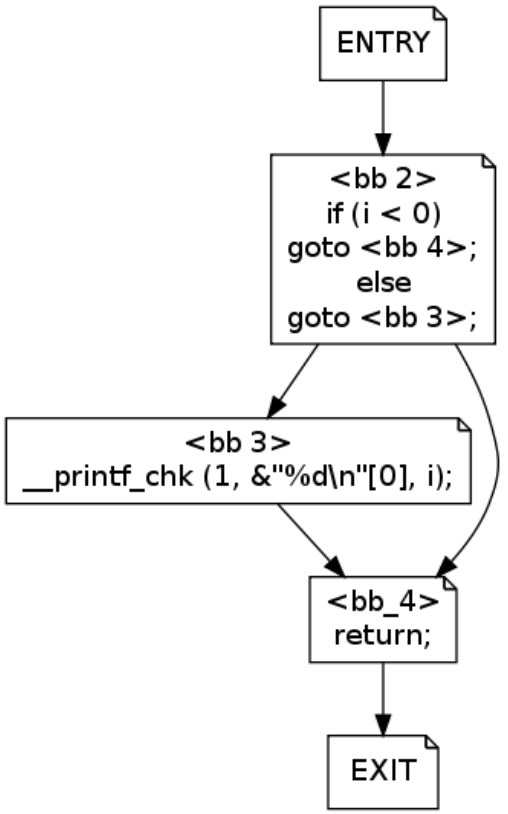
Generierung von Signaturen mittels statischer Kontrollflussanalyse

```
void rec1(int i) {
  if(i < 0) return;
  printf("%d\n",i);
}

;; Function void rec1(int)
void rec1(int) (i)
{
  # BLOCK 2
  # PRED: ENTRY
  if (i < 0)
    goto <bb 4>;
  else
    goto <bb 3>;
  # SUCC: 4 3

  # BLOCK 3
  # PRED: 2
  __printf_chk (1, &"%d\n"[0], i);
  # SUCC: 4

  # BLOCK 4
  # PRED: 2 3
  return;
  # SUCC: EXIT
}
```



Übersicht

- Motivation
- Host-basierte Einbrucherkennung
- Prozess der Signaturgenerierung
- Konstruktion des Programmmodelles
- Reduktion des Programmmodelles
- Generierung der Signatur
- Anwendungsbeispiele
- Zusammenfassung und Ausblick

Motivation

- Teilweise große Zeitabstände zwischen Entdeckung und Veröffentlichung/Behebung einer Sicherheitslücke
- laufendes Anwendungsbeispiel: fehlende Authentifizierung von netlink-Nachrichten im udev-Dienst (CVE-2009-1185)

●

31.03.2009:
Entdeckung der
Sicherheitslücke

Motivation

- Teilweise große Zeitabstände zwischen Entdeckung und Veröffentlichung/Behebung einer Sicherheitslücke
- laufendes Anwendungsbeispiel: fehlende Authentifizierung von netlink-Nachrichten im udev-Dienst (CVE-2009-1185)

31.03.2009:
Entdeckung der Sicherheitslücke

8.4.2009:
Stille Fehlerbehebung (Versionsverwaltung)

Motivation

- Teilweise große Zeitabstände zwischen Entdeckung und Veröffentlichung/Behebung einer Sicherheitslücke
- laufendes Anwendungsbeispiel: fehlende Authentifizierung von netlink-Nachrichten im udev-Dienst (CVE-2009-1185)

31.03.2009:
Entdeckung der Sicherheitslücke

8.4.2009:
Stille Fehlerbehebung (Versionsverwaltung)

15.4.2009:
Veröffentlichung + Fehlerbehebung + Angriffsprogramm

Motivation

- Teilweise große Zeitabstände zwischen Entdeckung und Veröffentlichung/Behebung einer Sicherheitslücke
- laufendes Anwendungsbeispiel: fehlende Authentifizierung von netlink-Nachrichten im udev-Dienst (CVE-2009-1185)

31.03.2009:
Entdeckung der Sicherheitslücke

8.4.2009:
Stille Fehlerbehebung (Versionsverwaltung)

15.4.2009:
Veröffentlichung + Fehlerbehebung + Angriffsprogramm

30.4.2009:
Letzte Fehlerbehebung (Mandriva-Linux)

Motivation

- Teilweise große Zeitabstände zwischen Entdeckung und Veröffentlichung/Behebung einer Sicherheitslücke
- laufendes Anwendungsbeispiel: fehlende Authentifizierung von netlink-Nachrichten im udev-Dienst (CVE-2009-1185)
- Minimierung dieses Zeitfensters → Einbrucherkennung

31.03.2009:
Entdeckung der Sicherheitslücke

8.4.2009:
Stille Fehlerbehebung (Versionsverwaltung)

15.4.2009:
Veröffentlichung + Fehlerbehebung + Angriffsprogramm

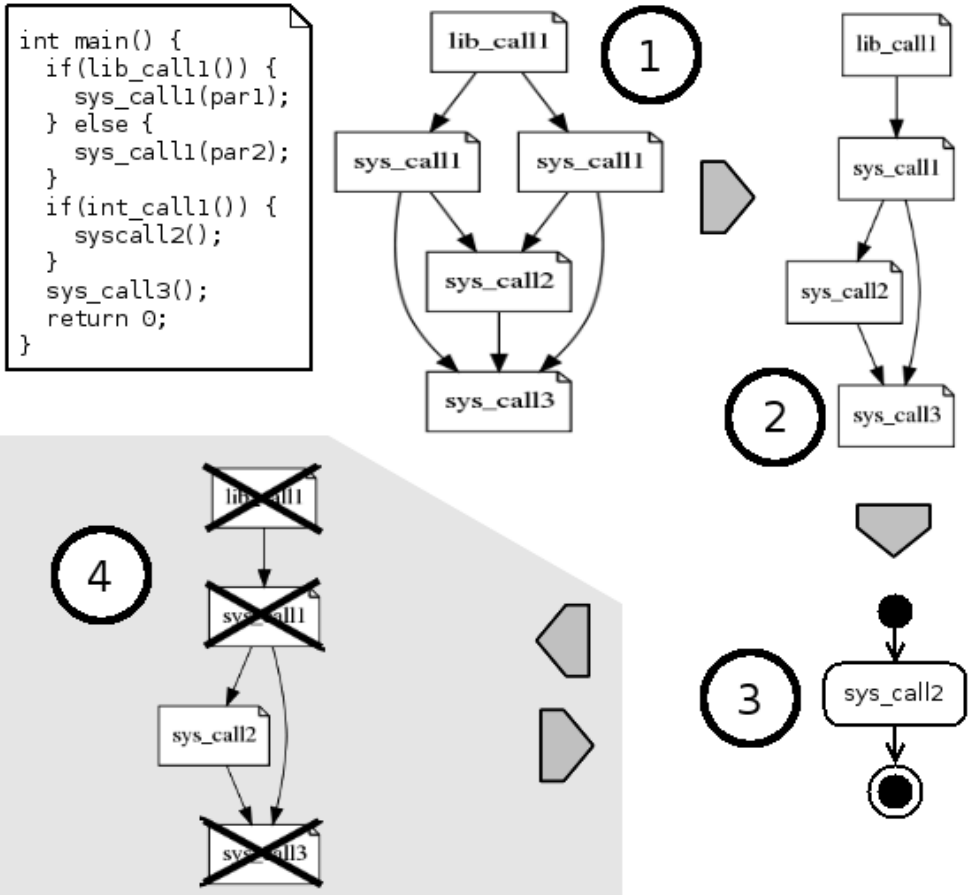
30.4.2009:
Letzte Fehlerbehebung (Mandriva-Linux)

Host-basierte Einbrucherkennung

- Missbrauchserkennung: Erkennung spezifischer Angriffe auf Basis von Signaturen
  → Signatur = Beschreibung der Angriffsspuren
- Anomalieerkennung: Vergleich des aktuellen Programmverhaltens mit einem Programmmodell
  → Annahme: Abweichung vom Modell = Einbruch
- im Bereich der Host-basierten *IDS*-Forschung hauptsächlich Anomalieerkennungssysteme betrachtet
  - teilweise obsolet durch Techniken zur Angriffsverhinderung
  - keine Erkennung von Anwendungslogikfehlern (siehe udev-Beispiel)

Prozess der Signaturgenerierung (Missbrauchserkennung)

- (1) Ermittlung des Laufzeit-verhaltens eines Programms
- (2) Überführung in Modell des *beobachtbaren* Programmverhaltens
- (3) Generierung einer Signatur zur Beschreibung des verwundbaren Programmabschnittes
- (4) Vergleich von Signatur und Programmmodell

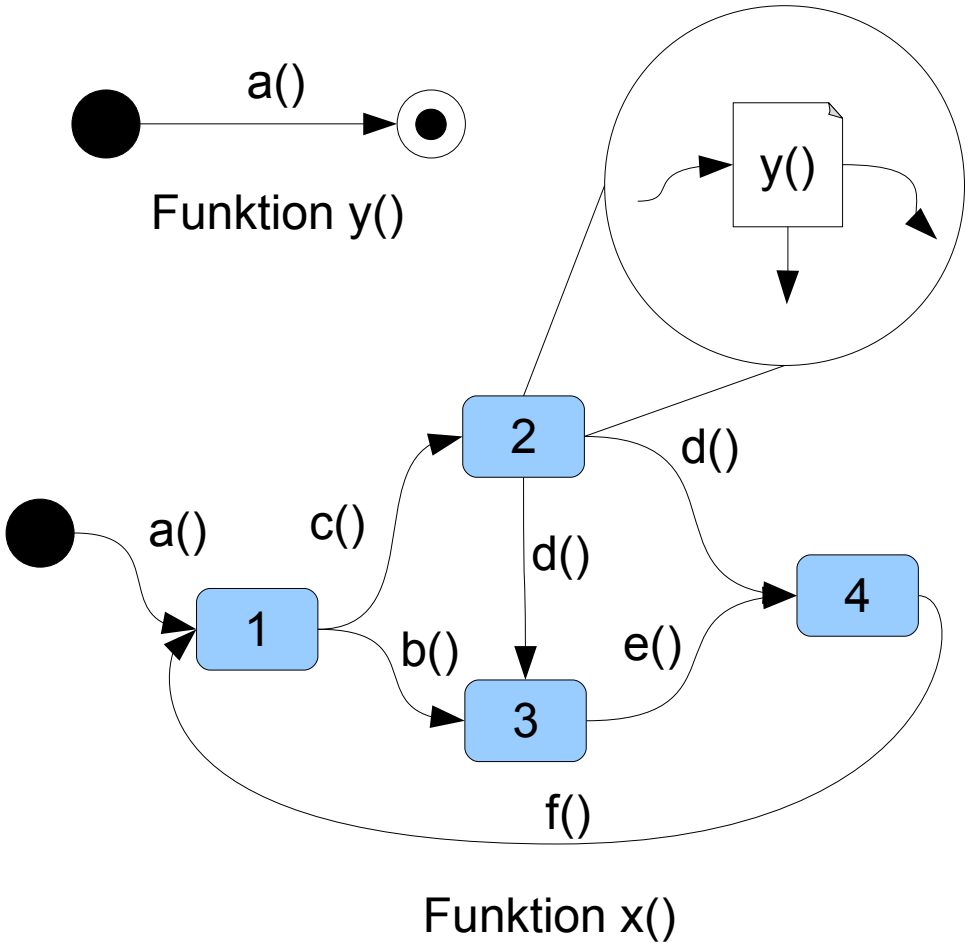


Konstruktion des Programmmodelles (Kontrollflussanalyse)

- statische Kontrollflussanalyse mit Hilfe eines Compilers
- in zwei Phasen unterteilt
  - Intraprozedurale Kontrollflussanalyse: Kontrollflüsse innerhalb von Funktionen (Schleifen, Verzweigungen) → Unterteilung der Programmanweisungen in Basisblöcke
    - keine Programmverzweigungen oder Sprünge innerhalb eines Blockes
    - Änderungen des Kontrollflusses an den Grenzen der Blöcke
    - Ergebnis: gerichteter Graph von Anweisungsblöcken einer Funktion
  - Interprozedurale Kontrollflussanalyse: Funktionsaufrufe eines Programms (direkt oder über Funktionszeiger) → Generierung von Funktionsaufrufgraphen

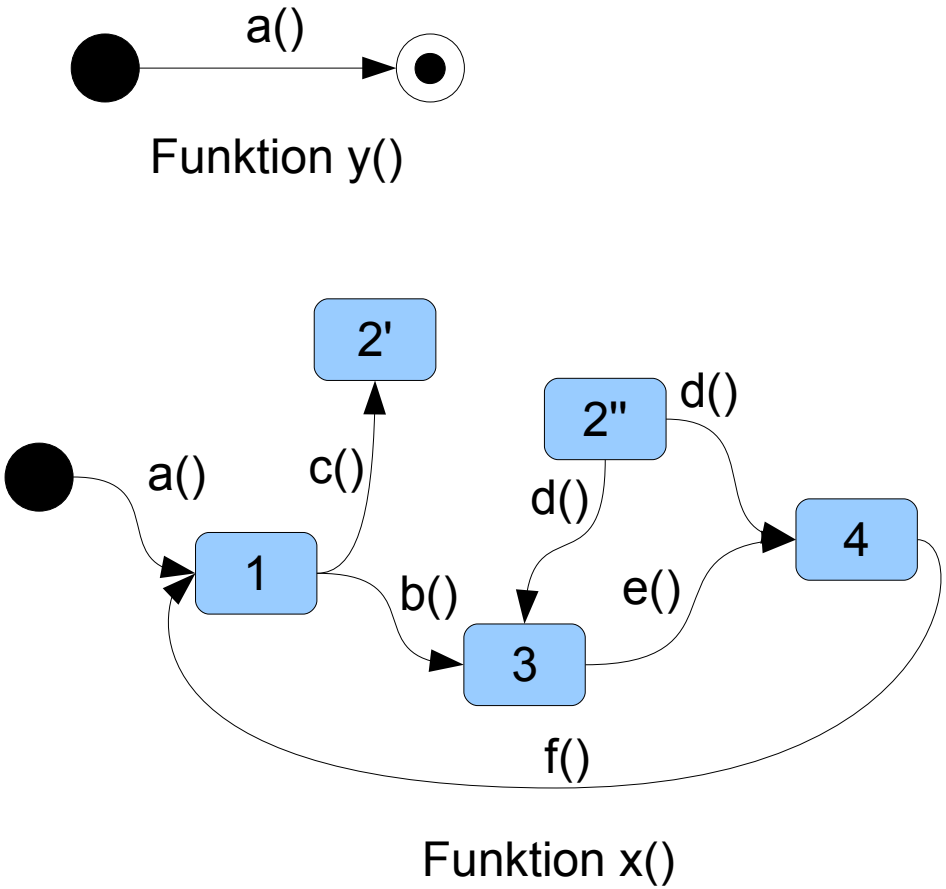
Reduktion des Programmmodelles (Integration von Funktionen)

- Einbrucherkennungssystem kann interne Funktionsaufrufe nicht erfassen
  - interne Funktionen können in ihre aufrufenden Stellen integriert werden
- Randbedingungen:
  - die aufgerufene Funktion darf keine weiteren internen Funktionen aufrufen



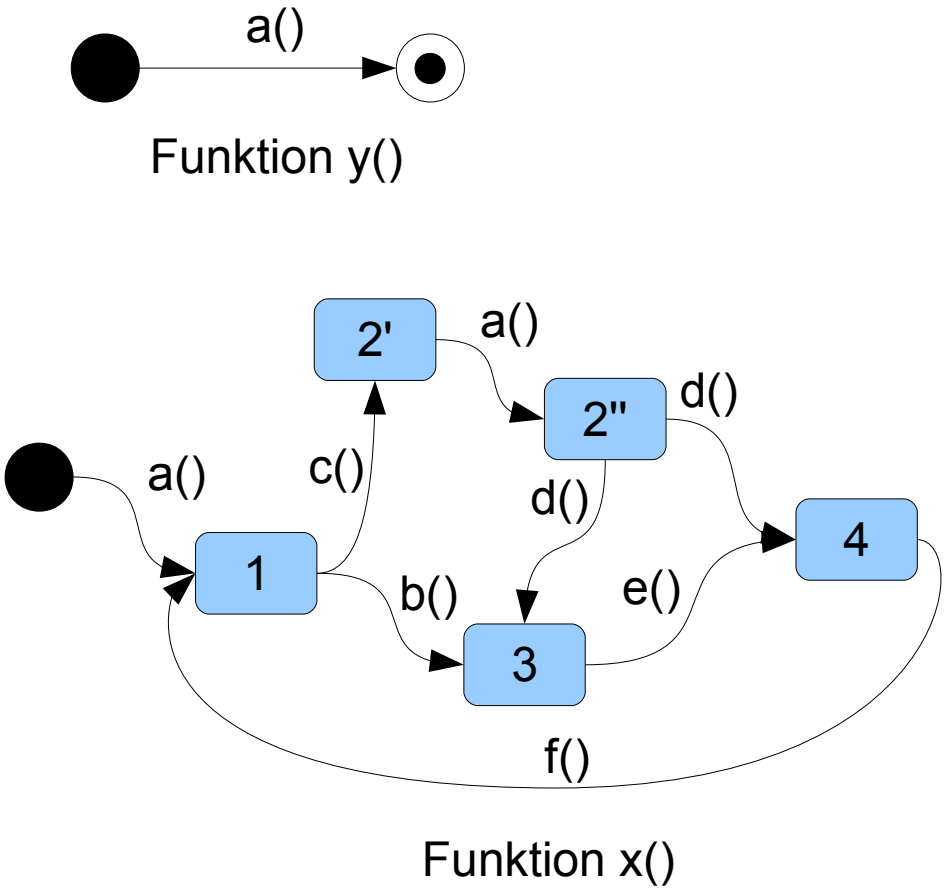
Reduktion des Programmmodelles (Integration von Funktionen)

- Einbrucherkennungssystem kann interne Funktionsaufrufe nicht erfassen
  - interne Funktionen können in ihre aufrufenden Stellen integriert werden
- Randbedingungen:
  - die aufgerufene Funktion darf keine weiteren internen Funktionen aufrufen



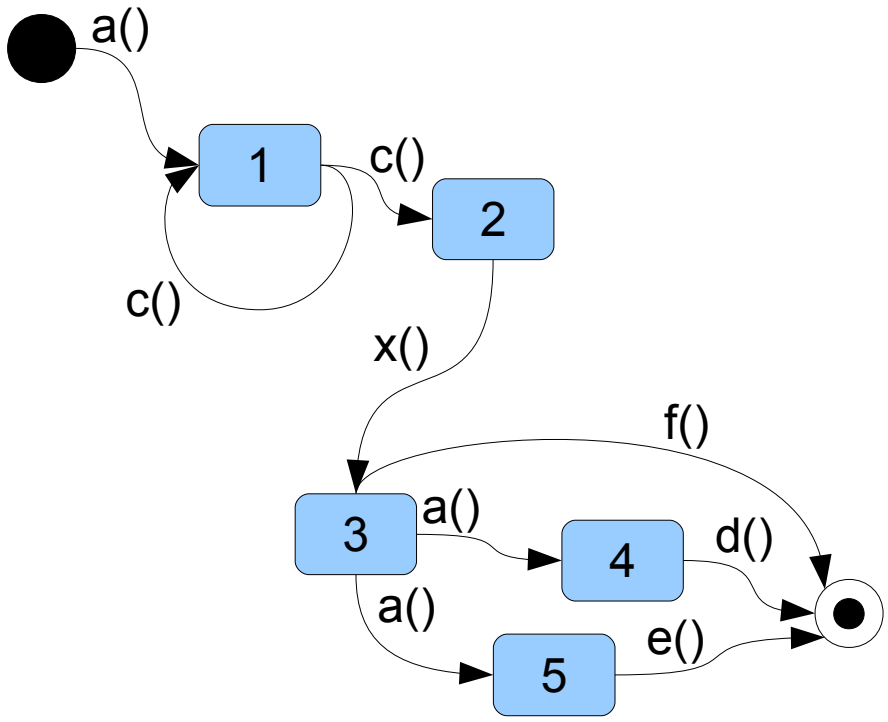
Reduktion des Programmmodelles (Integration von Funktionen)

- Einbrucherkennungssystem kann interne Funktionsaufrufe nicht erfassen
  - interne Funktionen können in ihre aufrufenden Stellen integriert werden
- Randbedingungen:
  - die aufgerufene Funktion darf keine weiteren internen Funktionen aufrufen



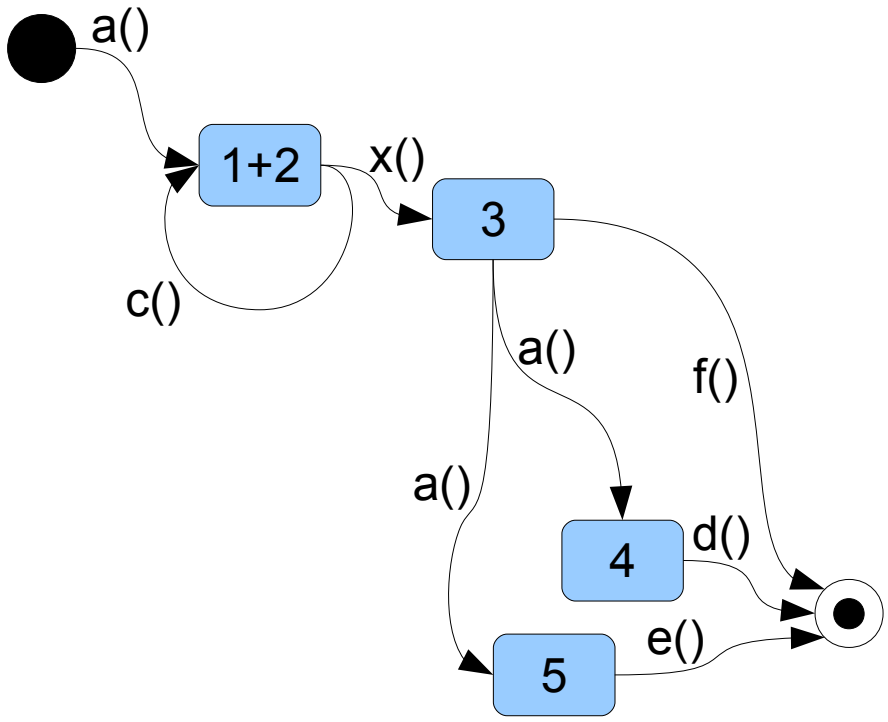
Reduktion des Programmmodelles (Reduktion von Nichtdeterminismen)

- einige Kontrollflüsse bei externer Programmobservierung nicht zu unterscheiden
  - Verzweigungen die jeweils mit identischen Funktionsaufrufen beginnen
  - Programmschleifen mit einer dem Schleifenrumpf folgenden identischen Aufrufsequenz
- nicht unterscheidbare Ereignisse können miteinander verschmolzen werden



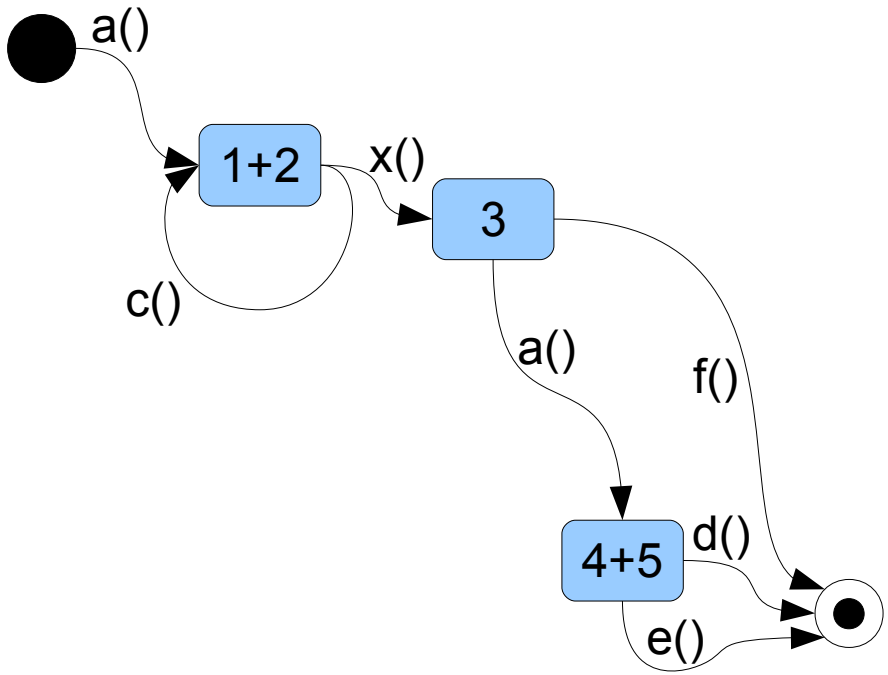
Reduktion des Programmmodelles (Reduktion von Nichtdeterminismen)

- einige Kontrollflüsse bei externer Programmobservierung nicht zu unterscheiden
  - Verzweigungen die jeweils mit identischen Funktionsaufrufen beginnen
  - Programmschleifen mit einer dem Schleifenrumpf folgenden identischen Aufrufsequenz
- nicht unterscheidbare Ereignisse können miteinander verschmolzen werden



Reduktion des Programmmodelles (Reduktion von Nichtdeterminismen)

- einige Kontrollflüsse bei externer Programmobservierung nicht zu unterscheiden
  - Verzweigungen die jeweils mit identischen Funktionsaufrufen beginnen
  - Programmschleifen mit einer dem Schleifenrumpf folgenden identischen Aufrufsequenz
- nicht unterscheidbare Ereignisse können miteinander verschmolzen werden

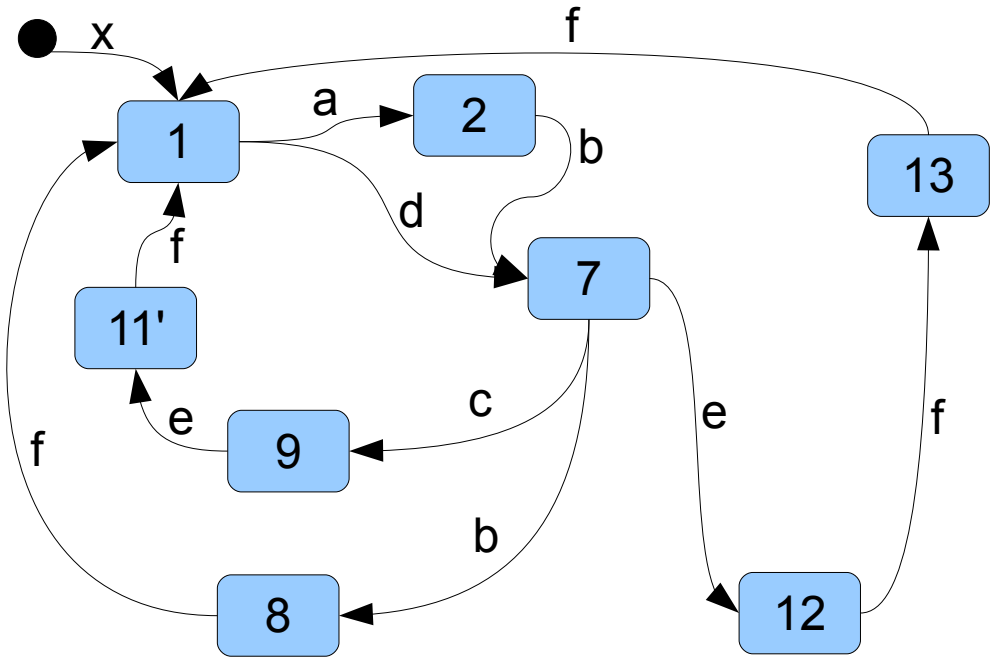


Generierung der Signatur (Grundidee)

- verwundbarer Programmabschnitt durch die Kontrollflüsse zwischen der Verwundbarkeitseinbringung und der Verwundbarkeitsausnutzung beschrieben
- Grundidee für die Generierung der Signatur:
  - Extraktion eines Ausschnittes aus dem Programmmodell, der zum Punkt der Verwundbarkeitsausnutzung führt
  - Vergleich dieses Ausschnittes mit anderen Programmab-schnitten, um Eindeutigkeit zu überprüfen
  - falls nicht eindeutig, Vergrößerung des Ausschnittes

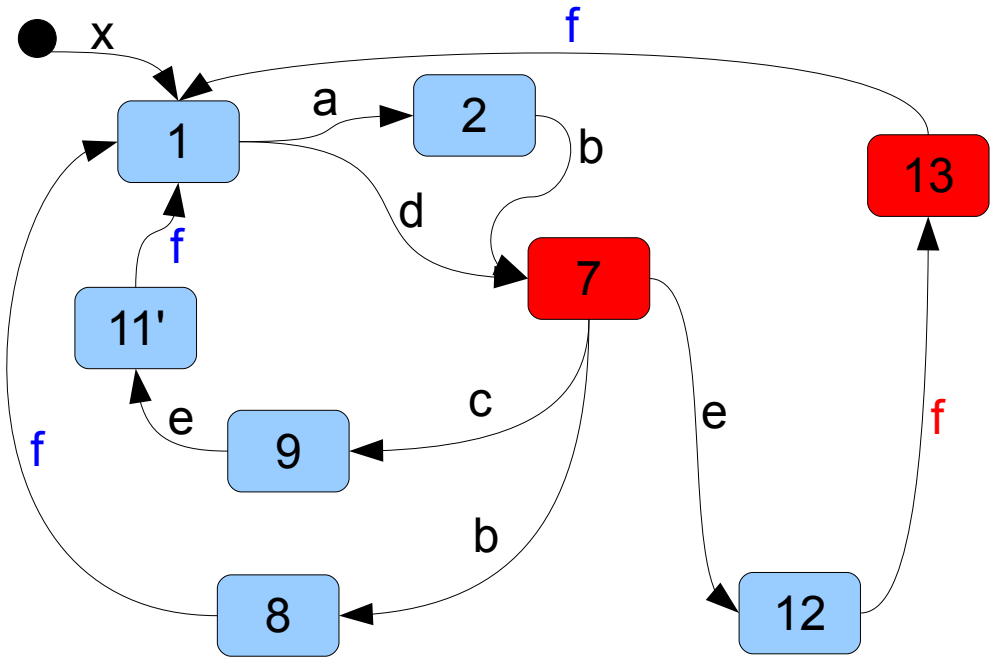
Generierung der Signatur (Vergleich von Kontrollflusspfaden)

- Generierung und Vergleich der Signatur über Vergleich von Kontrollflusspfaden
- Startknoten dieser Kontroll-flusspfade entsprechen jeweils Punkten der Ver-wundbarkeitsausnutzung
- Vergleichsmenge enthält alle zur Verwundbarkeitsaus-nutzung identischen Funk-tionsaufrufe



Generierung der Signatur (Vergleich von Kontrollflusspfaden)

- Generierung und Vergleich der Signatur über Vergleich von Kontrollflusspfaden
- Startknoten dieser Kontroll-flusspfade entsprechen jeweils Punkten der Ver-wundbarkeitsausnutzung
- Vergleichsmenge enthält alle zur Verwundbarkeitsaus-nutzung identischen Funk-tionsaufrufe

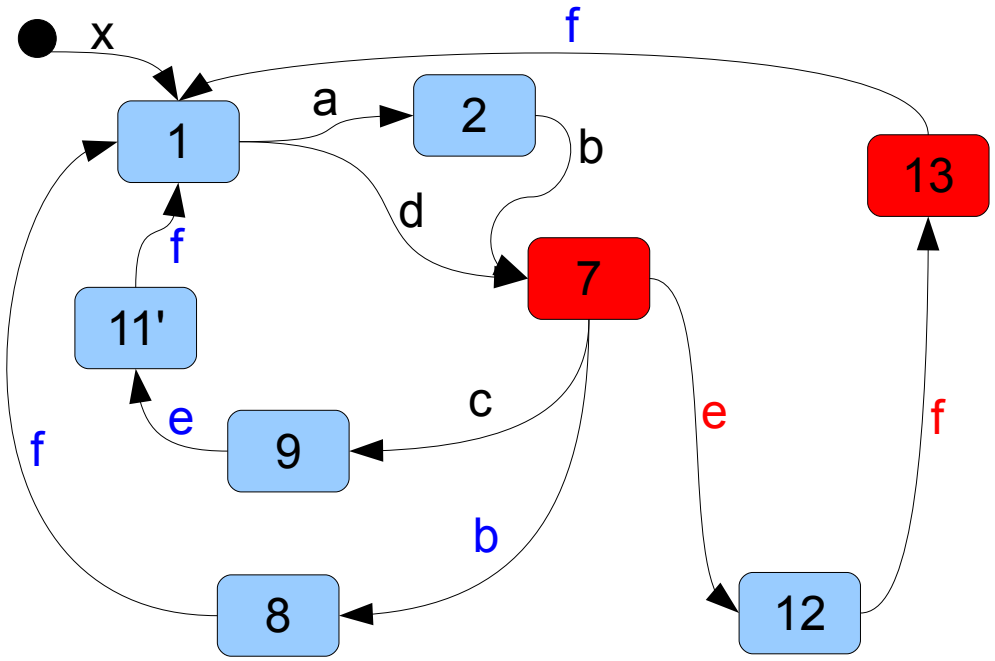


Signaturmenge: { f(12,13) }

Vergleichsmenge: { f(8,1),
f(11',1),
f(13,1) }

Generierung der Signatur (Vergleich von Kontrollflusspfaden)

- bei nicht leerer Vergleichsmenge Verlängerung der Kontrollflüsse
- im Anschluss Vergleich der Kontrollflusspfade
- zur Signaturmenge nicht identische Kontrollflusspfade werden aus der Vergleichsmenge entfernt
- Prozess wird wiederholt bis Vergleichsmenge leer

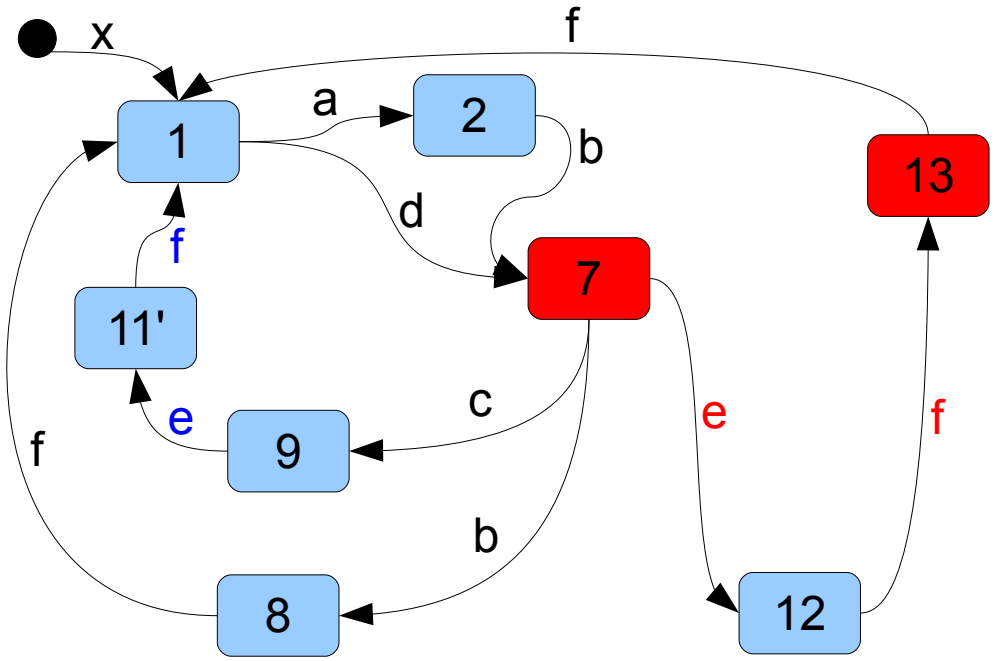


Signaturmenge: { e(7,12) → f(12,13) }

Vergleichsmenge: { b(7,8) → f(8,1),
e(9,11') → f(11',1),
f(12,13) → f(13,1) }

Generierung der Signatur (Vergleich von Kontrollflusspfaden)

- bei nicht leerer Vergleichsmenge Verlängerung der Kontrollflüsse
- im Anschluss Vergleich der Kontrollflusspfade
- zur Signaturmenge nicht identische Kontrollflusspfade werden aus der Vergleichsmenge entfernt
- Prozess wird wiederholt bis Vergleichsmenge leer

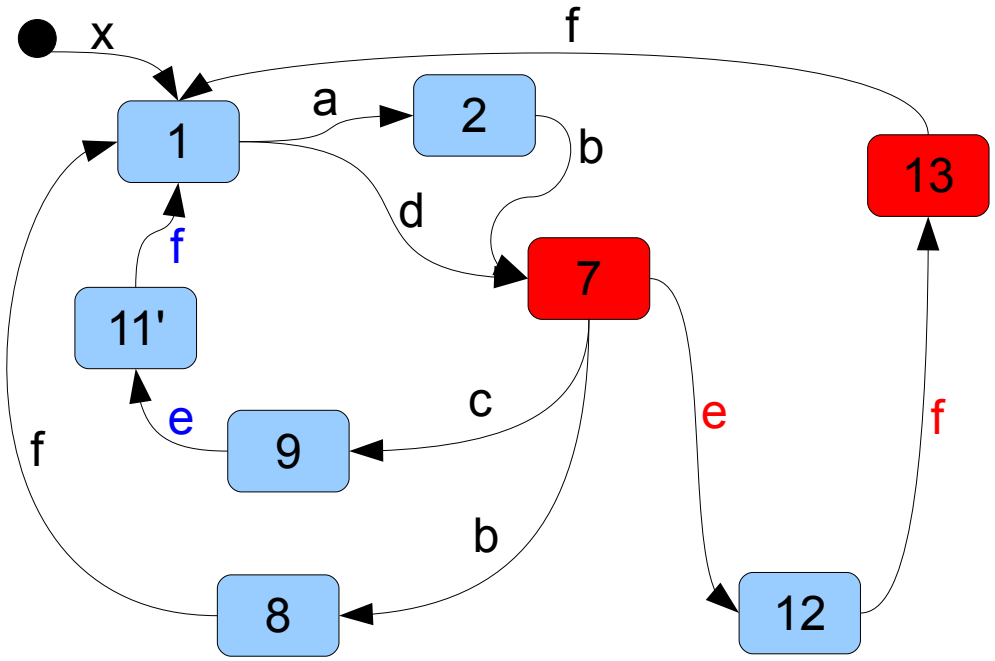


Signaturmenge: { e(7,12) → f(12,13) }

Vergleichsmenge: { b(7,8) → f(8,1),
e(9,11') → f(11',1),
f(12,13) → f(13,1) }

Generierung der Signatur (Vergleich von Kontrollflusspfaden)

- bei nicht leerer Vergleichsmenge Verlängerung der Kontrollflüsse
- im Anschluss Vergleich der Kontrollflusspfade
- zur Signaturmenge nicht identische Kontrollflusspfade werden aus der Vergleichsmenge entfernt
- Prozess wird wiederholt bis Vergleichsmenge leer

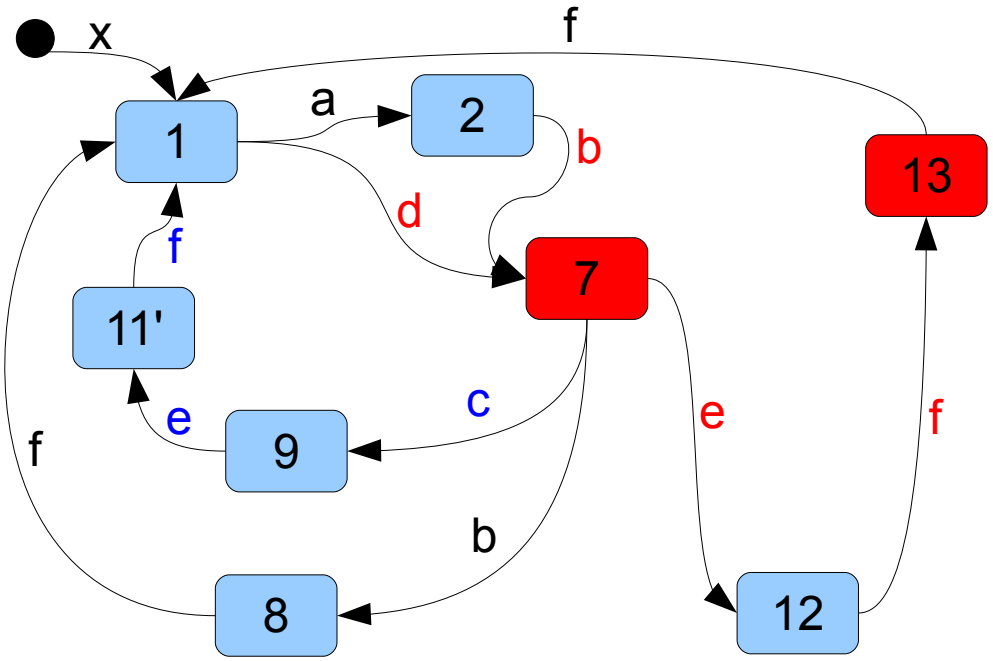


Signaturmenge: { e(7,12) → f(12,13) }

Vergleichsmenge: { e(9,11') → f(11',1) }

Generierung der Signatur (Vergleich von Kontrollflusspfaden)

- bei nicht leerer Vergleichsmenge Verlängerung der Kontrollflüsse
- im Anschluss Vergleich der Kontrollflusspfade
- zur Signaturmenge nicht identische Kontrollflusspfade werden aus der Vergleichsmenge entfernt
- Prozess wird wiederholt bis Vergleichsmenge leer



Signaturmenge: { b(2,7) → e(7,12) → f(12,13),
d(1,7) → e(7,12) → f(12,13) }

Vergleichsmenge: { }

Anwendungsbeispiele

- Sicherheitslücken von udev (CVE-2009-1185), clamav (CVE-2008-5314), ntpq (CVE-2009-0159) sowie splitvt (CVE-2008-0162) mittels Prototyp analysiert
- eines der vier Beispiele wegen zu hoher Komplexität nicht analysierbar (clamav)
- für udev und splitvt minimale Signaturen ermittelt

udev	# Funktionen	# Externe Aufrufe (red. Modell)	# System-funktionen	# Bibliotheks-funktionen
	104	22625	4548	18077

Signaturmenge: { time() }

splitvt	# Funktionen	# Externe Aufrufe (red. Modell)	# System-funktionen	# Bibliotheks-funktionen
	118	913	371	542

Signatur 1: {setuid(), utime() → getuid(), close() → getuid() }
Signatur 2: { execvp() }

ntpq	# Funktionen	# Externe Aufrufe (red. Modell)	# System-funktionen	# Bibliotheks-funktionen
	265	139	17	122

Signaturmenge: -

clamav	# Funktionen	# Externe Aufrufe (red. Modell)	# System-funktionen	# Bibliotheks-funktionen
	931	116141	9502	106639

Signaturmenge: -

Zusammenfassung und Ausblick

- Generierung von minimalen Signaturen bis zu einer bestimmten Programmkomplexität möglich
  - für die effiziente Identifizierung des verwundbaren Abschnittes geeignet → Ausgangspunkt für Überprüfung komplexerer Bedingungen zur Erkennung von Angriffen
- Für komplexere Programme weitere Untersuchungen notwendig
  - Möglicherweise Einbeziehung von Datenflussanalyse (Aufrufparameter einzelner Funktionen zur genaueren Identifizierung der Programmabschnitte)