KUNDEKONFIDENSIELL

Rapport

Arkitektur- og kildekodegjennomgang EVA

Valgdirektoratet

13.06.2018

mnemonic
- securing your business

1 Sammendrag

1.1 Forretningssikkerhet

Med forretningssikkerhet tenker vi sikkerhet relatert til de oppgavene som Valgdirektoratet er satt til å forvalte. Ut fra arbeidet som er gjort ser vi ikke om det brukes et helhetlig rammeverk for virksomhets(sikkerhets)arkitektur som f.eks TOGAF eller SABSA i arbeidet med EVA, men vi ser at det er gjort flere grep som naturlig passer inn i en slik (sikkerhets)forretningsarkitektur.

I 2017 ble det laget et strategidokument for perioden 2017-2021. Basis for en del av diskusjonen er en SWOT-analyse, som bekrefter våre funn og som tar opp temaer utenfor denne analysen. Det vi kan bekrefte, er bl.a. svakheter som bruk av gamle bibliotek, tett knytning mot testfunksjonalitet, ulik teknologibruk, vanskelig å sikre installasjon av EVA Skanning, men også styrker som at det jevnt over er god kodekvalitet og få alvorlige sikkerhetsfunn.

Noen av delrapportene fra dette prosjektet er lange, og det kan gi inntrykk av at det er store svakheter, men sammenliknet med andre prosjekter av liknende størrelse og omfang anser vi at det færre funn og færre alvorlige funn i EVA-porteføljen.

Vi ser dog ikke noen uoverkommelige barrierer, men noen endringer vil nødvendigvis være mer dyptgripende, som eventuell endring av EVA Skanning-arkitektur (og her anser vi ikke å bytte ut ReadSoft som en arkitekturendring).

Vi anser at det er mulig å bruke strategidokumentet som basis for innføring av en forretningssikkerhetsarkitektur, hvor tanken er at det skal være mulig å synliggjøre både hvilke endringer som skal til for å oppnå et sikkerhetsrelatert forretningsbehov, samt hvilke sikkerhetsrelaterte forretningsbehov blir påvirket ved endringer i teknologivalg og –bruk.

1.2 Kodekvalitet og arkitektur

Det er en del enkeltfunn kildekodegjennomgangen av de tre komponentene, men en del av disse er diskusjonsgrunnlag og som vi har videreformidlet fra skannerverktøyene da de ser ut til å peke på svakheter i koden, men som det kan være mottiltak mot, eller som ikke slår til, av grunner vi ikke har funnet. Det er forholdsvis få alvorlige funn, og noen av funnene vet vi at er relatert til funksjonalitet som Valgdirektoratet har et annet syn på kritikaliteten av da f.eks. EVA Admin og EVA Resultat kjører på infrastruktur som Valgdirektoratet har kontroll over. Vi vil dog være sikker på diskusjoner rundt kontroll av denne infrastrukturen er aktiv, da f.eks. plassering av infrastrukturen hos en driftsleverandør vil kunne påvirke om passord i konfigurasjonsfiler er en sårbarhet man kan leve med eller ikke.

Jevnt over er det få mønstre på gjennomgående feil. Vi ser noen tegn på at det kan være ulike utviklere som har vært i aksjon, eller at det er gjort opprydding i deler av koden, men ikke i andre, og hvor det ender opp med at samme funksjonalitet er implementert på to ulike måter. Det er dog enkelt å ta tak i, og det er bare å bestemme seg for hvilken måte som ønskes gjennomført.

Det var nødvendigvis en stor endring å fjerne grensesnittene mot elektronisk stemmegivning, og vi ser fortsatt kode som ligger under no.evote-hierarkiet og det er noen få referanser til leverandøren av evote-løsningen, Scytl. Her er det, som over, bare å fortsette med ryddingen og trekke ut den funksjonaliteten som skal brukes videre, og legge det under no.valg-hierarkiet.

Når vi ser på kildekoden i IntelliJ og Visual Studio får vi av og til forslag om opprydding av koden, hvor nyere språkutviklinger i f.eks. Java 8 eller C# 7.0 kan gjøre koden mer kompakt og lettlest. Vi oppfordrer til å bruke funksjonaliteten som ligger i disse verktøyene og som forenkler jobben med at alle utviklerne bruker samme kodestandard og –stil. SonarQube brukes aktivt i utviklingen, for å få ryddet i gammel kode og få den over i mindre, testbare enheter enn det som var tilfellet da koden ble tatt over.

Når det gjelder testing ser vi i SWOT-analysen at det påpekes at det mangler dekning, men vårt syn er at det er at det ser ut til å være mye testkode i løsningen. Vi ser heller problemet at testkoden er knyttet for tett opp mot løsningen, og for å klare å kompilere EVA Admin måtte vi fjerne bindinger mot testkoden. Nå vil det nok alltid være et ønske om flere gjennomgripende systemtester (etter at enhetstester er tilstrekkelig på plass), og som tar for seg ulike scenarioer, og med vært fokus foreslår vi å dekke feil bruk, for å dekke kvalitetskontroll, og også forsøk på misbruk, for å dekke sikkerhetsaspekter.

1.3 Komponenter med sårbarheter

EVA bruker noen eksterne programvarebiblioteker som inneholder kjente sårbarheter. Det anbefales å oppgradere disse til siste versjon. Det kan også være en god ide å implementere noen form for avhengighetskontroll i byggekjeden. OWASP Dependency Check er et verktøy som kan identifisere avhengigheter i koden og sjekke om det er noen kjente sårbarheter knyttet til de versjoner av ekstern kode som brukes. Som mye annen automatisert skannverktøy genererer den også falske positive, men den har støtte for å konfigurere hva som ikke skal rapporteres. Verktøyet kan med fordel kjøres automatisk ved f.eks. bygging av produksjonsversjoner, og har støtte for bland annet Maven og Jenkins.

Vi har brukt verktøyet i denne analysen, integrert i Maven for Java. For Windows kjørte vi kommandolinjeversjonen, noe som kan legges inn i TeamCity-konfigurasjonen.

I tillegg til automatisert overvåkning i byggeprosessen anbefaler vi å abonnere på informasjon om sårbarhetsinformasjon og feilrettinger i de komponentene som de ulik applikasjonen benytter seg av. Det gjelder JVM/CRL, bibliotek, applikasjonsservere, kjøringsmiljø, sikkerhetsprodukter og andre komponenter som brukes når applikasjonene kjøres.

Vi ser også i noen tilfeller at det benyttes biblioteker som ikke har kjente sårbarheter *som sådan*, men som har nådd «end of life». Eksempel på dette er OpenSAML 2.

Anbefaling 1 *Oppgrader utdaterte biblioteker til siste versjon*

Anbefaling 2 *Vurder bruk av OWASP Dependency Check (eller lignende) i byggekjeden*

Anbefaling 3 *Følg med på sårbarhetsinformasjon i komponentene som løsningene bruker*

Referanse: https://www.owasp.org/index.php/OWASP_Dependency_Check

1.4 Kryptografiske funn og observasjoner

Dette kapittelet beskriver generelle observasjoner relatert til kryptografifunn i gjennomgangen. Funn spesifikke til en av de tre hoveddelene ligger i den aktuelle delrapporten.

Et generelt funn er at det er forholdsvis mye kode som er skrevet for å håndtere kryptering (konfidensialitet), digital signatur (integritet og autentisitet) samt krypteringsnøkler og passord.

Dette er ofte ganske lavnivå kode, som brukes for å sy sammen biblioteksfunksjoner som implementerer selve algoritmene (f.eks. AES).

I den grad det er mulig, oppfordrer vi til å kapsle inn og forenkle denne type funksjonalitet så langt det lar seg gjøre, *helst* ved bruk av ferdige høynivåbibliotek som implementerer validerte og sikre funksjonaliteter og patterns.

- Mindre kode å forvalte
- Enklere å forstå hva som foregår i koden
- Enklere å etablere testcase

Generelt gir det lite verdi for Valgdirektoratet å vedlikeholde mye «tung» sikkerhetskode, og i den grad man vil bruke ressurser på videreutvikling av dette området i koden bør forenkling og konsolidering være et sentralt tema.

1.4.1 Shibboleth / OpenSAML

Bruken av OpenSAML 2.6.4 er observert i Eva Admin (i rot-POMen), og videre i admin-rbac, admin-frontend, og admin-common. Dette er nyeste versjon av OpenSAML for Java.

På wikien til Shibboleth advares det i store bokstaver:

> **End of Life Warning**
>
> As of July 31, 2016, all security maintenance for the OpenSAML V2 Java release branch will cease. A complete schedule of the dates can be found here. All deployments should upgrade to V3 or evaluate other alternatives. This does **not** apply to the C++ libraries, which remain supported indefinitely.

Med bakgrunn i dette, samt historikk for nyere sikkerhetsadvarsler for OpenSAML V3, virker det **ikke** tilrådelig å benytte OpenSAML versjon 2.x for sikkerhet i moderne løsninger.

Sett i et mer *strategisk* perspektiv så er det også tydelig at SAML er gårsdagens sikkerhetsteknologi. Nye implementasjoner benytter i stor grad lettvekts protokoller som OpenID Connect fremfor «tunge» XML-baserte protokoller som SAML, og ID-porten / Difi er også i ferd med å bevege seg i denne retningen.

Det kan derfor være lurt å se på mulighetene for å kvitte seg med SAML fullstendig, dersom man uansett er nødt til å gjøre større endringer på dette området for å ivareta sikkerheten.

Anbefaling 4 *Erstatt OpenSAML V2 med sikkerhetsbibliotek som blir aktivt forvaltet og videreutviklet*

Anbefaling 5 *Vurder om man samtidig skal erstatte bruken av SAML med en nyere og mer lettvekts teknologier*

1.4.2 Java Cryptography Architecture

Java Cryptography Architecture (JCA) er Java-plattformens utvidbare rammeverk for krypto, med Java Cryptography Extension (JCE) som en standard plattformutvidelse for dette.

Legion of the Bouncy Castle er en veldedig organisasjon i Australia som vedlikeholder gratis åpen-kildekode kryptobiblioteker for Java og .NET. Java-versjonen av biblioteket omfatter tilbydere for Java Cryptography Architecture (JCA) og Java Cryptography Extension (JCE), som er kjernebibliotekene / APIene i Java for krypto. Det tilbyr også implementasjoner av vanlige protokoller som PKCS #12 og sertifikater, CMS (S/MIME), TLS, med mer. Eva Admin bruker dette biblioteket relativt mye for å implementere sikkerhetsfunksjoner.

Den åpenbare fordelen med å bygge sikkerhetsfunksjoner på JCA / JCE er at dette er den standardiserte mekanismen i Java for dette – og at BouncyCastle er et av de mest utbredte bibliotekene.

En klar **ulempe** med kryptografi-arkitekturen i Java er imidlertid at grensesnittene er spesifisert på et svært lavt nivå. Dette betyr i stor grad at utviklere selv må forstå hvordan byggeklossene skal puttes sammen, og at man får relativt intrikat kode som kan være vanskelig å vedlikeholde, teste, og analysere. Et lavnivågrensesnitt kan forsåvidt være nyttig når man skal lage skreddersydde løsninger (for eksempel for elektronisk valg) som ikke kan baseres på kjente patterns, men i de fleste ordinære use-case vil det være lurt å ligge på et høyere abstraksjonsnivå.

På øvrige plattformer har trenden gått i retning høyere abstraksjonsnivå. For .NET-kode vil vi ofte foreslå å bygge på Microsoft sine egne sikkerhetsbibliotek (Secure Channel / SChannel), for C eller språk med gode bindinger mot C (f.eks. python) vil libsodium være et godt valg.

For Java er vi derimot usikre på hva som er det beste valget for implementasjon av kryptografiske sikkerhetsfunksjoner, men **mageølelse og erfaring** tyder på at JCA / JCE ofte kan til løsninger som er både risikable og ressurskrevende.

Anbefaling 6 *Se på muligheter for å erstatte kryptografisk kode med mer høynivå biblioteker.*