Toolbox für den Security Development Lifecycle

In der <kes> 2012#4 berichteten wir über den „Security Development Lifecycle“ (SDL) von Microsoft, einen frei verfügbaren Prozess zur Verbesserung der Softwaresicherheit und des Datenschutzes. Unsere Autoren geben nun eine Übersicht über Tools zur Identifizierung von Softwareschwachstellen, die in den jeweiligen Phasen des SDL zum Einsatz kommen können.

Von Gürkan Aydin, Anja Wallikewitz und Hartmut Pohl, Sankt Augustin

Sicherheitslücken in Software, die Angreifer nutzen können, um in Firmennetze einzudringen, sind bis heute ein Grundproblem der IT. Auch weil es so einfach wie nie ist, solche Lücken auszunutzen, denn im Internet stehen dafür eine ganze Reihe von Tools zum entgeltfreien Download zur Verfügung. Unternehmen versuchen sich davor durch bekannte Maßnahmen wie Virenschutz, Intrusion-Detection-Systeme (IDS) oder Firewalls zu schützen. Solche Maßnahmen sind zwar richtig, aber sie lindern lediglich die Symptome – und das ist eine völlig unzureichende Strategie. Um das Übel an der Wurzel zu packen, muss man bereits bei der Softwareentwicklung ansetzen und hier entsprechende Prozesse für mehr Sicherheit einführen.

In der <kes> 2012#4 stellte Michael Kranawetter ein dafür geeignetes Modell vor: Der „Security Development Lifecycle“ (SDL) von Microsoft besteht aus insgesamt 16 Methoden, die dem traditionellen Softwareentwicklungsprozess folgen (siehe Abbildung 1). Je nach Bedarf kann man ihn um weitere Richtlinien und Techniken ergänzen, um beispielsweise eine spezielle Softwaremethodik für eine Organisation zu erstellen. Ziel des SDL ist, die Anzahl der Verwundbarkeiten (Vulnerabilities) so gering wie möglich zu halten.

Tools für vier SDL-Methoden

In diesem Beitrag stellen wir daher für folgende vier Methoden des SDL passende Tools vor, mit denen sich Vulnerabilities in Software erfolgreich identifizieren lassen:

- SDL-Methode 7: Bedrohungsmodellierung (Phase 2: Entwurf)
- SDL-Methode 10: Statische Analyse des Quellcodes (Phase 3: Implementierung)
- SDL-Methode 12: Fuzzingtests (Phase 4: Überprüfung)
- optionale Maßnahmen: Penetrationstest (Phase 4: Überprüfung)

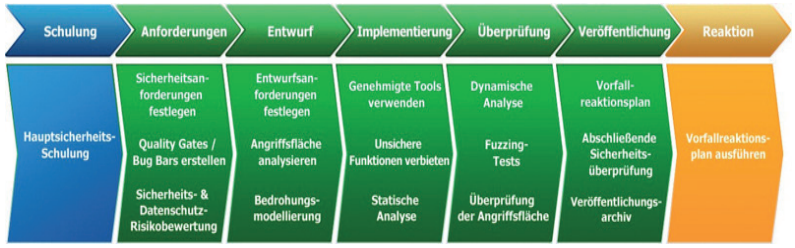

Abbildung 1: Der Security-Development-Lifecycle-(SDL)-Prozess mit seinen Methoden (Bild: Microsoft)

Bedrohungsmodellierung

Etwa die Hälfte aller Softwarefehler geht auf Designfehler zurück, daher ist die in der Designphase erstellte Sicherheitsarchitektur sorgfältig zu prüfen. Die Bedrohungsmodellierung (engl.: Threat-Modeling) unterstützt dabei als heuristische Methode die Sicherheitsüberprüfung eines Systementwurfs oder einer ganzen Sicherheitsarchitektur. Folgende Schritte sind dabei abzuarbeiten: Identifizierung der abzusichernden Assets, Generierung eines Überblicks der Sicherheitsarchitektur, Zerlegung des Sicherheitsdesigns der Zielsoftware und Identifizierung und Bewertung der identifizierten Vulnerabilities.

Ein wesentlicher Teil des Threat-Modeling ist die Analyse der Datenflüsse und ihre Prüfung auf mögliche Sicherheitslücken. Die dazu nutzbaren Tools sind in Tabelle 1 aufgeführt.

Die Nutzung eines dieser meist entgeltfreien Tools zur Unterstützung ist unverzichtbar. Unter den Aspekten Benutzerfreundlichkeit und Funktionsumfang wird häufig das kostenfreie „SDL Threat

Modeling Tool“ von Microsoft eingesetzt.

Statische Codeanalyse

Unter einer statischen Codeanalyse (engl. Static Source Code Analysis) versteht man die Analyse des Quellcodes, ohne dabei das Programm auszuführen. Dafür existieren Tools mit unterschiedlichen Wirkungsgraden, die sich in drei Klassen einteilen lassen:

_____ *Style-Checking-Tools* verifizieren den Quellcode bezüglich der Einhaltung syntaktischer Programmierrichtlinien. Diese einfachen Tools finden meist keine sicherheitsrelevanten Softwarefehler.

_____ *Semantic-Analysis-Tools* fügen zusätzliche semantische Informationen zum Syntaxbaum des Compilers hinzu. Diese werden mithilfe von Regeln auf statisch feststellbare Fehler verifiziert. Typische gefundene Fehler sind Datentypprobleme, nicht initialisierte Variablen und ungenutzte Methoden.

_____ *Deep-Flow-Static-Code-Analysis* ist die wirksamste Toolklasse: Die semantische Analyse wird um eine „Control-Flow-Graph“-Generierung und eine „Data-Flow“-Analyse ergänzt. Somit ist es möglich, komplexe Fehler, die etwa auf Race Conditions, Deadlocks oder falscher Pointerverwendung basieren, zu identifizieren.

Der Security-Analyst muss die festgestellten Anomalien allerdings aufwändig hinsichtlich False Positives verifizieren. False Negatives kann er später noch mit der Methode Dynamic-Analysis (Fuzzing) identifizieren.

Static-Source-Code-Analysis-Tools werden nach der Verwendung für eine bestimmte Programmiersprache unterschieden, eine Übersicht findet sich in Tabelle 2.

Tool	Lizenz	Link
SDL Threat Modeling Tool	Freeware	microsoft.com/en-us/download/details.aspx?id=2955
SeaMonster	GNU LGPL v3	sourceforge.net/projects/seamonster
Threat Analysis and Modeling	Microsoft Limited Permissive License (Ms-LPL)	archive.msdn.microsoft.com/tam
Trike	MIT License	octotrike.org
Coras	GNU LGPL	coras.sourceforge.net/downloads.html

Tabelle 1: Auswahl Threat-Modeling-Tools

Fuzzing

Fuzzing ist im SDL in der Überprüfungsphase (Phase 4) vorgesehen. Dabei testet man die Robustheit der Software mit zielgerichteten – jedoch unvorhergesehenen – Eingabedaten. Beispielsweise können in diese Eingaben Sonderzeichen eingestreut, bestimmte Zeichen mehrfach wiederholt oder auch überlange Zeichenketten generiert werden, um Buffer Overflows herbeizuführen. Die Software testen die Analysten entweder in einer virtuellen Maschine oder auf einem speziellen System. Der Quellcode wird beim Fuzzing nicht benötigt.

Die erste Generation von Fuzzing-Werkzeugen nutzte häufig das vorgeschlagene Brute-Force-Verfahren, also alle Eingabeschnittstellen mit allen technisch möglichen Eingaben (Bit-Kombinationen) zu bombardieren. Das führt jedoch nur selten zum Erfolg und benötigt einen überaus hohen Rechenzeitaufwand. Bei hinreichender Breite der Eingabeschnittstelle ist das Brute-Force-Verfahren auch außerdem nicht vollständig anwendbar.

Daher setzen Tester heute Tools ein, denen sie entweder Rahmendaten wie zum Beispiel ein Protokollaufbau vorgeben können oder solche, die aus „regulären“ Eingabedaten lernen, die relevanten – und damit die zur Identifizierung von Vulnerabilities führenden – Eingabedaten auszuwählen und

Tool	Kategorie	Lizenz	Link
PMD	Java	BSD License	pmd.sourceforge.net/
FindBugs	Java	GNU LGPL v3	findbugs.sourceforge.net/downloads.html
OWASP LAPSE	Java	GNU GPL v3	evalues.es/index.php/es/owasp-lapse-project.html
Google Code Pro Analytix	Java	propritär – free	https://developers.google.com/java-dev-tools/codepro/doc
cppcheck	C/C++	GNU GPL v2	sourceforge.net/projects/cppcheck
splint	C/C++	GNU GPL v3	splint.org
Frama-C	C/C++	GNU LGPL v2	frama-c.com/download.html
Yasca	Multi-Language	BSD License, GNU General Public License version 2.0 (GPLv2), GNU Library or Lesser General Public License version 2.0 (LGPLv2)	scovetta.com/yasca.html
Understand	Multi-Language	proprietär	scitools.com

Tabelle 2: Auswahl Static-Source-Code-Analysis-Tools

Tool	Kategorie	Lizenz	Link
Burp Suite Pro	Web-Application Fuzzer	proprietär	portswigger.net/burp/
Dfuz	Remote – Network-Protocol	GNU GPL v3	www.leetupload.com/dbindex2/index.php?dir=Linux/&file=dfuz.tar.gz
Defensics	Local – File-Format, Remote – Network-Protocol	proprietär	www.codenomicon.com
JavaFuzz	Local – API	GNU GPL v2	code.google.com/p/javafuzz/
Peach	Remote – Network-Protocol, Local – File-Format, Command-Line, Web-Browser, Database	MIT License	peachfuzzer.com
Scapy	Remote – Network-Protocol	–	www.secdev.org/projects/scapy/
Sulley	Remote – Network-Protocol, Datenbankanwendungen Local – File-Fuzzing	GNU GPL v2	github.com/OpenRCE/sulley
Tmin	Local – Command-Line	Apache License 2.0	code.google.com/p/tmin/
Untidy	Local – File-Format	GNU GPL	sourceforge.net/projects/untidy/
wsfuzzer	Remote – Web-Application	GNU LGPL	www.owasp.org/index.php/Category:OWASP_WSFuzzer_Project

Tabelle 3: Auswahl Fuzzing-Tools

einzusetzen. Dazu muss man die Eingabeschnittstellen identifizieren, an die solche Daten gesendet werden. Das können Netzwerkprotokollaufrufe (HTTP, SMTP, SIP, LDAP etc.), Remote Procedure Calls (RPC), Web-Services, File-Formats, Command-Line-Parameters sein. Im Anschluss analysiert der Security-Experte die bereitgestellten Daten, aus denen die Reproduzierbarkeit (Reproducibility), Ausnutzbarkeit (Exploitability) – auch aus dem Internet – und die Schwere von Vulnerabilities (Severity) bestimmt werden.

Im Internet gibt es jede Menge entgeltfreie und kostenpflichtige Werkzeuge, mit denen man solche Tests durchführen kann - eine Auswahl zeigt Tabelle 3. Dabei hängt die Eignung jedes Tools auch von der Zielsoftware und den individuellen Bedürfnissen ab – den perfekten Fuzzer gibt es (noch) nicht. Auf Grund des jeweils unterschiedlichen Funktionsumfangs werden meist mehr als fünf Tools aus einer Gesamtmenge von weltweit circa 300 Tools in Abhängigkeit von der Zielsoftware ausgewählt nach der Regel: Jedes Tool identifiziert (grundsätzlich) andere Sicherheitslücken.

	Tool	Lizenz	Link
1.	Acunetix Web Vulnerability Scanner	Proprietär	acunetix.com
2.	Burb	Proprietär	portswigger.net
3.	IBM Rational AppScan	Proprietär	ibm.com
4.	Metasploit	BSD	metasploit.com
5.	Nessus	Proprietär	Nessus.org
6.	Nmap	GPL	nmap.org
7.	OpenVAS	GPL	openvas.org
8.	OWASP-ZAP	Apache License 2.0	owasp.org
9.	w3af	GPL v2	w3af.org
10.	Wireshark	GPL	Wireshark.org

Tabelle 4: Auswahl Penetration-Testing-Tools

Penetrationstests

Der SDL definiert Penetrationstests als optionale Maßnahme – solche Maßnahmen sollen besonders bei Software zum Einsatz kommen, die später in sicherheitskritischen Umgebungen verwendet wird.

Grundsätzlich ist es unabdingbar, Softwaresysteme in kurzen Abständen anhand simulierter Angriffe auf die bekannten und besonders auf unbekannte Schwachstellen hin zu untersuchen. Ziel ist hier eine technische Analyse des Sicherheitsniveaus durch eine vollständige Untersuchung der Softwaresysteme. Dazu werden potenzielle Angriffe von Innen- und Außentätern (Intranet, Internet) simuliert sowie die organisatorischen Folgen erfolgreicher Angriffe identifiziert.

Im Vorfeld des Penetrationstests definiert der Analyst die zu untersuchenden Zielobjekte. Die Prüfung kann ein Blackbox-Test sein oder eine Whitebox-Analyse. Letztere ist kostengünstiger und wirksamer, da hier der Tester die zu überprüfenden Details der Zielumgebung wie IP-Adressen, Netzwerkpläne und interne Datenflüsse kennt. Alles Informationen, die auch Innentäter in Erfahrung bringen könnten. So können Experten auf eine langwierige Datensammlung wie bei der Blackbox-Analyse verzichten und stattdessen ihr Wissen auf die kritischen Gebiete fokussieren.

Pentests basieren in der Regel auf einer methodischen Vor-

gehensweise, die aus fünf Phasen besteht:

_____ Informationsbeschaffung: systematisches Sammeln grundlegender Informationen zur Zielumgebung zum Beispiel IP-Adressbereiche, Hostnamen, Who-is-Abfragen.

_____ Scanning: Identifizierung von Angriffsvektoren mit speziellen Schwachstellenscannern (z. B. Nmap, Nessus und Qualys) sowie Identifizierung exponierter Dienste (z. B. Mapping, Fingerprinting). In diesem Stadium setzen Tester oft mehrere Tools ein. Darüber hinaus werden manuelle Scanverfahren durchgeführt, um die Systeme zielgenauer zu untersuchen.

_____ Auswertung: Bewertung der identifizierten Schwachstellen, die sich im Rahmen eines Angriffsszenarios (Exploits) ausnutzen lassen.

_____ Verifikation: Partielles Ausnutzen identifizierter Schwachstellen zum Nachweis ihrer Existenz und Auswirkungen.

_____ Reporting: Aufbereiten der Ergebnisse in einem umfassenden Abschlussbericht.

Eine subjektive Auswahl aus der großen Menge der Penetration-Testing-Tools ist in Tabelle 4 aufgelistet.

Fazit

Aber wie wirkungsvoll sind die genannten vier Methoden für die Entwicklung sicherer Software? Eine Zufallsstichprobe von 40 Projekten zeigte, dass tatsächlich die verschiedenen Methoden auch unterschiedliche Softwarelücken identifizieren. Durch die Bedrohungsmodellierung fand man in der Stichprobe beispielsweise 112 von insgesamt 156 Zero-Day-Vulnerabilities. Das Fuzzing förderte 27 solcher Schwachstellen zutage.

Beim Testen müssen je Methode unverzichtbar mehrere – bis über 60 verschiedene Tools (Fuzzing) eingesetzt werden. Jedes von uns aufgelistete Tool findet mitunter andere Schwachstellen. Die vier vorgestellten Methoden lassen sich auf allen Prozessoren und Betriebssystemen einsetzen – nur in wenigen Fällen ist die Entwicklung eines speziellen Tools notwendig. Gleichermaßen lassen sich auch Apps sowie Netzwerkprotokolle, Embedded Systems (auch Hardware) und proprietäre Systeme auf Schwachstellen untersuchen. ■

Gürkan Aydin ist Seniorberater bei softScheck. Anja Wallikewitz ist im gleichen Unternehmen Assistentin der Geschäftsleitung. Prof. Dr. Hartmut Pohl ist geschäftsführender Gesellschafter der softScheck GmbH.

Literatur

[1] Michael Kranawetter: Security Development Lifecycle, <kes> 2012#4, S. 56

[2] Hartmut Pohl, Fuzzelarbeit, <kes> 2011#5, S. 66

[3] Peter Sakal, Hartmut Pohl: Entwicklungshelfer und Stresstester, <kes> 2010#2, S. 63