/119012011015 (11.11.2014)
18. Riigi Teataja - Infosüsteemide turvameetmete süsteemi kehtestamine [WWW] https://www.riigiteataja.ee/akt/791875 (11.11.2014)
19. Riigi Teataja - Isikuandmete kaitse seadus [WWW] https://www.riigiteataja.ee/akt/114032014031 (11.11.2014)
20. Riigi Teataja - Riigisaladuse ja salastatud välisteabe seadus [WWW] https://www.riigiteataja.ee/akt/121062014055 (11.11.2014)
21. Riigi Teataja homepage [WWW] http://www.riigiteataja.ee (11.11.2014)
22. SANS Software Security - Log Forging [WWW] http://software-security.sans.org/blog/2013/05/21/whatworks-in-appsec-log-forging (11.11.2014)
23. The Internet Engineering Task Force - Forwarded HTTP Extension [WWW] http://tools.ietf.org/html/rfc7239 (11.11.2014)

24. The Internet Engineering Task Force - Transmission Control Protocol [WWW] https://tools.ietf.org/html/rfc793#section-3.1 (11.11.2014)
25. The Linux Documentation Project - Multicast explained [WWW] http://www.tldp.org/HOWTO/Multicast-HOWTO-2.html (11.11.2014)
26. The Navy & Marine Living History Association - The Speed Log: History, Construction and Use [WWW] http://www.navyandmarine.org/planspatterns/speedlog.htm (11.11.2014)
27. The Open Web Application Security Project - HTTP Parameter Pollution [WWW] https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf (11.11.2014)
28. The PHP Group – Description of core php.ini directives [WWW] http://php.net/manual/en/ini.core.php#ini.variables-order(11.11.2014)
29. The PHP Group – trigger_error [WWW]http://php.net/manual/en/function.trigger-error.php (11.11.2014)
30. U.S. Navy Museum - Matthew Fontaine Maury (1806-1873) [WWW] http://www.history.navy.mil/branches/teach/ends/maury.htm (11.11.2014)
31. University of Cambridge, Computer Laboratory - A summary of the international standard date and time notation [WWW] http://www.cl.cam.ac.uk/~mgk25/iso-time.html (11.11.2014)
32. University of Miami - Privacy/Data Protection Project [WWW] http://privacy.med.miami.edu/glossary/xd_confidentiality_integrity_availability.htm (11.11.2014)
33. Web Application Security Consortium - Preventing Log Evasion in IIS [WWW] http://www.webappsec.org/projects/articles/082905.shtml (11.11.2014)
34. World Wide Web Consortium - Content Security Policy: report uri [WWW] http://www.w3.org/TR/CSP/#report-uri (11.11.2014)

35. Souppaya, Murugiah; Kent, Karen - Guide to Computer Security Log Management [WWW] http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf (23.12.2014)
36. Vaarandi, Risto - Tools and Techniques for Event Log Analysis [WWW] http://kodu.neti.ee/~risto/publications/thesis.pdf (23.12.2014)

Appendix 1 – Acronyms

ASCII - American Standard Code for Information Interchange

BSI – Bundesamt für Sichereheit in der Informationstechnik

CONNECT – CONNECT Request method for HTTP

CRLF – Carriage Return + Line Feed

CSI – Control Sequence Introducer

CSRF – Cross-Site Request Forgery

DNS – Domain Name System

EGPCS – Environment Get Post Cookie Server

FTP – File Transfer Protocol

GET – GET Request method for HTTP

GPC – Get Post Cookie

HTTP – Hypertext Transfer Protocol

ICMP – Internet Control Message Protocol

IDS – Intrusion Detection System

IIS – Internet Information Services

IP Address – Internet Protocol Address

IPv4 – Internet Protocol version 4

IPv6 – Internet Protocol version 6

ISKE – Infosüsteemide Kolmeastmeline Etalonturbesüsteem

IT – Information Technology

JSON – JavaScript Object Notation

LAN – Local Area Network

NAT – Network Address Translation

NNTP – Network News Transfer Protocol

NTP – Network Time Protocol

PC – Personal Computer

PHP – Hypertext Preprocessor

POST – POST Request method for HTTP

SIEM – Security Information and Event Management

SMTP – Simple Mail Transfer Protocol

URL – Uniform Resource Locator

UTC – Universal Time Coordinated

VT100 – Video Terminal Emulator

WORM – Write-Once, Read-Multiple

XSS – Cross-Site Scripting

XTERM – Terminal emulator for the X-Window System

Appendix 2 – Log samples

To give a better overview of the fields in the log event, the events will be represented in a form of a table. The field separator in the log file must be excluded from the field values.

Session log example

DateTime	UniqueID	Action	IPAddress	SessionID	Method	Result	Payload
2014-11-25T18:21:49.000Z	node-1_48127	LOGIN	192.168.1.13	0fd512414	IDCard	SUCCESS	{"**userID**":"Alice", "**password**":13, "**userAgent**":"Mozilla\/4.0 (com**%09**[6]patible; MSIE 9)"}
2014-11-25T18:22:17.000Z	node-1_65353	LOGOUT	192.168.1.13	0fd512414	Button_1	SUCCESS	{"**userAgent**":"Mozilla\/4.0 (compatible**%2509**[7]; MSIE 9)"}
2014-11-25T18:22:38.000Z	node-1_78872	TIMEOUT	Null	0fd512414	Null	SUCCESS	{"**userAgent**":"Mozilla\/4.0 (compatible; MSIE 9)"}

Request log example

DateTime	UniqueID	Action	IPAddress	Username	SessionID	Result	Payload
2014-11-25T19:19:50.000Z	node-1_4303	REQUEST	192.168.1.13	Alice	0fd512414	SUCCESS	{"**pageID**":"users.search.new", "**name**":"Bob", "**results**":4, "**userAgent**":"Mozilla\/4.0 (compatible; MSIE 9)"}
2014-11-25T19:19:50.000Z	node-1_4503	REQUEST	192.168.1.13	Alice	0fd512414	SUCCESS	{"**pageID**":"users.profile.view", "**id**":232, "**userAgent**":"Mozilla\/4.0 (compatible; MSIE 9)"}
2014-11-25T19:19:52.000Z	node-1_4574	REQUEST	192.168.1.13	Alice	0fd512414	FAILURE	{"**pageID**":"users.profile.edit", "**id**":232, "**userAgent**":"Mozilla\/4.0 (compatible; MSIE 9)"}

Appendix 3 - Log requirements (English)

Basic Guidelines for Logging

1. Standard component must be used for logging (e.g. *log4j* in case Java).
2. Events should be divided into separate log files as follows:
   2.1. **Session Log** – information about user authentication, authorization procedures including login to an application, application module or a part of an application with elevated security level, logout, session timeout and session invalidation and the respective outcome (*Success*, *Attempt*, *Failure*);
   2.2. **Request Log** – Information about the user activities and requests, including request type, session parameters (to correlate session logs with request logs) and input parameters given by the user (including information about external resource usage);
   2.3. **Debug Log** – technical details for debugging purposes. In production environments debug logging should be turned of by default;
   2.4. **Security Log** – pre-defined security-related events to monitor (IP change within a session, prohibited username usage attempts, faulty authentications etc);
   2.5. **Error Logs** – Information about various errors, that must be divided:
      2.5.1. **Technical Error Log** – technical error messages that originate from the system behaviour and unhandled user exceptions (problems with interfaces and connections to third party applications, problems with background jobs etc). Log can countain full error messages, stack traces and other error-related information;
      2.5.2. **User Error Log** – handled errors that show an error message to the user. The errors are related with the service functionality and are a result of a user activity (e.g. an attempt to open a file without proper privileges, make a large payment without the sufficient funds etc).
3. Logging must be optimized. Duplicated information in the logs must be avoided (if not required otherwise).

4. If one event creates entries to different log files, the log entris have to be correlated via a common field in both of those entries. The field must not be a timestamp, but it can be a unique event ID.
5. All systems that log have to synchronize their time with a central NTP server in order to provide the correctness in event timing and ordering.
6. All logs must be atleast file based; logs must be usable without the application and its components (e.g. its database). If the service requires that log events must be in the database, then it must be additionally to the file-based logs.
7. Logs files must be in a plain-text format (in ASCII printable symbols).
8. Logs must be extractable from the production system in a way that the analyzability of the log files would remain.
9. Logging should be based on functionality, not module or application component. I.e. all authentication requests should be in one session log file and user requests in a request log, not *module A* request and session on one module log and *module B* in the other module log.
10. Successful logging is mandatory – if it is not possible to write a log, the request must not be served and services should be closed (if needed).
11. All user activities must be logged.
    - 11.1. All authentication attempts (regardless of the outcome) must be logged, including attempts with empty or missing parameters.
    - 11.2. All administrative activities must be logged.
    - 11.3. Administrator must not modify the logs, delete the logs or stop logging.
    - 11.4. Both successful and failed activities must be logged.
    - 11.5. Parameters that were used by the application must be logged.
    - 11.6. Parameters and values that were given in the HTTP body (having the *content-type* as *www-url-encoded*) must be logged;
    - 11.7. The order of parameters in the log file must be the same as it was in the request sent to the server or the service;
    - 11.8. Requests with cached responses must be logged;
    - 11.9. When using SSL/TLS, the version and used cipher must be logged;

11.10. If the value of the parameter is not sent or is left empty, it must be marked as a placeholder (e.g. “null”).

11.10.1. If a placeholder is used, it must be distinguishable from the user-inserted value (i.e. logs should note if “null” was inserted by the user or a placeholder is used).

12. System must never log:

12.1. Passwords in plain-text format;

12.1.1. Password length can be logged (in bytes);

12.2. Private keys;

12.3. Value of the session ID (e.g. session tokens or cookies);

12.3.1. Hash or other irreversible derivate of the session ID can be logged;

12.4. Full-text responses from the database.

12.4.1. Fact about the response, response size (returned rows) or an error message can be logged.

13. All input-data given by the user must be coded before written to the log (i.e. all user input must be reproduced by the logs except data written in p. 12), to exclude log injections and attacks related to them;

13.1. All non-printable symbols (0x00..0x1f, 0x7f..0xff) and field separator characters in the input values must be coded. E.g. “\0” -> %00, “\n” -> %10, "%" -> *%25* etc.

14. Logging means and information about the logs must be protected against unauthorized access, modification and destruction.

Structure of the Log File

15. All field name descriptions, parameter names and other information in the log must be in English.

16. Reccomended format:

16.1. Fieldss must be tab-separated;

16.2. One event should be on one line. In case of multiline log entry JSON format should be used for input parameters if possible;

16.3. JSON should be used for only such cases when the amount of input parameters may vary or one parameter can have multiple parameters.

Fields that always exist (event initiator, timestamp, event type etc) must be in regular format.

17. Multiline log entries should be clearly distinguishable from each other.
    17.1. Multiline log entries must be kept in a separate log file;
    17.2. Multiline log events must be distinguishable by a pattern or a special set of characters (e.g. the event always starts with a timestamp format and ends with an empty row or a predefined set of character sequence that is never printed into the log message, e.g. "------");
18. User input parameters must be distinguishable from the parameters given by the application itself;
19. Maximum length of the log event must be at least 10kB;
    19.1. The length of the log event must be defined in the beginning of the event to detect possible attacks related to the log event length;
20. If possible, all input parameters should be analyzed. All values over 256B should be written to a separate log file with a timestamp, unique event ID and hash of the value. The request log should contain the value length and hash for the value;

Minimum Requirements for the Log Event

21. Logged event must contein enough information to answer the questions who, what, where, from where and when and present the result of the request.
    21.1. **WHO** – initiator of the request:
        21.1.1. Must be unique atleast within the service;
        21.1.2. Must be connectible with one physical responsible person;
        21.1.3. Automatic processes must be clearly recognizable;
        21.1.4. All automated process and activities must have one responsible person set.
    21.2. **WHAT** – the type or class or the activity or a request (authorization, authentication, operation, usage etc) and details of the activity:
        21.2.1. Object or a component that was used;
        21.2.2. Method and input parameters;
        21.2.3. Request parameters, file names, request objects etc.

21.3. **WHERE** – the identificator of the system, its node or instance name to define in which application and in which instance the request was processed.

21.4. **WHENCE** – unique identificator of the device where the request originated (name, IP address, device certificate etc);

21.4.1. Device ID mush provide enough data to uniquely define the origin of the request;

21.4.2. In case of an IP aadress, it should be the endpoints publicly visible IP address.

21.5. **WHEN** – timestamp that defines the exact date and time of the event;

21.5.1. Server time must be accurate, server time must be synchronized with a centralized time server (*NTP*);

21.5.2. Timestamp must be atleast with the accuaracy of a second;

21.5.3. Timestamp must be in the UTC timezone;

21.5.4. Timestamp must be in a format of "from larger to smaller" and should be machine readable, e.g. ISO8601 format (e.g. YYYY-MM-DD**T**HH:mm:ss.SSSZ).

21.6. **RESULT** – the outcome and output of the request or activity

21.6.1. Atleast the response code, outcome type (success, attempt, failure, error) and the response size (in bytes and returned results) must be logged;

22. Log peculiarity (what is logged, how the log events are divided, log examples etc) must be defined in the service documentation.
23. Documentation for logging and log event examples must be created with the use case scenarios during the development process.
24. During development process, correct logging and its documentation must be developed together with the functionality of the application.

Appendix 4 - Log requirements (Estonian)

Logimise põhimõtted

1. Logimiseks tuleb kasutada standardseid komponente (nt java's *log4j*).
2. Logikirjed jaotatakse erinevatesse failidesse järgnevalt:
   2.1. **Seansilogi** - info kasutajate tuvastamise, rakendusse või kõrgema turvalisusega moodulisse/rakenduse osasse sisenemiste, väljumiste, seansi aegumise jmt kohta (*Success*, *Attempt*, *Failure*);
   2.2. **Tegevuslogi** - kogu informatsioon kasutajate tegevuste kohta koos tegevuse tüübi, seansi parameetrite (korreleerimaks seansi- ja tegevuslogi) ja kasutaja poolt esitatud sisendparameetritega (sh. väliste ressursside kasutamise kohta);
   2.3. **Silumislogi** - arendajate jaoks vajalik *debug* info. Toodangu keskkonnas peaks *debug* olema vaikimisi välja lülitatud;
   2.4. **Turvalogi** - turvalisusega seotud sündmused, mida jälgida (IP aadressi muutumine seansi käigus, keelatud kasutajanimede kasutamised, vigased autentimised jms.);
   2.5. **Vealogi** - erinevate veaolukordade info, mida võimalusel jaotada kaheks:
      2.5.1. **Tehniline vealogi** - süsteemsed veateated ja kasutaja tegevusest tulenenud käsitlemata vead (probleemid liidestega, süsteemsete taustatööde veateated jms.). Logi sisu võib sisaldada täispikkuses veateateid, *trace* sisu ja mud veaga seotud informatsiooni;
      2.5.2. **Kasutajate vealogi** - kasutajate tegevuse tõttu esile kutsutud funktsionaalsuse käsitletud vead, mille peale kuvatakse kasutajatele veateade (nt. katse faili avada ilma vastavat õigust omamata, suurema ülekande sooritamine kui kontojääk lubaks jne.).
3. Logimine peab olema optimeeritud. Informatsiooni dubleerimist logides tuleb vältida kui ei ole nõutud teisiti.
4. Kui ühe päringu tõttu tekib kirjeid mitmesse logisse, peab olema võimalik neid kirjeid ühise välja abil siduda. Selleks ei sobi kellaaeg, aga sobib näiteks unikaalne päringu id.

5. Sündmuste ajalise korrektsuse tagamiseks peab logivatel süsteemidel olema õige kuupäev ja kellaaeg. Logivate süsteemide kellasid tuleb ajaserveriga sünkroniseerida.
6. Kõik logid peavad olema vähemalt failipõhised, st logifailid peavad olema kasutatavad ilma rakenduse ja selle komponentideta (nt andmebaas). Kui teenuse juures on nõutav ka andmebaasis hoitav tegevuselogi, on see lisaks failipõhisele logile.
7. Logifailid peavad olema loetavad tekstilisel kujul (ASCII prinditavad sümbolid).
8. Logisid peab olema võimalik toodangusüsteemidest kätte saada kujul, et neid oleks võimalik analüüsida (säilitaks nii masintöötlemise kui inimloetavuse).
9. Võimalusel logida infosüsteemi funktsionaalsuse-, mitte moodulipõhiselt. Nt. autentimispäringud ühte, tegevuste logi teise faili, mitte iga rakenduse mooduli jaoks eraldi logifail, kuhu kirjutatakse kõik sellega seonduv.
10. Logimise õnnestumine on kohustuslik - kui sündmuse kohta ei logi kirjutada, tuleb toiming jätta teostamata ning vajadusel teenus sulgeda.
11. Kõik kasutajate tegevused peavad olema logitud.
    - 11.1. Kõik autentimise katsed (hoolimata tulemusest) peavad olema logitud, sh. katsed tühja(de) või puuduva(te) parameetritega.
    - 11.2. Administraatori tegevusest peab logisse jääma jälg.
    - 11.3. Administraator ei tohi logisid muuta, logisid kustutada ega logimist peatada.
    - 11.4. Logida tuleb nii õnnestunud kui ebaõnnestunud tegevused.
    - 11.5. Logida tuleb vähemalt need parameetrid, mida rakendus kasutas.
    - 11.6. Parameetrid ja nende väärtused, mis edastatakse HTTP kehaga, kasutades *content-type*'na *www-url-encoded* väärtust tuleb logida;
    - 11.7. Logides tuleb säilitada sama parameetrite järjekord, millisena nad edastati serverile või teenusele;
    - 11.8. Logida tuleb päringuid, mille vastus on puhverdatud;
    - 11.9. SSL/TLS kasutamisel tuleb logida ka SSL versioon ja kasutatud sihver (*cipher*);
    - 11.10. Kui parameetri väärtus on tühi, tuleb see logis märkida asendusväärtusega (nt. "*null*").

11.10.1. Kui kasutatakse asendusväärtust, peab see olema eristatav kasutaja poolt sisestatud väärtusest (nt. Kui kasutaja sisestab väärtuseks “null”, peab logisündmusest olema järeldatav, kas tegu on asenduväärtuse või kasutaja poolt sisestatud väärtusega).

12. Mitte kunagi ei logita:
    12.1. Kasutajate salasõnu tekstilisel kujul;
        12.1.1. Logida võib parooli pikkust (baitides).
    12.2. Privaatvõtmeid;
    12.3. Seansivõtme väärtust (nt seansi tokeneid või -küpsiseid);
        12.3.1. Logida võib seansivõtmest tuletatud räsi või muu pöördumatu tuletise.
    12.4. Andmebaasidest tagastatud päringute täisvastuseid tekstilisel kujul;
        12.4.1. Logida võib andmete tagastamise fakti ja/või vastuse pikkust, vea korral veateadet.
13. Kasutajate sisend-andmed tuleb enne logifaili kirjutamist kodeerida (st kogu kasutaja sisendit peab olema võimalik taastada, va 2.12 toodud andmed), välistamaks logisüste ja sellega seonduvaid ründeid.
    13.1. Logitavas sisendinformatsioonis tuleb kodeerida kõik non-*printable* sümbolid (0x00..0x1f, 0x7f..0xff) ja väljaeraldajad. Näiteks \0 -> %00 ja \n -> %10 ja "%" -> %25 jne.
14. Logimisvahendid ja informatsioon logi kohta peab olema kaitstud volitamata muudatuste, hävitamise ja juurdepääsu eest.

Logifaili struktuur

15. Kõik väljanimede kirjeldused, parameetrite nimetused ja muu informatsioon peab olema võimalusel inglise keeles.
16. Soovituslik formaat:
    16.1. Väljad on tabeldus-eraldusega (*tab-separated*);
    16.2. Üks sündmus ühel real, mitmerealise logi asemel kasutada võimalusel JSON formaati sündmuse sisendparameetrite tarbeks;
    16.3. JSON’i kasutades tuleks seda teha vaid selliste sisendparameetrite korral, millel on mitu väärtust või mille olemasolu võib varieeruda.

Väljad, mis on alati olemas (sündmuse algataja, kuupäev, sündmuse tüüp jmt), peaks olema tavaformaadis.

17. Mitmerealiste logikirjete (*multiline log entry*) puhul peab olema võimalik selgelt eristada sündmusi teine-teisest (nt. kasutada sisendandmete salvestamiseks JSON formaati).
    17.1. Mitmerealised logikirjed tuleb salvestada eraldi faili;
    17.2. Mitmerealised logikirjed peavad olema üksteisest eraldatavad mustri või erisümboli alusel (nt. sündmus algab alati õiges formaadis, lõppeb alati tühja rea või kindla sümbolite jadaga, mida logikirjas ei esine, nt. "------");
18. Sisendandmed peavad olema eristatavad rakenduselt endalt pärinevatest andmetest.
19. Logirea maksimaalne lubatud pikkus peab olema vähemalt 10kB.
    19.1. Logirea kogupikkus tuleb märkida logirea alguses, tuvastamaks võimalikke logirea maksimaalse pikkusega seotud ründeid;
20. Võimalusel tuleb logi kirjutades analüüsida sisendparameetreid, üle 256B parameetrite puhul tuleks parameetrid kirjutada eraldi faili (koos kellaaja, unikaalse päringu ID ning räsi väärtusega), logisse peab maha märkima parameetri pikkuse ning räsi.

Miinimumnõuded logikirjele

21. Logikirjes (sündmuses) peab sisalduma piisavalt informatsiooni, et vastata küsimustele kes, mida, kus, kust, millal, kuidas ja tulemus.
    21.1. **KES** - tegevuse teostaja:
        21.1.1. Peab olema unikaalne vähemalt teenuse piires;
        21.1.2. Peab olema seostatav füüsilise isikuga kui vähegi võimalik;
        21.1.3. Automaatprotsessid peavad olema selgelt tuvastatavad;
        21.1.4. Automatiseeritud tegevuste kasutajatel peab olema isikuline vastutaja.
    21.2. **MIDA** - tegevuse/sündmuse liik või klass (kasutaja tuvastamine, administreerimine, operatsioon, kasutus) ning tegevuse detailid:
        21.2.1. Objekt või komponent, mida kasutati;
        21.2.2. Meetod ja sisend-andmed;

21.2.3. Tegevuse andmed, failide nimed, päringu objektid.

21.3. **KUS** - infosüsteemi identifikaator, mille abil on võimalik kindlaks teha täpne rakendus ja selle instants, mille suhtes tegevus teostati.

21.4. **KUST** - seadme unikaalne identifikaator (nimi, IP aadress, seadme sertifikaat), kust tegevus toime pandi:

21.4.1. Identifikaatori abil peab olema võimalik üheselt tuvastada seade, kust sündmus toime pandi;

21.4.2. IP aadressi puhul peab olema tuvastatav kliendi lõpp-seadme avalik IP aadress.

21.5. **MILLAL** - ajamärgistus, mis sisaldab täpset sündmuse kuupäeva ning kellaaega;

21.5.1. Serverite kellaaeg peab olema õige, serverite kellaajad peavad olema omavahel sünkroniseeritud (NTP);

21.5.2. Aeg peab olema vähemalt sekundi täpsusega;

21.5.3. Aeg peab olema UTC ajavööndis;

21.5.4. Ajaformaat peab olema formaadis "suuremast väiksemaks" ning masinloetaval kujul, nt ISO8601 formaadis (nt YYYY-MM-DD**T**HH:mm:ss.SSSZ).

21.6. **TULEMUS** – teostatud tegevuse väljund või tegevuse tulemus.

21.6.1. Kui tulemust ei ole mõistlik täies mahus maha salvestada, tuleb logisse kirjutada vastuse kood, tulemuse tüüp (*success, attempt, failure, error*) ning vastuse suurus (nii baitides kui ridade arvuna);

22. Logide spetsiifika (mida logitakse, kuidas sündmused logifailidesse on jagatud, logiridade näited) peab olema kirjeldatud teenuse dokumentatsioonis.
23. Rakenduse funktsionaalse kirjeldusega tuleb luuga ka logimise dokumenatsioon ja loginäidised.
24. Koos funktsionaalsuse arendamisega tuleb paralleelselt luua ka loodava funktsionaalsuse logimine ja selle dokumentatsioon.