GAI NetConsult GmbH— Am Borsigturm 58 — 13507 Berlin — www.gai-netconsult.de — Tel: 030/417898-0

Source Code Scanner - sinnvoll einsetzbar?

Martin Rahnefeld (GAI NetConsult GmbH)

Februar 2012 Sonderdruck aus Security *Journal* #59

In Zeiten immer komplexer werdender Software gestaltet sich die Einhaltung von Programmierstandards und -richt-linien immer schwieriger, besonders wenn das Programm einen gewissen Sicherheitsstandard aufweisen soll. Manuelle Überprüfungen des gesamten Quellcodes werden mit zunehmender Programmgröße immer umständlicher oder sogar unmöglich, sodass neue Wege zum Aufspüren von Schwachstellen gefunden werden müssen. Eine Antwort auf dieses Problem könnte die automatische Überprüfung mit Source Code Scannern sein. Der Artikel analysiert hierzu den aktuellen Stand der Technik und gibt die Erfahrungen aus ersten Projekteinsätzen wieder.

Es kann davon ausgegangen werden, dass Programmierer bei komplexen Anwendungen häufig Fehler machen, unabhängig davon, wie fachkundig sie sind oder wie sehr sie sich konzentrieren. Auf syntaktische Fehler, wie beispielsweise fehlende Semikolons oder Klammern, wird der Programmierer meistens durch den Compiler aufmerksam gemacht. Diese Fehler lassen sich relativ leicht beheben, da sie bereits während der Programmierung bemerkt werden können. Sicherheitslücken dagegen werden nicht durch den Compiler gefunden und können mehrere Jahre versteckt im Quellcode verbleiben. Je länger eine Schwachstelle unentdeckt bleibt, desto teurer ist aber auch erfahrungsgemäß die Behebung derselben.
Wird die Entwicklung der Softwarebranche betrachtet, so ist festzustellen, dass die Programmierer immer wieder die gleichen sicherheitsrelevanten Fehler machen. Aus diesem Grund gibt das „Open Web Application Security Project" (OWASP), eine offene Community mit dem Ziel die Sicherheit im Bereich Internet zu verbessern, seit Jahren neben anderen wichtigen Dokumenten eine Top-10 der Sicherheitslücken heraus [4, 6].

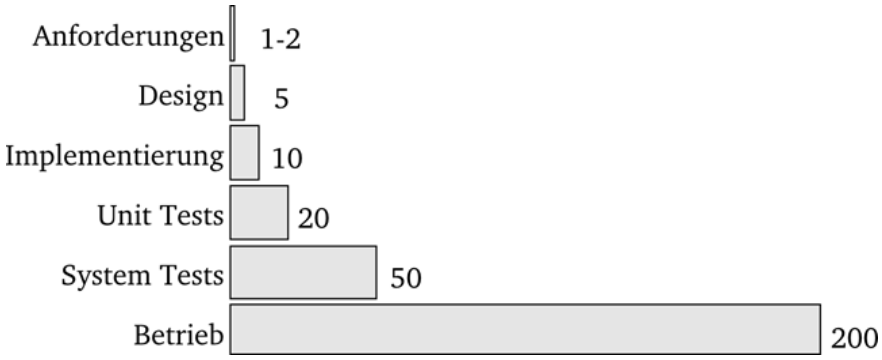

Abbildung-1: Kostenfaktoren der Behebung von Softwarefehlern nach Phasen [10]

Source Code Scanner versuchen die häufigsten Programmierfehler automatisch zu finden. Sie gehören somit zur Klasse der Analysewerkzeuge und haben das Ziel, die Softwareentwicklung, aber auch die Auditierung, zu unterstützen, indem die Durchführung von Prüfungen auf den Quellcode vereinfacht wird. Zur Erfüllung dieser Aufgabe bedienen sich Source Code Scanner eines festen Regelsatzes beziehungsweise Muster, anhand derer der Quellcode analysiert wird.
Es lassen sich zwei grundsätzliche Arten von Scannern unterscheiden, zum einen der statische und zum anderen der dynamische Analyseansatz. Im Gegensatz zur statischen setzt die dynamische Analyse die Ausführbarkeit der zu testenden Software voraus. Das grundsätzliche Prinzip ist hierbei die Ausführung des Programms mit vordefinierten Eingabefällen. Sofern sich das erzielte von dem erwarteten Ergebnis unterscheidet, liegt eine mögliche Schwachstelle vor. Demzufolge lassen sich dynamische Scanner erst nach Abschluss der Programmierung verwenden, da erst dann eine ausführbare Version der Software vorliegt.

Während dynamische Scanner bereits seit Jahren erfolgreich eingesetzt werden, sind die statischen Source Code Scanner noch nicht wirklich ausgereift und in vielen Bereichen durchaus noch Forschungsgegenstand. Die Gartner Group veröffentlicht seit Jahren den sog. „Hype Cycle", auf dem die Entwicklung neuer Technologien mit ihren typischen Phasen verzeichnet ist. Hier sieht man deutlich die unterschiedlichen Reifegrade der dynamischen bzw. statischen Scanner.

Aufgrund dessen, dass für die Überprüfung der Software nicht immer vollständiger Quellcode vorliegt und damit eine Ausführung der Software nicht möglich ist, wird nachfolgend nur auf die statische Analyse eingegangen.

Statische Source Code Scanner

Eine statische Analyse kann auch bei einem unvollständigen Quellcode durchgeführt werden. Grundsätzlich können statische Analysen aber auch ausschließlich manuell durchgeführt werden, dies hat jedoch den Nachteil, dass der Zeitaufwand bei komplexen Projekten stark an-

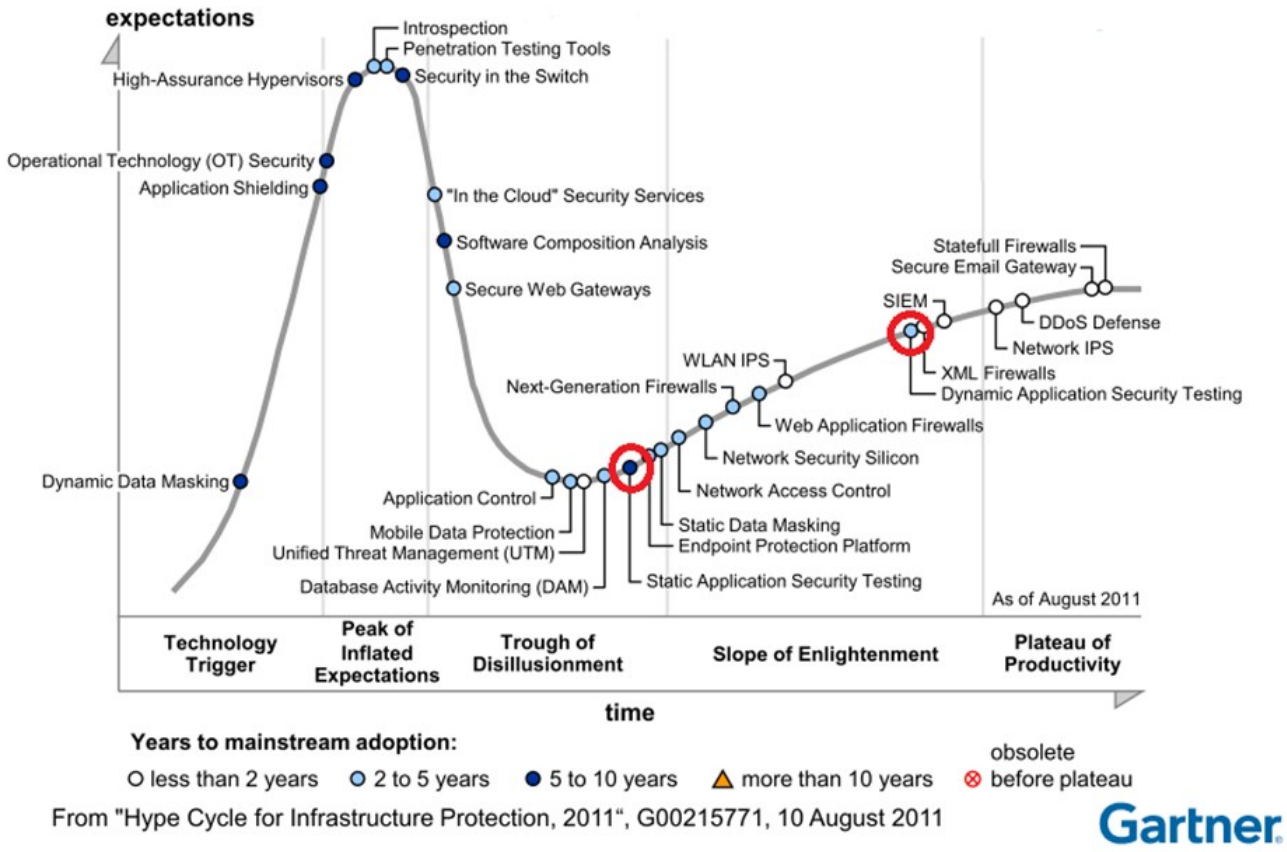

Abbildung-2: Entwicklungsstufe der Source Code Scanner [11]

steigt. Aufgrund der oftmals monotonen Tätigkeit, die sich aus der Anwendung der recht einfachen Regeln auf den komplexen Quellcode ergibt, ist zu erwarten, dass eine solche Tätigkeit schnell ermüdend ist und somit zu qualitativ eingeschränkten Ergebnissen führt. Somit eignen sich für einfache Überprüfungen automatische Source Code Scanner, die einen Großteil der ansonsten manuellen Arbeit übernehmen können.

Source Code Scanner verrichten ihre Arbeit ähnlich wie ein Compiler. In einem der ersten Schritte werden die verwendete Programmiersprache und das jeweilige Projektverzeichnis ausgewählt. Anschließend besteht für den Anwender die Möglichkeit, den Quellcode nach bestimmten Regelsätzen durchsuchen zu lassen. Einige fortschrittlichere Source Code Scanner geben dem Anwender die Möglichkeit die anzuwendenden Regeln gezielt zu selektieren oder eigene Regelsätze hinzuzufügen. Nach Start des Scanners wird dieser als Erstes die Struktur des gegebenen Quellcodes analysieren. Dabei wird u. a. geprüft, ob der Quellcode den syntaktischen Anforderungen der Programmiersprache entspricht und ob grundlegende semantische Bedingungen gegeben sind. Beispielsweise kann geprüft werden, ob Variablen entsprechend deklariert worden sind oder ob die Typzuweisungen korrekt sind. Dies sind jedoch Aufgaben, die rudimentär auch von einem Compiler durchgeführt werden und nicht zwingenderweise einen Source Code Scanner voraussetzen. Im zweiten Schritt findet eine erweiterte Analyse statt. Im einfachsten Fall wird beispielsweise nach Schlagwörtern gesucht, die nicht vorkommen dürfen. Mit dieser Vorgehensweise können Funktionen gefunden werden, die nicht mehr verwendet werden sollten oder ein potenziell unerwünschtes Verhalten aufweisen. In der Programmiersprache PHP würde so zum Beispiel der Aufruf „shell_exec()" eine Fehlermeldung auslösen, da dieser Befehl dazu verwendet werden kann beliebigen Code auf dem Hostsystem auszuführen. Diese Suche könnte jedoch auch im einfachsten Fall durch ein „grep"-Kommando im Zusammenspiel mit einer Liste von unerwünschten Funktionen durchgeführt werden.

Fortschrittlichere Source Code Scanner sind zudem in der Lage den Programmablauf zu deuten und können nicht nur nach vordefinierten Mustern suchen, sondern diese auch im eingeschränkten Kontext deuten. So kann beispielsweise geprüft werden, ob vermeintlich schädlicher Quellcode ausgenutzt werden kann. Möglicherweise hat der Entwickler dieses schon bedacht und fängt falsche Parameter im Vorfeld ab? Einfache Source Code Scanner versagen an dieser Stelle. Eine weitere Möglichkeit der Analyse stellt die Beobachtung der Datenströme innerhalb des Programms dar. Bei dieser Variante wird überprüft, ob die Eingabedaten nach der Verarbeitung das erwartete Er-

gebnis liefern. Innerhalb des Datenstroms könnte es Stellen geben, die fehlerhaft sind. Beispielsweise könnte geprüft werden, ob eine Variable vor dem Zugriff initialisiert wurde und gültige Werte enthält. Wenn eine Variable innerhalb des Datenstroms freigegeben wurde und anschließend wieder referenziert wird, so wird der Datenstrom durch nicht vorhersagbare Werte aus der Variablen beeinflusst. Dieser Vorgang stellt eine Anomalie des Datenstroms dar und würde bei einem Source Code Scanner zu einer Fehlermeldung führen. Ein weiterer Nutzen aus der Erstellung von Programmablaufplänen ergibt sich aus der verbesserten Lesbarkeit für den Anwender, da der vorhandene Quellcode durch den Source Code Scanner in einem Kontrollflussdiagramm dargestellt wird und somit komplexe Zusammenhänge visualisiert werden können [1].

Vor– und Nachteile von Source Code Scannern

In diesem Abschnitt soll auf die einzelnen Vor- und Nachteile bei der Verwendung von Source Code Scannern eingegangen werden. Einer der Vorteile wurde bereits angesprochen, bei der Verwendung der statischen Methode kann auch unvollständiger Quellcode analysiert werden. Bereits ab der ersten Zeile des Programmierens kann die Analyse durchgeführt werden. Daraus ergibt sich, dass potenziell vorhandene Schwachstellen bereits in einer sehr frühen Phase gefunden werden können. Da ein Source Code Scanner das Problem direkt im Quellcode markiert, im Gegensatz zum üblichen Ansatz der Überprüfung einer laufenden Anwendung von „außen", können Fehler schnell behoben werden. Somit lassen sich auch die Kosten der Fehlerbehebung senken. Wenn zudem die Analyse in den bereits vorhandenen Erstellungsprozess integriert wird, entstehen weitere Synergieeffekte, beispielsweise kann auf diese Art der Quellcode jede Nacht automatisch überprüft werden. Dieser Vorgang kann selbst auf große Projekte angewandt werden, da Source Code Scanner in der Lage sind, selbst große Mengen an Quellcode in einer vertretbaren Zeit zu analysieren. Durch die Wiederholbarkeit lassen sich zudem Metriken bilden, anhand derer die Qualitätssicherung den Zustand des Quellcodes bewerten kann. Für den Programmierer haben Source Code Scanner den Vorteil, dass für die Analyse keine Testfälle geschrieben werden müssen. Es ist lediglich eine Anpassung der Regelsätze an den Quellcode notwendig. Dementsprechend können Source Code Scanner bestimmte Arten von Schwachstellen mit einer relativ hohen Wahrscheinlichkeit entdecken, dazu gehören beispielsweise auch Speicherüberläufe oder SQL-Injektionen.

Positiv hervorzuheben ist weiterhin, dass fortschrittlichere Scanner ausführliche Beschreibungen der gefundenen Fehler bieten und somit dem Entwickler die Schwachstellen erklären. Der Entwickler lernt auf diese Weise die gefundenen Schwachstellen in Zukunft zu vermeiden, indem sicherere Programmiermethoden angewandt werden. Aus diesem Aspekt leitet sich jedoch sogleich eine weniger erfreuliche Eigenschaft der Scanner ab. Die Programmierer können nur aus den Fehlern lernen, die ein Source Code Scanner meldet. Schwachstellen, die nicht gefunden werden, verbleiben weiterhin im Quellcode. Dementsprechend hat der Einsatz dieser Source Code Scanner auch erhebliche Nachteile, die im Folgenden angesprochen werden sollen. Gerade ältere Source Code Scanner neigen dazu, übermäßig viele Meldungen zu produzieren, die zudem nicht immer zutreffen („false positives"). Der Scanner meldet also Schwachstellen, obwohl der Quellcode an dieser Stelle unkritisch ist. Je mehr false positives durch ein Programm gemeldet werden, desto ineffizienter wird die Analyse, da der Aufwand der manuellen Nacharbeit deutlich steigt. Selbst fortschrittlichere Scanner, die die Rate der false positives zu verringern suchen, arbeiten noch relativ ungenau, sodass weiterhin ein hoher manueller Aufwand vorzusehen ist. Es bedarf zumeist mit der Sicherheitsthematik vertrautem Personal, das die gemeldeten Schwachstellen manuell analysiert und bewertet. Selbst dann ist es oftmals problematisch, die Schwachstelle genau einzuschätzen, da es einer gewissen Einarbeitungszeit in den Quellcode erfordert, wenn beispielsweise ein Fremder die Überprüfung durchführt.

Ein weiteres Problem stellen die „false negatives" dar, das sind Schwachstellen, die nicht vom Scanner gefunden worden sind. Dies liegt daran, dass Source Code Scanner nur auf die Schwachstellen aufmerksam machen können, für die sie entsprechende Regeln besitzen. Dementsprechend können in aller Regel nicht alle vorhandenen Schwachstellen gefunden werden und das Programm ist weiterhin nicht völlig sicher. Erschwerend kommt hinzu, dass einige Sicherheitsschwachstellen nur sehr schwer automatisiert auffindbar sind, dies betrifft beispielsweise das Sessionmanagement, Authentifizierungsprobleme oder Zugriffskontrollen. Auch wenn die Source Code Scanner langsam besser werden, so kann doch nur ein kleiner Prozentsatz der möglichen Schwachstellen automatisiert gefunden werden. Ebenfalls sind Source Code Scanner kaum in der Lage, Probleme in der Architektur oder im Design von Programmen feststellen zu können. Auch wenn die statische Analyse das Scannen von unvollständigem Quellcode als einen der Vorteile herausstreicht, so ergeben sich aus dieser Unvollständigkeit weitere Probleme. Sofern nur ein Teil des Quellcodes geprüft wird, kann nicht auf das gesamte Programm geschlossen werden, oftmals fehlen externe Komponenten, wie beispielsweise Datenbanken oder bestimmte Frameworks. Schwachstellen, die erst in Kombination mit diesen Komponenten auftreten, können bei Teilüberprüfungen nicht gefunden werden. Auch beeinflussen Umgebungsvariablen die Sicherheit eines Programms, da sie in der Regel für den Source Code Scanner unbekannt sind, sodass dieser Annahmen über die Umgebung treffen muss [2, 3 und 5].

Auswahl von Source Code Scannern

Soll der Quellcode eines Programms geprüft werden, so ist zuerst ein entsprechender Scanner auszuwählen, doch welcher bietet die besten Ergebnisse und wie ist die Auswahl zu treffen?

Der Vergleich zwischen den Fähigkeiten der einzelnen Scanner ist nicht trivial, da die Auswahl der Vergleichsgrößen schwierig ist. Die Anzahl der enthaltenen Regeln oder Muster erscheint auf den ersten Blick als ein guter Ansatz. Es ist zu vermuten, dass ein Scanner besser ist, je mehr Regeln und Muster enthalten sind. Dies trifft nach unseren Erfahrungen jedoch nicht immer zu, es ist nicht primär bedeutend wie viele Regeln insgesamt enthalten sind, wichtiger ist die Anzahl der Regeln, die auf den eigenen Anwendungsfall zutreffen. Denn Regeln, die nicht verwendet werden, erhöhen nur die Komplexität des Scanners und erschweren die Anpassung. Dementsprechend könnte versucht werden die Anzahl der relevanten Regeln als Kennzahl der Vergleichbarkeit zu verwenden, aber auch dieser Ansatz birgt Risiken, da ein Regelsatz mehrere Schwachstellen finden kann. Je nach Aufbau der Regeln unterscheidet sich zudem die Qualität der gefundenen Schwachstellen, sodass diese Kennzahl sich nur bedingt zum Vergleich eignet.

Eine weitere Möglichkeit wäre die Arbeitsgeschwindigkeit. Es könnte der Scanner genommen werden, der am schnellsten die Ergebnisse liefert. Jedoch wird eine Überprüfung des Quellcodes zumeist nachts durchgeführt, sodass es ohne große Bedeutung ist, wann genau der Scan beendet ist. Es ist nur wichtig, dass am nächsten Tag die Ergebnisse vorliegen, somit ist die Geschwindigkeit höchstens ein sekundärer Vergleichsfaktor.

Zur Auswahl eines Source Code Scanners eignet sich daher besser die Benutzbarkeit. Beispielsweise ist es wichtig, ob sich die Ergebnisse einer Analyse zwischenspeichern lassen oder ob alle Dateien erneut gescannt werden müssen, obwohl keine Änderungen aufgetreten sind. Es sollte auch ausschließlich der Quellcode und keine irrelevanten Ressourcendateien oder Kommentare gescannt werden. Weiterhin ist die Darstellung der Ergebnisse wichtig. Ein effizientes Arbeiten wird erst dann möglich, wenn Ergebnisse gefiltert und „false positives" als solche markiert werden können. Ebenso sollte auf eine Integration in die Entwicklungsumgebung des Programmierers geachtet werden, sofern die Source Code Analyse entwicklungsbegleitend eingesetzt wird [7, 8, und 9]. Allgemeinere Empfehlungen für die Auswahl eines Source Code Scanners gibt beispielsweise das OWASP [3]:

- Unterstützt der Scanner die entsprechende Programmiersprache?
- Benötigt der Scanner einen vollständigen Quellcode?
- Welche Schwachstellen kann der Scanner entdecken?
- Können auch kompilierte Programme überprüft werden?
- Kann der Scanner in die Entwicklungsumgebung integriert werden?
- Wie sehen die Lizenzkosten aus?

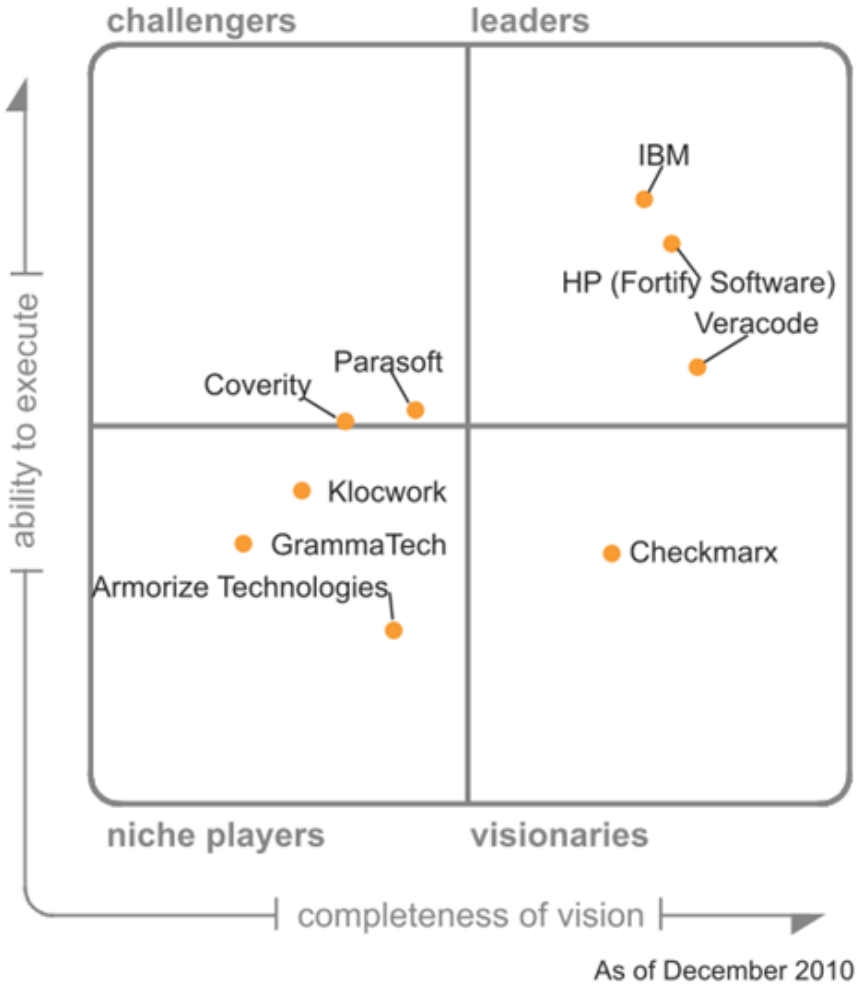

Abbildung-3: Gartner Magic Quadrant for Static Application Security Testing [12]

Die Analysten von Gartner veröffentlichten in ihrer Marktanalyse eine Übersicht der Reife der unterschiedlichen kommerziellen Anbieter von Source Code Scannern, anhand derer sich auch eine erste Vorauswahl treffen lässt. Bei dieser Analyse sind jedoch die kostenfreien Scanner nicht berücksichtigt worden.

Fazit

Zusammenfassend kann festgestellt werden, dass trotz einiger negativer Punkte Source Code Scanner durchaus eine Daseinsberechtigung haben, da sie in der Lage sind einfache Fehler relativ verlässlich aufzudecken. Besonders bei wiederkehrenden trivialen Prüfschritten können Source Code Scanner die Effizienz einer Überprüfung steigern. Zudem können sie zur Kontrolle auf Einhaltung von Programmierrichtlinien eingesetzt werden. Durch die hohe Automatisierung und die Möglichkeit der Integration in den Entwicklungsprozess können Metriken erstellt werden, anhand derer auf die Qualität und Komplexität des Quellcodes geschlossen werden kann. Wird weiterhin der Aspekt betrachtet, dass die Technologie noch recht neu ist, so ist zu erwarten, dass die Ergebnisse in Zukunft besser werden. Sofern Source Code Scanner nicht als alleinige Überprüfungsmaßnahme zur Feststellung der Sicherheit eingesetzt werden, ist eine Verwendung empfehlenswert. Es sollte der Scanner ausgewählt werden, der am besten zu der zu überprüfenden Software passt. Dies sind üblicherweise Scanner, die bestimmte Arten von Schwachstellen mit hoher Zuverlässigkeit erkennen können und zudem ein breites Spektrum an Überprüfungen zulassen, u.a. auch durch die Definition von eigenen Regelsätzen. Ein Scanner, der alle Schwachstellen mit hoher Zuverlässigkeit erkennen kann, scheint gegenwärtig nicht zu existieren. Dementsprechend ist zu überlegen, ob die Überprüfung des Quellcodes durch eine Kombination von Scannern verbessert werden kann, indem beispielsweise ein Produkt mit einem großen und zuverlässigen Erkennungsspektrum durch weitere Scanner unterstützt wird, die nur eine bestimmte Art von Schwachstellen überprüfen. Werden die Lizenzkosten der kommerziellen Source Code Scanner betrachtet, so empfiehlt sich aus wirtschaftlichen Gründen aber auch die Nutzung von kostenfreien Lösungen für die Analyse von speziellen Schwachstellen.

Literaturverzeichnis

[1]Blum, Dan. 2011. Cloud Security & Risk – Global Standards & Trends. Gartner. [Online] 11 24, 2011. [Cited: 02 17, 2012.]

[2]Chess, Brian and Gary, McGraw. 2004. Static Analysis for Security. US CERT. [Online] 12 2004.

[3]Faulk, Stuart R. 1997. Software Requirements: A Tutorial. Department of Computer Science. [Online] 1997. [Cited: 02 17, 2012.]

[4]Feiman, Joseph and MacDonald, Neil. 2010. Magic Quadrant for Static Application Security Testing. veracode. [Online] 12 13, 2010. [Cited: 02 17, 2012.]

[5]Harrison, Brendan. 2011. All static analysis tools are not created equal. klocwork. [Online] 03 08, 2011.

[6]Juuso, Anna-Maija and Takanen, Ari. Building Secure Software using Fuzzing and Static Code Analysis. Codenomicon. [Online]

[7]Karpov, Andrey and Ryzhkov, Evgeniy. 2011. Difficulties of comparing code analyzers, or don't forget about usability . viva64. [Online] 03 31, 2011.

[8]Karpov, Andrey. 2010. Static analysis of source code by the example of WinMerge. viva64. [Online] 10 30, 2010.

[9]Michael, Christoph, Lavenhar, Steven. 2005. Source Code Analysis Tools - Business Case. US CERT. [Online] 09 28, 2005.

[10]OWASP. 2012. Static Code Analysis. OWASP. [Online] 01 06, 2012.

[11]Steven, John and McGraw, Gary. 2011. Software [In]security: Comparing Apples, Oranges, and Aardvarks (or, All Static Analysis Tools Are Not Created Equal). informIT. [Online] 01 31, 2011.

[12]Vigilant Software. The Cost of Defects. Vigilant Software. [Online]