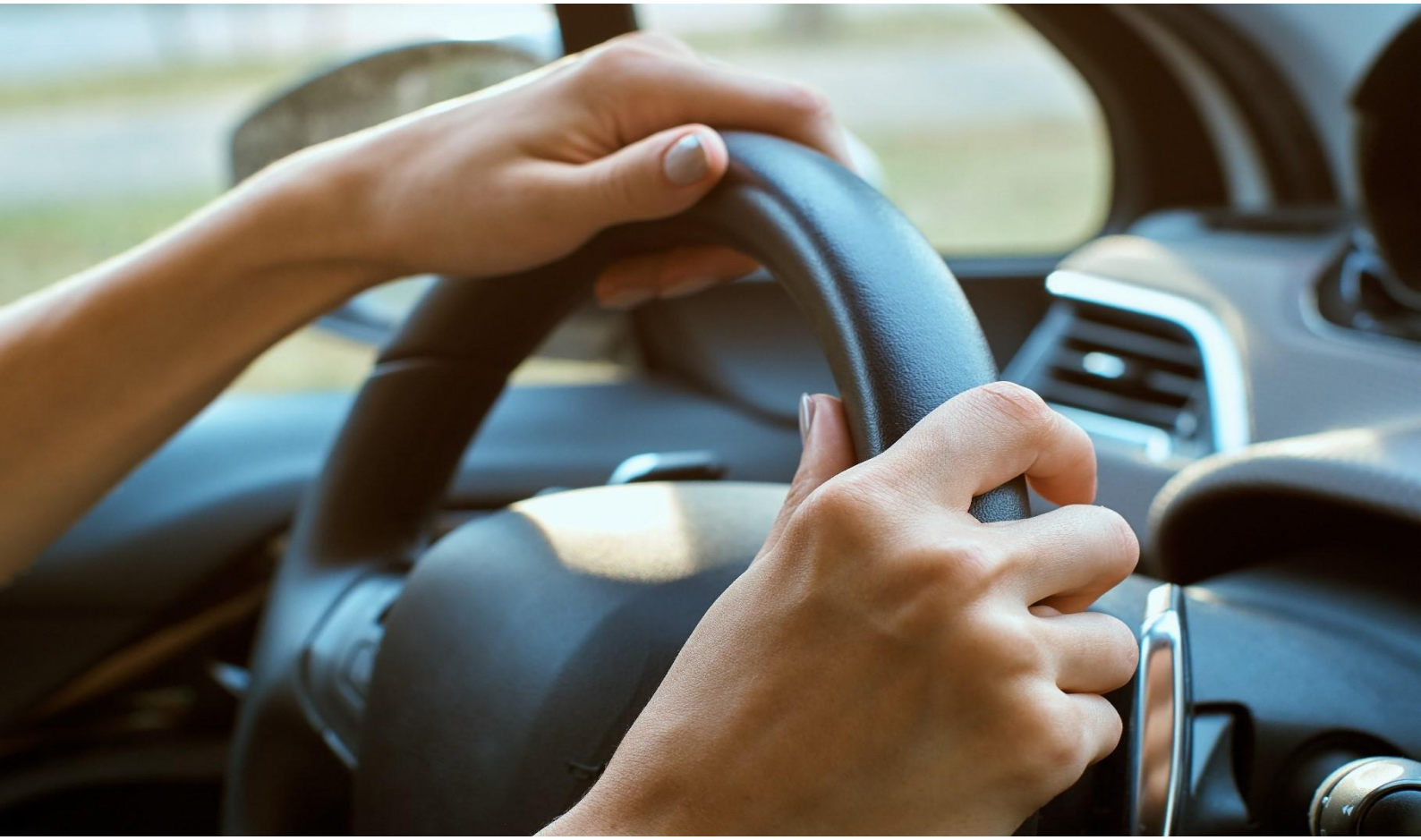
veiligheid.nl
kenniscentrum letselpreventie

Richtlijn Veilig Ontwikkelen

Versie 1.2



veiligheid nl
kenniscentrum letselpreventie

Over VeiligheidNL

VeiligheidNL is hét kenniscentrum voor letselpreventie. Wij zetten ons in om het leven van mensen veilig(er) te maken door veilig gedrag in een veilige omgeving te stimuleren.

Veiligheid is niet vanzelfsprekend. Het is het resultaat van onderzoek, van wetenschap, van interventies, van gedrag. Wij richten ons op de meest voorkomende en meest ernstige letsels, waar preventie belangrijk én mogelijk is. Dit doen we vanuit de thema's Kinderveiligheid, Valpreventie, Gezond gehoor, Sportblessurepreventie, Verkeersveiligheid en Productveiligheid.

We werken in een doelgerichte cyclus aan onderzoek, strategie- en interventieontwikkeling, implementatie en evaluatie. Relevante kennis en inzichten zetten wij om in hoogwaardige gedragsinterventies en slimme veiligheidsoplossingen en we verbinden wetenschappelijke inzichten met de dagelijkse praktijk. En, dat doen we niet alleen. We werken samen met partners en professionals en samen strijden we voor maximale impact.

Voor de monitoring van letsels werken we met ons eigen Letsel Informatie Systeem (LIS). Een uniek systeem dat letsels registreert bij een representatieve steekproef van Spoedeisende Hulpafdelingen van ziekenhuizen in Nederland.

veiligheid.nl
kenniscentrum letselpreventie

Documentbeheer

Documentinformatie	
Bestandsnaam	Richtlijn Veilig Ontwikkelen
Document type	Operationeel beleid
Eigenaar	Nils Meijer
Geldig tot	12-2026
Classificatie	Intern

Distributielijst

Versie	Distributielijst
1.x	Nils Meijer, Hidde Toet, Onno Valkering, Rianne Kaptein, Dorien Heijting

Versiebeheer

Versie	Datum	Auteur(s)	Status	Opmerking
0.1	01-2022	Dorien Heijting, Onno Valkering, Rianne Kaptein, Nils Meijer	Concept	
1.0	12-2022	Dorien Heijting, Onno Valkering, Rianne Kaptein, Nils Meijer	Definitief	
1.1	01-2024	Onno Valkering	Definitief	Toegevoegd: 2.5, 3.5, 3.6
1.2	01-2026	Onno Valkering	Definitief	Bijgewerkt: 3.3, 3.4, 3.5, 3.7, 4.1, 5.2, 5.3, 5.4, 5.5, 5.6 Toegevoegd: 3.8, 5.7, 5.8, B.2

Inhoudsopgave

1 Inleiding

Dit document is geschreven om uitvoer te geven aan NEN 7510 normeis A.14 (Acquisitie, ontwikkeling en onderhoud van informatiesystemen). Het beschrijft alle maatregelen en richtlijnen die benodigd zijn om volgens het “security-by-design” principe softwaretoepassingen te ontwikkelen.

Het document is onderdeel van het Informatie Beveiligingsbeleid (IB) dat VeiligheidNL voor de gehele organisatie toepast. De bijlage bevat een tabel met een overzicht van relevante secties per NEN7510 normeis(groepen), alsook een tabel met een overzicht van alle relevante (Azure) compliance checks.

2 Werkafspraken

2.1 Algemene taakverdeling

Er wordt ontwikkeld volgens de DevOps werkwijze. Dit betekent, o.a., dat er geen strikte functie scheiding bestaat tijdens ontwikkeling, maar dat taken door ieder kan worden opgepakt. De taakverdeling wordt in overleg bepaald en kan tijdens de ontwikkeling aangepast worden. Om foutgevoeligheid van (beheer)taken te minimaliseren wordt er zo veel mogelijk geautomatiseerd.

Om wijzigingen definitief door te voeren aan applicaties wordt het 4-ogen principe toegepast. Een PR (zie Sectie 3.5) wordt door tenminste één iemand anders goedgekeurd. Ook een deployment (zie Sectie 2.3) naar de productieomgeving wordt bij voorkeur door iemand anders goedgekeurd.

2.2 Voortgang bijhouden

Voor elke werkzaamheid wordt een *work item* aangemaakt in Azure DevOps. Afhankelijk van de grootte van de aanpassingen kan deze opgeknipt worden in sub-werkzaamheden. In *work items* wordt voorgang bijgehouden en worden design keuzes vastgelegd d.m.v. comments. Aan commits messages (zie Sectie 3.5), worden de relevante *work items* gerefereerd. Op deze manier worden daadwerkelijk gemaakte aanpassingen aan de codebase gelinkt aan de omschrijvingen en afspraken.

2.3 Deployments

Om de continuïteit van de productieomgeving als ook het voortbrengingsproces te kunnen garanderen wordt er gebruik gemaakt van het OTAP-concept, waarbij de verschillende ontwikkelstadia plaats vinden in de daarvoor bestemde omgeving. Namelijk de Ontwikkel, Test, Acceptatie en Productie omgeving. Waarbij Acceptatie en Productie identieke omgevingen zijn. Minimaal aanwezig zijn een de Test en Productie omgevinging, Ontwikkel en Accepatie indien nodig. Deployments worden enkel automatisch uitgevoerd om foutgevoeligheid te minimaliseren. Voor deployment naar een productieomgeving zijn altijd (in totaal) ten minste twee goedkeuringen nodig. Indien noodzakelijk en mogelijk, wordt gebruik gemaakt van *swap slots* i.c.m. (automatische) regressie tests om downtime te minimaliseren en *smoke tests* (zie Sectie 4.1) te kunnen uitvoeren.

2.4 Incident afhandeling

Als er een incident (systeemfout) voordoet op de productieomgeving wordt er altijd een nieuwe *work item* met daarin zo veel mogelijk (context-)informatie over het incident. Op basis van deze informatie wordt er direct een prioriteit ingesteld en iemand toegewezen om het incident verder te onderzoeken en er zorg voor te dragen dat het opgelost wordt. Elke incident wordt gecheckt op mogelijke dataleks.

2.5 Cocreatie

De richtlijn in dit document is ook van toepassing op *cocreatie* projecten (in samenwerking met externe leveranciers). Aanvullende werkafspraken worden vastgelegd, bij voorkeur in een SLA.

3 Codeerafspraken

Codeerafspraken zijn essentieel in de ontwikkeling van productiewaardige en veilige applicaties. Een belangrijk doel van codeerafspraken is het behalen en behouden van consistentie in codebases, dit draagt, o.a., bij aan de leesbaarheid en onderhoudbaarheid. Ook worden er met codeerafspraken best practices, op verschillende aspecten waaronder security, geïntroduceerd. Met deze best practices kunnen, o.a., bugs en kwetsbaarheden in de beveiliging worden voorkomen.

Het streven is om de controle op codeerafspraken zo veel mogelijk te automatiseren.

3.1 Naamgevingconventies en codestijl

Voor naamgevingconventies en codestijl worden breed-geadopteerde community codeergidsen gevolgd (zie Tabel 1). Verder wordt er geprogrammeerd in het Engels (code, comments, en config).

Tabel 1 – De gebruikte codeergids per programmeertaal.

Programmeertaal	Codeergids
C#	Microsoft C# Coding Conventions (**link**)
JavaScript	Airbnb JavaScript Style Guide (**link**)
Python	PEP8 (**link**) + "Comments en Docstrings" (sec. 3.8) uit Google Python Style Guide (**link**)
R	The tidyverse style guide (**link**)

Voor elk van deze codeergidsen zijn formatters en/of linters beschikbaar die de controle op naamgevingsconventies en codestijl (deels) kan automatiseren en in bepaalde gevallen ook automatisch code kan corrigeren. Voorbeelden hiervan zijn *black* (**link**) en *dotnet-format* (**link**).

3.2 Wachtwoord- en sleutelbeheer in code

Voorkomen moet worden dat wachtwoorden, sleutels en andere geheimen direct (hardcoded of in een los configuratiebestand) in codebases worden opgeslagen. De mechanismes voor het beheer van codebases zijn namelijk niet toereikend voor het beheer van geheimen. Zo ontbreekt bijvoorbeeld encryptie en de mogelijkheid tot rol-gebaseerd toegangsbeheer per geheim(groep), zie hoofdstuk 5.

Het gebruik van de Azure Key Vault (AKV, **link**) heeft de voorkeur. AKV slaat geheimen versleuteld op en biedt veel beheermogelijkheden, o.a. het roteren en van invalideren van geheimen wordt vergemakkelijkt. Als alternatief kunnen geheimen ook geïnjecteerd worden. Dit kan op het moment van deployment of via voorgeconfigureerde omgevingsvariabelen per deployment-omgeving. Een andere optie is het gebruik maken van *variable groups*, i.c.m. de *secrets* functie, in Azure DevOps.

3.3 Foutafhandeling en logging

Om bugs correct te kunnen identificeren en op te lossen is het belangrijk dat wanneer er een fout optreedt context-informatie wordt gelogd. Ook wanneer deze fout onverwachts optreedt. Om te voorkomen dat deze informatie niet meer beschikbaar is zodra de applicatie stopt; dient er gelogd te worden naar minstens één (semi)langdurige opslag in aanvulling op de *stdout* en *stderr* stromen.

Op fouten die enigszins te verwachten zijn moet worden geanticipeerd. Op (sub)component niveau is *fail fast* (**link**) hierbij de standaard. Overkoepelend (i.e. top-level) dient er met fouten om te gaan middels een *unhappy path* in aanvulling tot de *happy path*. Naast logging moet ook het opschonen van (tijdelijke) data werkkopieën, indien van toepassing, onderdeel zijn van een *unhappy path*.

Er mag geen gevoelige (persoons)informatie leesbaar in de logs terecht komen.

Als logging mechanisme heeft Azure's Application Insight (AppI, **link**) de voorkeur. Met AppI is het, o.a., mogelijk alerts in te stellen voor bepaalde foutmeldingen. Voor applicaties die niet in Azure draaien kan naar een bestand gelogd worden. Logging kan op verschillende levels (zie Tabel 2).

Tabel 2 – Logging levels zoals aanwezig in de meeste programmeertalen.

Level	Omschrijving
Debug	Bedoeld voor informatie dat handig is tijdens development en debugging. Bijvoorbeeld: wat is de waarde van een bepaalde variabele op een bepaald moment tijdens uitvoer.
Information	Informatie over het algemene functioneren van een applicaties. Bijvoorbeeld: welke (belangrijke) dynamische beslissing heeft de applicatie zelfstandig genomen, en waarom.
Warning	Geeft onverwachte/abnormale invoer of gebeurtenissen aan, die niet leiden tot een error. Bijvoorbeeld: een invoerbestand is leeg of standaard niet in een ideale bestandscodering.
Error	Voor foutmeldingen van onoverkoombare fouten in een (sub)component of in binnen de context van een verwerkbaar verzoek. Bijvoorbeeld: niet genoeg toegangsrechten.
Criticial	Voor foutmeldingen van onoverkoombare fouten op applicatie of systeem niveau en andere zaken die direct aandacht vereisen. Bijvoorbeeld: geen schijfruimte meer.

Azure (audit) logging wordt opgeslagen in een Log Analytics Workspace met een bewaartermijn van 30 dagen. Voor SQL Server audit logging wordt gebruikgemaakt van Azure Blob Storage met een langere bewaartermijn van 90 dagen, wat de kosten drukt ten opzichte van opslag in Log Analytics.

3.4 Statische codeanalyse

Het gebruik van statische codeanalyse tools helpt bij het vroegtijdig identificeren van fouten en kan het gebruik van bepaalde best practices afdwingen. Het kan nodig zijn om meerdere tools te moeten gebruiken om alle relevante aspecten (o.a. kwaliteit, security, correctheid) af te dekken. Minimaal dienen, waar beschikbaar, de facto open-source tools gebruikt worden (zie Tabel 3). Afhankelijk van de applicatie kan gekozen worden om *GitHub Advanced Security for Azure DevOps* aan te zetten.

Tabel 3 – De gebruikte statische codeanalyse tools per programmeertaal.

Programmeertaal	Statische codeanalyse	
C#	Tool	Omschrijving
	SCS (**link**)	Checkt de code op OWASP beveiligingsproblemen. Bijvoorbeeld: er wordt geen gebruik gemaakt van parameterized SQL queries.
JavaScript	Tool	Omschrijving
	ESLint (**link**)	Checkt de code op standaard kwaliteitseisen. Bijvoorbeeld: er komen ongebruikte en/of ongedefinieerde variabelen voor in de code.
Python	Tool	Omschrijving
	Flake8 (**link**)	Checkt de code op standaard kwaliteitseisen. Bijvoorbeeld: er komen ongebruikte en/of ongedefinieerde variabelen voor in de code.
	Bandit (**link**)	Checkt de code op basis beveiligingsproblemen. Bijvoorbeeld: er wordt geen of verkeerd gebruik gemaakt van het TLS (SSL) protocol.
	MyPy (**link**)	Type checker die *type safety* afdwingt in met types geannoteerde Python code. Bijvoorbeeld: is het return value van het juiste type.
R	Tool	Omschrijving
	Lintr (**link**)	Checkt de code op standaard kwaliteitseisen. Bijvoorbeeld: er komen ongebruikte en/of ongedefinieerde variabelen voor in de code.
	Goodpractice (**link**)	Checkt R packages op best practices. Bijvoorbeeld: functies hebben te hoge cyclomatic complexity of er zijn ontbrekende unit tests.

De statische codeanalyse tools dienen gedraaid te worden als onderdeel van CI (zie Sectie 3.6).

Statistische codeanalyse tools zijn geen vervanging van "hardning" (zie Sectie 5.2).

3.5 Versiebeheer

Voor versiebeheer van codebases worden, als onderdeel van Azure DevOps, *git* (**link**) repositories gebruikt. Het gebruik hiervan stelt in staat vorige versies van code te herstellen en (deels) geautomatiseerd los van elkaar ontwikkelde code samen te voegen, d.m.v. een *merge* of *rebase*.

Er wordt gewerkt met een *feature branch workflow* (**link**) met een *main* en *develop* branch. Alle feature branches worden gebaseerd op de *develop* branch en dienen middels een prefix aan te geven wat voor soort change het betreft (zie Tabel 4), bijvoorbeeld *feat/add-extra-option* of *fix/bug-xyz*.

Op het moment dat een commit gemaakt wordt kan middels pre-commit (**link**) de wijzigingen aan de code automatisch gecheckt worden met formatters/linters (zie Sectie 3.1) en statische codeanalyse tools (zie Sectie 3.4). Hiermee wordt, ten minste, consistentie in broncode gewaarborgd.

Tabel 4 – Prefixes om het type change aan te geven.

Prefix	Omschrijving
feat	Algemene change, meestal voor toevoeging of aanpassing van een feature.
fix	Change specifiek om een bug op te lossen.
style	Aanpassingen ten behoeve van naamgevingconventies en codestijl.
docs	Aanpassingen aan de codebase documentatie (bijv. comments en docstrings).
refactor	Refactorings: geen aanpassing aan functionaliteit enkel beter/anders geïmplementeerd.
chore	Onderhoudstaken zonder impact op code functionaliteit (bijv. afhankelijkheden update).
security	Aanpassingen die beveiligingslekken verhelpen of security verbeteren.

3.6 Continuous Integration

Om vroegtijdig problemen in software te detecteren wordt *Continuous Integration* (CI) toegepast. Dit houdt in dat alle wijzigingen aan een reeks van automatisch checks worden onderworpen. Als deze checks slagen, kan met een bepaalde zekerheid gezegd worden dat de code goed uitvoerbaar is. De checks die uitgevoerd worden varieert, maar een standaard is het uitvoeren van statische codeanalyse (zie Sectie 3.4), audit tools (zie Sectie 5.3), en automatische tests (zie sectie 4.1).

Voor CI wordt, waar mogelijk, gebruik gemaakt van templates ter standaardisatie.

3.7 Versienummers

Alle software releases worden voorzien van een versienummers, volgens *semantic versioning* (**link**). Een semantisch versienummer bestaat uit drie cijfers: *major*, *minor*, en *patch* (zie Tabel 5). Versienummering voor interne releases begint bij 0.1.0, externe releases bij 1.0.0 (beeldvorming).

Tabel 5 – Onderdelen van een semantisch versienummer

Onderdeel	Omschrijving
Major	Opgehoogd wanneer de nieuwe release breaking changes bevat.
Minor	Opgehoogd wanneer de nieuwe release *backward compatible* is.
Patch	Opgehoogd wanner de nieuwe release (uitsluitend) *backward compatible* bugfixes bevat.

Elke release wordt als checkpoint/snapshot (tag) vastgelegd in het versiebeheersysteem (zie Sectie 3.5). Indien van toepassing wordt een release als package gepubliceerd in *Azure DevOps Artifacts*.

3.8 Infrastructure as Code

Voor het beheren en provisioneren van Azure wordt Infrastructure as Code (IaC) toegepast. Dit zorgt voor reproduceerbaarheid, consistentie tussen omgevingen, en vermindert de kans op menselijke fouten bij handmatige configuratie. Als tooling wordt *Bicep* i.c.m. Azure DevOps gebruikt.

4 Testen & Acceptatie

Bij het ontwikkelen van software worden de OTAP (Ontwikkelen, Testen, Acceptatie, Productie) stadia gevolgd. Tijdens de het "Testen" stadium wordt vastgesteld dat de software voldoet aan de eisen die worden gesteld functioneel, technisch en kwaliteit van de code. In het "Acceptatie" stadium wordt vervolgens door de gebruikers van het product getest of het product ook voor hen acceptabel is.

4.1 Testen

Binnen de organisatie wordt er gewerkt met verschillende soorten test sets. Er zijn testsets in gebruik voor de: unit tests; integratie tests; systeem tests; en de gebruikersacceptatietests. Testsets voor de unit tests en integratie tests worden opgeslagen in de code repository (Azure DevOps). Deze testsets mogen alleen bestaan uit dummy data en geen productie data bevatten. Testsets voor de systeem tests en gebruikersacceptatietests staan in de database van de betreffende OTAP omgeving.

Kwestbaarheden, o.b.v. de OWASP top 10, worden beoordeeld tijdens systeem tests.

4.2 Gebruikersacceptatietesten

Voor wijzigingen die invloed hebben op gebruikers is de product owner (PO) verantwoordelijk voor het opstellen van acceptatiecriteria. Input voor de acceptatiecriteria kan worden aangeleverd door de (interne en/of externe) stakeholders en door het ontwikkelteam.

Voor uitgebreide gebruikersacceptatietests wordt er gebruik gemaakt van testprotocollen. In deze test protocollen worden verschillende aspecten van de gebruikersacceptatietests omschreven (zie Tabel 6). Voorbeelden van wanneer testprotocollen gebruikt kunnen worden is wanneer er wordt overgestapt op een nieuw softwaresysteem, of wanneer een software update veel wijzigingen bevat.

Tabel 6 – Onderdelen van een testprotocol

Onderdeel	Omschrijving
Testopzet	Algemene informatie over hoe de gebruikersacceptatietests georganiseerd is.
Testscope	Welke onderdelen van het systeem worden er getest met de gebruikersacceptatietest.
Testdata	Welke data er gebruikt kan/moet worden tijdens het uitvoeren van de tests.
Tests	Een overzicht van de tests die uitgevoerd moeten worden, eventueel met taakverdeling.
Testresultaten	Hoe voortgang van de tests, en de uiteindelijke resultaten, worden vastgelegd.
Instructies	Extra handelingen die tests wel/niet moeten uitvoeren, indien van toepassing.

5 Security & Autorisaties

5.1 Wachtwoord- en sleutelbeheer

Met wachtwoorden, sleutels, en andere geheimen moet zorgvuldig omgegaan worden. Binnen de Azure omgeving worden alle geheimen opgeslagen in een Azure Key Vault (AKV, **link**) met daarop access ingesteld in overeenkomst met de geldende autorisatie matrix. Geheimen die (ook) gebruikt moeten worden buiten de Azure omgeving worden (ook) in LastPass opgeslagen, eveneens in overeenkomst met de geldende autorisatie matrix. Per geheim moet ten alle tijden duidelijk zijn wie er toegang tot heeft en sinds wanneer (die laatste versie van) het geheimen in gebruik is genomen.

Wachtwoorden en sleutels moeten minstens elke twee jaar geroteerd worden.

5.2 Applicatiebeveiliging

Applicaties worden 'gehardend' conform voorschriften van de leverancier en conform best-practices, zoals richtlijnen vanuit OWASP. Hardening wordt toegepast tijdens het ontwikkelproces (middels Pull Requests) en vervolgens periodiek (middels pentests) om de continuïteit van applicaties te borgen.

5.3 Software afhankelijkheden

Voordat derde-partij software afhankelijkheden (i.e. *packages*) worden gebruikt wordt goed nagegaan of dit daadwerkelijk nodig is of dat de gewenste functionaliteit zelf geïmplementeerd kan worden. Pas als er een meerwaarde bestaat, wordt overgegaan tot het gebruik van software van derde partijen. Dagelijks moet er bekeken worden of deze software bekende beveiligingsproblemen hebben. Middels *audit* tools i.c.m. CI (zie sectie 3.6) kan dit worden geautomatiseerd. Minimaal dienen open-source tools gebruikt te worden (zie Tabel 7). Afhankelijk van de applicatie kan gekozen worden om *GitHub Advanced Security for Azure DevOps* te gebruiken voor dependency scanning.

Tabel 7 – De gebruikte open-source audit tool per programmeertaal

Programmeertaal	Codeergids
C#	.NET CLI (**link**)
JavaScript	NPM CLI (**link**)
Python	Safety (**link**)
R	OysteR (**link**)

Alle afhankelijkheden worden gecontroleerd op licensies middels *Software Bill of Materials* (SBOM) generatie en analyse. Open-source afhankelijkheden mogen alleen gebruikt worden als de licentie geen copyleft voorwaarden bevat die vereisen dat onze eigen code openbaar gemaakt moet worden.

5.4 Data encryptie

Data-in-transit wordt versleuteld op basis van industriestandaard encryptieprotocollen, met minimaal TLS1.2 maar bij voorkeur TLS1.3. Data-at-rest (databases, object storage, etc.) wordt standaard versleuteld. Hiervoor worden in Azure *customer-managed keys* (CMKs) gebruikt. Voor informatietransport over netwerk wordt encryptie toegepast, zoals hierboven vermeld. Tevens is het gebruik van onveilige protocollen, zoals FTP, HTTP, en SMTP, niet toegestaan voor data uitwisseling.

5.5 Authenticatie

Gebruikers identificeren zich door middel van een persoonsgebonden gebruikersnaam. Gebruikersnamen zijn dus altijd herleidbaar naar een natuurlijk persoon. Voor toegang tot codebases moet altijd 2-factor authenticatie (2FA) gebruikt worden. Waar mogelijk wordt ook single single-on (SSO) ingezet om het aantal wachtwoorden en aanmeldingen te reduceren. Applicaties and services in Azure authentiseren, waar mogelijk, *passwordless* m.b.v. *(user-assigned) managed identities*.

5.6 Autorisatie

Gebruikers en applicaties krijgen enkel de rechten die noodzakelijk zijn om hun taak te kunnen uitvoeren. Rechten worden niet aan individuele personen toegekend maar aan rollen (groepen) waar personen lid van zijn. Voor data (database, object storage, etc.) toegang wordt gewerkt met een drie standaard niveaus (zie Tabel 8). Voor veelvoorkomende rollen in Azure zijn er standaard groepen Azure (zie Tabel 9). Per omgeving en/of project zijn er aparte groepen voor toegangsautorisatie.

Tabel 8 – Standaard indeling voor rollen omtrent data toegang.

Niveau	Omschrijving
Tier 1	Gebruikers/applicaties op dit niveau hebben rechten voor lezen, schrijven, en uitvoering.
Tier 2	Gebruikers/applicaties op dit niveau hebben rechten voor lezen en schrijven.
Tier 3	Gebruikers/applicaties op dit niveau hebben alleen rechten voor lezen.

Tabel 9 – Standaard groepen voor veelvoorkomende rollen in Azure.

Groep	Omschrijving
Data Admins	Deze groep heeft volledige rechten op/in Azure Databases en Storage Accounts.
Infra Admins	Deze groep heeft contributor rechten op een specifieke Azure subscriptie.
RBAC Admins	Beheren toegangsrechten en rol toewijzingen voor een specifieke Azure subscriptie.
Secrets Admins	Beheren secrets, keys en certificaten in Key Vaults voor een specifieke Azure subscriptie.
Cloud Observers	Deze groep heeft lees rechten op kosten, security, en compliance rapportages.

5.7 Data residency

Voor opslag en verwerking van data mogen uitsluitend Azure locaties binnen de Europese Economische Ruimte (EER) gebruikt worden. West-Europa is de primaire Azure regio voor alle resources, met Noord-Europa als secundaire regio voor redundantie en disaster recovery doeleinden.

5.8 Networking

Voor netwerkisolatie wordt gebruikgemaakt van *Virtual Networks* (VNets), waarbij strikte scheiding tussen de OTAP omgevingen (zie sectie 2.3) wordt gehandhaafd. Verbinding tussen Azure resources en applicaties verloopt bij voorkeur via *Private Endpoints*, zodat dataverkeer binnen het Azure-netwerk blijft en niet over het publieke internet gaat. Elk subnet wordt beveiligd met *een Network Security Group* (NSG). Hierbij wordt voor uitgaand verkeer een allowlist principe gehanteerd; toegang tot het internet is standaard geblokkeerd tenzij expliciet toegestaan. Publiek toegankelijke applicaties maken bij voorkeur gebruik van een custom domein geconfigureerd met DNSSEC. Waar ondersteund, worden *Virtual Machines* (VMs) t.b.v. compute aangemaakt zonder publiek IP.

Bijlage 1

In de onderstaande tabel wordt de link gelegd tussen dit document en de NEN7510 normeisen.

Normeisen	Titel	Sectie(s)	Samenvatting
14.2.1 14.2.2 14.2.3 14.2.4	Wijzigingsbeheer	2.2, 3.5, 3.7	Wijzigingen aan systemen en applicaties worden beperkt tot het noodzakelijke, en worden beheerst door middel van het change management proces. Hierdoor worden wijzigingen centraal geregistreerd en beoordeeld. Wijzigingen aan code worden bijgehouden middels een versiebeheer systeem.
14.2.1 14.2.6	Functiescheiding	2.1, 5.6	Er wordt ontwikkeld volgens de DevOps werkwijze. Dit betekent, o.a., dat er geen strikte functie scheiding bestaat tijdens ontwikkeling. Foutgevoeligheid wordt gemitigeerd door automatisering en continue back-ups.
14.2.1 14.2.5 14.2.6	OTAP	2.3, 3.6	Om de continuïteit van de productieomgeving als ook het voortbrengingsproces te kunnen garanderen wordt er gebruik gemaakt van het OTAP-concept.
14.2.1 14.2.5 14.2.6	Hardening	3.4, 4.1, 5.2	Systemen worden 'gehardend` conform best practices, zoals OWASP richtlijnen.
14.2.1 14.2.8 14.2.9 14.3.1	Software testing	2.1, 4.3	Alvorens software of code in productie genomen wordt, wordt deze getest en gecontroleerd door een teamlid.
14.2.1 14.2.5 14.2.6	Gebruikers-identificatie	5.5	Gebruikers identificeren zich door middel van een persoonsgebonden gebruikersnaam. Gebruikersnamen zijn dus altijd herleidbaar naar een natuurlijk persoon.
14.2.1 14.2.5 14.2.6	2FA	5.5	Toegang tot codebases zal altijd op basis van 2 factor authenticatie (2FA) ingeregeld zijn.
14.2.1 14.2.5 14.2.6	SSO	5.5	Waar mogelijk wordt gebruik gemaakt van Single-Sign-On (SSO), zodat het aantal wachtwoorden en aanmeldingen gereduceerd wordt.
14.2.1 14.2.5 14.2.6	Autorisatiebeheer	5.6	Gebruikers krijgen enkel de rechten die noodzakelijk zijn om hun taak te kunnen uitvoeren. Rechten worden niet aan individuele personen toegekend maar aan rollen (groepen) waar personen lid van zijn.
14.2.1 14.2.5 14.2.6	Verzamelen van (systeem)logs	3.3	Relevante handelingen van gebruikers en componenten worden geregistreerd en weggeschreven in de logbestanden van de applicatie of het systeem. Logbestanden kunnen lokaal op de applicatie worden opgeslagen, of in een logmanagement omgeving.
14.2.1 14.2.5 14.2.6	Versleuteling	5.4	Data-in-transit wordt versleuteld op basis van hedendaagse encryptieprotocollen zoals TLS1.2 of TLS1.3. Data-at-rest wordt in principe niet versleuteld, tenzij er voldoende aanleiding is om dit, in lijn met het beveiligingsbeleid.
14.2.1 14.2.5 14.2.6	Gebruik van veilige protocollen	5.4	Voor informatietransport over netwerk wordt encryptie toegepast, zoals hierboven vermeld. Tevens is het gebruik van onveilige protocollen, zoals FTP, HTTP en SMTP, niet toegestaan voor data uitwisseling.
14.2.1 14.2.5 14.2.6	OWASP top 10	3.4, 4.1, 5.2	Gedurende het ontwikkelproces wordt in de testfase beoordeeld of de applicatie / het systeem vatbaar is voor veelvoorkomende kwetsbaarheden zoals beschreven in de OWASP top 10.

Bijlage 2

In de onderstaande tabel wordt de link gelegd tussen dit document en **compliance checks**.

Sectie	Link
5.1	- Key Vault keys should have an expiration date (**link**) - Key Vault secrets should have an expiration date (**link**) - Keys should have a rotation policy ensuring that their rotation is scheduled (**link**) - Secrets should have more than the specified number of days before expiration (**link**) - Secrets should have the specified maximum validity period (**link**)
5.4	- App Services should have minimum TLS version 1.3 (**link**) - SQL servers should use customer-managed keys to encrypt data at rest (**link**) - Storage accounts should use customer-managed keys for encryption (**link**)
5.5	- Deny direct user RBAC role assignments (**link**) - Use centralized identity and authentication system (**link**)
5.7	- Allowed locations (**link**) - Allowed locations for resource groups (**link**) - Audit resource location matches resource group location (**link**)
5.8	- App Services should have at least one custom domain configured (**link**) - NSGs should have deny internet outbound rule (**link**) - Key Vaults should only be accessible via private endpoint (**link**) - Storage accounts should only be accessible via private endpoint (**link**) - Azure SQL servers should only be accessible via private endpoint (**link**)