SOURCE-CODE: DE BRON VAN APPLICATIEBEVEILIGING

DE VOLGENDE STAP IN APPLICATIEBEVEILIGINGSONDERZOEKEN

Ing. M. (Michiel) van Veen RE MSc. CISA CISSP is als adviseur bij KPMG IT Advisory werkzaam in de groep Information Protection and Business Resilience. Michiel is specialist op het gebied van IT-beveiligingsadvies, IT auditing, IT-beveiligingsonderzoeken en penetratietesten. Zijn kennisgebied reikt van IT Governance tot technische beveiligingsimplementaties in detail: from board to bit. Hij is te bereiken via vanveen.michiel@kpmg.nl.

De laatste jaren neemt het aantal IT-beveiligingsincidenten sterk toe. Recente berichten over inbraken bij de FBI, CIA, Amerikaanse Senaat en bedrijven als Sony en Nintendo laten de ernst zien van deze dreiging. Dergelijke incidenten bevorderen het bewustzijn dat in vrijwel elk apparaat software zit. Van horloge tot spaceshuttle, alles wordt bepaald door software. Hierdoor ontstaat de behoefte om meer controle te krijgen en te houden op beveiliging in applicaties en het pad naar de data die het ontsluit, om zo te bewaken dat de software precies doet wat het moet doen en niet wat het niet moet doen.

Om meer controle te krijgen over de applicatiebeveiliging is het noodzakelijk inzicht te krijgen in de aanwezige maatregelen en de benodigde maatregelen. Typisch worden applicatiebeveiligingsonderzoeken uitgevoerd om dit in kaart te brengen. In een applicatiebeveiligingsonderzoek staan de IT-controlemaatregelen centraal en daarmee dus de (functie) scheiding tussen verschillende (gebruikers)groepen. Een veel gehanteerde scheiding is die tussen eindgebruikers, beheerders en ontwikkelaars.

- Ontwikkelaars staan aan de basis van de behoeftevoorziening naar de gebruikers. Zij vertalen de wensen en eisen in ondersteunende software. Daarmee dragen zij de verantwoordelijkheid om te zorgen dat de software correct werkt.
- Beheerders zorgen ervoor dat de software in het IT-landschap wordt opgenomen. Daartoe configureren zij de software en eventuele ondersteunende IT-infrastructuur zoals databases, servers en netwerk. Dit wordt zodanig uitgevoerd dat de eindgebruikers op de juiste manier toegang krijgen tot de software.
- De eindgebruikers zetten de software in voor de ondersteuning van de uit te voeren taken en maken zo dus gebruik van de software en de daarin besloten data.

Om de scheiding tussen deze groepen te implementeren is het belangrijk het pad van de gebruiker naar de data onder controle te hebben. Met behulp van bijvoorbeeld authenticatie- en autorisatiemiddelen zoals een gebruikersnaam en wachtwoord, wordt de scheiding technisch afgedwongen. Een applicatiebeveiligingsonderzoek richt zich op de technisch afgedwongen scheidingen. Typisch worden configuratiereviews en penetratietesten gebruikt voor het onderzoek.

Fig. 1. De nadruk van een applicatiebeveiligingsonderzoek ligt meer op IT-beheer in plaats van softwareontwikkeling.

Configuratiereviews hebben tot doel te valideren of alle instellingen de juiste scheidingen technisch afdwingen.Voor veel gebruikte software (zoals Windows besturingssystemen of Oracle databases) worden zogeheten beveiligingsconfiguratiebaselines gebruikt. Deze baselines zijn in feite controlelijsten met configuratie-instellingen zoals die geïmplementeerd moeten zijn. Afwijkingen kunnen soms leiden tot het doorbreken van de scheidingen en daarmee tot ongeautoriseerde toegang tot de software en data. Penetratietesten hebben tot doel inzicht te verschaffen in de aanwezige beveiligingszwakheden. Vaak wordt hiertoe met speciale applicaties en IT-beveiligingspecialisten (ethical hackers) getracht zwakheden te identificeren door de software op zowel gebruikelijke als ongebruikelijke manieren te benaderen. De controles in de software worden omzeild, waardoor ongeautoriseerd toegang wordt verkregen tot de software en data.

Configuratiereviews richten zich vooral op de beheerders. Bevindingen uit dit type onderzoek richten zich op het afdwingen van controle vanuit de software naar de eindgebruiker. Deze worden vervolgens naar beheerders gecommuniceerd die de nodige aanpassingen doorvoeren in de IT-infrastructuur. Verder dragen zij zorg voor de periodieke controle van deze instellingen en waarborgen zo de effectiviteit van de maatregelen over tijd.
Penetratietesten richten zich vooral op de eindgebruikers. Bevindingen uit dit type onderzoek richten zich op de mogelijkheden die de eindgebruiker heeft in de

interactie met de software. Ook deze bevindingen komen uiteindelijk terecht bij beheerders die de nodige maatregelen implementeren of aanscherpen. Daarnaast worden bevindingen die voortkomen uit fouten in de software zelf gecommuniceerd naar de ontwikkelaars. Kortom, de bovenstaande onderzoeksmethodes laten zien dat veel nadruk ligt op het beheren van IT en minder op softwareontwikkeling. Dit is vreemd aangezien juist de softwareontwikkeling bepaalt in welke mate vereiste controlemaatregelen technisch worden afgedwongen in de source-code. Om in de behoefte van meer controle over applicatiebeveiliging te voorzien, dient ook de source-code zelf te worden onderzocht. Fig.1. geeft het onderzoeksgebied duidelijk aan.

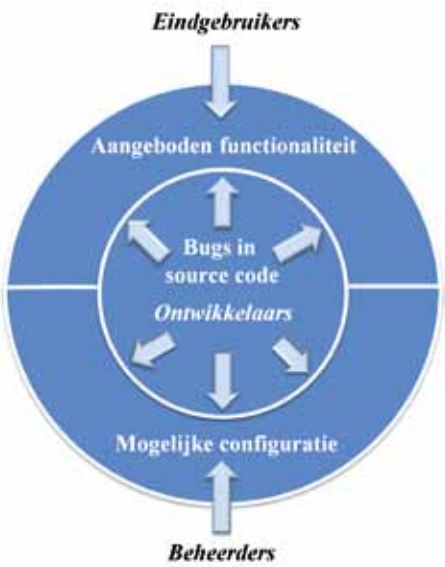

Fig. 2. Bugs in de source-code hebben een grote invloed op de applicatiebeveiliging.

Source-code en haar rol in de applicatiebeveiligingsproblematiek

Applicaties worden geschreven door een softwareontwikkelaar waarbij deze de wensen en eisen van de opdrachtgever in een ontwerp van source-code vertaalt. Die source-code wordt geschreven door ontwikkelaars die daarmee bepalen wat de applicatie wel en niet doet. Het is dus belangrijk vast te stellen dat de applicatie aan de wensen en eisen voldoet. De applicatie moet doen wat zij belooft. Dit lijkt triviaal maar dat is het niet. Immers, een applicatie wordt geschreven door mensen en die maken soms fouten. Deze fouten (bugs) kunnen ervoor zorgen dat de applicatie zich anders gedraagt dan verwacht. Omdat een applicatie al vrij snel complex wordt door de vele commando's in de source-code en hun onderlinge samenhang en afhankelijkheden, is een bug snel geïntroduceerd. Bovendien zijn bugs door de complexiteit moeilijk op te sporen. Een softwareontwikkelaar moet immers beredeneren hoe een computer in een bepaalde situatie omgaat met het opgegeven commando. Ook het vervolgens verhelpen van bugs is geen sinecure. Zeker bij grote applicaties is het vrijwel onmogelijk alle bugs in de software te ontdekken en te verhelpen.

De invloed van bugs op applicatiebeveiliging

Het valideren van de beoogde functionaliteit (eisen en wensen) ten aanzien van de applicatie is nog relatief eenvoudig. Ingewikkelder wordt het om vast te stellen dat de applicatie ook niet doet wat ze niet belooft. Biedt de applicatie meer mogelijkheden aan voor eindgebruikers om data te benaderen dan de bedoeling is? Bijvoorbeeld zonder authenticatie en autorisatie? Of biedt de applicatie, zonder dat het de bedoeling is, toegang tot onderliggende IT-infrastructuur (databases, servers en netwerk)?
In applicaties speelt controle dus een belangrijke rol. Een belangrijk aspect van de controle is het vaststellen of een gevraagde actie wel is toegestaan. Alleen op die manier is het mogelijk preventief te garanderen dat de computer op een juiste manier reageert op een verzoek en zo dus doet wat hij belooft. Daarmee is het authenticeren en autoriseren van gebruikers en hun acties een belangrijke functionaliteit binnen een applicatie. Met authenticatie en autorisatie kan een organisatie controle uitoefenen op bijvoorbeeld het aanpassen van financiële data, het lezen van vertrouwelijke gegevens of het beschikbaar houden van systemen. Vaak maakt een organisatie een vertaalslag van rollen en verantwoordelijkheden en voert deze door in de applicatie.
Het testen van de authenticatie en autorisatiemechanismen in een applicatie is een belangrijk onderdeel van een applicatiebeveiligingstest. Bugs in de authenticatie- en autorisatiemechanismen kunnen het gebruik van de applicatie ondermijnen omdat de scheidingen tussen gebruikers onderling wordt doorbroken. Daarmee wordt feitelijk het model van rollen en verantwoordelijkheden uit de organisatie doorbroken. Dat dit een impact kan hebben op de beveiliging blijkt uit de vele fraude-incidenten, privacyschendingen en recente beschikbaarheidproblemen van websites. Een recent voorbeeld is het stelen van meer dan 130.000.000 creditcardnummers van de betalingsverwerker Hartland door een aanval van buiten het bedrijf. [ITA,2010]
Naast bugs die een functioneel karakter hebben zoals authenticatie- en autorisatiemechanismen, zijn er ook non-functionele bugs. Een bekend voorbeeld is het gebrek aan invoervalidatie. Afhankelijk van het gebruik van deze invoer in de applicatie kan dit grote beveiligingsgevolgen hebben. Wanneer de ongevalideerde invoer bijvoorbeeld wordt gebruikt om een achterliggende database te bevragen, kan de gebruiker zelf extra database-commando's (SQL) toevoegen en zo ongeautoriseerd alle data opvragen, aanpassen of verwijderen. Hierdoor kan een gebruiker alle controles in de applicatie omzeilen. Neem als voorbeeld een salarisadministratie. De software zorgt ervoor dat personeel dat de salarisbetaling uitvoert niet dezelfde medewerkers zijn als diegene die de betaling controleren en autoriseren: functiescheiding. Wanneer een gebruiker de invoervalidatiebug misbruikt kan hij zelf de salarisbetaling direct via de database doorvoeren en omzeilt daarmee alle controles met alle gevolgen van dien.
Een top-10 lijst van veelvoorkomende

Zeker bij grote applicaties is het vrijwel onmogelijk alle bugs in de software te ontdekken en te verhelpen

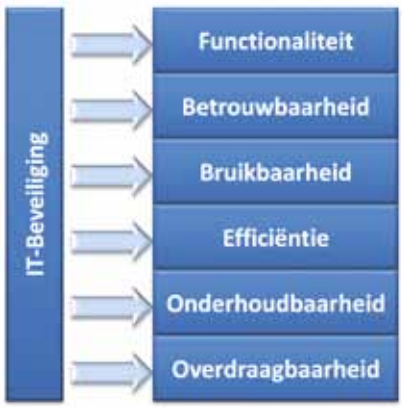

Fig. 3. IT-beveiliging is van toepassing op elk van de zes aspecten van de softwarekwaliteit ISO-norm.

applicatiefouten is terug te vinden in de OWASP top 10 [OWASP,2010].
Kortom, de source-code heeft grote invloed op applicatiebeveiliging. Fig. 2. laat duidelijk zien welke invloed de bugs in source-code hebben op de applicatiebeveiliging. Dit betekent dat het belangrijk is om in een applicatiebeveiligingsonderzoek ook onder de motorkap te kijken van de software om de interne werking vast te stellen, een source-codeonderzoek.

Het is belangrijk om in een applicatiebeveiligingsonderzoek ook onder de motorkap te kijken: source-codeonderzoek

Het uitvoeren van een source-codeonderzoek

De aanpak voor het onderzoeken van source-code is sterk afhankelijk van de vereiste breedte en diepgang.Twee bekende aanpakken zijn het onderzoeken van de kwaliteit van software en source-code reviews.
Een bekende invulling van de kwaliteit van software is gegeven in de ISO-norm 9126. De norm verdeelt de kwaliteit van software op basis van de volgende kenmerken: functionaliteit, betrouwbaarheid, bruikbaarheid, efficiëntie, onderhoudbaarheid en overdraagbaarheid. Door het onderzoeken van deze aspecten ontstaat het beeld of men kan vertrouwen op de software. Daarmee beslaat dit onderzoek meer dan de source-code alleen. De kwaliteit van de software wordt in zijn geheel bekeken. Zo wordt deze aanpak ook vaak gebruikt om de software te koppelen aan de (beheer)kosten daarvan, omdat software met lage kwaliteit veel en moeizaam onderhoud vergt.
In de ISO-norm is applicatiebeveiliging als aspect gehangen onder 'functionaliteit'. Deze classificering staat in contrast met die van functionele en non-functionele eisen. Applicatiebeveiliging kan namelijk zowel functionele als non-functionele eisen hebben. Een functionele eis is bijvoorbeeld als een gebruiker moet zich kunnen authenticeren met behulp van een gebruikersnaam en wachtwoord. Een non-functionele eis is als het wachtwoord opgeslagen dient te worden met behulp van een one-way-hash-functie. Dit betekent dat applicatiebeveiliging niet te plaatsen is onder 'functionaliteit' in de ISO-norm. Wanneer applicatiebeveiliging met de softwarekwaliteittaxonomie wordt vergeleken, blijkt dat applicatiebeveiliging op elk van de zes kenmerken invloed heeft en dus als integraal onderdeel van softwarekwaliteit beschouwd dient te worden. Daarmee heeft de implementatie van de applicatiebeveiliging in software een grote invloed op de kwaliteit van de software en dus op het functioneren van de software in brede zin. In fig.3. is de positie van IT-beveiliging weergegeven in relatie tot de ISO-normaspecten.
Source-code reviews richten zich op de diepgang en minder op de breedte. Het reviewen van source-code specifiek voor applicatiebeveiliging is minder bekend dan softwarekwaliteitsonderzoeken. Praktische methodes die zich op applicatiebeveiliging focussen zijn [McGraw,2001] [McGraw,2004] [McGraw,2006]. Bij het reviewen van source-code wordt onderscheid gemaakt tussen kwantitatief en kwalitatief onderzoek. In een kwantitatief onderzoek worden vaak aspecten bekeken zoals complexiteitsmetrieken, codeduplicatie, functiegroottes, overervingen in objectenstructuren enz. Met behulp van tools wordt de volledige source-code doorgelicht en worden statistische berekeningen gemaakt. De uitkomst geeft vaak goede handvatten in een onderzoek naar de beheerbaarheid, overdraagbaarheid en uitbreidbaarheid van de software.
Kwalitatieve onderzoeken worden veelal met de hand uitgevoerd waar een specialist de source-code in de diepte onderzoekt. Voor een applicatiebeveiligingsonderzoek betekent dit, dat kwetsbaarheden worden gezocht die de beveiliging kunnen beïnvloeden. Zoals eerder besproken kan applicatiebeveiliging functioneel en non-functioneel worden geïmplementeerd. In een source-code review is het belangrijk om beide benaderingen mee te nemen.

- De eerste stap in het onderzoek is een lexicale analyse. In deze stap wordt onderzocht of de source-code gestructureerd is, alles goed is gedocumenteerd in de code, programmeerstandaarden worden toegepast, variabelendeclaraties consistent zijn en de code goed leesbaar is. De lexicale analyse geeft een eerste indruk van de source-codekwaliteit en mogelijke probleemgebieden.
- De tweede stap in het onderzoek is de control flow relation analyse. In

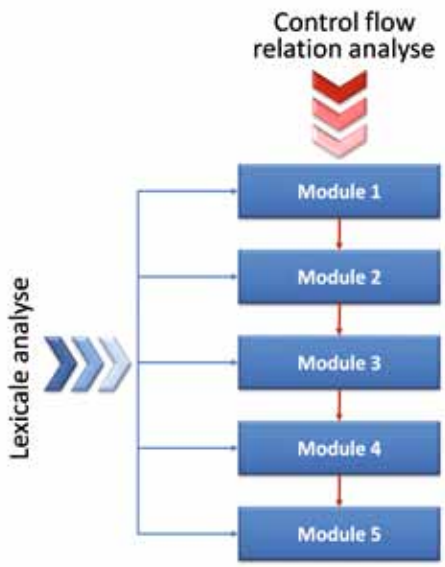

Fig. 4. De relatie tussen lexicale analyse en control flow relation-analyse van de source-code.

deze stap loopt de onderzoeker door de source-code heen op basis van de functieaanroepen in de applicatie. Zo worden de stappen gevolgd die de applicatie zelf doorloopt bij het uitvoeren van acties. De nadruk ligt op aspecten als foutafhandeling, (technische) aannames, design patterns, code hergebruik en logische objectstructuren.

De control flow relation analyse richt zich op de keten als geheel. De lexicale analyse op iedere schakel in die keten. In fig. 4. is deze relatie goed te zien. De resultaten uit beide stappen leveren een goed overzicht op van beveiligingskwetsbaarheden veroorzaakt door de manier van source-code schrijven of door de functionele samenhang van applicatieonderdelen.

Source-codeonderzoek als onderdeel van een applicatiebeveiligingsonderzoek
Zoals besproken bestaat het uitvoeren van een applicatiebeveiligingstest typisch uit een penetratietest, eventueel gecombineerd met een configuratiereview. Een source-codereview is geen vervanging van deze aanpak maar juist een belangrijke aanvulling daarop. Deze aanvulling helpt bij het identificeren van meer en complexe kwetsbaarheden in applicaties binnen een kortere tijd. Dit is duidelijk te zien in fig.5.
Het uitvoeren van een source-codereview in het kader van een applicatiebeveiligingstest kan op twee manieren worden ingezet.

- *Bottom-up-aanpak*
  In een bottom-up-aanpak wordt eerst begonnen met het analyseren van de source-code. Daarmee worden mogelijke zwakheden geïdentificeerd. Deze zwakheden worden vervolgens in een penetratietest gevalideerd. Hierdoor worden 'false-positives' eruit gehaald.

- *Top-down-aanpak*
  In een top-down-aanpak wordt eerst begonnen met een penetratietest. Wanneer vermoedens bestaan van een beveiligingslek, dan wordt de desbetreffende source-code opgezocht. In de source-codeanalyse wordt vervolgens gevalideerd of er sprake is van een beveiligingslek.

De keuze van de aanpak hangt samen met het doel van de applicatiebeveiligingstest. Wanneer het doel van de test is om zoveel mogelijk beveiligingslekken op te sporen en hiervoor aanbevelingen aan te dragen, dan is een bottom-up-aanpak het meest geschikt. Door het integraal analyseren van de source-code worden alle mogelijkheden geïdentificeerd en reële dreigingen zichtbaar gemaakt met een penetratietest. Wanneer het doel is om vast te stellen wat een anonieme aanvaller zou kunnen (ontdekken) is een top-down-aanpak logischer. Immers, de anonieme aanvaller heeft geen beschikking over de source-code (afhankelijk van het open source-karakter) en zal die dus niet als basis gebruiken. Wel kan de source-codeanalyse worden gebruikt om de efficiëntie van de penetratietest te verhogen. Bij een penetratietest is de tijd vaak gelimiteerd, terwijl een echte aanvaller meer tijd en resources tot zijn beschikking heeft.
Het is verstandig om de top-down- en bottom-up-aanpak te combineren. Hierdoor worden zoveel mogelijk beveiligingskwestbaarheden ontdekt en wordt tevens bepaald wat een anonieme hacker in een realistische situatie zou kunnen doen. Samen met een configuratiereview kan de applicatiebeveiligingstest compleet worden gemaakt.

Top-down en bottom-up: wat kan een anonieme hacker in een realistische situatie doen?

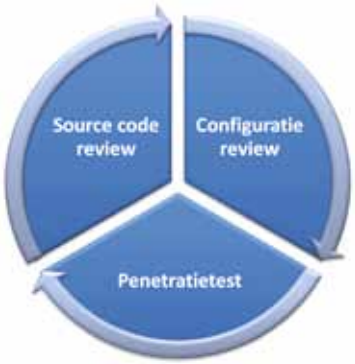

Fig. 5. Het verband tussen een penetratietest, configuratiereview en een source-code review.

Conclusie
Traditionele benaderwijzen van penetratietesten en configuratiereviews voor het testen van applicatiebeveiliging komen onder druk te staan door het toenemend aantal inbraakpogingen op applicaties. Dit stimuleert de vraag naar een aanvulling in de huidige aanpak: source-codereview. Door een source-codereview kan applicatiebeveiliging, als belangrijke invloed van de softwarekwaliteit, verder worden beheerst. Dit voorkomt dat eindgebruikers de software anders laten gedragen (manipulatie) of de software meer laten doen dan de bedoeling is (injectie). Net als de voor de hand liggende functionele en non-functionele eisen, dienen ook applicatiebeveiligingseisen expliciet te worden opgesteld aan het begin van de software-ontwikkelcyclus. Dit waarborgt dat de applicatiebeveiliging al tijdens de software-ontwikkeling wordt getest. Met voldoende aandacht voor applicatiebeveiliging en in juiste balans met gebruikersvriendelijkheid wordt zo veiligere software ontwikkeld en gebruikt. Daarmee kan de samenleving verder steunen en vertrouwen op een juiste ondersteuning door software.

Literatuur

Identity Theft Awareness. 2010 Security Incidents, www.identity-theft-awareness.com/2010-security-incidents.html, 2010

ISO/IEC 9126-1:2001. Software engineering - Product quality - Part 1: Quality model. www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749, 2011

McGraw et al. Secure Software: How to Avoid Security Problems the Right Way, 2001

McGraw et al. Exploiting Software: How to Break Code, 2004

McGraw et al. Software Security: Building Security In, 2006

OWASP. OWASP top 10,www.owasp.org, 2010