Technische Universität Ilmenau 10. April 2024

Vorlesung

Security Engineering

Dr. Peter Amthor

NeXTeX

Vorlesungsmitschrift von
Adrian Schollmeyer

Inhaltsverzeichnis

1 Introduction

1.1 What is Security Engineering?

- Zugriffssteuerung
- Methoden zur Absicherung von Software
- Spezialisierte Methoden für Softwareentwicklung; zieht sich durch alle Schritte des Softwareentwicklungsprozesses, ist also ein *Cross-Cutting-Concern* in der Softwareentwicklung: Security Requirements, Security Model, Security Testing (z. B. SAST), SecDevOps

Konkret ist Security Engineering eine Spezialform der Softwareentwicklung, bei der besondere security-spezifische Teilmethodiken/Artefakte/Modelle benutzt werden, um Sicherheitsanforderungen umzusetzen. Ziel ist die Garantie nichtfunktionaler Sicherheitsanforderungen. Dies umfasst alles von der Technologie bis zum Projektmanagement.

Beispiel: Vertraulichkeit als Anforderung muss zu jedem Zeitpunkt berücksichtigt werden. Im Design müssen verschiedene Benutzerrollen vorgesehen werden, die mit Authentifizierung und Autorisierung umgesetzt werden müssen. Die korrekte Funktion dieser muss getestet werden.

Damit dieser Prozess beherrschbar bleibt, ist Automatisierung mit entsprechenden tools zwingend notwendig.

1.2 Why do we need Security Engineering?

Egal ob Code offen zugänglich oder versteckt ist, sollten wir davon ausgehen, dass Angreifer auf allen Phasen des Softwareentwicklungsprozesses unterwegs waren.

Angreifer laufen den Softwareentwicklungsprozess normalerweise rückwärts ab:

- Ausnutzung einer laufenden Software (z. B. infizierter E-Mail-Anhang wird geöffnet)
- Privilege Escalation: Einsatz eines Programms, um mehr Berechtigungen zu erlangen
- Implementierung von Schadcode auf der Opfer-Infrastruktur, um den Zugang dauerhaft zu erhalten.
- Entwurf von Zielen, die der Angreifer auf dem Opfer-System erreichen will

Wenn jedoch formal gezeigt werden kann, dass die Umsetzung von Sicherheitsanforderungen entlang des Softwareentwicklungsprozesses formal verifizierung lässt, kann man zeigen, dass es nicht am Sicherheitsmodell gelegen haben kann, wenn etwas schief geht. Dies ist allerdings schwierig bis unmöglich zu beweisen.

2 Model Engineering

Der Schritt der Modellierung wird jetzt aufgeweitet in mehrere Teilschritte:

- **Security Model**
- Iterativer Prozess zur Analyse und Verbesserung des Security Model
- ...
- UML

Ein zentraler Punkt des Security Engineering ist das Model Engineering.

2.1 Security Engineering Workflow

In diesem Workflow werden neben dem üblichen Modell basierend auf funktionalen Anforderungen auch eine Security Policy in die Modellierung aufgenommen. Eine **Security Policy** wird aus den Sicherheitsanforderungen erstellt und legt Regeln fest, um dieben diese Sicherheitsanforderungen umzusetzen. Die Security Policy wird nicht auf Basis natürlicher Sprache definiert, sondern mit einem formalen Sicherheitsmodell, welches sich später analysieren und idealerweise auch verifizieren lässt. So kann sichergestellt werden, dass bestimmte Vertraulichkeits- und Integritätsverletzungen nicht auftreten können.

Beispiel: beim Aufbau einer Patientendatenbank gehört es zur Security Policy, dass Rollen und Regeln festgelegt werden sollen, wer welche Daten lesen/schreiben darf (Zugriffssteuerungsregeln). Nicht zum Modell würde üblicherweise gehören, wie genau die Datenbank die Daten speichert.

2.2 Models We Know

ACF/ACM Access Control Function / Access Control Matrix. Es gibt Subjekte, Objekte und Operationen, die in Relationen zueinander stehen. Subjekte können Operationen auf Objekten ausführen. In einer ACM wird aufgetragen, welche Subjekte Operationen auf welche Objekte ausführen düfen. Formal ist eine ACM $m : S \times O \to \mathfrak{P}(Op)$, wobei S die Menge aller Subjekte, O die Menge aller Objekte und Op die Menge aller Operationen ist.

ACF/ACM ist abstrakt und extrem mächtig. Alle in der Vorlesung vorgestellten und in Betriebssystemen eingesetztn Modelle basieren auf ACF/ACM bzw. lassen sich darauf zurückführen. In der Praxis sind Spalten der ACM häufig in Metadaten (z. B. Berechtigungsinformationen in inodes des Dateisystems) hinterlegt.

HRU (Harrison, Russo, Ullman). Die ACM wird in einem Automaten verpackt, um auch Zustand bei der Berechtigungsprüfung zu berücksichtigen. Der Zustandsautomat kann vom Startzustand q_0 in andere Zustände wechseln, die ihrerseits wiederum mit einem **STS** (State Transition Scheme) in andere Zustände wechseln können. Die Zustände wiederum enthalten dann jeweils eine Menge von Subjekten, Objekten und eine dazugehörige ACM. So kann man bspw. abbilden, dass neue Nutzer angelegt werden können, wodurch die wirksame ACM um ein Subjekt erweitert wird (sprich: ein Zustandübergang). Dies ermöglicht Änderungen der Sicherheitspolitik in Reaktion auf Zustandsänderungen.

RBAC Role-base Access Control. Hier werden Benutzer durch Rollen abstrahiert, die die Berechtigungen mehrerer Nutzer auf einmal definieren.

2.3 HRU Model Safety

Eine Eigenschaft des HRU-Modells ist **Safety**. Es ist safe, wenn es von einem gegebenen Ausgangszustand unmöglich ist, durch Zustandsübergänge ein bestimmtes Recht in die ACM einzutragen, was nicht bereits im Ausgangszustand vergeben ist. Es darf sich kein Recht irgendwohin in die ACM „ausbreiten“.

Auf dem HRU-Modell ist die Frage, ob man irgendwie von einem Startzustand q_0 ein bestimmtes Recht erlangen kann, äquivalent zu der Frage, ob man vom Startzustand q_0 einen Zustand erreichen kan, in dem dieses Recht gegeben ist.

2.4 Analyzability Tradeoff

Modelle klönnen danach klassifiziert werden, wie gut sie sich analysieren lassen bzw. wie gut sich beliebige Semantiken der Security Policy modellieren lassen. Ein Beispiel für ein sehr ausdrucksstarkes Modell ist RBAC, bei dem man einfach eine Liste von Rollen mit erlaubten Operationen auf Objekten hinterlegt (leicht verständlich).

Allgemein gibt die **Ausdrucksstärke** an, wie nah das Modell an der Semantik der Security Policy ist. Bei den weniger ausdrucksstarken Modellen ist es meist deutlich einfacher, diese auf ihre Sicherheit zu analysieren. Allgemein ist es hier ein Tradeoff zwischen Ausdrucksstärke und Analysierbarkeit.

2.5 The Take-Grant Model

Ziel war es, ein möglichst domänenspezifisches Modell zu erstellen, um Sicherheitsanforderungen für einzelne Anwendungsbereiche zu erstellen. Das **Take-Grant Model** wurde speziell für die Anwendungsdomäne Betriebssystem und Access Control entworfen.

Wichtig! Generell ist Safety *nicht* entscheidbar. Take-Grant zeigt uns hingegen eine untere Grenze der Entscheidbarkeit, indem ein minimales Modell gebaut wird, dessen Safety-Eigenschaften noch in Polynomialzeit entscheidbar sind.

Im Take-Grant-Modell ist ein kompletter Graph Teil des Zustands. Zusätzlich ist Teil die Menge R, die **Rewriting Rules**, die erklären, wie sich Zustände verändern dürfen. Dies entspricht δ im HRU-Modell.

Die Frage ist jetzt, ob von einem Ausgangszustand q_1 mit Graph G_1 ein Pfad existiert, der diesen Graphen mithilfe der in R definierten Regeln ein Graph G_n erzeugt wird, der eine (sicherheitsrelevante) Eigenschaft X aufweist.

Im Take-Grant-Model werden Subjekte und Objekte zu **Entitäten** zusammengefasst.

2.5.1 Graph Rewriting Rules

take Kopieren von Rechten von einer Entität auf eine andere. Wenn also Ann take auf Bob hat und Bob read auf Doc hat, kann Ann sich read auf Doc geben (das Recht erlangen).

grant Eine Entität darf Rechte an eine andere Entität zu geben, die sie selbst hat. Wenn also Ann read auf Doc hat, darf Ann das Recht read auf Doc an Bob übertragen.

create Eine Entität darf neue Entitäten mit optionalen Rechten erstellen. Problem hierbei ist, dass das Modell nicht abdeckt, dass möglicherweise bestimmte Berechtigungen nötig sind, um entsprechende Entitäten erstellen zu können (Beispielsweise ist das Erstellen von Dateien im Betriebssystem daran geknüpft, dass Schreibberechtigungen für das Verzeichnis existieren). Dies ist jedoch per Design so, da Take-Grant nur ein minimales Modell sein soll.

call Eine Entität darf ein bestimmtes Programm ausführen. Dabei kommt ein neuer Knoten hinzu (ein Prozess wird erstellt). Dieser hat read auf das Image des Programms (er darf den eigenen Code lesen) und er darf auf Argumente mit den gleichen Rechten zugreifen wie die Entität, die das Programm gestartet hat. Diese Semantik ist bereits sehr spezifisch auf Betriebssysteme ausgelegt.

Beispiel: ein erstellter (Kind-)Prozess darf auf eine gemeinsame Systemressource zugreifen.

remove Eine Entität darf Kanten löschen (Umkehroperation zu grant).

2.5.2 Achievements

Es enthält Operation mit mehr als einer Vorbedingung und mehr als einer Nachbedingung. Dies bedeutet bereits, dass Safety im HRU-Modell nicht allgemein entscheidbar ist. Durch die Spezifität des Take-Grant-Modells ist es jedoch in linearzeit entscheidbar, ob das Modell safe ist.

In der Praxis ist Take-Grant allerdings auch nicht sehr ausdrucksstark, da man nur auf fünf Operationen beschränkt ist.

Es zeigt sich also, dass der Entwurf eines sehr spezifischen Modells zu entscheidbaren Safety-Eigenschaften führen kann. Nachteil ist natürlich das Problem, dass es sehr Domänenspezifisch ist. Bereits kleine Änderungen am Modell (bspw. zusätzliche Regeln)

können schnell dazu führen, dass der Zustandsraum und damit die Komplexität des Zustandsgraphen stark (bspw. exponentiell) wächst. Es ist beim Entwurf des Modells auch darauf zu achten, ob das Modell tatsächlich die nötige Analysierbarkeit erzielen lässt.

Stichwortverzeichnis

Literatur

[1] Peter Amthor. “Security Engineering”.