UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Karin Klooster

Applying a Security Testing Methodology: a Case Study

Bachelor's Thesis (9 ECTS)

Supervisor: Meelis Roos, MSc
Supervisor: Margus Freudenthal, PhD

Tartu 2016

Applying a security testing methodology: a case study

Abstract: Security testing is a software testing discipline that aims to verify that the functionality of the software is resistant to attacks and data processed by the software is protected. To establish common requirements that the software must fulfill, software security standards are published. This thesis aims to describe and apply a process necessary to verify the security of a web application. A checklist of security requirements was gathered combining OWASP ASVS web application security standard and OWASP Top Ten project. Test cases were developed and web application UXP Portal was tested to verify the security requirements in the checklist. Numerous security vulnerabilities were identified by security testing. The recommendations based on lessons learned during the case study were presented.

Keywords: web application, security testing

CERCS code and name: P175 Informatics, systems theory

Turvatestimise metoodika rakendamine: juhtumiuuring

Lühikokkuvõte: Turvatestimine on tarkvara testimise haru, mille eesmärgiks on kontrollida, kas tarkvara on haavatav rünnete suhtes ning kas andmed, mida tarkvara töötleb, on kaitstud. Tarkvara turvalisuse standardeid töötatakse välja selleks, et tekitada ühine arusaam turvanõuetest, mida turvaline tarkvara peab täitma. Selles bakalaureusetöös kirjeldatakse ja rakendatakse tegevused, mis on vajalikud veebirakenduse turvalisuse kindlaks tegemiseks. Kombineerides OWASP ASVS veebirakenduste turvastandardit ja OWASP Top 10 riskide nimekirja, töötati välja turvanõuete nimekiri. Turvanõuete testimiseks töötati välja testjuhtumid ning testiti veebirakendust UXP Portal. Turvatestimise tulemusena tuvastati arvukalt turvaprobleeme. Juhtumiuuringu läbiviimise kogemuse põhjal vormistati õpitust lähtuvad soovitused.

Võtmesõnad: veebirakendus, turvatestimine

CERCS kood ja nimetus: P175 Informaatika, süsteemiteooria

Contents

1 Introduction

Software applications are integrated in almost all areas of life — business, medicine, social media, transportation, energy supply, home appliances and warfare, to name a few. Due to the fact that software plays a major role in maintaining critical infrastructure and processing sensitive data, software security is of paramount importance. On the other hand, according to the Global State of Information Security Survey 38% more cyber security incidents were detected in 2015 compared to 2014 and the theft of intellectual property increased by 56% [1]. In software development industry the ability to prove to clients that the developed software is secure, is a precondition for success.

Security testing is a software testing discipline that aims to verify that the functionality of the software is resistant to attacks and data processed by the software is protected. Similarly to other areas of software testing, a security tester can never test every possible attack scenario. To establish common requirements that the software has to fulfill, software security specialists community and governmental institutions have published security standards such as OWASP Application Security Verification Standard, CMU CERT secure coding standards, Penetration Testing Execution Standard, Common Criteria and NIST standards. Security requirements that make up a security standard give security tester a rough overview what has to be tested, but the tester still has to analyze what the requirements mean in the context of a specific software and develop a set of test cases to verify the requirements.

The starting point for a problem that this thesis aims to solve is the one of a software company that has developed a new web application and faces two challenges — firstly, how to verify that the software is secure and secondly, how to reassure clients that the software is secure. This thesis aims to describe and apply a process necessary to verify the security of a web application. The subject of testing is Unified eXchange Platform (UXP) Portal — a web application developed by Cybernetica AS [2]. The UXP Portal is a web application that allows users to make web service requests through their browser. UXP Portal can be used to quickly deploy end-user access points to UXP infrastructure supporting role-based user access control.

There are two types of software security standards — the standards developed by a community of software security specialists and the standards developed by governmental or international institutions. To achieve the aim, different software security standards are reviewed and compared to find the security standard most suitable for the application under test. After that, the security requirements of the chosen standard are analyzed in context of the UXP Portal to make sure which requirements are applicable to the application under test. Then the test cases to verify the security requirements are developed and UXP Portal is tested. Lastly the results of testing and recommendations to anyone facing a similar problem based on lessons learned during the process are given.

The first chapter aims to describe what web application security means, which software security standards exist, which security testing techniques can be applied to verify software security and gives an overview about related work. The second chapter describes the application under test and the security testing process: security requirements analysis, the development of test cases and automatic security testing tools used. The third chapter presents the results from the security testing and recommendations based on lessons learned during the process.

2 Security testing

2.1 The concept of web application security

Secure software is usually defined by uncompromisable integrity, confidentiality and availability of application's data and services [3]. Software security means that software is developed in a way that it remains secure even under a malicious attack [4]. Software security depends on four factors: the data stored in the application and the functionality being offered by the application, the motivation of potential attackers and the cost of ensuring security [5]. Ensuring software security has an important role in software risk management.

Risk analysis is typically utilized in software design phase to identify security threats and their impact [5]. After potential risks have been identified and classified according to probability and impact, these can be used to steer security testing. Risk analysis provides a base for deciding which test cases must be created and executed [6]. Security testing should aim to test the areas of the system that contain the biggest risks. Risk analysis results can be used for repairing security faults as well as communicating risks that are not yet addressed to stakeholders to create trust [6]. Having described how risk analysis can be useful, Dukes et al. also bring out three major problems with software risk analysis.

1. Risk analysis is a complicated and expensive task because it needs to be carried out by experts of the field.
2. Risk assessment is subjective and therefore unreliable.
3. Methodologies for integrating risk analysis in security testing are still in developmental phase.

In case the functionality supplied by the application is not critical and application does not collect any data that would be of interest to potential attackers, it is not reasonable to spend vast resources on protecting the application. Of course one must take into account that many attackers do not behave rationally and may target random web applications, so a basic level of security is necessary in any case. Potter and McGraw accentuate that understanding attacker's way of thinking is important for anyone measuring or testing software security [4].

Potter and McGraw suggest that there are two types of problems that can obstruct software functions and cause security vulnerabilities: bugs in implementation and faults in software design. The faults in software design are harder to discover and more widespread than implementation bugs [4]. Hence software security testing should begin already in the software design phase.

OWASP Testing Guide [3] also emphasizes that software security cannot be achieved by only one measure, but rather various security assurance methods should be applied in all phases of software development life cycle (SDLC) from requirements engineering and code reviews to penetration testing. There exist situations where this approach is not applicable, for example when the application has already been developed. In this case it is not possible to go back in time and apply all security assurance methods that should have been applied during requirements engineering and development. In worst case security testers don't even have access to source code, but penetration testing is still a useful technique [3].

In real life software development world there are usually time and (human) resource boundaries that limit the amount of security assurance measures that can be applied during software development process. Security is not a priority for all software developers. In many cases software developers may also have a mindset that it is most important to develop the functionality of the software first and security features can be added later. According to Potter and McGraw the main problem with perception of software security is that many security measures are reactive instead of being proactive and security is perceived as a product that protects software while the real problem is the design and implementation of the software itself [4].

In conclusion, software security is a compromise on which risks are dangerous enough to be worth fighting against and how much resources can be allocated for ensuring security. There is no one measure that ensures software security, different security measures should be applied during the whole software development life cycle.

2.2 Software security standards

To establish mutual criteria a secure application must comply with, software security standards have been developed. In software development industry software security standards are used as a selling point. Verified compliance with a software security standard creates trust among potential clients and gives a competitive edge when software is compared to similar software systems. Compliance to software security standards is often used as a default requirement during software development procurement or as a precondition for using the system in certain sectors (for example electronic banking). Next an overview of various software security standards is given.

Open Web Application Security Project (OWASP)[1] is a non-profit organization aiming to improve software security. OWASP has organized a worldwide community of software security professionals to publish free materials on software security. The OWASP Application Security Verification Standard (ASVS) is a checklist of security requirements that application designers, developers and testers can use to verify software security. The main objectives of OWASP ASVS are providing a metric to base contracts on and assess software security as well as providing direction for software developers.

ASVS security verification requirements are classified as three security levels: to pass level 3 verification, 179 requirements have to be verified, 146 for level 2 and 86 for level 1. Level 1 is suitable for all web applications and contains a minimum set of requirements, level 2 grants protection against most typical security risks for applications that contain sensitive data and level 3 provides solid protection for critical applications [7].

A positive side of OWASP ASVS is that in addition to the standard, OWASP also provides other materials to help apply the standard. OWASP Testing Guide [3] provides detailed instructions on how to test many of the requirements found in OWASP ASVS. Another OWASP project that has received a lot of attention is OWASP Top Ten that aims to order the 10 most critical software vulnerabilities [8].

OWASP wiki pages also contain much useful material for users of OWASP ASVS like OWASP Cheat Sheets Series that gather detailed information about specific software security matters [9]. OWASP materials on software security are of high value because these are constantly renewed by the community of software security specialists. Software security is a subject field that develops very rapidly and obsolete information may prove to be hazardous while maintaining software security.

CERT division of Carnegie Mellon University's Software Engineering Institute organizes community effort to develop secure coding standards for C, C++, Java, Perl, and the Android platform. The standards are published as free wiki pages [10].

Penetration Testing Execution Standard (PTES) is a penetration testing standard published as a website that aims to establish a common understanding about which activities quality penetration testing consists of. This standard also aims to standardize

[1]https://www.owasp.org/

penetration testing reporting. PTES technical guidelines describe detailed exploits to try and apply during penetration testing [11].

Besides community efforts to establish software security standards, there are also governmental and international organizations that are actively developing software security standards. US National Institute of Standard and Technology (NIST) produces numerous standards and guidelines in the field of software security that are mandatory for federal information systems and government contractors. Besides standards NIST also has a Special Publications (SP) 1800 subseries of NIST Cybersecurity Practice Guides that consists of practical guides on how to apply cybersecurity standards [12].

Common Criteria (CC) is an international agreement that provides a framework for assuring that software products have been specified, implemented and tested in a way that ensures security. CC uses licensed laboratories to evaluate software product's correspondence to the criteria [13]. CC has been typically used to evaluate components like firewalls, routers, switches and smart cards rather than web applications [14]. According to Jackson [15], CC is generic and does not provide concrete security requirements. Due to the fact that validation is only provided by licensed laboratories, it is very expensive. What's more, the evaluation depends more on the software specification documents written rather than the security of the software product itself [15].

The software security standard to be used as a source of security requirements for UXP Portal should fulfill the following requirements:

1. Using the standard is free of charge.
2. The standard is web application specific.
3. The standard is accompanied by materials that provide guidance on how to verify compliance with the standard.
4. The standard is up to date.

When comparing the software security standards developed by a community of software security specialists and the standards developed by governmental or international institutions it is clear that the negative side of software security standards that are developed by community effort is that these may contain errors or be incomplete. CMU CERT secure coding standards and PTES technical guidelines contain many incomplete pages. Complete standards are preferred and these standards are not used in this case study. Contrarily, OWASP Application Security Verification Standard and OWASP Testing Guide do not contain any incomplete sections.

NIST standards from SP 1800 subseries of practice guides contain very detailed recommendations about one specific software security aspect (for example “Securing Electronic Health Records on Mobile Devices” or “Identity and Access Management for Electric Utilities”), but there is no practice guide that would cover all aspects of web application security yet [12]. For these reasons, NIST is not used as a basis for security testing in this thesis as it is not web application specific. CC is also not suitable as a base for deriving security test cases as CC is not specifically developed for web applications and validation is not free of charge.

It can be concluded that OWASP ASVS (most recent version is 3.0) is the most suitable software security standard to be used as a source for security requirements for UXP Portal, as it is specifically developed to be used with web applications, using the standard is free of charge and besides the standard there are various other resources like OWASP Testing Guide and OWASP Cheat Sheet series that provide guidance on how to verify compliance with the standard. What is more, OWASP ASVS 3.0 was published in October 2015 and therefore one can be certain that it contains up-to-date security requirements.

2.3 Security testing techniques

Security testing techniques can be classified as black box testing and white box testing. Black-box security testing techniques involve trying to analyze and find vulnerabilities in running software by manipulating the input without having any knowledge about the source code. White-box security testing techniques aim to analyze the source code and application architecture from security assurance viewpoint. Next a more detailed overview of the following security testing techniques is given: manual inspection, threat modeling, code review and penetration testing.

According to OWASP Testing Guide manual inspection is a white box security testing technique that involves analyzing documentation and interviewing software designers to inspect the decisions made in the software design process from the security viewpoint. The purpose of manual inspections is to test the software development process rather than test the application itself. Usually manual inspection is used early in the SDLC and during such reviews the documents reviewed include architectural documents, security requirements and secure coding policies [3]. Manual inspection assumes that there are various documents to inspect — in case of small-scale software development projects the volume of documents produced before starting development might be scarce.

Threat modeling is another white box security testing technique that is used early in the SDLC. According to Steven [16] threat modeling is a process that starts with acknowledging possible threats to the web application. Next, risk probabilities and their possible impacts on the system are assessed [16]. After probable risks have been found the strategies for risk mitigation can be developed. Threat modeling is a software risk assessment method that can provide necessary input in penetration testing — it must be verified that the threats detected are successfully avoided.

Source code review is a white box security testing technique that involves reviewing the application source code for possible security vulnerabilities. According to OWASP Testing Guide code review can often reveal critical security problems that would not be discovered by using black-box security testing techniques. In order to be effective and fast, security code review requires very competent security oriented code reviewers [3].

Penetration testing is a security testing technique that is applied in the latest phase of SDLC and in many cases used as a synonym to security testing. Penetration testing means that tester is trying to penetrate the system by fingerprinting it and probing it with invalid input. A penetration tester acts like a hacker with a purpose of attacking the system, causing the system to stop working, viewing the data contained in the system or performing other activities beneficial to the attacker and unwanted for system owner. According to OWASP Testing Guide penetration testing historically originates from network security testing [3].

There are areas of penetration testing that can be automated and there has been a rapid development in the field of applications performing automated penetration testing. OWASP Testing Guide cautions testers about using automatic tools, insisting that although an automatic scanner can find many vulnerabilities, it cannot be considered the one and only necessary testing tool and manual testing cannot be skipped. OWASP emphasizes that "security is a process and not a product" [3]. Dukes et al. [6] also conclude from the web application security testing case study conducted that as there are vulnerabilities that can only be discovered by manual testing, it is paramount to test manually in addition to using scanners and automatic tools.

In conclusion, the white box security testing techniques used early in the SDLC include manual inspection and threat modeling. Security oriented code review and penetration testing are used later in the SDLC. Penetration testing can be performed manually as well as by using automated tools.

2.4 Security testing case studies

This section gives an overview about similar works and their results. Numerous researchers have conducted software security testing case studies. Potter and McGraw [4] emphasize that security testing case studies are very useful tools for novice security testers for understanding how security testing should be approached.

Potter and McGraw [4] conducted a case study on Java card security testing where risk analysis is used as a base for deriving test cases. Similarly, Apvrille and Pourzandi [17] present a case study on instant messenger security testing where security requirements are derived from a threat model developed. Authors come to the conclusion that code review, despite being very labour-intensive and dependent on the reviewer, proved to be the best technique for security testing. On the other hand, the amount of vulnerabilities found by automatic scanner was limited [17]. Contrarily, Tóth et al. [18] conclude their security testing case study with a certainty that it is not feasible to test complex software systems manually and automated tools are the only option for testing complex systems efficiently.

Wang et al. [19] describe a threat model driven security testing case study that suggests that modeling threat scenarios with UML sequence diagrams would make security testing more comprehensible for IT industry as UML diagrams are widely used in the software development process. Similarly, Jürjens [20] uses a threat model for deriving security testing test cases and uses UMLsec, an UML extension, to model threats and generate test sequences for testing the Common Electronic Purse Specifications. It is concluded that the systematic approach of UMLsec modeling prevents security flaws arising from poor implementation.

Dukes et al. [6] describe a case study about web application security testing. The authors conclude that manual testing is important because it can discover the vulnerabilities that automatic tools cannot find. OWASP Top Ten is used as the list of potential web application security issues that should be tested. Vibhandik and Bose [21] also use OWASP Top Ten as a reference to decide what should be tested in their web application security testing case study. It is concluded that automatic tools are useful, but a few different automatic tools should be used to discover more vulnerabilities as different automatic tools are capable of discovering different vulnerabilities.

Similarly, Acharya et al. [22] utilize OWASP Top Ten Mobile Risks to develop a checklist of risks that should be tested when verifying the security of mobile healthcare applications tested in the case study. It is decided that OWASP Top Ten Mobile Risks framework proves to be very useful when used by mobile app developers. Knorr and Aspinall [23] also conduct a case study about mobile health apps security testing, but use a custom classification of attacker kinds as a basis for deriving test cases. The main conclusion of the case study is that security testing is very expensive because of the vast amount of manual testing required [23].

From previous work in the field of security testing case studies it can be concluded that there are two groups of web application security testing case studies. The first group of case studies uses threat modeling or risk analysis to derive security requirements to test. The other approach is using OWASP Top Ten (Mobile Risks) as a reference to what should be tested.

The case studies reviewed present very different views on which security testing techniques are the most useful. UML modeling, manual testing, code reviews and automatic scanning are all described as useful techniques. Negative sides of manual testing, code reviews and automatic scanning are also discussed. It can be concluded that the choice of security testing methods to be used for getting the best results depends heavily on the application under test and the resource limits imposed on the security testing.

OWASP ASVS and OWASP Top Ten were chosen to be used to develop a checklist of security requirements to be tested in this case study. Carrying out a risk analysis to find the security requirements that should be tested would have required more resources than were available for this case study.

3 Description of the security testing process

3.1 Application under test

3.1.1 Overview of UXP

The Unified eXchange Platform (UXP) is a technology developed by Cybernetica AS that is used by organizations to provide services to each other [2]. UXP offers a standardized communication channel that provides confidentiality, strong authentication and long-term proof value of the relayed messages. In order to ensure consistently high quality and high degree of interoperability, all the UXP members use standardized security components, called security servers.

In the UXP system, members communicate directly without intermediaries. As the exchanges usually contain personal information and are subject to regulation (for example, transmitting medical information is highly regulated), the security requirements for this kind of communication are high. All the messages are signed and time-stamped and sent over encrypted and mutually authenticated channel. Figure 1 gives an overview of the system components in UXP installation.

An UXP installation consists of the following components.

- UXP Central Server maintains information about approved certificate authorities, approved trust services, UXP members and security servers. This information is distributed to the security servers.
- UXP Central Monitoring Server receives monitoring information from the security servers and makes it available to central system administrators.
- UXP Management Security Server receives management requests from the security servers and forwards them to the UXP central server.
- UXP Security Server acts as a gateway between the organization's information system and the UXP infrastructure. Security server relays request and response messages while adding a protective layer. All the messages are passed through cryptographically secure channel. Additionally, the messages are digitally signed and time-stamped to ensure long-term proof value of the transactions.

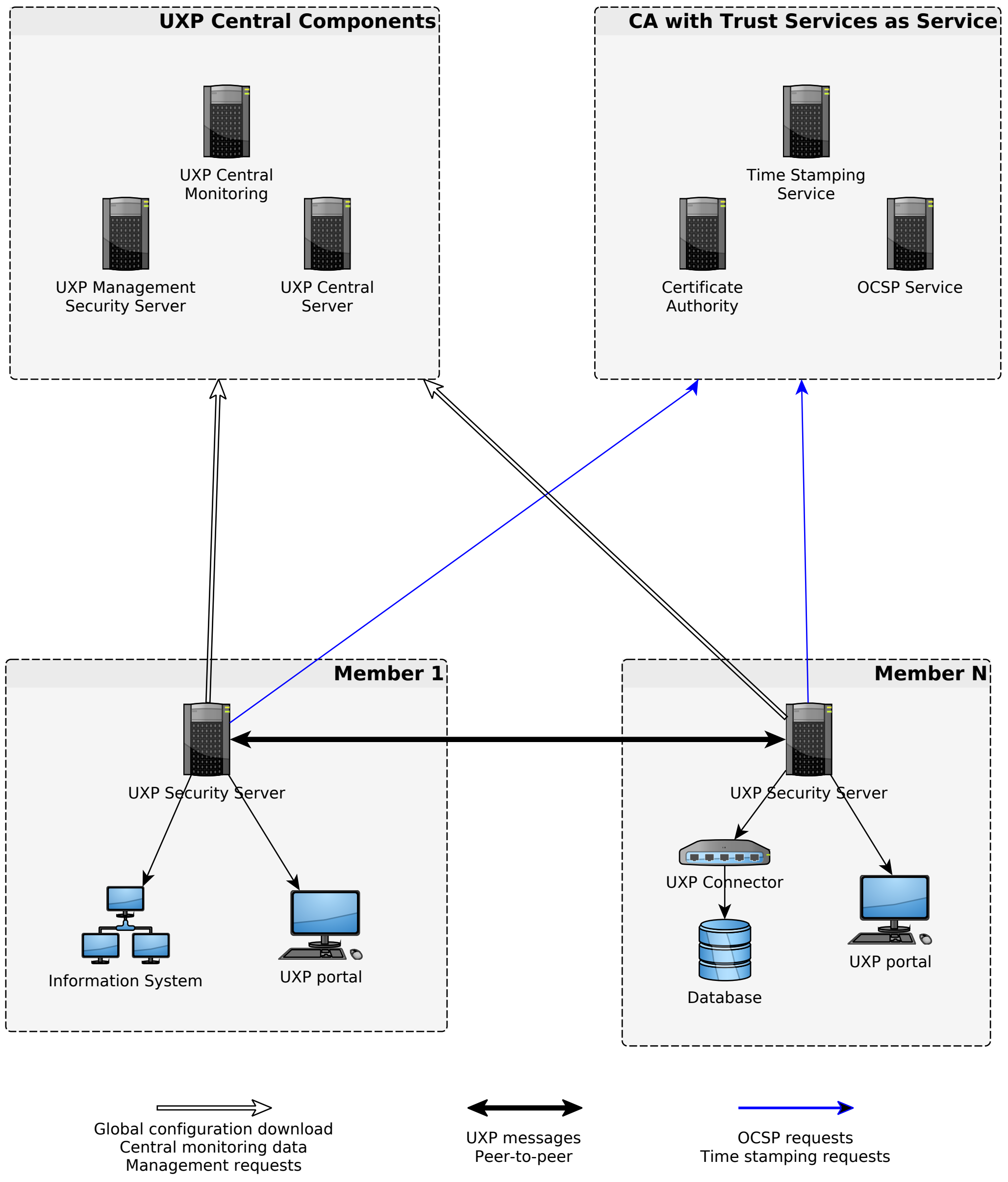

Figure 1: System components of an UXP installation

- UXP Connector can be installed at an organization to implement UXP services based on SQL databases.

- UXP Portal is a web-based application which allows UXP service discovery through UXP platform and creates dynamic query/input forms for service consuming.
- Certificate Authority (CA) creates certificates that prove ownership of public keys.
- Online Certificate Status Protocol service distributes certificate validity information to security servers.
- Time-stamping service issues time stamps that ensure long-term validity of the digital signatures created in the UXP system.

UXP members communicate directly with approved trust service providers. When joining the communication infrastructure, each member must acquire a certificate for signing the messages and another certificate for transport security. The member must also select a time-stamping service provider that will be used to provide long-term security to the exchanged messages. During communication, the member interacts with service providers to acquire certificate validity information and to time-stamp signed messages. In UXP, the service provider maintains and enforces access control list for each service.

Before using a service, the service client and service provider enter into an agreement that specifies liabilities of both parties. From one side, the service provider agrees to provide service with given Service Level Agreement. From the other side, the service client agrees to use the service and process the received data according to conditions defined by the service provider.

3.1.2 Overview of UXP Portal

UXP Portal is a web application that allows users to make UXP web service requests through their browser. The UXP Portal is a part of the UXP platform that is connected to the service consumer's security server and uses this security server to retrieve information about the UXP instance it is part of. This information includes registered UXP members, lists of services provided by specific UXP members and service descriptions for services that are being registered. UXP Portal can be used to quickly deploy end-user access points to UXP infrastructure supporting role-based user access control and several authentication methods. UXP portal supports role-based user authorization, there are three roles that can use different functionality.

Portal administrator can add organizations that can use the UXP Portal. Organizations can be chosen from a list of UXP members registered at the security server that the UXP Portal is connected to. Organization management component is accessible only to portal administrator. All users, roles and permissions (except for portal administrator) are always related to some organization and are available only in the context of that organization. Organizations are used by portal administrator to separate responsibilities for management of organization employees and available services to organization administrators.

User administrators can use the user and group management functionality:

- Create and remove user accounts.
- Add users to groups. User groups are used to assign groups of users to services these users are allowed to use.
- Assign roles (service administrator and user administrator) to the users. Limited functionality (registration of user administrators and service administrators) is available also to portal administrator.

Service administrators can use the service management functionality:

- Register and remove service providers for the organization. A service provider is a UXP member that offers services to other UXP members via UXP platform.
- Update the list of available services. The available services are services that the service provider has made available to the organization using the UXP Portal. The list of services is retrieved from the Security Server that the UXP Portal instance is connected to.
- Register, edit and remove registered services. Services can be chosen from the available services pool of the registered service providers.
- Classify services into service categories to group related services together.

Portal users can submit UXP service requests to the services they have been given access to.

3.2 Security requirements analysis and testing process

To determine a checklist of security requirements to be verified for UXP Portal, it had to be decided which level of OWASP ASVS standard should be used. As UXP Portal is not a critical application, but contains sensitive data, level 2 would be suitable as level 2 grants protection against all most typical security risks. Level 2 consists of 146 security requirements to be verified. All 146 requirements were analyzed to find out how many of these requirements are applicable to UXP Portal. It was concluded that 113 requirements are applicable. It was estimated that in case verification of one requirement would take one day, it would take over 22 working weeks to verify 113 requirements.

This volume of work would be excessive in context of this thesis and so it was decided that OWASP ASVS level 1 is used. Level 1 consists of 86 requirements of which 67 are applicable to UXP Portal. Level 1 contains a minimum set of requirements for a web application. If verification of one requirement would take one day, verifying 67 requirements would take over 13 working weeks which still exceeds the volume of a thesis. To further reduce the number of requirements it was decided to combine OWASP Top Ten with OWASP ASVS level 1 requirements and verify the requirements that are part of level 1 as well as OWASP Top Ten.

OWASP does not provide a link between OWASP ASVS requirements and OWASP Top Ten risks. The materials of OWASP Top Ten project were analyzed to determine which level 1 requirements must be fulfilled to mitigate OWASP Top Ten risks. A mapping between OWASP Top Ten 2013 risks and OWASP ASVS 3.0 level 1 requirements was created (see appendix 6.1). It was concluded that there are 62 security requirements that belong to OWASP ASVS level 1 as well as OWASP Top Ten according to the mapping.

The next step after determining the list of security requirements to be verified was analyzing which requirements are applicable to UXP Portal and which requirements are not. 6 security requirements were found not to be applicable in UXP Portal context. These security requirements and the reasons why these are not applicable are shown in table 1.

OWASP ASVS 3.0 security requirement [7]	**Reason**
2.17 Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user	No password recovery functions
2.22 Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism	No password recovery functions
2.24 Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application	Knowledge based questions are not used
5.11 Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.	LDAP is not used
5.22 Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task.	No WYSIWYG editor
18.7 Verify that the REST service is protected from Cross-Site Request Forgery	REST web service is not provided

Table 1: The security requirements that are not applicable to UXP Portal

The remaining list of 56 security requirements was analyzed to find the requirements that are not satisfied because the corresponding functionality has not been implemented in UXP Portal. According to the analysis results there are 6 such OWASP ASVS requirements [7]:

1. *2.20 Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as brute force attacks or denial of service attacks.*

2. *2.27 Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases.*

3. *3.16 Verify that the application limits the number of active concurrent sessions.*
4. *3.17 Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session.*
5. *3.18 Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.*
6. *10.1 Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid.*

The analysis of security requirements showed that when one wants to verify the OWASP level 1 requirements that need to be verified to mitigate OWASP Top Ten risks, 62 security requirements have to be verified. There are 6 requirements that are not applicable to UXP Portal and 6 requirements that are certainly not satisfied due to not being implemented. There are 50 requirements that have to be verified by security testing techniques.

Before security testing, information about the architecture of UXP Portal that would be useful during security testing was gathered. Information about system components and interfaces, inputs and outputs, pages and access rights, the location of the logs was gathered.

After establishing a list of security requirements that needed to be verified by security testing, each OWASP Top Ten risk was assigned an amount of time available for security testing considering the number and contents of security requirements in each risk category. After that the test cases were developed to test UXP Portal. Code review and penetration testing techniques where used to test the UXP Portal. Manual penetration testing as well as automated tools were used.

OWASP Testing Guide and OWASP Cheat Sheets were used as a primary reference while choosing the testing techniques and developing the security testing test cases. All security tests except the code reviews were carried out by the author of this thesis. The author decided which requirements needed a code review but did not review the code herself because the author does not have necessary competence and experience in software development field.

After testing a test result was reported about every requirement tested. In case of failed test cases, the risks threatening the UXP Portal due to a failed test were explained and the recommendations to mitigate the vulnerability were proposed. Links to additional information on the subject were also provided as it is very important for the developers who are going to fix the security problems to acquire a clear understanding what and why has to be fixed to mitigate the security risks. See appendix 6.2 for an example of test cases and test results for verifying OWASP ASVS requirement 3.6.

3.3 Automatic security testing tools

This section gives an overview about automatic security testing tools that were used for testing UXP Portal which are: OWASP Zed Attack Proxy, SSLScan and Firefox browser add-ons.

OWASP Zed Attack Proxy

OWASP Zed Attack Proxy (ZAP) is a very popular security testing tool that can be used to assist a tester in manual security testing as well as for automatic scanning. OWASP ZAP is free and open-source software that is developed by volunteers. Due to being open-source software OWASP ZAP has many add-ons that are developed separately and allow more thorough testing of different subjects. ZAP main functionality includes intercepting proxy, spiders, various scanners, fuzzer, forced browsing, authentication and session support and REST API [24]. What is more, as ZAP is a very popular security testing tool, there is a very active big community of ZAP users that provides support and discussion forums for ZAP users.

OWASP ZAP was used as an intercepting proxy server to intercept and modify HTTP requests. OWASP ZAP scanning possibilities were used to test SQL injection, directory browsing, URL rewriting, session fixation, various HTTP response headers, cookie attributes, cross-site request forgery and application error disclosure vulnerabilities. Fuzzing was used to test potential reflected cross site scripting vulnerabilities.

SSLScan

SSLScan is a scanner for testing TLS settings [25]. SSLScan was used to test supported cipher suites, secure session renegotiation support, Heartbleed vulnerability and TLS certificate attributes.

Browser add-ons

Firefox browser add-ons were used during security testing. Cookies Manager+ [26] was used to view, edit and create cookies while testing session management. Firebug [27] was used to inspect HTML and monitor network activity. Groundspeed [28] was used to view the HTML form fields.

4 Results from security testing

4.1 UXP Portal security testing results

Security testing of UXP Portal started with a checklist of 62 security requirements. 6 of those requirements were not applicable to UXP Portal as the requirements targeted the components UXP Portal does not contain. 26 requirements were found to be satisfied, 27 requirements were not satisfied and the verification of 3 requirements was unclear at the time of writing this thesis as the code review that was part of the verification process for these requirements was not completed yet (see table 2). The six requirements that targeted the functionality that was not implemented in UXP Portal belong to "not satisfied" category. See appendix 6.1 for detailed results.

OWASP Top Ten 2013 risk [8]	Number of requirements	Satisfied	Not satisfied	Not applicable	Not ready
A1-Injection	9	5	3	1	
A2-Broken Authentication and Session Management	27	11	11	3	2
A3-Cross-Site Scripting	4		3	1	
A4-Insecure Direct Object References	4	3	1		
A5-Security Misconfiguration	3		3		
A6-Sensitive Data Exposure	10	6	4		
A7-Missing Function Level Access Control	1				1
A8-Cross-Site Request Forgery	2		1	1	
A9-Using Components with Known Vulnerabilities	1		1		
A10-Unvalidated Redirects and Forwards	1	1			
Total	62	26	27	6	3

Table 2: UXP Portal security testing results

Security vulnerabilities discovered by security testing are presented according to OWASP Top Ten risk categories. A1-Injection risk is not fully mitigated in UXP Portal as it is susceptible to XML external entity attack. What is more, HTTP responses do not contain anti-sniffing headers and some HTTP responses do not contain a type header specifying a safe character set.

There are various unaddressed risks in the category A2-Broken Authentication and Session Management. In addition to 5 security requirements that are not verified due to corresponding functionality not being implemented (see section 3.2), there are also other unaddressed risks. Firstly, it is possible to view the directory listing and some UXP Portal resources without authentication. Secondly, administration manuals are accessible for non-administrative users. In addition, session ID is enclosed in URL, session ID is not changed during logging in and session ID can be sent in HTTP request. There are also shortcomings concerning cookie attributes — cookie paths are set too loosely and cookie attributes “HttpOnly” and “secure” are not set. Lastly, the log out button is not directly visible in UXP Portal pages.

The requirements necessary to protect against A3-Cross-Site Scripting risk were not satisfied. It is possible to execute stored arbitrary HTML code and upload a HTML file in place of a TLS key store. Furthermore, HTTP responses do not set Content-Security-Policy headers and X-XSS-Protection headers. The risk A4-Insecure Direct Object References is not fully mitigated due to possibility to view the directory listing.

There are unaddressed risks in the category A5-Security Misconfiguration in UXP Portal. Firstly, UXP Portal outputs stack traces that reveal sensitive information about application’s structure and software used to develop it. Error pages and HTTP responses also contain running web server type and version information. UXP Portal server also accepts other HTTP methods besides GET and POST.

The risk A6-Sensitive Data Exposure is not fully mitigated because autocomplete attribute is not disabled on login pages and anti-caching HTTP response headers could be stricter. UXP Portal is not protected from the risk A8-Cross-Site Request Forgery as anti-CSRF tokens are not used in UXP Portal’s HTML submission forms. There are also vulnerabilities in the risk category A9-Using Components with Known Vulnerabilities, as the security of nginx, Jetty and PostgreSQL configuration could be improved.

In conclusion, almost half of the requirements in the security requirements checklist that were tested were not satisfied in case of UXP Portal. Among the vulnerabilities discovered there are no critical vulnerabilities, but some medium risk and many low risk vulnerabilities that should nevertheless be taken seriously as vulnerabilities can be combined by a skilled attacker to craft an attack.

4.2 Lessons learned from the security testing process

In this section recommendations to anyone facing a challenge of verifying their software is secure are presented. The recommendations are based on lessons learned from the security testing case study.

In this case study a security testing methodology that combines OWASP ASVS and OWASP Top Ten to determine a checklist of security requirements to be verified is used. After forming the checklist a time-box for testing the requirements in each OWASP Top Ten risk category was allocated. It can be concluded that this methodology is suitable for getting an initial overview of web application's security. It is also suitable for projects where security of the web application must be verified but the time resources available for doing it are scarce.

The case study made clear that after determining the list of security requirements that must be verified it is a good idea to work closely with the application developers to quickly determine which requirements are not applicable to the application under test due to respective components missing in the application. Similarly, it can be determined which requirements are surely not satisfied because the functionality required is not implemented. These steps can considerably lower the number of requirements that must be verified by testing.

It is also advisable to gather the information about application's components, inputs and outputs, list of pages and which user roles should have access to which pages and where are the logs written before starting security testing. This kind of information about application's architecture will surely be necessary during security testing and it saves time to have the information ready in one place.

Experience from the case study shows that when allocating time for testing the security requirements in OWASP ASVS, it must be taken into account that the volume of testing required to verify one requirement can vary considerably. While there are requirements that can be verified by just checking one attribute, there are also more generic requirements that require multiple test cases and extensive manual testing. What is more, when testers are not familiar with using automatic tools, a separate time must be allocated for getting to know the automatic security testing tools as well.

Analysis of the security testing results shows that OWASP ZAP tool can be used to obtain information about a wide variety of security vulnerabilities. Automatic scanner is especially efficient when the objective is to scan a large amount of input, for example scanning all HTTP responses. On the other hand, findings of automatic scanner can have false positives and should be manually verified. For example OWASP ZAP reported time-based SQL injection vulnerabilities that were not confirmed in manual testing.

Security testing results also accentuate the importance of manual penetration testing. Manual testing revealed various security vulnerabilities that would not have been discovered by using only automatic tools. A good example of such vulnerability is XML external entity attack vulnerability.

5 Conclusion

The objective of this thesis was to describe and apply a process necessary to verify the security of a web application. To achieve the aim, various software security standards were compared to find the standard that fits the requirements of this case study the best. OWASP ASVS was chosen to be the source of security requirements. A mapping between OWASP Top Ten 2013 risks and OWASP ASVS 3.0 level 1 requirements was created. A checklist of security requirements was determined according to the mapping.

Test cases were developed and web application UXP Portal was tested to verify 62 security requirements in the checklist. Test results and the security risks threatening UXP Portal were reported. Numerous security vulnerabilities were identified by security testing. It was concluded that in case of UXP Portal numerous injection, authentication and session management, cross-site scripting, insecure direct object reference, security misconfiguration, sensitive data exposure and cross-site request forgery risks are not mitigated. Lastly, the recommendations based on lessons learned during the case study were presented.

The work undertaken in this case study can be developed further. Future work on UXP Portal should include verifying all OWASP ASVS level 1 security requirements. Furthermore, after that it is possible to move on to verifying OWASP ASVS level 2 security requirements. It is also important to fix the security vulnerabilities discovered in this case study.

References

[1] (2016, March) The Global State of Information Security® Survey 2016. Visited 13.03.2016. [Online]. Available: http://www.pwc.com/gx/en/issues/cyber-security/information-security-survey.html

[2] (2016, April) UXP. Visited 24.04.2016. [Online]. Available: http://cyber.ee/en/e-government/uxp/

[3] (2016, March) OWASP Testing Guide 4.0. Visited 13.03.2016. [Online]. Available: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

[4] B. Potter and G. McGraw, "Software security testing", IEEE Security & Privacy, 2(5), 81-85 (2004).

[5] D. Verndon and G. McGraw, "Risk Analysis in Software Design", IEEE Security & Privacy, 2(4), 79–84 (2004).

[6] L. Dukes, X. Yuan, F. Akowuah, "A case study on web application security testing with tools and manual testing", Southeastcon, 2013 Proceedings of IEEE, 1-6 (2013).

[7] (2016, March) OWASP Application Security Verification Standard 3.0. Visited 13.03.2016. [Online]. Available: https://www.owasp.org/images/6/67/OWASP ApplicationSecurityVerificationStandard3.0.pdf

[8] (2016, March) OWASP Top Ten Project. Visited 13.03.2016. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[9] (2016, March) OWASP Cheat Sheet Series. Visited 13.03.2016. [Online]. Available: https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series

[10] (2016, March) SEI CERT Coding Standards. Visited 13.03.2016. [Online]. Available: https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards

[11] (2016, March) The Penetration Testing Execution Standard. Visited 13.03.2016. [Online]. Available: http://www.pentest-standard.org/index.php/Main_Page

[12] (2016, March) NIST Computer Security Publications. Visited 13.03.2016. [Online]. Available: http://csrc.nist.gov/publications/PubsSPs.html

[13] (2016, March) About The Common Criteria. Visited 13.03.2016. [Online]. Available: https://www.commoncriteriaportal.org/ccra/

[14] (2016, March) Certified Products. Visited 13.03.2016. [Online]. Available: https://www.commoncriteriaportal.org/products/

[15] W. Jackson, "Under attack" (2007). Visited 13.03.2016. [Online]. Available: https://gcn.com/articles/2007/08/10/under-attack.aspx

[16] J. Steven, "Threat Modeling-Perhaps It's Time", Security & Privacy, IEEE, 8(3), 83-86 (2010).

[17] A. Apvrille and M. Pourzandi, "Secure software development by example", IEEE Security & Privacy, 1(4), 10-7 (2005).

[18] G. Tóth, G. Kőszegi, Z. Hornák, "Case study: automated security testing on the trusted computing platform", Proceedings of the 1st European workshop on system security, 35-39, (2008).

[19] L. Wang, E. Wong, D. Xu, "A threat model driven approach for security testing", Proceedings of the Third International Workshop on Software Engineering for Secure Systems 2007, 10, (2007).

[20] J. Jürjens, "Model-based security testing using umlsec: A case study", Electronic Notes in Theoretical Computer Science, 220(1), 93-104 (2008).

[21] R. Vibhandik, A. K. Bose, "Vulnerability assessment of web applications-a testing approach", e-Technologies and Networks for Development (ICeND), 2015 Forth International Conference, 1-6 (2005).

[22] S. Acharya, B. Ehrenreich, J. Marciniak, "OWASP inspired mobile security", Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference, 782-784, (2015).

[23] K. Knorr and D. Aspinall, "Security testing for Android mHealth apps", InSoftware Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on 2015 Apr 13, 1-8 (2015).

[24] (2016, March) OWASP Zed Attack Proxy Project. Visited 29.03.2016. [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[25] (2016, April) SSLScan. Visited 24.04.2016. [Online]. Available: https://github.com/rbsec/sslscan

[26] (2016, April) Cookies Manager+. Visited 24.04.2016. [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/cookies-manager-plus/

[27] (2016, April) Firebug. Visited 24.04.2016. [Online]. Available: http://getfirebug.com/

[28] (2016, April) Groundspeed. Visited 24.04.2016. [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/groundspeed/

6 Appendix

6.1 OWASP ASVS 3.0 level 1 requirements mapped to OWASP Top Ten 2013 risks and UXP Portal requirements verification results

Number	OWASP ASVS 3.0 requirement [7]	OWASP Top 10 2013 risk [8]	Satisfied
1.1	Verify that all application components are identified and are known to be needed.		
2.1	Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation).	2013-A2-Broken Authentication and Session Management	no
2.2	Verify that all password fields do not echo the user's password when it is entered.	2013-A2-Broken Authentication and Session Management	yes
2.4	Verify all authentication controls are enforced on the server side.	2013-A2-Broken Authentication and Session Management	not ready
2.6	Verify all authentication controls fail securely to ensure attackers cannot log in.	2013-A2-Broken Authentication and Session Management	not ready
2.7	Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered.	2013-A2-Broken Authentication and Session Management	yes
2.8	Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.	2013-A2-Broken Authentication and Session Management	yes
2.9	Verify that the changing password functionality includes the old password, the new password, and a password confirmation.	2013-A2-Broken Authentication and Session Management	yes

2.16	Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link.	2013-A2-Broken Authentication and Session Management	yes
2.17	Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user.	2013-A2-Broken Authentication and Session Management	n/a
2.18	Verify that information enumeration is not possible via login, password reset, or forgot account functionality.	2013-A2-Broken Authentication and Session Management	yes
2.19	Verify there are no default passwords in use for the application framework or any components used by the application (such as "admin/password").	2013-A2-Broken Authentication and Session Management	yes
2.20	Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as brute force attacks or denial of service attacks.	2013-A2-Broken Authentication and Session Management	no
2.22	Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism.	2013-A2-Broken Authentication and Session Management	n/a
2.24	Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application.	2013-A2-Broken Authentication and Session Management	n/a
2.27	Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases.	2013-A2-Broken Authentication and Session Management	no
2.30	Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism.	2013-A2-Broken Authentication and Session Management	yes
2.32	Verify that administrative interfaces are not accessible to untrusted parties.	2013-A2-Broken Authentication and Session Management	no

3.1	Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks.	2013-A2-Broken Authentication and Session Management	yes
3.2	Verify that sessions are invalidated when the user logs out.	2013-A2-Broken Authentication and Session Management	yes
3.3	Verify that sessions timeout after a specified period of inactivity.	2013-A2-Broken Authentication and Session Management	yes
3.5	Verify that all pages that require authentication have easy and visible access to logout functionality.	2013-A2-Broken Authentication and Session Management	no
3.6	Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.	2013-A2-Broken Authentication and Session Management	no
3.7	Verify that all successful authentication and re-authentication generates a new session and session id.	2013-A2-Broken Authentication and Session Management	no
3.12	Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the "HttpOnly" and "secure" attributes	2013-A2-Broken Authentication and Session Management	no
3.16	Verify that the application limits the number of active concurrent sessions.	2013-A2-Broken Authentication and Session Management	no
3.17	Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session.	2013-A2-Broken Authentication and Session Management	no
3.18	Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.	2013-A2-Broken Authentication and Session Management	no

4.1	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.	2013-A4-Insecure Direct Object References, 2013-A7-Missing Function Level Access Control	yes
4.4	Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account).	2013-A4-Insecure Direct Object References, 2013-A7-Missing Function Level Access Control	yes
4.5	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	2013-A4-Insecure Direct Object References, 2013-A5-Security Misconfiguration	no
4.8	Verify that access controls fail securely.		
4.9	Verify that the same access control rules implied by the presentation layer are enforced on the server side.	2013-A7-Missing Function Level Access Control	not ready
4.13	Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism.	2013-A8-Cross-Site Request Forgery	no
4.16	Verify that the application correctly enforces context-sensitive authorisation so as to not allow unauthorised manipulation by means of parameter tampering.		
5.1	Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.	2013-A1-Injection	yes
5.3	Verify that server side input validation failures result in request rejection and are logged.	2013-A1-Injection	yes

5.5	Verify that input validation routines are enforced on the server side.	2013-A1-Injection	yes
5.10	Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection	2013-A1-Injection	yes
5.11	Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.	2013-A1-Injection	n/a
5.12	Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection.	2013-A1-Injection	yes
5.13	Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file.	2013-A4-Insecure Direct Object References	yes
5.14	Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks.	2013-A1-Injection	no
5.15	Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks.	2013-A3-Cross-Site Scripting	no
5.22	Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task.	2013-A3-Cross-Site Scripting	n/a
7.2	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding.		

7.7	Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard.	2013-A6-Sensitive Data Exposure	yes
8.1	Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information	2013-A5-Security Misconfiguration	no
9.1	Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features.	2013-A6-Sensitive Data Exposure	no
9.3	Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data).	2013-A6-Sensitive Data Exposure	yes
9.4	Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: Expires: Tue, 03 Jul 2001 06:00:00 GMT Last-Modified: {now} GMT Cache-Control: no-store, no-cache, must-revalidate, max-age=0 Cache-Control: post-check=0, pre-check=0 Pragma: no-cache	2013-A6-Sensitive Data Exposure	no
9.9	Verify that data stored in client side storage - such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies - does not contain sensitive or PII).	2013-A6-Sensitive Data Exposure	yes
10.1	Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid.	2013-A6-Sensitive Data Exposure	no

10.3	Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm.	2013-A6-Sensitive Data Exposure	yes
10.11	Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains	2013-A6-Sensitive Data Exposure	no
10.13	Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic.		
10.14	Verify that proper certification revocation, such as Online Certificate Status Protocol (OSCP) Stapling, is enabled and configured.		
10.15	Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority.	2013-A6-Sensitive Data Exposure	yes
10.16	Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure.	2013-A6-Sensitive Data Exposure	yes
11.1	Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, and DELETE) are explicitly blocked.	2013-A5-Security Misconfiguration	no
11.2	Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).	2013-A1-Injection	no

11.5	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	2013-A5-Security Misconfiguration	no
11.6	Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	2013-A1-Injection	no
11.7	Verify that the Content Security Policy V2 (CSP) is in use in a way that either disables inline JavaScript or provides an integrity check on inline JavaScript with CSP noncing or hashing.	2013-A3-Cross-Site Scripting	no
11.8	Verify that the X-XSS-Protection: 1; mode=block header is in place.	2013-A3-Cross-Site Scripting	no
16.1	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	2013-A10-Unvalidated Redirects and Forwards	yes
16.2	Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities.		
16.3	Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content.		
16.4	Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local file inclusion vulnerabilities.		

16.5	Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content.		
16.8	Verify the application code does not execute uploaded data obtained from untrusted sources.		
16.9	Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards.		
17.1	Verify that ID values stored on the device and retrievable by other applications, such as the UDID or IMEI number are not used as authentication tokens.		
17.2	Verify that the mobile app does not store sensitive data onto potentially unencrypted shared resources on the device (e.g. SD card or shared folders).		
17.3	Verify that sensitive data is not stored unprotected on the device, even in system protected areas such as key chains.		
17.7	Verify that the application sensitive code is laid out unpredictably in memory (For example ASLR).		
17.9	Verify that the app does not export sensitive activities, intents, content providers etc., for other mobile apps on the same device to exploit.		
17.11	Verify that the app's exposed activities, intents, content providers etc. validate all inputs.		
18.1	Verify that the same encoding style is used between the client and the server.		

18.2	Verify that access to administration and management functions within the Web Service Application is limited to web service administrators.		
18.3	Verify that XML or JSON schema is in place and verified before accepting input.		
18.4	Verify that all input is limited to an appropriate size limit.		
18.5	Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum.		
18.6	Verify the use of session-based authentication and authorization. Please refer to sections 2, 3 and 4 for further guidance. Avoid the use of static "API keys" and similar.		
18.7	Verify that the REST service is protected from Cross-Site Request Forgery.	2013-A8-Cross-Site Request Forgery	n/a
19.1	All components should be up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users.	2013-A5-Security Misconfiguration, 2013-A9-Using Components with Known Vulnerabilities	no

6.2 Test cases and test report to verify OWASP ASVS 3.0 requirement 3.6

OWASP ASVS 3.0 security requirement 3.6: "Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies" [7].

Test case 1

- Review the UXP Portal site tree composed by OWASP ZAP, make sure there are no URLs that contain the value of JSESSIONID.
- Perform active scan with OWASP ZAP, see if medium priority alert "Session ID in URL Rewrite" is in alerts list.

Test case 2

Make sure that UXP Portal does not support interchanging POST request for logging in with GET request:

- Try to log in, using the following URLs (first insert valid portal address, organization index, username and password):
- `https://<portal address>/login?loginForm1 _hf_0=&organization= <valid index>&username=<username>&password=<pass>`
- `https://<portal address>/admin/login?loginForm1_hf_0=&username=<username>&password=<pass>`
- Make sure that you are not logged in.

Test case 3

Make sure that UXP Portal does not support URL rewriting of session cookies:

- Delete all UXP Portal cookies from browser.

- Use the browser to send a GET request to the following URL: `https://<portal address>/login;jsessionid=lcfjqjhad73crgm4l4moa7m`
- Make sure that server does not accept the cookie in URL - server response contains a Set-Cookie header to set a new JSESSIONID value.

Test case 4

Make sure that it is not possible to manipulate UXP Portal to send JSESSIONID unencrypted by changing HTTP to HTTPS.

- Make a GET request to an authenticated page while logged in, but change HTTPS to HTTP.
- View the request and response headers with OWASP ZAP, make sure that JSESSIONID is not sent via HTTP.

Test case 5

Review the logs:

- /var/log/nginx/access.log
- /var/log/nginx/error.log
- /var/log/uxp/portal/jetty/jetty.log

Make sure JSESSIONID is not enclosed in log entries.

Results

- In UXP Portal session IDs are by default sent over encrypted transport (HTTPS). Session IDs are transferred in GET and POST request headers. POST request for logging in cannot be successfully interchanged with GET request and URL rewriting of session cookies is not possible.

- Session IDs are enclosed in URLs. In case the session cookie is deleted from the browser (test case 3) and a GET request made to portal login page after that, server reponds with code 302 and includes JSESSIONID in the Location header and after that the URL in Location header is loaded in browser. An example of a server response:

```
HTTP/1.1 302 Found

...

Set-Cookie:  JSESSIONID=2tusft03upnf1kjoy9hw7i3eq;Path=/

...

Location:  https://<portal address>:443/login;jsessionid=2tusft03upnf1kjoy9hw7i3eq?0
```

Due to session ID being enclosed in URL, in that case it is also logged in /var/log/nginx/ access.log.

- It is possible to manipulate client to send JSESSIONID unencrypted by changing HTTPS to HTTP. In case HTTPS is changed to HTTP in a GET request (for example http://<portal address>/home), server answers with status code 301 Moved Permanently and redirects to https://<portal address>/home. One HTTP request is sent that contains a session ID as well as orgLoginPath:

```
GET http://<portal address>/home HTTP/1.1

...

Cookie:  JSESSIONID=16d6o140qag07j6i5et0ejva9;

orgLoginPath=d6cf8ffb-c841-451f-9c5c-678bb5655846

...
```

Server responds with:

```
HTTP/1.1 301 Moved Permanently

Server:  nginx/1.8.1

Date:  Mon, 29 Feb 2016 16:57:32 GMT
```

```
...
Location: https://<portal address>/home
```

Risks:

- Enclosing session ID in URL might disclose the session ID to attackers as the URL containing the session ID can end up in logs, web browser history, browser bookmarks and the HTTP header referer field.
- HTTP to HTTPS redirection produces a single unprotected HTTP request/response exchange that can be used by a malicious attacker to sniff a valid session ID. If a user is logged in to portal and attacker is able to convince a portal user to click on a HTTP link to an authenticated portal page, valid JSESSIONID is sent by HTTP request and in case attacker is able to sniff the network, attacker obtains a valid session ID.

Recommendations:

- Session ID should never be enclosed in the Location header sent by the server.
- UXP Portal should avoid HTTP to HTTPS redirection. Instead HTTP Strict Transport Security (HSTS) should be enforced to HTTPS connections, for example: Strict-Transport-Security: max-age=15724800; includeSubdomains.

Additional information:

- https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
- https://www.owasp.org/index.php/HTTP_Strict_Transport_Security

6.3 Glossary

ASVS Application Security Verification Standard

CA Certificate Authority

CC Common Criteria

CERT Computer Emergency Response Team

CMU Carnegie Mellon University

CSRF Cross-Site Request Forgery

HSTS HTTP Strict Transport Security

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

LDAP Lightweight Directory Access Protocol

NIST US National Institute of Standard and Technology

OWASP Open Web Application Security Project

PTES Penetration Testing Execution Standard

REST Representational State Transfer

SDLC software development life cycle

SP Special Publications

SQL Structured Query Language

TLS Transport Layer Security

UML unified modeling language

URL Uniform Resource Locator

UXP Unified eXchange Platform

WYSIWYG what you see is what you get

XSS Cross-site scripting

XML EXtensible Markup Language

ZAP OWASP Zed Attack Proxy

6.4 Licence