Masterprojekt i Softwarekonstruktion

IT-sikkerhed

ASVS som fundament for IT-sikkerheds awareness

Studerende
Torben Krogh Johansen

Vejleder
René Rydhof Hansen

Indholdsfortegnelse

1 Resumé

Summary of the master thesis "ASVS as the foundation for IT security awareness"

The purpose of my thesis is to analyse if OWASP's ASVS (Applications Security Verification Standard) can be used as the foundation for creating IT security awareness among application developers. The reason for my research is my interest in awareness and my fascination of ASVS's way of focusing on what to do, instead of what not to do.

Chapter two encompasses the introduction to the thesis and the problem statement. In the introduction it is emphasised why awareness is important, and it is specified that the awareness in this thesis is awareness among application developers. The first part of the problem statement is to analyse if ASVS can be used as the foundation for awareness. The second part is to pinpoint elements from the theory "Three Pillars of Software Security" which is needed to be able to create awareness using ASVS.

In chapter three my motivation for writing this thesis is explained by three war stories. Each of the stories shows a situation where obvious IT security goals were not fulfilled at all. The point is that with the proper amount of IT security awareness among the application developers, it is very likely that the stories would have never happened.

Chapter four encompasses a description of a fictive banking system. It is named a standard bank. The purpose of the standard bank is divided into two. Firstly, it is needed in order to have a simplified system to apply ASVS onto. Secondly, it has to be a fictive system, since exposing a real system could be dangerous.

Chapter five outlines the overall purpose of IT security by the CIA model (Confidentiality, Integrity and Availability). The goals of the CIA model is explained and related to a banking customer, to awareness and to the standard bank.

Chapter six describes the theory "Three Pillars of Software Security". Part one is about the subject "Risk Management". Shortly this is about linking business goals and security demands together. Part two explains "Seven Touchpoints" which is a set of security best practices that can be applied in any software development lifecycle. Part three is about "Knowledge" which is quite related to awareness.

Chapter seven is about ASVS. It is easier to understand why ASVS is so different if the reader is familiar with the OWASP Top 10. Hence, it is shortly described in part one. Part two is about ASVS. The three different security verification levels are explained and related to a bank. ASVS consists of 16 different verifications, which are each explained shortly. Afterwards, ASVS's relation to the theory "Three Pillars of Software Security" is examined. This shows that ASVS is directly related to "Knowledge". At the end of chapter seven, the organisation and history of OWASP is described.

Chapter eight is where ASVS is applied to the standard bank. Before applying it, it is decided, upon which parameters is should be validated. Parameters are: Relevance to the developer, level of difficulty and reusability of training material. Instead of applying all 16 ASVS verifications to the standard bank, only four have been selected. “V1 Architecture, design and threat modelling”, “V2 Authentication”, “V8 Error handling and logging” and “V10 Communications”. During the process of applying these four verifications, four challenges appear.

Chapter nine is about the above mentioned four challenges, which are each described separately. The first challenge is about deciding to which level a system should be verified. The challenge is named “Classification”. The second challenge is named “Security requirement” and describes the need for knowing about requirements, when making applications. The third challenge is “Reuse of code”. Different ways of reusing code provide different levels of problems. The fourth challenge is “Audience” and is about the audience for awareness training. Different audience applies different responsibilities and the need for coordination.

Chapter 10 encompasses the discussion. It is stated that resources and management commitment are needed in order to achieve awareness. In the end, it is a management responsibility to achieve awareness. Using ASVS as the foundation for IT security awareness is not as simple as expected. It requires coordination and depends on a combination of target groups, relevance and reusability. Despite it is not as simple as expected, it is recommended to use it. An alternative based on the OWASP Top 10 is shown. The need for elements from the theory “Three Pillars of Software Security” was expected. But if the purpose mainly is to achieve awareness, there is no need for additional elements. If the purpose is to create more secure systems, it is desirable to implement “Three Pillars of Software Security”.

In chapter 11 it is concluded that:
ASVS can be used as the foundation for IT security awareness. But management should be aware of the resource requirements needed.
ASVS can be used as the foundation for IT security awareness, without the need for elements from the theory “Three Pillars of Software Security”

2 Indledning

I dag er mange virksomheder afhængige af deres webløsninger. Fokus er ofte mest på brugervenlighed, da det er en parameter, som er direkte afgørende for kundens oplevelse. Men der er også en enorm afhængighed af IT-sikkerheden. Brud på fortrolighed, integritet eller tilgængelighed vil koste virksomheden et direkte økonomisk tab eller et indirekte økonomisk tab grundet tab af omdømme. IT-sikkerhed er derfor meget vigtig.

En af de bløde værdier indenfor IT-sikkerhed er awareness. Den måde, man oftest tænker på awareness, er for slutbrugere af et system. Et eksempel er at alle ansatte i en virksomhed modtager e-mails. Det er derfor vigtigt, at de har en bevidsthed - awareness - omkring håndtering af vedhæftede filer fra ukendte afsendere. Hvis ikke denne awareness er til stede, udsættes virksomheden for risikoen for ransomware. Et eksempel på, hvor galt det kan gå, er NotPetya Ransomware angrebet hos Mærsk ([Version2-1]).

Jeg vil arbejde med en anden type awareness i denne rapport. Mere specifikt den awareness udviklere bør have for at lave sikre webløsninger. I min hverdag ser jeg nemlig en stor forskel på de af mine kollegaer, som besidder awareness, og de som ikke gør. De, som besidder awareness, skaber sikre løsninger - uanset hvilke metoder de arbejder efter. Derfor er awareness så vigtigt.

En organisation, som er blevet mere og mere anerkendt, er OWASP (Open Web Application Security Project). De har en anden - mere praktisk - tilgang til, hvordan man opnår sikre webløsninger. Deres ASVS (Application Security Verification Standard) ([OWASP-4]) er en tjekliste, som bruges til at bestemme om sikkerhedsniveauet for en webløsning er tilstrækkelig.

Da OWASP er meget konkret har jeg behov for et system at arbejde med. Jeg arbejder i en bank, men for at give mig frihed under udførelsen af denne rapport vælger jeg at beskrive en standard bank og de tilhørende nødvendige IT-systemer.

Min forventning er at ASVS ikke kan bruges alene, og jeg bliver derfor nødt til at supplere med teori. Teorien jeg vil arbejde med i denne rapport er:

- “Three Pillars of Software Security” [SS]

2.1 Problemformulering

Jeg vil i denne rapport bruge ASVS på et tænkt system - en standard bank - for at:

- vurdere om ASVS kan bruges til at skabe awareness.
- udpege elementer fra teorien, som mangler for at skabe awareness med ASVS.

3 Motivation

Jeg vil i det følgende komme med tre historier fra min hverdag.

Af fortrolighedsmæssige grunde oplyser jeg ikke, hvor og hvornår historierne er fra. Jeg tror ikke historierne er unikke, og mit bud er, at flere i branchen har oplevet lignende.

3.1 Ny serviceoperation

Et gammelt internt system til vedligeholdelse af brugere skulle moderniseres. Det var et sikkert system, som fik input via filer. Der var kontrol over adgangen til disse filer. Eneste problem med systemet var, at læsningen af filerne var scheduleret. Så når et bruger objekt blev ændret, skulle man vente på næste schedulerede kørsel.

Løsningen havde en del år på bagen, så man valgte at kode det helt om. Alt input til systemet kom til at foregå med REST services. En god beslutning da en serviceoperation giver en løs kobling.

Efter to år opdagede jeg ved et tilfælde at REST servicen, hverken krævede autentifikation eller autorisation. Alle ansatte i virksomheden kunne altså modificere alle brugere!
Et helt igennem skrækscenarie, der sandsynligvis ikke var opdaget af nogen.

3.2 Omlæggelse til serviceoperation

En ældre kundevendt webløsning skulle udstille data til andre systemer. Løsningen blev at flytte en del logik ud i en serviceoperation, og efterfølgende bruge serviceoperationen fra webgrænsefladen. Det er en god afkobling og arkitektonisk det rigtige valg.

Men man havde glemt, at den gamle webløsning havde kontrol over både autentifikation og autorisation. Den nye løsning havde kun kontrol over autentifikation. Så alle brugere, som var logget på systemet, kunne nu med lidt snilde kalde serviceoperationen.

Det var altså muligt med “url hacking” at se andre kunders data.
Igen et skrækscenarie og igen kan man håbe, at det ikke blev opdaget af nogen.

3.3 Kodeord i klar tekst

En ældre kundevendt webløsning havde brug for at modtage data fra andre systemer. Løsningen blev at lave en web service. Den helt rigtige beslutning, da det giver en løs kobling.

Da jeg blev ansat i virksomheden opdagede jeg, at webservicen ikke kunne kaldes over HTTPS men udelukkende over HTTP. Løsningen krævede autentifikation - altså brugernavn og kodeord - som derfor blev transporteret over internettet i klar tekst.

Endnu engang et skrækscenarie og endnu engang kan man håbe, at det ikke blev opdaget af nogen.

3.4 Min motivation

Ovenstående tre historier fortæller om IT-sikkerhedsproblemer, der sandsynligvis aldrig ville være opstået, hvis udviklerne bag dem havde haft tilstrækkelig awareness om IT-sikkerhed.

I den første historie om web servicen til vedligeholdelse af brugere burde arkitekten bag løsningen have tænkt på autentifikation og autorisation til et så kritisk system. Men det gjorde arkitekten ikke. Det næste sted at se det er i implementeringsfasen. Men heller ikke her blev det opdaget.
Hvis udvikleren havde haft en basal IT-sikkerheds awareness, ville udvikleren som et minimum i testfasen have undret sig over, at web servicen kunne kaldes uden at angive credentials. En dårlig løsning kunne være undgået.

Den anden historie med omlæggelse til serviceoperation kunne også have været undgået. Igen er ansvaret delt mellem arkitekt og udvikler. Arkitekten burde have vidst, at der blev udført en autorisation. Tilsvarende burde udvikleren have tænkt, at der mangler en autorisation. Men den er lidt sværere at fange i en test. Når udvikleren har testet, har der jo været en angivelse af credentials. Men en god udvikler med en god IT-sikkerheds awareness burde have opdaget problemet.

Den tredje historie med kodeord i klar tekst burde aldrig være opstået. Uanset hvad bør et kodeord aldrig sendes i klar tekst. Hvis udvikleren havde haft den mindste IT-sikkerheds awareness kunne det have været undgået.

Fælles for alle tre ovenstående historier er, at den dårlige løsning kunne have været undgået, hvis udvikleren havde haft tilstrækkelig med awareness.

Men der er andre faktorer, som kan have været skyld i de dårlige løsninger.

Manglende udviklingsressourcer kan gøre, at der er pres på for at levere en løsning. Udvikleren kan have påpeget at dette kun er en første-udgave, og at den bør gøres færdig. Men da første-udgaven af løsningen består den funktionelle test, bliver løsningen aldrig færdig modnet. I sådanne tilfælde er der tale om en ledelsesmæssig årsag til de dårlige løsninger.

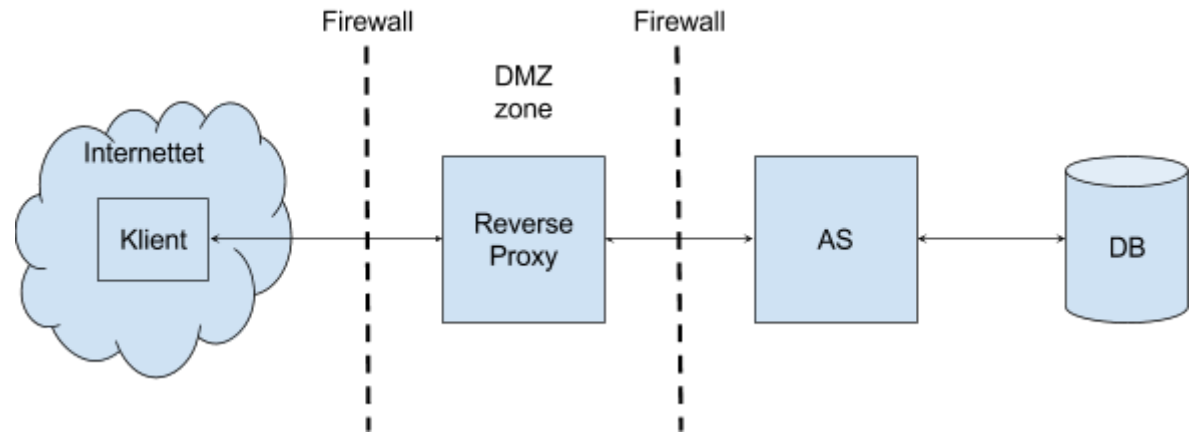
Manglede udviklingsprocesser kan gøre, at udviklingsforløbet har været uden hensyntagen til IT-sikkerhed. Hvis udviklingsprocessen havde indeholdt krav til arkitektur review eller udførelse af en penetrationstest, tror jeg ikke de dårlige løsninger ville have været lavet. Igen er der så tale om en ledelsesmæssig årsag til de dårlige løsninger.

Min motivation er at skabe awareness blandt udviklere. Uanset manglende udviklingsressourcer eller -processer vil det skabe mere sikre systemer.

Forhåbentlig kan jeg så være med til at reducere antallet af tilsvarende nye hverdagshistorier i fremtiden.

4 En standard bank

I figur 1 har jeg tegnet en typisk systemarkitektur, som kan bruges til de fleste webapplikationer.



Figur 1

Jeg vil i det efterfølgende gennemgå de enkelte komponenter.

Det jeg har set i min dagligdag er, at man enten hælder til Linux eller til Windows. Så for hver enkelt komponent vil jeg angive to specifikke valg.

4.1 Systembeskrivelse af en standard bank

4.1.1 DB - Database

For at en applikation kan blive dynamisk, bliver den nødt til at kunne persistere data. Behovet dækkes med en **database**. Der findes rigtig mange forskellige databaser. Der er de helt typiske relationelle databaser som Oracle, SQL server, MySql, MariaDB, DB2, hvor data tilgås med SQL statements. I det sidste stykke tid er der dukket flere alternativer op, f.eks. MongoDB, som er en NoSQL database.

Men for at holde min standard bank simpel, vælger jeg disse to relationelle databaser:

- **Linux: Oracle Database**
- **Windows: SQL server**

4.1.2 AS - Applikationsserver

For at kunne lave en HTML-side, har man behov for noget, som henter data fra databasen. Det er typisk enten en hel applikation, som danner en HTML-side og beriger den med data fra databasen. Eller, som man ser oftere og oftere, en applikation, der henter data ud med servicekald, f.eks. restful services.

Applikationen skal afvikles et sted, og stedet er en applikationsserver. Gennem tiden har jeg set mange forskellige slags. Til Java applikationer har jeg set fra helt simple Tomcat servere til mere udviklede JBoss applikationsservere. Jeg har også arbejdet med IBM's WebSphere Application Server til Java webapplikationer.

Microsoft har i lang tid arbejdet med at gøre deres applikationsserver mere robuste. I dag er en Microsoft IIS (Internet Information Server) fuldt ud lige så accepteret som de andre applikationsservere.

For min standard bank vælger jeg disse applikationsservere.

- **Linux: JBoss**
- **Windows: IIS**

En applikationsserver skal have deployed en eller flere applikationer for at give mening.
De applikationer, som min standard bank skal bestå af er:
Konto applikation (KontoApp) og Valuta applikation (ValutaApp).

Begge applikationer skal indeholde en webside og tilhørende restful serviceoperationer. Formålet med serviceoperationerne er at lave en afkobling og gøre det muligt for f.eks. mobil-applikationer at bruge de samme serviceoperationer.

4.1.2.1 KontoApp

Formålet er at overføre penge fra kundens egne konti til andre konti. Det kræver, at kunden er autentificeret. Denne applikation vil være et oplagt mål for hackere, da de vil kunne opnå økonomisk gevinst med det samme.

Webside / serviceoperation
Give mulighed for at overføre penge fra egne konti til andre konti.

4.1.2.2 ValutaApp

Formålet er at vise omregningskurser mellem forskellige valuta. Dette er at betragte som en offentlig service, så det vil ikke kræve, at kunden er autentificeret. Tilsvarende vil den ikke være et oplagt mål for hackere. Men applikationen skal også indeholde mulighed for at administrere valutakurser. Denne del vil kun være internt tilgængeligt, men skal kræve autorisation.

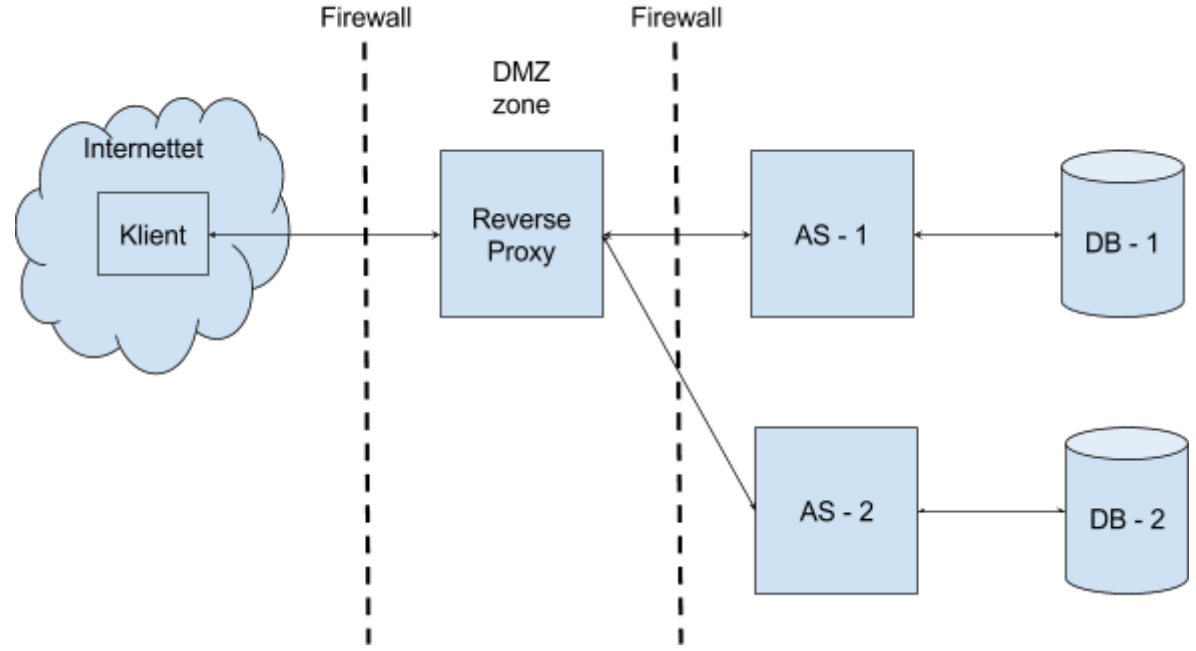
Webside / serviceoperation
Give mulighed for at vise og administrere valutakurser.

4.1.3 Reverse Proxy

Denne del af arkitekturen er altid et smertens barn. Er den nødvendig eller ej? Jeg vil argumentere for, at den er nødvendig ud fra dens formål.

Navnet på komponenten er en reverse proxy. Men den har tre formål.

Første formål er, som navnet angiver, at være reverse proxy. Det gør, at trafikken ude fra internettet termineres på én IP-adresse, men at reverse proxyen dirigerer trafikken flere steder hen, se figur 2. Formålet er at muliggøre load balancering og kompartmentalisering.



Figur 2

Det andet formål - og måske det vigtigste - er at have en komponent, som er skabt og derfor hærdet til at stå i en DMZ. Komponenten er udsat for trafik direkte fra internettet. Den vil typisk kun lytte på port 80 (HTTP) og 443 (HTTPS). Dens rolle er at være første bastion i at modstå et angreb. Det er derfor vigtigere, at reverse proxyen er hærdet, end at applikationsserveren er det.

Det tredje formål er autentificering. Information om hvem som er autentificeret videregives til applikationsserveren.

Jeg har arbejdet med IBM's reverse proxy. Den hed tidligere WebSeal, men har nu skiftet navn til ISAM (IBM Security Access Manager).

På Windows siden har jeg ikke arbejdet med dette område endnu. Men deres produkt Web Application Proxy skulle gøre det samme. En forløber for produktet er Forefront Unified Access Gateway (UAG).

For min standard bank vælger jeg disse reverse proxy's:

- **Linux: ISAM**
- **Windows: Web Application Proxy**

4.1.4 Klient

Her er der stort set frit valg. Det vigtige er, hvilke klienter forretningen vil understøtte. Ved komplekse webapplikationer kan man sagtens opleve store forskelle på, hvordan applikationen virker i FireFox, Chrome og Internet Explorer. Forskellen er også at finde mellem forskellige versioner af de enkelte browsere. Så jo flere forskellige platforme din forretning vil understøtte, jo større arbejde med at teste og vedligeholde.

En tendens er at man bevæger sig væk fra browsere og mere mod mobil-applikationer. Mobil-applikationen vil så typisk hente data med servicekald. Det giver en fin afkobling, således at applikationsudvikleren ikke skal tilpasse sin applikation til klienten.

4.2 Autentifikation og autorisation

Sammenhængen mellem autentifikation og autorisation er som følger. Autentifikation er, hvor man identificerer brugeren, f.eks med brugernavn og kodeord. Når brugeren er autentificeret, kan det afgøres, om brugere har adgang. Altså om brugeren er autoriseret.

Den måde autentifikation og autorisation håndteres på i standard banken er som beskrevet herunder.

Autentifikation håndteres af reverse proxyen

Reverse proxyen er konfigureret således, at hvis en HTML-side kræver autentifikation, vises siden ikke før, brugeren er autentificeret. Dette kan gøres på flere måde. Enten ved at redirecte til en login-side eller ved at bede klienten om at autentificere med BA (Basic Authentication).

Autorisation håndteres af applikationsserveren.

Når brugeren er autentificeret, videresender reverse proxyen HTTP-requestet til applikationsserveren med information om brugeren. Enten via et specifikt felt i HTTP-headeren, eller med en cookie.

Applikationen er nu istand til at afgøre om brugeren er autoriseret til at udføre den ønskede handling.

4.2.1 Autentifikation og autorisation i ValutaApp

Når en bruger forsøger at vise HTML-siden, ved reverse proxyen, at siden ikke kræver autentifikation. Den videresender derfor requestet til applikationsserveren. Applikationen returnerer en passende HTML-siden til brugeren uden at foretage yderligere kontroller.

4.2.2 Autentifikation og autorisation i KontoApp

Når en bruger forsøger at vise HTML-siden, ved reverse proxyen, at siden kræver autentifikation. Den redirecter til en login-side, hvor brugeren bliver bedt om at autentificere. Når brugeren har autentificeret sig, videresender reverse proxyen requestet til applikationsserveren sammen med en cookie.

Applikationen læser informationen om brugeren i cookien. Informationen om brugeren anvendes af applikationen til at afgøre, om brugeren er autoriseret til at udføre den ønskede handling eller ej. Applikationen returnerer en passende HTML-side til brugeren.

4.3 Opsummering

Jeg har i ovenstående skitseret en typisk arkitektur, som kan bruges til at drive en standard bank med. Da min viden hovedsageligt er baseret på Linux og Windows operativsystemerne, tager løsningen udgangspunkt i disse platforme. Men en standard bank kan selvfølgelig baseres på andre teknologier. I disse tider sker der en enorm udvikling inden for cloud-løsninger.

5 IT-sikkerhed

Hvad er IT-sikkerhed? Dette spørgsmål er så grundlæggende og alligevel, er der divergerende opfattelser af, hvad det er. Så derfor en præcisering af, hvad der opnås med IT-sikkerhed.

Overordnet set defineres formålet med IT-sikkerhed i FIT-modellen.
FIT - Fortrolighed, Integritet, Tilgængelighed (på engelsk CIA - Confidentiality, Integrity, Availability).

Fortrolighed:
Sikre at data hemmeligholdes, således at de ikke kan ses af andre end dem, som er autoriserede til at se data. For at selve datatrafikken hemmeligholdes er det oplagte valg at kryptere trafikken via SSL. For at sikre den rigtige bruger kræves autentifikation. På de fleste netbanker er der i dag to-faktor login med NemID.

Integritet:
Sikre at data ikke ændres uautoriseret på noget tidspunkt i datas livscyklus. Her er kryptering af trafikken via SSL en måde at undgå, at data ændres under transporten fra brugeren og ind til datacenteret. Men man skal også sikre sig, at data ikke ændres uautoriseret i datacenteret. F.eks. ved at en database-administrator ændrer data uden at forretningen ved det.

Tilgængelighed:
Sikre at data er tilgængelige for dem, som er autoriserede - når de skal bruge dem. En måde at sikre høj tilgængelighed er ved at have dubleret servere, således at systemerne altid kan afvikles. Der tages også backup af data, så at data altid kan fremfindes.

Her er et eksempel på hvert punkt for mig som bankkunde:

- Det er altafgørende, at mine data ikke kan ses af andre end mig selv (Fortrolighed).

- Hvis jeg betaler en regning, er det altafgørende, at jeg efterfølgende kan dokumentere overfor f.eks. skat, at jeg har betalt min regning. Ellers har jeg f.eks. ikke dokumentation for mit håndværkerfradrag (Integritet).

- Det er desuden altafgørende for mit kundeforhold, at jeg kan bruge systemet, når det passer ind i min dagligdag (Tilgængelighed).

FIT-modellen set fra en bankkundes synspunkt
Fra en kundes synspunkt er tilgængelighed det vigtigste mål. Det er så vigtigt, at man i bankverdenen i mange tilfælde har slækket på to-faktor login, således at man kan nøjes med en-faktor login for at logge på netbanken og se f.eks. saldi. Hvis man så efterfølgende skal lave en transaktion, afkræves to-faktor login.

Dette er et klart krav fra kunderne om øget tilgængelighed på bekostning af fortrolighed. Integritet og fortrolighed er også vigtige.

FIT-modellen set fra en banks synspunkt
Fra en banks synspunkt er integritet det vigtigste mål. Hvis netbanken ikke er tilgængelig, eller kunder kan se hinandens data, vil det være meget pinligt. Men hvis en kunde ikke kan stole på saldi, mister kunden tilliden til banken. Hvis mange kunder mister tilliden til banken, er det kun et spørgsmål om tid før banken må lukke.
Fortrolighed og tilgængelighed er også vigtige.

Er FIT-modellen tilstrækkelig for en bank?
En ting, man ikke må glemme, er den traditionelle fysiske sikkerhed:

- Hvem kan komme ind i den fysiske afdeling?
    - Til trods for den digitale tidsalder, er der værdier i afdelingerne.

- Overvågning af pengeautomater
    - Der kan monteres skimmers på pengeautomaterne.

- Hvem kan komme ned i maskinstuen i datacenteret?
    - Hvis man har fysisk adgang, vil man hurtigt kunne påvirke tilgængeligheden.

Et mål, FIT-modellen ikke direkte håndterer, er uafviselighed (non-repudiation).
Såfremt en handling er udført i systemet af en bruger, der er autentificeret, er vi som systemejere sikre på, at det er brugeren, der har udført handlingen. Men uafviselighed er mere end det. Vi kan ikke bevise over for en tredjepart, f.eks. en domstol, at det er brugeren, der har udført handlingen. Årsagen er, at vi som systemejere kan konstruere data, der viser at brugeren har udført en handling.

En måde man ser uafviselighed udført på i webløsninger, er i forbindelse med underskrift af dokumenter - f.eks låneansøgninger. Her bruges NemID til at underskrive med. I praksis bruges den private nøgle tilhørende NemID til at lave en signatur af dokumentet. På den måde er det entydigt, at man har læst og accepteret dokumentet. Også over for en tredjepart, da vi som systemejere ikke kan konstruere signaturen.

Uanset hvilke faktorer, der er de vigtigste, og om FIT-modellen er fuldstændig tilstrækkelig, er den ekstrem vigtig.

Hvis FIT-modellens mål ikke overholdes, vil det være tvivlsomt, om en virksomhed fortsat vil kunne eksistere.

5.1 FIT-modellen i relation til awareness

For konstant at kunne sikre FIT-modellen er awareness vigtigere end nogensinde. Årsagen er hovedsageligt disse to parametre:

- Skærpet trusselsbillede
- Agil udvikling - DevOps

Det skærpede trusselsbillede gør, at sårbarheder bliver udnyttet hurtigere og oftere end tidligere. Det at udnytte en sårbarhed er desværre blevet til en forretningsmodel.

Der stilles store krav til agilitet i udviklingen. Kunderne efterspørger ny funktionalitet, og time-to-market er en konkurrenceparameter. Med DevOps [Wiki-1] foretages deployments oftere end tidligere i produktionsmiljøet. Hyppigere deployments gør, at testforløb er kortere. Ofte er det kun udvikleren, som er garant for sikkerheden.

Alt i alt gør det, at awareness er vigtigere end nogensinde for at sikre FIT-modellen.

5.2 FIT-modellen i relation til en standard bank

I standard banken har alle komponenterne en rolle i forhold til FIT-modellen.

Klienten skal være robust, således at

- data ikke lækkes (Fortrolighed)
- det ikke er muligt at udføre handlinger på vegne af andre (Integritet)
- den altid er tilgængelig (Tilgængelighed)

Reverse Proxy'en skal sikre korrekt autentifikation, således at

- data ikke vises til de forkerte brugere (Fortrolighed)
- handlinger ikke registreres med forkert bruger (Integritet)

Reverse Proxy'en skal desuden være robust overfor angreb, således at

- standard banken altid er tilgængelig (Tilgængelighed)

Applikationsserveren skal sikre korrekt autorisation, således at

- data ikke vises til de forkerte brugere (Fortrolighed)
- handlinger ikke registreres med forkert bruger (Integritet)

Applikationsserveren skal desuden være robust, således at

- standard banken altid er tilgængelig (Tilgængelighed)

Databasen skal være robust, således at

- standard banken altid er tilgængelig (Tilgængelighed)

6 Three Pillars of Software Security

I bogen Software Security [SS] beskrives tre byggesten som en måde til at opnå IT-sikkerhed.

Det er ikke en let og ubesværlig process at implementere sikker software. Det er almindelig kendt, at der i vores organisationer er personer, som ikke har haft forståelsen for, hvorfor IT-sikkerhed er vigtig og bestemt heller ikke viden om, hvordan man opnår det.

Ifølge forfatteren af Software Security [SS] får man en metode, som virker, samt en organisatorisk synliggørelse af indsatsen ved at implementere “Three Pillars of Software Security”.

De tre byggesten er: Risk Management, Seven Touchpoints og Knowledge. Jeg vil i de næste underafsnit kort beskrive dem samt relatere dem til awareness.

6.1 Risk Management

Som beskrevet i afsnittet om IT-sikkerhed er en virksomhed dybt afhængig af alle parametre på FIT-modellen. Men hvor afhængige? Og til hvilke risici?

For at en IT-virksomhed kan navigere i den enorme mængde af risici, skal den have et værktøj til at belyse disse risici. Værktøjet hedder Risk Management.

I Risk Management ser man på sammenhængen mellem virksomhedens mål i forhold til sikkerhedskrav. Man finder ud af, hvilke risici man løber. Hvor sandsynligt er det, at risikoen indtræffer, og hvilke konsekvenser for virksomhedens mål, vil det i så fald have. Når man har lavet analysen, kan man derfor prioritere håndteringen af de forskellige risici. Skal den enkelte risiko mitigeres, elimineres, flyttes eller accepteres?

En af de sværeste ting i Risk Management er balancen mellem forretningskendskab og IT-kendskab. En person med IT kendskab kan vurdere, at det vil tage f.eks. 12 timer at restore fra backup efter et ransomware angreb, men kan ikke vurdere om 12 timer er kritisk for forretningen. Omvendt kan en person med forretningskendskab vurdere at f.eks 6 timers nedbrud er kritisk for forretningen, men kan ikke vurdere, om der er situationer som resulterer i nedbrud af mere end 6 timers varighed.

Det vil derfor give stor værdi for Risk Management, hvis de personer, som er involverede i at lave risikoanalysen, også besidder IT-sikkerheds awareness.

Bemærk, at både trusselsscenariet og virksomhedens mål ofte er af dynamisk karakter, og det er derfor vigtigt, at processen gentages regelmæssigt.

6.2 Seven Touchpoints

I bogen [SS] beskrives Seven Touchpoints. Beskrivelsen af disse touchpoints er efter min mening hovedformålet med bogen.

Forfatterens mening med et Touchpoint er:
"*... a set of security security best practices that I call touchpoints*." ([SS], side 83).

Desuden ligger der følgende i hans måde at definere det på:
"*... the software security touchpoints are designed to be process agnostic. That is, the touchpoints can be applied no matter which software process you use to build your software.*" ([SS], side 84).

Kort sagt har forfatteren altså beskrevet syv best practices for at udvikle sikker software, som desuden har den egenskab, at de kan implementeres uanset, hvilke metoder og processer en organisation udvikler software efter.

Grunden til, at der kun er syv touchpoints, er ifølge forfatteren, at der tidligere er lavet beskrivelser af, hvordan man opnår sikker software på en måde, som har været meget kompliceret. Så kompliceret, at det aldrig bliver implementeret. Han mener derfor, at hvis man begrænser sig til disse syv touchpoints, er det så overskueligt, at det rent faktisk bliver implementeret. Jeg synes at det giver rigtig god mening.

I det følgende er der en meget kort beskrivelse af de enkelte touchpoints samt deres relation til IT-sikkerheds awareness.

6.2.1 Code Review

Alle softwareprojekter producerer kode og review af koden er meget vigtig. Formålet med reviewet er at finde manuelt introducerede fejl, som udvikleren har lavet. Der er to måder at udføre review'et på. Den første er ved klassisk review, hvor en anden udvikler vurderer koden. Den sidste er at bruge værktøjer, som automatisk laver statisk analyse af koden.

Et manuelt kode review kræver forståelse for IT-sikkerhed for at vide, hvad man skal kigge efter. Med andre ord bliver kode review's bedre jo højere niveau af IT-sikkerheds awareness revieweren besidder.

6.2.2 Architectural Review

I dette afsnit analyseres arkitektur, design og specifikationer.
Et eksempel på ting, der skal fanges her, er om en webservice er beskyttet korrekt med autentifikation og autorisation. Eksempelvis kan det være fint, hvis en webservice, som viser nogle generelle oplysninger, ikke kræver andet end, at brugeren er autentificeret. Hvorimod en webservice, som laver en opdatering, skal kræve, at brugeren er autoriseret til det.

Udover eksemplet med webservices kan viden om cipher suites også give værdi her. Jo højere IT-sikkerheds awareness revieweren besidder, jo bedre review.

6.2.3 Penetration Testing
Her laver man typisk en test på et system i sit endelige produktionsmiljø. Det giver en bedre penetrationstest, hvis testen er beriget med informationer om systemet. Eksempelvis input fra Architectural review.

En penetrationstest kan slet ikke udføres, hvis man ikke besidder IT-sikkerheds awareness. Det er en absolut forudsætning for at vide, hvor man skal starte testen.

6.2.4 Risk-based Security Testing
Udgangspunktet for denne test er resultatet fra risikoanalysen, "Abuse case" og kendte angrebsmønstre. Man prøver altså at teste, hvor det giver mest mening ud fra risikoanalysen, og prøver at tænke som en angriber.
Præcis her bliver det klart hvad forskellen på funktionalitetstest og sikkerhedstest er: *"QA is about making sure good things happen. Security testing is about making sure bad things don't happen."* ([SS], side 87).

Grundlaget er risikoanalysen, men det at sammenholde den med det at "tænke som en angriber" kræver viden og bevidsthed om IT-sikkerhed. Igen er awareness vigtigt.

6.2.5 Abuse Case
Her forestiller man sig, hvordan systemet kan misbruges. At lave abuse cases kræver, at man tænker som en angriber. Abuse cases beskriver, hvordan systemet opfører sig under et angreb.

Ligesom i ovenstående skal man her "tænke som en angriber". Det kræver IT-sikkerheds awareness.

6.2.6 Security Requirements
Her sikrer man sig, at sikkerhedskrav er beskrevet i systemets kravspecifikation. Det kan f.eks være, at dataudveksling via en webservice sker kryptereret.

Her er behovet for IT-sikkerheds awareness knap så stort. Det er mere en sikring af, at firmaets politikker og tilhørende regler er tilgodeset i systemets kravspecifikation.

6.2.7 Security Operations
Vores systemer bliver angrebet uanset, hvad vi tror. Det er derfor vigtigt at kunne monitorere vores systemer, også ud fra et sikkerhedsmæssigt perspektiv. Det er ekstremt betydningsfuldt at have logfiler, som dokumenterer et angreb i forhold til en domfældelse.

Når der opstår en situation, som skal efterforskes, vil det altid give værdi at besidde IT-sikkerheds awareness. Det vil være med til at afgøre, hvad man skal kigge efter i logfiler. F.eks. vil det være mistænkeligt, hvis en bruger tilgår systemerne samtidigt fra to forskellige IP-adresser.

6.3 Knowledge

Viden er en meget stor del af IT-sikkerhed, og det er her, hvor IT-sikkerheds awareness beskrives. Jeg ser “knowledge” inddelt i to overordnede grupper.

Den første er en overordnet sikkerhedsfunktion. Her skal man følge med i det generelle trusselsbillede, samt bestemme strategier for f.eks. awareness.

Den anden del af viden er det, at de enkelte medarbejdere skal vide noget om IT-sikkerheds awareness. Awareness er, at alle ansatte er klar over den risiko, som de potentielt udsætter virksomheden for ved f.eks. ikke at være opmærksomme på potentiel ransomware eller social engineering.

Awareness for udviklere resulterer i, at de følger fastlagte retningslinjer for god og sikker udvikling, samt følger sin sunde fornuft. Udviklere skal også være bevidste om den risiko, de udsætter virksomheden for. Og det er lige netop denne bevidsthed om IT-sikkerhed blandt udviklere, jeg arbejder med i denne rapport.

I bogen [SS] sætter forfatteren viden i system. Forfatteren er bevidst om, at dette kan virke mere som en akademisk øvelse end noget, som er praktisk anvendeligt.

Systemet er at inddele viden i syv kataloger ([SS], side 265-267), som igen kan inddeles i tre kategorier ([SS], side 263 -264).

6.3.1 Seven knowledge catalogs

Principles
Et princip er en udtalelse, som er af generel sikkerhedskarakter. Den oprinder fra erfaring.

Guidelines
En guideline er en anbefaling af ting, man kan gøre, og ting man ikke må gøre for at bygge sikker software. En guideline implementeres med en “menneskelig” analyse.

Rules
En regel er en anbefaling af ting, man skal gøre, og ting man ikke skal gøre for at bygge sikker software. En regel kan implementeres med automatiseret skanning af kode.

Vulnerabilities
En sårbarhed er resultatet af en fejl i software, som kan bruges til at bemægtige sig adgang til eller negativt påvirke et computersystem.

Exploits
Et exploit er en specifik udgave af et angreb på et computersystem, som udnytter en specifik sårbarhed eller en samling af sårbarheder.

Attack Patterns
Et angrebsmønster er et mønster, som er fremkommet ved at analysere store mængder af software exploits.

Historical risks
En historisk risiko er en risiko, som tidligere er identificeret under en normal software udviklingscyclus.

6.3.2 Three knowledge categories

Prescriptive knowledge
Samlet set indeholder denne kategori retningslinjer for, hvad man skal gøre, og hvad man skal undgå for at bygge sikker software.

Den præskriptive viden er derfor en samling af katalogerne: principles, guidelines og rules.

Diagnostic knowledge
I stedet for at være præskriptiv bruges den diagnostiske viden til at genkende og håndtere kendte problemer, som potentielt kan føre til et angreb.

Denne kategori indeholder derfor en samling af katalogerne: attack patterns, exploits og vulnerabilities.

Historical knowledge
Denne kategori er en samling af specifikke risici, som er fundet i praksis. Den bruges til at afdække nye tilsvarende angreb.

Kategorien indeholder kataloget historical risks. Kan også indeholde vulnerabilities.

7 ASVS - Application Security Verification Standard

ASVS er et OWASP-projekt. Selvom OWASP i sig selv er perifert i denne rapports sammenhæng, er den interessant i forståelsen af ASVS's oprindelse. Jeg har derfor valgt at beskrive OWASP ganske kort i afsnit "7.4 OWASP".

ASVS har en sammenhæng til et andet OWASP-projekt, nemlig "TOP 10". Dette projekt beskriver jeg i afsnit "7.1 Top 10", lige inden jeg i afsnit "7.2 ASVS" beskriver ASVS.

Jeg har tidligere i rapporten beskrevet teorien "Three Pillars of Software Security". I afsnittet "7.3 ASVS i relation til Three Pillars of Software Security" vil jeg relatere teorien til ASVS.

7.1 Top 10

Dette projekt er efter min mening et af de vigtigste OWASP-projekter. Man ser ofte referencer til det. Jeg har set "spisesedler" hos kursusudbydere, hvor de tilbyder "Certified Ethical Hacker" kursus med udgangspunkt i OWASP Top 10. Jeg har også oplevet forelæsninger omkring websikkerhed, hvor der refereres til OWASP Top 10. Alt i alt er der rigtig mange, der laver referencer til denne Top 10. Og det giver god mening, da OWASP Top 10 er en liste over de 10 mest kritiske sikkerhedsrisici for webapplikationer.

Den seneste udgave af listen er fra 2013 (OWASP Top 10 - 2013). Der arbejdes lige nu med et initiativ hvor de leder efter kandidater til en ny liste. Den nuværende arbejdstitel er "OWASP top 10 - 2017 Release Candidate". Om den når at blive publiceret i indeværende år, kan man godt være i tvivl om. Men arbejdet er i gang.

Den nuværende liste består af disse 10 punkter:

OWASP Top 10 - 2013
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

([OWASP-2])

Listen er et godt udgangspunkt for at diskutere vigtigheden af IT-sikkerhed med udviklere. Min oplevelse er, at når udviklere ser listen, forstår de vigtigheden af den, men de bliver mere skræmte end handlingsorienterede. Med andre ord, de ved ikke, hvor de skal starte.

OWASP Top 10 er nemlig en liste over ting, du skal undgå at gøre.

7.2 ASVS

ASVS er knapt så kendt som “TOP 10”, men er efter min mening lige så vigtig.
“TOP 10” og ASVS supplerer nemlig hinanden. I stedet for ting man ikke skal gøre, beskriver ASVS ting, man skal gøre.

I ASVS står der følgende:

“... Instead, secure development courses can use the ASVS with a strong focus on the proactive controls found in the ASVS, rather than the Top 10 negative things not to do...”
([OWASP-4], side 17)

I stedet for at vise, hvad man ikke skal gøre, vender ASVS om, og viser hvad man skal gøre. På den måde bliver det mere håndgribeligt, konkret og dermed praktisk anvendeligt.

For at gøre brugen af ASVS mere konkret, vil jeg senere i rapporten bruge ASVS. Se afsnit “8 ASVS brugt på en standard bank”.

Jeg vil i det efterfølgende gennemgå de to væsentligste elementer i ASVS, som er “Application Security Verification Levels” og “Detailed Verification Requirements”

7.2.1 Application Security Verification Levels

En af de første ting, som ASVS påpeger, er at ikke alle systemer behøver at være lige sikre. Derfor defineres tre niveauer: Level 1, Level 2 og Level 3.

Level 1 - Opportunistic
Dette er det laveste niveau. Her er en applikation beskyttet mod sårbarheder, som er lette at opdage, bla. vil de ting som OWASP Top 10 beskriver, være håndteret.

Niveauet vil være passende for applikationer, hvor der ikke er det store behov for at sikre overholdelse af sikkerhedsmekanismer. Dette niveau kan valideres med værktøjer eller manuelt uden adgang til kildekode.

“We consider Level 1 the minimum required for all applications”
([OWASP-4], side 9)

Trusler mod et Level 1 system vil være fra angribere, som udnytter sårbarheder der er lette at finde og nemme at udnytte.

Level 2 - Standard
Dette niveau opnås, hvis applikationen er beskyttet mod hovedparten af de gængse risici.

Niveauet sikrer, at sikkerhedsmekanismer er implementeret korrekt. Niveauet er passende for forretningsapplikationer, som håndterer følsomme og kritiske data.

Angreb mod et Level 2 system vil være fra motiverede og trænede angribere.

Level 3 - Advanced
Dette er det højeste niveau, en applikation kan opnå i ASVS. Her er typisk applikationer, som kræver et meget højt sikkerhedsniveau. I ASVS nævnes militære, sundhed-, sikkerhed- og kritiske infrastrukturapplikationer.

For at opnå dette niveau kræver det meget mere analyse, arkitektur, kode og test end de andre to niveauer. Det er bestemt ikke trivielt at nå Level 3.

Hvilket level skal man så vælge?
For at give en idé om et passende niveau for ens applikation er der i ASVS et skema med udgangspunkt i fire industrier. For hver af disse industrier kan man se hvilken trusselsprofil der er gældende og se ASVS's anbefaling af, hvilke applikationer, der skal være Level 1, Level 2 og Level 3.

Den industrikategorisering der er passende for en bank er: "Finance and Insurance", se tabel 1.

Industry	Threat Profile	L1 Recommendation	L2 Recommendation	L3 Recommendation
Finance and Insurance	Although this segment will experience attempts from opportunistic attackers, it is often viewed as a high value target by motivated attackers and attacks are often financially motivated. Commonly, attackers are looking for sensitive data or account credentials that can be used to commit fraud or to benefit directly by leveraging money movement functionality built into applications. Techniques often include stolen credentials, application-level attacks, and social engineering. Some major compliance considerations include Payment Card Industry Data Security Standard (PCI DSS),Gramm Leech Bliley Act and Sarbanes-Oxley Act (SOX).	All network accessible applications.	Applications that contain sensitive information like credit card numbers, personal information, that can move limited amounts of money in limited ways. Examples include: (i) transfer money between accounts at the same institution or (ii) a slower form of money movement (e.g. ACH) with transaction limits or (iii) wire transfers with hard transfer limits within a period of time.	Applications that contain large amounts of sensitive information or that allow either rapid transfer of large sums of money (e.g. wire transfers) and/or transfer of large sums of money in the form of individual transactions or as a batch of smaller transfers.

Tabel 1 - ([OWASP-4], side 11)

Giver kategoriseringen "Finance and Insurance" de rigtige levels for en bank?
Dette er et ikke helt trivielt spørgsmål at svare på. Svaret er "måske", og "det kommer an på".

Måden ASVS kommer frem til vurderingen, er at der er lavet en trussels profil, hvori der beskrives, at den største trussel er fra økonomisk motiverede angribere, som går efter følsomme data, der kan bruges til at opnå økonomisk gevinst. Efterfølgende kategoriseres systemerne efter, hvor mange sensitive data, systemet indeholder, samt om man kan overføre store summer af penge med systemet.

Dette er for så vidt en helt igennem valid betragtning. Det er bare ikke ASVS der skal lave vurderingen for den enkelte bank. Den enkelte bank skal i en Risk Management proces (se afsnit "6.1 Risk Management") vurdere, hvilken risiko man vil løbe, altså virksomhedens risikoappetit. Og på baggrund af den vurdere hvilket level de enkelte systemer skal verificeres til. ASVS beskriver det derfor også kun som en vejledning:

Organizations are strongly encouraged to look more deeply at their unique risk characteristics bases on the nature of their business.
([OWASP-4], side 10)

7.2.2 Detailed Verification Requirements
Nu kommer vi ind til kernen i ASVS. Det er her de egentlige verifikationer er beskrevet.

Der er 16 kategorier i alt. V1 - V19 (V6, V12 og V14 er udgået). Hver kategori består af en titel. f.eks "V1 Architecture, design and threat modelling" samt tre punkter: "Control objective", "Requirements" og "References".

Control objective
Her beskrives, hvad der opnås.

Requirements
Her listes de enkelte punkter, som skal verificeres. Ud for hvert punkt er listet, hvilket af de tre niveauer (Level 1, Level 2 eller Level 3) de hører til.

References
Her er der beskrevet, hvor yderligere information kan findes.

For at give et overblik over hvor mange kontroller, der er påkrævet for hvert niveau, har jeg lavet en optælling. Den ses i tabel 2.

Et punkt, som kræves for at opnå Level 1, er også krævet for Level 2 og 3.

Et punkt, som kræves for at opnå Level 2, er også krævet for Level 3, men ikke nødvendigvis for Level 1.

Det ses, at det kræver 90 kontroller at verificere Level 1, og godt det dobbelte - 182 - for at opnå Level 3 verifikation. Bemærk, at det udover mængden også er yderligere komplicerede kontroller. Så arbejdsbyrden fra Level 1 til Level 3 er meget større end det dobbelte.

	Level 1	**Level 2**	**Level 3**
V1	1	8	11
V2	17	24	26
V3	11	13	13
V4	7	11	12
V5	10	20	21
V7	2	7	10
V8	3	9	13
V9	4	8	11
V10	7	9	13
V11	6	8	8
V13	0	0	2
V15	0	2	2
V16	7	9	9
V17	7	10	11
V18	7	10	10
V19	1	5	10
I alt	90	153	182

Tabel 2

For at skabe et overblik over indhold og omfang af de 16 kategorier, vil jeg i det følgende ganske kort gennemgå hver enkelt af dem.

V1 Architecture, design and Threat modelling
Fokus er arkitektur, design og trusselsmodellering. Fordelt på de tre levels er formålet:

Level 1
Alle komponenter i applikationen er identificeret og nødvendige.

Level 2
Arkitekturen er defineret, og koden passer til arkitekturen.

Level 3
Arkitektur og design er på plads, i brug og optimeret.

V2 Authentication
Dette er helt klassisk autentifikation. Altså at sikre den som kommunikerer er den han/hun udgiver sig for at være, og at autentifikations credentials bliver udvekslet på en sikker måde.

V3 Session management
En webapplikation er hvis ikke altid, så næsten altid, stateful. Denne state håndteres med sessionsbegrebet.

Det er derfor nødvendigt, at sessioner er unikke og ikke kan gættes eller deles.

Sessioner skal invalideres, når de ikke længere er nødvendige. F.eks. når brugeren logger af et system eller efter en passende inaktivitets periode.

V4 Access control
For at kunne opnå adgang til et system skal man være autoriseret til det. Desuden skal brugere kunne associeres med roller og tilhørende privilegier.

V5 Malicious input handling
Den mest almindelige sårbarhed i webapplikationer er dårlig input validering.
A1 – Injection, fra OWASP Top 10. Et kendt eksempel er SQL injection.

Så her verificeres følgende:

- Alt input er valideret, så det er korrekt og som forventet.
- Eksterne data skal man aldrig stole blindt på.

V7 Cryptography at rest
I denne sektion verificeres implementeringen af kryptografiske funktioner.

- Alle kryptografiske moduler skal fejle sikkert, og fejlhåndtering skal ske korrekt.
- Hvis tilfældige tal skal bruges, skal der bruges en passende tilfældige tals generator.
- Adgang til nøgler skal ske på en sikker måde.

V8 Error handling and logging
Det primære formål med fejlhåndtering og logning er ikke at skabe uanede mængder af logs. Formålet er i en fejlsituation at kunne afhjælpe problemet, samt sikre sig dokumentation i forhold til lovgivning. Eksempelvis for at overholde bogføringsloven.

Resultatet af dette er, at logfiler ofte indeholder en masse sensitive data. De skal derfor beskyttes på tilstrækkelig forsvarlig vis.

- Lad være med at logge følsomme informationer medmindre det er nødvendigt.
- Sørg for at logfiler er beskyttet på passende vis.
- Logfiler skal gemmes så kort tid som nødvendigt.

Vær opmærksom på, at alt efter hvad der logges, kan logfilerne blive et eftertragtet mål for angribere.

V9 Data protection
Dette afsnit omhandler det at data skal behandles efter FIT-modellen.
Se afsnit “5 IT-sikkerhed” for uddybning af FIT-modellen.

V10 Communications
Her verificeres, at applikationen kommunikerer på en tilpas sikker måde.

- TLS bruges, når sensitive data flyttes.
- Stærke algoritmer og ciphers bruges.

V11 HTTP security connections
Her verificeres grundlæggende HTTP parametre. F.eks skal applikationen afvise PUT og DELETE requests, hvis der kun er behov for GET og POST.

V13 Malicious controls
Her skal verificeres, at applikationen ikke indeholder skadelig kode. Dette er sjældent tilfældet, men til gengæld ekstremt svært at se.

V15 Business logic
I dette punkt verificerer man, at applikationen overholder forretningslogikken. Hvis en bruger forsøger at bruge applikationen på en måde som ikke understøtter forretningslogikken, skal den stoppe eksekveringen.

V16 File and resources
Her verificeres at

- ukendte filer skal håndteres på sikker og forsvarlig måde.
- filer, som er hentet fra ukendte kilder, skal gemmes uden for applikationens webroot og med begrænsede rettigheder.

V17 Mobile
Dette afsnit omhandler specielle kontroller for mobile applikationer.

Sådanne kontroller er meget relevante, da der laves flere mobile applikationer end nogensinde.

V18 Web services
Her håndteres REST og SOAP baserede webservices.

De skal have tilstrækkelig autentifikation, autorisation og session management samt input validering.

Se eventuelt afsnittet “3 Motivation” for, hvad der kan ske, hvis man ikke sikrer sine webservices tilstrækkeligt.

V19 configuration
I dette punkt arbejdes med platformen, hvor webapplikationen afvikles.

Her verificeres at

- libraries og platform er opdaterede.
- applikationsserveren har en sikker default konfiguration.

7.3 ASVS i relation til Three Pillars of Software

I foregående afsnit er ASVS beskrevet som en mængde af kontroller, der gør, at man opnår sikre systemer.

Det relaterer sig direkte til byggestenen “Knowledge”. I relation til “Knowledge” vil jeg kategorisere ASVS som præskriptiv viden. Altså forebyggende viden, som indeholder retningslinjer for at bygge sikker software.

At bruge ASVS som fundament for IT-sikkerheds awareness giver derfor god mening.

7.4 OWASP

OWASP (Open Web Application Security Project) blev grundlagt i 2001. I bilag 13.1 ses indholdet af den oprindelige mail, som startede OWASP.

OWASP er et åbent fællesskab (open community) dedikeret til at sætte virksomheder og organisationer i stand til at konstruere, udvikle, købe, drifte og vedligeholde sikre applikationer.

Når man betragter OWASP udefra er det mest af alt en struktur og organisering, de tilbyder. Hvis man har en god ide til et projekt, som man finder værdigt til OWASP, er det bare at byde ind. Hvis man har lyst til at bidrage til et eksisterende projekt, er det også bare at byde ind.

De udtrykker det ambitiøst med disse kerneværdier og kerneformål:

Core Values

- OPEN
    - Everything at OWASP is radically transparent from our finances to our code.
- INNOVATION
    - OWASP encourages and supports innovation and experiments for solutions to software security challenges.
- GLOBAL
    - Anyone around the world is encouraged to participate in the OWASP community.
- INTEGRITY
    - OWASP is an honest and truthful, vendor neutral, global community.

Core Purpose

- Be the thriving global community that drives visibility and evolution in the safety and security of the world's software.

([OWASP-1])

OWASP er organiseret omkring projekter. Projekterne kategoriseres efter formål i tools, code eller documentation.

Det er meget naturligt, at et projekt ikke nødvendigvis har modenhed fra start. Det skal jo udvikles og forædles. Så derfor kategoriseres projekterne efter modenhed og værdi i disse tre kategorier: "OWASP Incubator Projects", "OWASP Labs Projects" og "OWASP Flagship Projects"

Jeg vil kort gennemgå de tre kategorier.

“OWASP Incubator Projects”
Dette beskriver OWASP som en eksperimentel legeplads. Det er her hvor ideerne og tankerne bag projektet ikke er bevist endnu. Der er stadigvæk lang vej til et modent og færdigt projekt.

“OWASP Labs Projects”
Her er projekter, som har produceret værdi. Værdien er bevist via OWASP review.

“OWASP Flagship Projects”
Her er projekter, som demonstrerer strategisk værdi til OWASP og applikationssikkerhed som et hele.

Et hurtig sammentælling på deres hjemmeside af de forskellige kategorier:

- “OWASP Incubator Projects” - ca 70 projekter
- “OWASP Labs Projects” - ca 30 projekter
- “OWASP Flagship Projects” - 12 projekter

Denne måde at inddele i kategorier giver god værdi. Hvis de ikke havde gjort det, tror jeg ikke, at deres “Flagship Project” havde opnået den seriøsitet, som det har.

De tolv “Flagship Projects” er pt.:

Tools

- OWASP Zed Attack Proxy
- OWASP Web Testing Environment Project
- OWASP OWTF
- OWASP Dependency Check

Code

- OWASP ModSecurity Core Rule Set Project
- OWASP CSRFGuard Project
- OWASP AppSensor Project

Documentation

- OWASP Application Security Verification Standard Project
- OWASP Software Assurance Maturity Model (SAMM)
- OWASP AppSensor Project
- OWASP Top Ten Project
- OWASP Testing Guide Project

8 ASVS brugt på en standard bank

I dette afsnit vil jeg bruge ASVS på min standard bank for på den måde at undersøge, hvor effektiv ASVS er som fundament for awareness.

Inden jeg starter, er det nødvendigt at vide, hvilke parametre jeg vil vurdere ASVS efter. Med andre ord, hvilke parametre er vigtige for at skabe awareness for en udvikler?

Jeg vælger at vægte på disse tre parametre: relevans, sværhedsgrad og genbrugelighed.

Relevans:
Det skal være relevant for udvikleren. De må ikke føle, at de spilder deres tid. Udviklerne føler, at deres primære opgave er at producere applikationer. Hvis de ikke har opfattelsen af, at deres tid bliver brugt fornuftigt, er slaget tabt på forhånd.

Men der kan være en manglende opfattelse af, hvad en udvikler synes er relevant, og hvad ledelsen synes er relevant for udvikleren. Der vil i processen komme diskussioner om, hvorvidt en kontrol er relevant eller ej. Min erfaring er at udviklere gerne vil lave sikre løsninger, men de skal naturligvis forstå it-sikkerheds problemstillingen, inden de vil betragte den som relevant for dem. I processen med at bruge ASVS, skal der være tid til at forklare, hvorfor en kontrol er vigtig. Man kan ikke forvente at alle udviklere har kendskab til vigtigheden af en kontrol fra start af.

Sværhedsgrad:
Hvis et emne er for svært i forhold til ens aktuelle viden, kan motivationen forsvinde. Alt, man lærer, skal være på et passende niveau. Hvis en udvikler, der har fuldstændig styr på kryptering, bliver tvunget til at gennemgå trivielle spørgsmål omkring kryptering, er udviklerens motivation væk. Omvendt vil en udvikler uden viden om kryptering føle sig unødvendigt presset, hvis man forventer, at han ved alt om nøglehåndtering.

Her skal man være bevidst om, hvad man vil opnå.

Mit fokus er at opnå øget awareness, og det er derfor en balancekunst at have fokus på den enkelte udviklers kompetenceniveau. Den enkeltes awareness niveau skal bibeholdes, hvis den er tilpas, eller forøges, hvis den er under forventet niveau. Man kan argumentere for at verifikationen af den enkelte applikation er underordnet.

Hvis fokus derimod er at opnå en verifikation af applikationen, og som et biprodukt opnå awareness blandt udviklerne, bliver det straks mere kompliceret. Kan man sætte en udvikler med en høj awareness til at lave helt simple kontroller? Faren er, at udvikleren mister fokus, fordi det simpelthen er for elementært og trivielt. Hvis det er tilfældet, kan man komme i en situation, hvor man i alle afdelinger skal have nogen ansat som har et lavt awareness niveau. Et slags A- og B-hold i forhold til IT-sikkerheds awareness, for at være sikker på korrekt ASVS verifikation af applikationerne.

Genbrugelighed:
Her er der ikke tale om genbrug af kode, men om genbrug af materialet til awareness undervisningen. Det vil lette arbejdet betydeligt jo mere materialet kan genbruges.
F.eks. vil et materiale omkring generel brug af certifikater være meget genbrugeligt, da det er relevant for alle udviklere. Mens et materiale omkring valg af cipher suites vil være mindre genbrugeligt, da det højst sandsynligt kun er en lille gruppe af infrastruktur-udviklere, det er relevant for.

8.1 Klassifikation

Inden man starter med verifikationerne opdager man den første udfordring. Verifikationen er nemlig afhængig af, om det er en Level 1, Level 2 eller Level 3 verifikation, man har som mål.

I afsnit “9.1 Klassifikation” gennemgås udfordringen mere detaljeret.

Min Standard bank har to applikationer: KontoApp og ValutaApp.

KontoApp bruges til at flytte penge mellem konti. Alt efter mulighederne for at flytte små eller store summer vil applikationen skulle verificeres til Level 2 eller Level 3, jvf. tabel 1 side 24.

ValutaApp er nærmest at betragte som en offentligt applikation samt en privat administrationsdel. Ingen af disse kan bruges til at overføre penge, så derfor skal ValutaApp kun verificeres til Level 1, jvf. tabel 1 side 24.

Når man skal verificere til level 2 og level 3 skal man til at kigge ind i applikationen. Jeg har ikke lavet en implementering af min standard bank. Derfor foretager jeg et bevidst valg og vælger udelukkende at kigge på ValutaApp applikation i udførelse af verifikationen.

Det giver ikke mening at vurdere klassifikation ud fra de tre parametre relevans, sværhedsgrad og genbrugelighed. Det er nemlig ikke udvikleren der skal klassificere applikationen. Det bør være forretningen, der gør det.

8.2 Afgrænsning af kontroller

Der kræves i alt 90 kontroller for at opnå Level 1 verifikation. Formålet med at bruge ASVS på standard banken er ikke at opnå en Level 1 validering, men derimod at vise om ASVS kan bruges til at opnå awareness med. Det vil derfor være uinteressant at gennemføre samtlige 90 kontroller. I stedet udvælger jeg nedenstående kontroller.

8.2.1 - V1 Architecture, design and threat modelling

Denne kontrol er taget med, fordi den tydeligt viser omfanget af bare en enkelt kontrol. Kontrollen er vigtig, men er faktisk svær at opfylde i en hverdag med mange ældre applikationer.

8.2.2 - V2 Authentication

Denne er efter min mening den vigtigste kontrol af dem alle. Autentifikation er grundpillen i adgangsstyring i en webapplikation. Hvis ikke autentifikationen fungerer, kan man hverken autorisere eller logge korrekt.

8.2.3 - V8 Error handling and logging

Dette emne bliver ofte overset. Det er en stor fejl. Forkert fejlhåndtering i en webapplikation kan gøre utrolig stor skade. Et stacktrace kan f.eks. give information om applikationsserveren, således at et angreb bedre kan målrettes. Fejlagtig eller manglende logning er også kritisk.

8.2.4 - V10 Communications

Brugen af HTTPS protokollen er en af de ting, som udviklere finder sværest. Det er til gengæld et af de emner, jeg nyder at lære fra mig i min hverdag. Så disse kontroller er taget med, fordi jeg synes det er utroligt spændende, og fordi det er relevant i forhold til den awareness træning, jeg allerede er aktiv med.

8.3 - V1 Architecture, design and Threat modelling

Her er der kun et krav for at opnå level 1 verifikation:

8.3.1 - 1.1 Verify that all applications components are identified and are know to be needed.

([OWASP-4], side 20)

I tilfældet med ValutaApp applikationen fra min standard bank forestiller jeg mig en simpel og relativ ny applikation. Der er derfor kontrol over applikationens elementer. Hvis applikationen havde været kompleks og gammel, ville dette krav ikke nødvendigvis være så let at opfylde. En ældre applikation, som har været vedligeholdt af flere forskellige personer gennem tiden, kan let blive ustruktureret. At skulle sige den kun indeholder identificerede og nødvendige komponenter vil derfor være svært. Omvendt vil det være en god lejlighed til at påpege nødvendigheden af en refaktorering af koden.

Efter at have arbejdet med dette krav, opstår to nye udfordringer.

Kravet er åbenlyst, men kunne og burde det ikke være kendt på forhånd af udvikleren? Et krav til applikationen bør ikke opstå i forbindelse med en verifikation. Såvel funktionskrav som sikkerhedskrav skal være kendte inden en applikation udvikles. Og de skal også være kendte i forbindelse med vedligehold af applikationen.

I afsnit “9.2 Sikkerhedskrav” gennemgås denne udfordring mere detaljeret.

En anden udfordring er omkring kravet, at koden kun må indeholde nødvendige komponenter. Inden for OO programmering er der meget genbrug af kode. Hvis man genbruger kode, kan og skal man så være sikker på, at koden kun indeholder nødvendige komponenter?

I afsnit “9.3 Genbrug af kode” gennemgås denne udfordring mere detaljeret.

<u>Relevans</u>
Min vurdering er, at relevansen er høj. De fleste vil kunne forstå, at applikation kun skal indeholde det nødvendige kode. Der vil dog være nogle udviklere, som skal arbejde anderledes, da de oftere end andre bruger 3. parts kode.

<u>Sværhedsgrad</u>
Det kræver ikke sikkerhedsviden at udføre denne kontrol, men det kræver stor viden og kendskab til applikationen. Det vil ikke være alle, der vedligeholder en applikation, som kan udføre denne kontrol.

<u>Genbrugelighed</u>
Kontrollen er overordnet set meget genbrugelig og let at forklare.

8.4 - V2 Authentication

Autentifikation er en ekstrem vigtig ting for sikkerheden i en webapplikation. Det er derfor ikke overraskende, at dette er den verifikation, som indeholder flest kontroller. Jeg vil i det følgende gennemgå de 17 kontroller, der er for at opnå Level 1 verifikation.

8.4.1 - 2.1 Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation).

([OWASP-4], side 22)

Her vil det være let at verificere, hvad default settings er for ValutaApp applikationen. ValutaApp applikation indeholder både en offentlig del og en privat del. Hvis man ikke har en default værdi, som gør, at alle sider kræver autentifikation, vil alle nye dele være offentlige. Det er derfor bedst eksplicit at angive, at siden skal være offentligt.

En del settings af den slags bør ligge i rammeværket, som en standard template, der bruges ved oprettelse af nye applikationer. Men stadigvæk kan der være dele af en applikation, som ikke kræver autentifikation. Det er derfor vigtigt, at udviklerne kender principperne.

Relevans

Relevansen for udviklere er høj. Det giver mening.

Sværhedsgrad

Sværhedsgraden er lav. For langt de fleste vil det være trivielt at udføre kontrollen. Hvis det er en ældre applikation, kan det dog være omfangsrigt at skulle rette default værdien.

Genbrugelighed

Denne kontrol er meget genbrugelig.

8.4.2 - 2.2 Verify that forms containing credentials are not filled in by the application. Pre-filling by the application implies that credentials are stored in plaintext or a reversible format, which is explicitly prohibited.

([OWASP-4], side 22)

Dette bør ikke ske. Credentials må aldrig gemmes i klar tekst.

ValutaApp applikationen antages ny og simpel. En hurtig kontrol vil kunne bekræfte, at den ikke indeholder kodeord eller andre credentials i forms.

Relevans

Relevansen for udviklere er høj. Det vil dog ikke være en af de første punkter, jeg tager op med en udvikler. Mange ville blive fornærmede, hvis jeg spurgte dem om dette, da det er meget trivielt.

Sværhedsgrad
Sværhedsgraden er lav. Hvis det er en stor applikation, kan det dog være omfangsrigt at skulle verificere. Hvis applikationen er bygget over gode templates, bør dette ikke kunne lade sig gøre, og kontrollen er hurtigt gennemført.

Genbrugelighed
Denne kontrol er meget genbrugelig.

8.4.3 - 2.4 Verify all authentication controls are enforced on the server side.
([OWASP-4], side 22)

ValutaApp kræver kun autentifikation af administrationsdelen.

Som jeg forestiller mig det, vil dette kun være tilfældet for ældre JavaScript tunge webapplikationer. Nye applikationer, som bruger et fornuftigt rammeværk, bør ikke kontrollere autentifikation på klienten. Applikationen bør være opbygget, så det er let at se, om der udføres autentifikation på klienten. En søgning i koden efter navne som password, kodeord og lignende vil være en god start. Se også efter om der læses eller skrives til cookies fra javascript. Hvis der gør, vurder om det kan have noget med autentifikation at gøre.

Selve autentifikationen sker ikke i webapplikation, men i reverse proxyen. Så denne kontrol kan ikke udføres alene ved at kigge på ValutaApp. Der er også brug for at kigge på reverse proxyen. Den bliver typisk håndteret af infrastrukturudviklere.

Relevans
Stor relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Meget let at forstå ud fra en sikkerhedsmæssig betragtning. Er let at finde for en applikationsudvikler. Hvis koden indeholder autorisation på klient siden, kan det dog kræve stor forståelse af hele applikationen for at kunne rette.

Genbrugelighed
Denne kontrol er meget genbrugelig for applikationsudviklere. Men for at have effekt skal den også håndteres i forhold til reverse proxyen. Den bliver typisk håndteret af en infrastruktur afdeling, og det er derfor nødvendigt at gennemgå ASVS med flere afdelinger.

Denne udfordring gennemgås mere detaljeret i afsnit “9.4 Målgruppe”.

8.4.4 - 2.6 Verify all authentication controls fail securely to ensure attackers cannot log in.

([OWASP-4], side 22)

Her vil opgaven være at finde alle steder i koden, hvor autentifikation håndteres. Kontroller efterfølgende at applikation fejler på en sikker måde, hvis autentifikation ikke er tilfredsstillet.

For reverse proxyen er opgaven større, da det er her selve autentifikation udføres.

Relevans
Stor relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
For applikationsudvikleren er opgaven let at forstå og let at håndtere.

For infrastrukturudvikleren er opgaven let at forstå, men svær at udføre.

Men opgaven får en anden dimension, da opgaven er et samarbejde mellem applikationsudviklere og infrastrukturudviklere. Der skal testes på livet løs, og det skal sikres, at applikationen ikke tillader adgang, hvis reverse proxyen får invalid input.

Genbrugelighed
Denne kontrol er meget genbrugelig for applikationsudviklere. Men bliver specifik i forhold til infrastruktur.

8.4.5 - 2.7 Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent password managers, long passphrases or highly complex passwords being entered.

([OWASP-4], side 22)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen, da det er den, som redirecter til en login side.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.6 - 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.

([OWASP-4], side 22)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.7 - 2.9 Verify that the changing password functionality includes the old password, the new password, and a password confirmation.

([OWASP-4], side 22)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.8 - 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.

([OWASP-4], side 23)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Så kontrollen af, at brugerens indtastede kodeord overføres med en krypteret metode, giver ikke mening. Men det giver mening at kontrollere, hvordan webapplikationen kommunikerer med f.eks. databaser. Altså hvis webapplikationen autentificerer sig mod databasen, skal kommunikationen være krypteret.

Relevans
Stor relevans.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Denne kontrol er meget genbrugelig for applikationsudviklere. Men bliver specifik i forhold til infrastruktur.

8.4.9 - 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.

([OWASP-4], side 23)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.10 - 2.18 Verify that information enumeration is not possible via login, password reset, or forgot account functionality.
([OWASP-4], side 23)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.11 - 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").
([OWASP-4], side 23)

Det er let at kigge applikationen igennem efter konfigurationsfiler som indeholder kodeord. Så den kontrol udføres hurtigt. Opgaven bliver mere omfangsrig, når der skal kigges på rammeværk og komponenter.

Kontrollen er afhængig af kontrollen "V1 Architectur, design and threat modelling", hvor alle komponenter identificeres.

Relevans
Stor relevans.

Sværhedsgrad
Kontrollen er ikke svær, men bliver hurtigt omfangsrig.

Genbrugelighed
Umiddelbar høj genbrugelighed, men da webapplikationer bruger forskellige kompnenter, kan den blive kompliceret at genbruge.

8.4.12 - 2.20 Verify that anti-automation is in place to prevent breached credential testing, brute forcing, and account lockout attacks.
([OWASP-4], side 23)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.13 - 2.22 Verify that forgotten password and other recovery paths use a TOTP or other soft token, mobile push, or other offline recovery mechanism. Use of a random value in an e-mail or SMS should be a last resort and is known weak.
([OWASP-4], side 23)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.14 - 2.24 Verify that if shared knowledge based questions (also known as "secret questions") are required, the questions do not violate privacy laws and are sufficiently strong to protect accounts from malicious recovery.
([OWASP-4], side 23)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.15 - 2.27 Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases.
([OWASP-4], side 24)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.4.16 - 2.32 Verify that administrative interfaces are not accessible to untrusted parties.

([OWASP-4], side 24)

Tit er fokus på hoved funktionalitet. Altså i tilfældet med ValutaApp applikationen i standard banken, at brugere kan få oplyst kurser på valuta. Men hvordan kommer data ind i systemet? Der er også en administrationsdel af applikationen, som også skal være beskyttet. Så i denne kontrol sikres, at alle dele af applikationen er tilstrækkeligt beskyttet.

Hvis applikationerne har en fælles struktur for administrative interfaces, kunne dette løses med et rammeværk, som bruges, når nye applikationer oprettes. F.eks hvis alle applikationer har en url, som hedder /admin, kunne rammeværket sikre, at /admin altid er beskyttet. Men da det er i applikationen, man definerer, hvilke grupper, der skal have adgang til /admin, er det svært at lave en skabelon, som sikrer, at der ikke gives adgang til en forkert gruppe.

Relevans
Høj relevans. Virkelig vigtig kontrol.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Høj genbrugelighed.

8.4.17 - 2.33 Browser autocomplete, and integration with password managers are permitted unless prohibited by risk based policy.

([OWASP-4], side 24)

Den enkelte webapplikation udfører ikke nogen kodeordskontrol. Dette håndteres i reverse proxyen.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.5 - V8 Error handling and logging

I det følgende vil jeg gennemgå de tre kontroller, det kræver at opnå Level 1 verifikation.

8.5.1 - 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.

([OWASP-4], side 36)

Hvor simpel denne kontrol end lyder, er den faktisk ret svær at gennemføre. Den slags meddelelser skal man være forsigtige med at give videre til slutbrugeren. Hvis f.eks et kodeord er forkert, kan man logge i system logs, at kodeordet var forkert. Slutbrugeren skal kun have at vide, at brugernavn eller kodeord er forkert. På den måde viser man ikke, at brugernavnet er korrekt, og det bare er at gætte kodeordet.

For ValutaApp applikationen vil kontrollen kræve to ting:

- Gennemgå applikationen for logning og dermed kontrollere hvad der logges.
- Gennemgå applikationen og sikre, at alle fejlsituationer håndteres forsvarligt.

Det ville være let, hvis reverse proxyen kunne sikre det, men det kan den ikke. En HTML-side, som indeholder et stacktrace, er jo bare en HTML-side med et indhold. Siden kunne jo lige så godt være bestemt for en slutbruger.

Der er desværre ingen anden udvej end at sikre applikationens logning, og infrastrukturens settings, således at sådan noget aldrig vises.

<u>Relevans</u>
Høj relevans. Virkelig vigtig kontrol.

<u>Sværhedsgrad</u>
Lav sværhedsgrad, men kræver stort applikations kendskab.

<u>Genbrugelighed</u>
Denne kontrol er meget genbrugelig.

8.5.2 - 8.10 Verify that an audit log or similar allows for non-repudiation of key transactions.

([OWASP-4], side 37)

I de systemer jeg har arbejdet med, har log-funktionaliteten været behandlet af en infrastruktur afdeling. På den måde har de enkelte applikationsudviklere ikke skulle tage stilling til, om deres logninger er uafviselige. Uafviselig overfor tredjepart kan dog være vanskelig.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Stor sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.5.3 - 8.13 Time sources should be synchronized to ensure logs have the correct time.

([OWASP-4], side 37)

Denne kontrol synes jeg ikke er så vigtig. Det skyldes nok, at jeg altid har arbejdet steder, hvor samtlige servere har været tidsmæssig i synk. Men hvis man har logfiler fra forskellige komponenter, og disse tidsmæssigt ikke er synkroniserede, er anvendeligheden reduceret.

Relevans
Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad
Lav sværhedsgrad, men kan være kompleks, da den er på tværs af infrastruktur.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.6 - V10 Communications

TLS / SSL er et svært emne for mange udviklere. Min erfaring siger, at der er to hovedgrunde til, at det er så svært.

Den første grund er, at det består af mange elementer:

- certifikat
- PKI
- signatur
- nøgler
- revokation
- keystores
- cipher suites
- handshake
- etc.

.

Ulogiske elementer:

- Der findes ikke nøglepar i vores hverdag, så hvordan kan man forholde sig til dem?

Men når alle elementerne først falder på plads, har jeg hørt flere udtale, at det jo ikke er svært. Bare kompliceret.

Her gennemgås de syv kontroller det kræver for at opnå Level 1 verifikation.

8.6.1 - 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.

([OWASP-4], side 41)

For at udføre denne kontrol skal man vide, hvor de enkelte TLS-sessioner initieres og termineres. Ofte ser man som applikationsudvikler ikke, at applikationen selv kaldes over HTTPS. Det håndteres af infrastrukturen.

Men når applikationen selv kalder f.eks webservices bliver man nødt til at håndtere det. Så hvis ValutaApp applikationen udfører HTTPS kald, skal alle rodcertifikater verificeres.

Relevans

Høj relevans. Dette er vigtigt og meget grundlæggende.

Sværhedsgrad

Høj sværhedsgrad. Min erfaring siger mig, at emnet bare er svært.

Genbrugelighed

Høj genbrugelighed. Meget generelt.

8.6.2 - 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.

([OWASP-4], side 41)

Denne kontrol kræver stor viden om, hvilke forbindelser applikationen laver, samt hvilke data der overføres. Normalt vil jeg anbefale altid at bruge TLS, uanset om det er sensitive data eller ej. På den måde sikres at sensitive data altid overføres over TLS.

Når det er sikret, at de forskellige forbindelser sker over TLS, kommer øvelsen i at sikre forhandlingen af cipher suites. Altså, at det altid er den stærkeste cipher suite, som vælges, og at svageste cipher suite er tilstrækkelig stærk. Valg af cipher suites bør være noget, som styres fra centralt hold. Eventuelt med et rammeværk.

Relevans
Høj relevans omkring TLS-forbindelser. Mindre relevans omkring cipher suites.

Sværhedsgrad
Lav sværhedsgrad omkring TLS-forbindelser. Høj sværhedsgrad omkring cipher suites.

Genbrugelighed
Høj genbrugelighed. Generel problemstilling.

8.6.3 - 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

([OWASP-4], side 41)

HSTS er en response header. Det er en god ting at få den sat, således at alt trafik forbliver krypteret til applikationen. Ofte er dette ikke noget, den enkelte applikationsudvikler angiver, men noget som angives af et rammeværk, der er under infrastrukturkontrol.
Men det er utroligt let at kontrollere. Start ValutaApp applikationen op i en browser, og start et debug værktøj og kontroller om HSTS-headeren er angivet.

Relevans
Jeg vurderer denne til mellem relevans. Det vil være fornuftigt for udviklerne at kende til HSTS, men det er typisk en infrastruktur, som håndterer denne header.

Sværhedsgrad
Lav sværhedsgrad.

Genbrugelighed
Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.6.4 - 10.13 Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic.

([OWASP-4], side 42)

Simpelt og alligevel svært. Dette er mere en infrastrukturopgave end en udvikleropgave. Undersøg hvilke cipher suites, der kan komme i spil, og vælg alle fra, som ikke understøtter forward secrecy. I praksis vil det sige tillad kun Diffie-Helmann eller Elliptic curve synkron nøgleudveksling. Min erfaring siger, at man skal være varsom med at ændre cipher suites. Der skal en grundig test til for at sikre, at alle klienter fortsat kan kommunikere med serveren.

Relevans

Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad

Høj sværhedsgrad.

Genbrugelighed

Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.6.5 - 10.14 Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.

([OWASP-4], side 42)

Alle steder i applikationen, hvor der initieres TLS-forbindelser, skal det kontrolleres, at applikationen eller rammeværket tjekker, om certifikatet er revoket. Det er en grundlæggende kontrol i brugen af certifikater. Hvis en privat nøgle tilhørende et certifikat kompromitteres, er den eneste mulighed, man har at revoke certifikatet. Det er derfor meget dårligt, hvis brugeren af certifikatet ikke tjekker, om det er validt.

Relevans

Høj relevans. Dette er vigtigt og meget grundlæggende.

Sværhedsgrad

Høj sværhedsgrad. Min erfaring siger mig, at emnet bare er svært.

Genbrugelighed

Høj genbrugelighed. Meget generelt.

8.6.6 - 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.

([OWASP-4], side 42)

Denne opgave er meget svær. Et rammeværk, som håndterer dette, vil være væsentligt at have implementeret. Så kan kontrollen være at sikre, at alle forbindelser håndteres gennem rammeværket.

Rammeværket skal sikre, at nøglelængder er tilstrækkelige, og at hash-algoritmer er forsvarlige. Og det er hele certifikatkæden, det drejer sig om. Sådan et rammeværk skal holdes ved lige. Der skal tages højde for, at sikre cipher suites bliver usikre. F.eks SHA-1. ([Version2-2]). Der skal tages højde for, at TLS 1.0 på et tidspunkt ikke må bruges længere.

Relevans

Lav relevans. Til trods for, at dette er vigtigt, vurderer jeg relevansen til lav, da en udvikler typisk ikke vil finde det relevant.

Sværhedsgrad

Høj sværhedsgrad.

Genbrugelighed

Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

8.6.7 - 10.16 Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure.

([OWASP-4], side 42)

En svær opgave som ofte overses. I foregående kontrol beskrev jeg vigtigheden af at have et rammeværk, som håndterer TLS-forbindelser. Rammeværket skal håndteres af en infrastrukturfunktion. Men ret beset er det en IT-sikkerhedsfunktion, som skal udstikke retningslinjerne for, hvad der er forsvarlige konfigurationer. Og det skal ske regelmæssigt på baggrund af opdateret viden.

Denne udfordring gennemgås mere detaljeret i afsnit “9.4 Målgruppe”.

Relevans

Meget lille relevans for applikationsudviklere. Stor relevans for infrastrukturudviklere.

Sværhedsgrad

Høj sværhedsgrad.

Genbrugelighed

Lav genbrugelighed, da kontrollen er meget specifik i forhold til infrastruktur.

9 Udfordringer med at bruge ASVS på en standard bank

I afsnittet “8 ASVS brugt på en standard bank” opstod der fire væsentlige udfordringer. Disse udfordringer vil jeg håndtere i det efterfølgende. Hver udfordring samt en løsning vil blive beskrevet.

9.1 Klassifikation

Når man skal bruge ASVS, skal man vælge, hvilket level man vil opnå verifikation til.
Så hvis man vil bruge ASVS efter hensigten, skal alle applikationer kategoriseres som enten Level 1, Level 2 eller Level 3.

OWASP har lave en guide, hvor det er funktionalitet, som afgør, hvilket level der skal vælges. Se tabel 1 på side 24. Det er efter min mening for simplificeret.

Den rigtige måde at vælge level på vil være i forbindelse med risikovurderingen af de enkelte applikationer.

Så for at kunne bruge ASVS korrekt skal vi have udført en risikovurdering af applikationen, således at vi kan kategorisere den. Men det er ikke nok.

Som demonstreret med standard banken, afhænger en applikation af flere infrastrukturkomponenter. Hver enkelt af disse komponenter, skal kunne understøtte det level, som applikationen er kategoriseret til.

Hvis man som i standard bank eksemplet har to applikationer med forskellig klassifikation, der er afhængig af de samme infrastrukturkomponenter, har man to valg:

Den første løsning vil være at opdele infrastrukturen i grupper. F.eks. kunne man lave tre forskellige database servere, som opfylder forskellige krav. Denne løsning ser man sjældent, da det vil kræve mange ressourcer at drifte den en løsning.

Den anden løsning er at sikre, at alle infrastrukturkomponenter overholder det højeste applikationskrav. Så i tilfældet med standard banken skal de komponenter, som håndterer Level 1 applikationen (ValutaApp), opfylde de samme krav, som Level 2 applikationen (KontoApp) kræver.

Jeg vil anbefale sidstnævnte løsning. Det er også den jeg oftest ser.t.

9.2 Sikkerhedskrav

Denne udfordring opstod i forbindelse med verifikation af en “V1 Architecture, design and threat modelling” kontrol. Men den er bestemt ikke unik for denne kontrol. Det er samme udfordring for alle kontroller.

Udfordringen er, at kontrollerne udover at tjene et formål med at opnå en verifikation, jo er et reelt krav til applikationen. Dette krav skal en udvikler selvfølgelig overholde, når han/hun laver applikationen. Udviklere vil gerne lave sikre applikationer, men de vil naturligvis også gerne kende kravene til applikationer, inden de laver dem.

Så i stedet for at være reaktiv og først verificere applikationen, når den er implementeret, vil det give værdi at forankre ASVS i allerede eksisterende sikkerhedskrav. På den måde kan processen flyttes fra at være reaktiv til at være proaktiv.

I teorien om Knowledge er der to kataloger, hvor denne viden skal forankres. Det er Guidelines og Rules.

Måden hvorpå viden kan forankres, er enten ved at omskrive viden, så de passer ind i eksisterende materiale eller specificere, at software skal overholde de krav, som ASVS kræver.

I softwarens livscyclus (SDLC - Software Development Life Cycle) vil punktet Security Requirements fra de syv touchpoints så sikre at software, som udvikles eller vedligeholdes, nu overholder de krav, som ASVS stiller.

Hvis ikke denne indsats gennemføres, vil det kun være applikationer, hvor ASVS verifikation gennemføres, som vil få løftet sikkerhedsniveauet.
Over tid vil viden fra verificerede applikationer selvfølgelig spredes, fordi udviklere flytter mellem applikationer. Men en helt ny applikation, som udvikles af nye udviklere, vil kun overholde kravene til en ASVS verifikation, hvis viden er forankret som et sikkerhedskrav.

9.3 Genbrug af kode.

Denne udfordring opstod i forbindelse med verifikation af, at koden kun må indeholde nødvendige komponenter. Men hvad er nødvendige komponenter? Jeg ser to slags kode, som det er nødvendigt at forholde sig til genbrug af: tredjeparts kode og fælles kode.

Tredjeparts kode:
Her er der tale om kode, som ikke produceres i firmaet. Der kan være tale om kode, som købes eller kode, som bare deles.

Kode som købes, mener jeg, at der oftest er kontrol over. Typisk vil man have en kontrakt med leverandøren og en aftale om vedligeholdelse af koden.

Kode som er offentligt tilgængelig er straks mere vanskeligt. Hvis koden hentes, har man mulighed for visuelt at inspicere den inden idriftsættelse. Men det kan være en meget uoverskuelig ting at gøre. Nogle gange ved man som udviklere ikke engang hvilken tredjeparts kode, der bruges. Et eksempel herpå, er brugen af JavaScript funktionen left-pad distribueret fra økosystemet NPM. En udvikler valgte at fjerne en funktion han havde delt via NPM. Efterfølgende kunne applikationer, som brugte funktionen, ikke bygges.
([The Register-1]).

Man bør have en klar holdning til tredjeparts kode. Vil man tillade brugen af NPM? Det kommer an på ens risikoappetit. For langt de fleste virksomheder vil jeg ikke mene, at det er forsvarligt at bruge kode fra ukendte tredjeparter, og slet ikke uidentificeret kode. Hvis man alligevel vælger at gøre det, skal der være nogle kompenserende foranstaltninger. F.eks. værktøjer til automatiseret kodeanalyse.

Fælles kode:
Fælles kode kan være en styrke, men også en svaghed. Jeg deler fælles kode op i to grupper. Den ene er kode som “bare” genbruges. Den anden gruppe er deciderede rammeværk.

Om koden skal genbruges eller ej, er en evig diskussion. Der er ikke noget entydigt svar på, om det er godt eller skidt. Der er både fordele og ulemper ved at gøre det.

<u>Fordele</u>
Man sparer tid ved at genbruge kode, da man ikke skal skrive så mange linjer kode.

Koden man genbruger bør have en høj kvalitet. Den er testet af mange.

Fejlrettelser i kode, der genbruges, rammer bredt. Giver generelt mere robust kode.

Ulemper
Kode, der skal genbruges, har en tendens til at blive overkompliceret. Det skal kunne bruges i flere situationer. Hvis koden, som genbruges, bliver for kompliceret, kan det være vanskeligt at overskue, og man kan argumentere for at en simpel og mere specialiseret løsning vil være bedre.

For at kode, der genbruges, er berettiget til at blive genbrugt, skal det have en høj kvalitet. Det skal derfor testes rigtig meget. Måske af en central funktion. Man skal derfor være bevidst om, at denne høje kvalitet har en pris.

Fejlrettelser og ny funktionalitet i kode, der genbruges, kan skabe en bølge af vedligehold. Alle steder, hvor koden bruges, skal koden bygges og testes. Så selv om koden, der genbruges, er testet og derfor har en høj kvalitet, undgår man ikke at teste sin egen kode. Tværtimod skabes der et endnu større behov for test.

Rammeværker:
Et rammeværk er en specialiseret form for genbrug af kode. Der er dog den forskel, at et rammeværk typisk omhandler en kompliceret funktionalitet. Hvis man bruger rammeværket, er det derfor ikke nødvendigt at forstå den komplicerede funktionalitet. Man indpakker viden sammen med koden.

Fordele
Man sparer tid ved udvikling, da det ikke er alle, der skal have specialist viden.
Man opnår en mere robust implementering, da rammeværket tager hånd om f.eks. alle fejlsituationer.

Ulemper
Den største ulempe jeg ser ved brug af rammeværker er, når vidensafkoblingen er så stor, at de, som bruger rammeværket, ikke ved, hvad rammeværket gør. Hvis f.eks. et rammeværk sikrer den rigtige brug af kryptering, mister udvikleren viden om vigtigheden af krypteringen. Min frygt er, at awareness blandt udviklere falder, hvis viden om rammeværkets funktionener gemmes af vejen.

Løsning
Det er et stort problem at navigere i ovenstående landskab. Den eneste farbare vej er at have klare beskrivelser af, hvad der er accepteret. Der skal være kendte krav til genbrug af kode. Kravene skal udfærdiges i et samarbejde mellem infrastruktur- og sikkerhedsfunktionen.
Derudover er det så let at implementere tredjeparts kode i f.eks. JavaScript, at der skal udføres kontroller.

Hvor ofte og hvordan må afhænge af den enkelte organisation.

9.4 Målgruppe

En stor udfordring med at bruge ASVS på standard banken er, hvem der skal/kan udføre kontrollen, og ikke mindst hvem der efterfølgende skal/kan ændre i systemet, hvis nødvendigt.

De målgrupper, jeg har observeret, er:

- Applikationsudviklere
- Infrastrukturudviklere
- IT-sikkerhedsfunktion

Mit fokus har været på applikationsudviklere, men for at lave en komplet verifikation af en applikation skal alle tre målgrupper involveres.

Det kræver koordinering og styring. En praktisk anvendelig måde at gøre det på vil være, hvis repræsentanter fra de tre målgrupper sætter sig sammen og får sat ansvarlige på de enkelte kontroller. Nogle kontroller udføres af én målgruppe, mens andre kontroller udføres af to målgrupper. Dog vil jeg anbefale, at der altid kun er en målgruppe, som er ansvarlig for kontrollens udførelse.

Som jeg ser det, vil det være naturligt, hvis det er IT-sikkerhedsfunktionen, som er overordnet ansvarlig for processen.

Udover at være ansvarlig for processen mener jeg også, at IT-sikkerhedsfunktionen skal verificere, at kontrollerne er udført korrekt. En måde at gøre dette på vil være at lave regelmæssige stikprøvekontroller, hvor man i en dialog med de som har udført kontrollerne, gennemgår kontrollen overfladisk. På den måde sikres, at kontrollen er udført på en tilfredsstillende måde.

10 Diskussion

I denne rapport har mit fokus været på udviklere som mål for awareness og IT-sikkerhedsfunktionen som styrende for processen. Jeg har helt undgået at præcisere, hvem der skal tage beslutningen om at starte processen, og hvem der er ansvarlig for awareness-niveauet blandt udviklerne.

Der skal allokeres ressourcer hos IT-sikkerhedsfunktionen til at styre processen, samt ressourcer hos applikations- og infrastrukturudviklere til både awareness-træning og til at lave sikre løsninger. Det er virksomhedens ledelse, som er ansvarlig for ressourcefordelingen og i sidste ende, om virksomheden overholder sine IT-sikkerhedspolitikker. Det er derfor essentielt, at ledelsen tager IT-sikkerhed alvorligt.

En lille historie fra hverdagen. Tre af mine bekendte har skiftet arbejde inden for de sidste seks måneder. Alle tre er rutinerede udviklere. Af interesse spurgte jeg dem, om de til deres jobsamtaler var blevet spurgt ind til deres viden om IT-sikkerhed. Alle tre svarede nej.

Min påstand er, at når det bliver naturligt at spørge ind til en jobkandidats viden om IT-sikkerhed, så har virksomheden nået en bevidsthed om vigtigheden af IT-sikkerhed.

IT-sikkerhed er en faktor, som bør vægtes på samme niveau som funktionalitet.

10.1 Awareness med ASVS

Hvordan ser den ideelle måde at lave en awareness-proces så ud? Det kan der være flere bud på. Jeg forestiller mig følgende: Materialet kan genbruges, således at alle gennemfører samme træning. Det skal være let at gennemføre, et selvstudie vil være perfekt. Der skal være minimal styring af processen. Og sidst skal det have en høj effekt.

Denne pakke får man ikke, hvis man vælger at bruge ASVS som fundament for awareness.

For at bruge ASVS som fundament for awareness kræver det en høj grad af styring.
Som vist i afsnit “8 ASVS brugt på en standard bank” varierer de enkelte kontroller på faktorerne:

- Sværhedsgrad
- Relevans
- Genbrugelighed

Desuden er der forskellige målgrupper for kontrollerne:

- Applikationsudviklere
- Infrastrukturudviklere
- IT-sikkerhedsfunktion

Alle disse faktorer og målgrupper skal håndteres og styres, således at det altid er de rigtige personers IT-sikkerheds awareness, der styrkes.

Ud over den høje grad af styring skal man være bevidst om en potentiel faldgrube.

Emnet omkring IT-sikkerhed kan hurtigt komme meget langt væk fra en udviklers normale arbejdsområde. Det vil derfor være naturligt, hvis nogles interesse fanges mere end andres. Men vi vil lave awareness for alle udviklere. Ikke kun de, som er mest interesserede i det.

En måde at genbruge materialet på kunne være at køre temaer. Et tema kunne bestå af en eller flere kontroller. F.eks.”V10 - Communications”. På et afdelingsmøde kunne man gennemgå kontrollerne med udviklerne. Efterfølgende skal udviklerne selv eller sammen i små grupper verificere deres applikation, og man kan på et efterfølgende afdelingsmøde gennemgå resultatet.

Da jeg startede med at skrive denne opgave var min forventning, at ASVS kunne danne rammen om den ideelle awareness-træning. Jeg tog fejl. Som vist og demonstreret i denne rapport, kræver ASVS en høj grad at styring for at kunne bruges som fundament for awareness træning.

Til trods for det store krav til styring, er min anbefaling at bruge ASVS som fundament for IT-sikkerheds awareness.

Min vurdering er, at værdien i forhold til awareness er meget stor, og det er endda set i forhold til den store indsats, der skal bruges for at styre processen.

Inden man begynder processen med ASVS, kan man overveje alternativer. Et alternativ kunne være med udgangspunkt i f.eks. OWASP Top 10 at lave månedlige fællesmøder med alle applikationsudviklere. På hvert møde kunne man gennemgå udvalgte sikkerhedsrisici, og på den måde gøre applikationsudviklerne opmærksomme på de sikkerhedsrisici, som de skal undgå. Uanset hvad, vil dette øge awareness på en langt billigere og lettere måde end ved at bruge ASVS. Men jeg tror desværre, at interessen for emnet falder hurtigt. Nogle vil kede sig, og nogle kan slet ikke forstå det. Der er ingen differentiering.

Netop differentiering er det, jeg mener er nødvendigt for at holde en konstant interesse for IT-sikkerhed. Og det kan awareness-træning baseret på ASVS give.

I rapporten har jeg indtil nu ikke forholdt mig kritisk til ASVS. Jeg synes ASVS har fat om rigtig mange ting, som højner IT-sikkerheden. Men jeg mener, der mangler noget elementært: dokumentation og kodekvalitet.

<u>Dokumentation:</u>
Kode er aldrig selvforklarende. Der bør forefindes dokumentation på alt vigtig kode.

Dokumentation skal være lavet på et niveau, så den er anvendelig. Aldrig dokumentation for dokumentationens skyld. Den rigtige dokumentation kan gøre underværker i forhold til nedbrud og videreudvikling.

Kodekvalitet:
God kode er robust og let at vedligeholde. Det opnås på flere måder.

Brug patterns. Patterns er kendetegnet ved, at de er robuste og virker, samt at andre udviklere kender dem, og derved lettere forstår koden.

Brug automatiserede unit tests. Det gør det lettere for andre at verificere funktionalitet, samt idriftsætte uden problemer.

Robust kode fejler sjældnere end ikke-robust kode. Der vil derfor sjældent opstå uforudsete fejlsituationer med tilhørende uforudset udfald, som kan resultere i sikkerhedsfejl.

10.2 De tre byggesten og awareness med ASVS.

For at starte med at bruge ASVS mangler der en risikovurdering af applikationen. Ellers vil man ikke vide, hvilket level man skal verificere applikationen til.

Hvis man isoleret set kun vil kigge på awareness, er der dog ingen afhængighed til teorien om de tre byggesten. Så man kan bare gå i gang. Der vil så være et behov for at definere, hvilket level applikationerne skal verificeres til, eller hvilke specifikke kontroller der skal verificeres.

Men hvis formålet med at skabe awareness er at skabe mere sikre løsninger, bør man implementere teorien fra de tre byggesten. Årsagen er, at awareness i grove træk er den tredje byggesten - Knowledge, og fokus skal derfor flyttes fra udelukkende at se på awareness til at se det hele billede.

Altså: Risk Management, Seven Touchpoints og Knowledge. På den måde sikres, at viden opbygges, forankres og bruges.

Jeg er absolut tilhænger af teorien om de tre byggesten.

Som jeg ser det, er der er ingen tvivl om, at Risk Managment kommer til at blive en endnu større del af hverdagen i de fleste virksomheder. Den naturlige forklaring er, at risikobilledet er under konstant forandring, og man har derfor behov for at kunne synliggøre den øgede risiko. Det er den eneste måde, hvorpå man kan foretage bevidst navigation i et miljø hvor risikobilledet er dynamisk.

De syv touchpoints er ifølge forfatteren agnostiske. Altså, de kan tilføjes i allerede eksisterende software livscyklus. Mit bud er, at det ikke vil være tilfældet i en mere agil tilgang til udvikling af software. F.eks. i DevOps sammenhæng. Her skal man altså gentænke, hvilke touchpoints man vil bruge og modificere dem, således at de giver mening.

Den sidste byggesten er knowledge. Jeg er helt enig med forfatteren i, at inddeling af viden i kataloger og kategorier mere er en akademisk øvelse end praktisk anvendeligt. Så for at have fokus på anvendelighed, bør man mere tænke på IT-sikkerheds awareness, end på katalogisering og kategorisering af viden.

11 Konklusion

I starten af rapporten definerede jeg en infrastruktur til en standard bank. Standard banken er baseret på gængse komponenter fra Linux og Windows miljøet. Denne standard bank har været gennemgående i hele rapporten.

Jeg har beskrevet det overordnede formål med IT-sikkerhed - FIT-modellen - som jeg har relateret til standard banken.

Af teori har jeg beskrevet de tre byggesten, som kan bruges til at opnå IT-sikkerhed. I min gennemgang af teorien har jeg relateret de tre byggesten til awareness.

Baggrunden for og opbygningen af OWASP er beskrevet. Især er ASVS gennemgået for efterfølgende at blive brugt til at verificere en del af standard banken.

På baggrund af den verifikation har jeg konkluderet at

- ASVS vil give god værdi som fundament for IT-sikkerheds awareness. Men det kræver en stor indsats i form af styring af processen.

- ASVS kan bruges som fundament for awareness, uden det er nødvendigt at supplere med elementer fra teorien. Men det vil være bedst at implementere de sidste to byggesten fra teorien om de tre byggesten.

12 Referencer

[SS] McGraw - Software Security: Building Security In

[OWASP-1] - About The Open Web Application Security Project
https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project

[OWASP-2] - OWASP Top 10 - projekt
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[OWASP-3] - OWASP ASVS - projekt
https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

[OWASP-4] - OWASP ASVS - pdf
https://www.owasp.org/images/3/33/OWASP_Application_Security_Verification_Standard_3.0.1.pdf

[Wiki-1] - DevOps
https://en.wikipedia.org/wiki/DevOps

[The Register-1] - How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript
https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/

[Version2-1] - NotPetya-cyberangreb koster Mærsk milliardbeløb
https://www.version2.dk/artikel/notpetya-cyberangreb-koster-maersk-milliardbeloeb-1079124

[Version2-2] - Google efter 6.500 CPU-år: Nu er kollisioner med SHA-1 virkelighed
https://www.version2.dk/artikel/google-efter-6500-cpu-aar-nu-kollisioner-med-sha-1-virkelighed-1073741

13 Bilag

13.1 The eMail that Started OWASP

https://www.owasp.org/index.php/History_of_OWASP

From: Mark Curphey (markcurphey.com)

Date: Mon Sep 24 2001 - 01:52:35 CDT

Messages sorted by: [date] [thread] [subject] [author]

As some of you may know, someone on the www-mobile-codesecurityfocus.com (to be renamed webappsecsecurityfocus.com real soon) list suggested setting up a project to define an industry standard testing methodology for the security of web applications. I was asked to help set it up and co-ordinate and am pleased to be involved. Several people have already volunteered with various degrees of commitment. As I discussed ideas with various people, it became clear there was a need for a much wider project to include the design, development, deployment and testing of web application security as well as the standard categorization of attacks.

So we are pleased to announce the creation of the "Open Web Application Security Project" known as OWASP. This is a community effort that will be open source and available to all. I have created a quick and dirty web site at http://www.owasp.org until we can get a real webmaster to volunteer. As this was created on the mailing list, most of the work is expected to be driven on mailing list traffic.

How will the project work ? Over the coming months the project will seek to define security recommendations, specifications and explanations in key areas. Security professionals will be able to use the output to incorporate in their work. Security vendors will be able to base services and products on these standards and consumers will be able to baseline and test applications or services they consume.

It seems to make sense to initially start by defining standard web application Attack Categories and develop the testing methodology. The methodology will probably include "white box" testing (where the tester has full access to source code), "black box" testing where the tester has access to the application as a user and "glass box" testing where the tester has both.

A broad based schedule will be set over the next few weeks after initial administrivia has been worked out. That includes a licensing model such as GPL to prevent commercial companies taking the output and using it as there own, whilst still promoting its widespread adoption. Each part of the project will need to be lead by individual volunteers, initial ones will hopefully be determined this week.

We are currently looking for;
Technical - We are looking for additional people with technical security skills in various web technologies including HTTP, XML, HTML, ASP, Java, C, C#, PHP, CGI's, Perl, JavaScript, .NET, J2EE and others.
Translators - We have two translators ready to port documentation to French and Portuguese. However we will be looking for other volunteers in particular German.
Graphic Designer - We need some simple graphics for the web-site and may need illustrations etc.
Webmaster - We need someone to design and maintain this web site.

Much of the success of open source projects comes from individuals adding value within his or her individual area of expertise. This community welcomes your contribution.

http://www.owasp.org

Kind Regards,
Mark Curphey