Automatische Sourcecode-Analyse

An der Quelle

Die Analyse des Quellcodes ist seit jeher eines der wirkungsvollsten Instrumente zur Herstellung sicherer Software. Doch dieser Ansatz war bisher stets mit großem Aufwand verbunden und in aller Regel nicht umfassend durchführbar. Dies hat sich mit der Verfügbarkeit von Werkzeugen zur automatischen Sourcecode-Analyse (SCA) grundlegend verändert.

von Mirko Richter

Dass die Zahl erfolgreich durchgeführter Angriffe auf Webanwendungen dramatisch zunimmt, ist ein Fakt, der von Statistiken wie CERT [1] belegt ist. Beinahe erschreckender sind Analysen über den inoffiziellen Zustand von Internetauftritten. So analysierte das Web Application Security Consortium (WASC, [2]) im Jahr 2006 rund 30 000 zufällig ausgewählte Webanwendungen und identifizierte pro Anwendung durchschnittlich fünf voneinander unabhängige Schwachstellen. Am anfälligsten waren die Testkandidaten dabei für Cross-Site Scripting (85 Prozent), SQL Injection (27 Prozent), Information Leakage (16 Prozent) und HTTP Response Splitting (10 Prozent). Für die Abwehr dieser und vieler weiterer Angriffsarten stellt die automatische Sourcecode-Analyse (SCA) ein mächtiges Werkzeug dar.

SCA arbeitet direkt auf dem Quellcode (Whitebox) und geschieht ohne eine Ausführung des betroffenen Programms. Dabei konzentriert sich die Quellcodeanalyse auf die Bewertung der Struktur, der Qualität und/oder der Sicherheit.

Dieser Artikel stellt Möglichkeiten für ein Sicherheits-Review mit speziellen Werkzeugen vor. Bei dieser Art der Analyse wird der Code der Anwendung mithilfe von Tools wie beispielsweise Lapse [3] analysiert und für die Weiterverarbeitung durch einen menschlichen Auditor aufbereitet. Das Tool meldet dabei eine Reihe von Findings, also mögliche Schwachstellen, die dann nach ihrer Ausnutzbarkeit bewertet und nach Bedrohungspotential eingestuft werden müssen.

Wie funktioniert SCA?

Im einfachsten Fall werden mithilfe von regulären Ausdrücken Stellen im Code identifiziert, die ein potenzielles Sicherheitsrisiko bedeuten. Dies sind zum Beispiel Aufrufe von gefährlichen

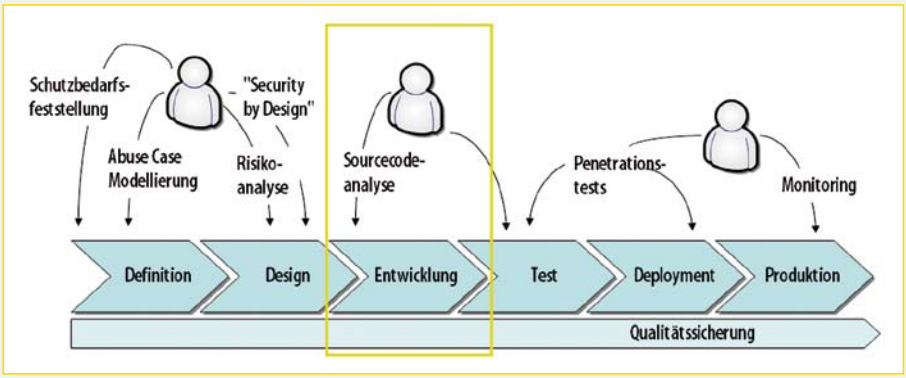

Abb. 1: Optimale Einbettung von SCA in den Software-Entwicklungsprozess (aus [6])

Operationen (wie *strncpy* in C, die einen Buffer-Overflow-Angriff ermöglichen) oder Eintritts- und Austrittspunkte von Nutzereingaben. Ein möglicher Eintrittspunkt könnte der Aufruf von *HttpServletRequest.getParameter(String)* oder *get*()* in einem Struts-Formular [4] sein. Ein denkbarer Austrittspunkt wäre der Aufruf von *Connection.executeSQL(String)*, durch den eine direkte SQL-Anfrage an die Datenbank gesendet wird.

Zusätzlich verstehen die leistungsstärkeren Werkzeuge wie Fortify SCA [5] sicherheitsrelevante Zusammenhänge bei der Verwendung verbreiteter APIs und können so beispielsweise erkennen, dass von bestimmten API-Klassen abgeleitete Anwendungsklassen immer zustandslos implementiert werden müssen, wie zum Beispiel Struts Actions. Erweiterungen für hausinterne APIs oder zur Durchsetzung nahezu beliebiger Coding Guidelines können in der Regel als nutzerspezifische Erweiterungen – so genannte Custom Rules – in das Tool eingebracht werden.

Der durch diese Analyse als relevant eingestufte Sourcecode wird anschließend in unterschiedliche Arten von Graphen umgewandelt. Dies sind im Allgemeinen Aufrufgraphen (wie rufen sich Klassen, Methode, Seiten et cetera gegenseitig auf), Kontrollflussgraphen (welche möglichen Ablaufpfade existieren bei der Ausführung), Kontrollabhängigkeitsgraphen (wovon hängt der Aufruf einer Codezeile ab) und Datenflussgraphen (wie werden Daten durch das System gereicht).

Mithilfe des Kontrollfluss- und Kontrollabhängigkeitsgraphen kann beispielsweise festgestellt werden, welche Codezeile bei jedem Durchlauf, manchmal oder nie durchlaufen wird. Diese Art der Analyse ermöglicht es zu prüfen, ob alle geöffneten Ressourcen auch immer wieder geschlossen werden. Listing 1 zeigt die oft anzutreffende falsche Behandlung: Wird bei der Benutzung der geöffneten Ressource eine Ausnahme geworfen, wird sie nicht wieder freigegeben. Die richtige Behandlung wäre hier eine Freigabe im *finally*-Block.

> Problem der manuellen Analyse: Wer kontrolliert den Auditor?

Die Analyse des Datenflussgraphen ermöglicht es, einen Zusammenhang zwischen Nutzereingaben und Austrittspunkten als Bestandteil potenziell gefährlicher Aufrufe zu erkennen. Alle zu Beginn erwähnten Angriffsarten können hierdurch erkannt werden.

Der Aufrufgraph wird letztlich dazu verwendet, ein größeres Verständnis für die Zusammenhänge im System herzustellen.

Vorteile der automatischen Analyse

Der größte Vorteil der SCA liegt in der Unterstützung bei der konsistenten Betrachtung des gesamten Codes. Niemand muss – potenziell falsch – entscheiden, welcher Codeteil einem Sicherheits-Review unterzogen werden sollte und welcher nicht. Zudem kann bei einem komplett manuellen Review (beispielsweise auch bei einem Black-Box-Test) unabhängig von der Expertise des Sicherheitsexperten, keine Aussage über die Zuverlässigkeit des Ergebnisses gemacht werden: Wie viele Fehler existieren? Wie viele wurden gefunden? Würde der Auditor diese dennoch ableiten, wäre die Folge ein trügerisches Sicherheitsgefühl, das auf Grund seines Stichprobencharakters auf nichts anderem als Teilwissen beruht. Ein Angreifer mit mehr Geld, Zeit und Erfahrung kann hier möglicherweise mehr Glück haben und einen Schwachpunkt finden.

Grenzfälle und schwer erreichbare Zustände innerhalb der Anwendung können durch SCA konsistent analysiert werden. Schwachstellen, die aus solchen Spezialfällen resultieren würden, können somit auch erkannt werden.

Der White-Box-Charakter der Analyse ermöglicht es, direkt auf den Fehler und seinen genauen Hergang (Datenfluss et cetera) hinzuweisen und nicht, wie bei einem Black-Box-Test, nur die Symptome aufzuzeigen. Eine Korrektur wird dadurch deutlich vereinfacht. Weiterhin kann SCA jederzeit mit geringem Aufwand durchgeführt werden. Dadurch können Probleme bereits während der Entwicklung erkannt und somit Kosten gespart werden. Der Lerneffekt für die Programmierer ist zudem enorm, da sie zeitnah auf Fehler und ihren genauen Hergang hingewiesen werden und ihnen der Effekt einer Korrektur sofort am Tool veranschaulicht wird. Für Entwickler, die ein solches Tool nicht selbst einsetzen können, besteht in der Regel die Möglichkeit, durch den Auditor gefundene und als gefährlich eingestufte Fehler mit ausführlichen Beschreibungen

Listing 1

```
try{
  resource.open();
  <... do something with resource ...>
  resource.close();
}catch(ResourceException e){
  log.error(e.getMesage());
}
```

vollautomatisch in ein eigenes Bug-Tracking-System wie Trac oder Bugzilla zu exportieren.

Haben Sie eine SCA erst einmal in den Softwareprozess integriert (Abbildung 1 verdeutlicht eine optimale Integration), kommen Sie zusätzlich in den Genuss eines Features, bei dem neue Angriffe und Schwachstellen vom Hersteller als Regel formuliert in die Auswertung des Tools integriert werden. Ein Nachprüfen alter, möglicherweise sehr großer Codebasen auf Anfälligkeit gegen diese neuen Probleme ist somit mit geringem Aufwand möglich. Kommerzielle Tools bieten zudem eine Trendanalyse, über die Sie während der Projektlaufzeit verfolgen, wie sich wichtige Sicherheitskennzahlen wie die Arten von Schwachstellen oder ihre Anzahl entwickeln.

Die Kehrseite der Medaille

Die SCA unterliegt natürlich auch Beschränkungen, die in erster Linie in der Komplexitätstheorie zu suchen sind. Das so genannte Halteproblem – eine Prüfung würde inakzeptabel lange laufen – verhindert an einigen Stellen, zum Beispiel bei Schleifen mit dynamischer Lauflänge, ein Prüfen aller möglichen Zustände. Das Resultat der damit also beschränkten Analyse ist zwar nicht perfekt, aber dennoch wesentlich besser als gar keins oder das Ergebnis einer teilweise manuellen Analyse.

Weiterhin können logische Fehler in der Anwendung nicht erkannt werden, da die bloße Betrachtung des Quellcodes kein Wissen über den Kontext der Anwendung vermitteln kann. Ein Beispiel hierfür wäre eine Bankanwendung, mit der Geld von einem Konto auf ein anderes übertragen wird. Wird hier als Betrag zum Beispiel -1000 Euro angegeben, wäre das gewünschte Verhalten, dass die Anwendung dies erkennt und eben nicht beim Zielkonto 1000 Euro abzieht und auf dem Quellkonto gutschreibt. Diese Prüfung kann keine Maschine out-of-the-box, sondern nur ein Mensch mit spezifischen Tests, Custom Rules et cetera leisten.

Ein weiteres Problem liegt in der Natur der SCA, lieber zu viele False Positives, also gemeldete Fehler, die keine sind, als auch nur einen False Negative, einen nicht gemeldeten Fehler, zu verursachen. Da, wie bereits erwähnt, die reine Komplexität kein perfektes Ergebnis zulassen kann, müssen sich die Werkzeuge zwangsläufig für einen Mittelweg aus Laufzeit, Präzision, Tiefe und Skalierbarkeit entscheiden. Die Folge sind einige False Positives – die Rate ist bei leistungsstarken Tools sehr gering –, die allerdings durch den Auditor als ungefährlich eingestuft und für nachfolgende Analysen unterdrückt werden können.

Fazit

Sourcecode-Analyse alleine kann die Sicherheit einer Anwendung nicht garantieren, jedoch stellt sie einen wichtigen Baustein im Prozess für Qualitätssicherung dar. Sie vermeidet, dass sich bekannte sicherheitskritische Programmierfehler immer und immer wieder in den Anwendungscode einschleichen und trägt darüber hinaus zur Sensibilisierung der Entwickler bei.

Die ständige Erweiterung der zu Grunde liegenden Regeln – eigene oder durch den Hersteller bereitgestellte – lässt ein SCA-Werkzeug zusätzlich zu einem wachsenden Nachschlagewerk und Know-How-Fundus im Programmieralltag werden. Der Einfluss bekannter Angriffsarten und speziell zu betrachtender, potenzieller Schwachstellen in APIs oder Systembibliotheken auf korrektes Programmierverhalten wird somit selbst bei sehr großen Anwendungen mit relativ geringem Aufwand beherrschbar.

Mirko Richter *ist Mitarbeiter der Securenet GmbH in München und verfügt über mehr als zehn Jahre Erfahrung in der Softwareentwicklung und -architektur. Seit fünf Jahren beschäftigt er sich intensiv mit der Sicherheit von (Web-)Anwendungen. Seine Schwerpunkte liegen in den Bereichen IT-Sicherheit, Java und MDSD/MDA.*

Links & Literatur

[1] www.cert.org

[2] www.webappsec.org

[3] suif.stanford.edu/~livshits/work/lapse

[4] struts.apache.org

[5] www.fortify.com

[6] www.securenet.de/download/Best_Practices_fuer_sichere_Webanwendungen_iX0703.pdf