Vereinfachung der Signaturentwicklung durch Wiederverwendung

Sebastian Schmerl[1], Ulrich Flegel[2], Michael Meier[1]

[1] Brandenburgische Technische Universität Cottbus
Lehrstuhl Rechnernetze und Kommunikationssysteme
Postfach 10 13 44, 03013 Cottbus
Sebastian.Schmerl{at}tu-cottbus.de
Michael.Meier{at}tu-cottbus.de

[2] Universität Dortmund
D-44221 Dortmund
ulrich.flegel{at}udo.edu

Abstract: Die meisten heute eingesetzten IDS realisieren eine Signaturanalyse. Hierbei werden protokollierte Ereignisse mit vordefinierten Mustern (Signaturen), die auf IT-Sicherheitsverletzungen hindeuten, verglichen. Der Entwurf einer Signatur zu einem neuen Angriff erfolgt typischerweise empirisch auf der Grundlage von Expertenwissen. Es gibt bisher kaum Methodiken für eine systematische Vorgehensweise. Ebenso existieren keine automatisierbaren Ansätze, um Entwurfs- bzw. Modellierungsentscheidungen bereits spezifizierter Signaturen wieder zu verwenden. Der Beitrag stellt einen neuen Ansatz vor, der aus einer Menge von Signaturen, diejenigen mit der größten Ähnlichkeit mit dem neuen Angriff selektiert. Dem Signaturmodellierer können so Signaturvorschläge unterbreitet werden. Anhand eines Beispiels wird der praktische Einsatz des Ansatzes beschrieben.

1 Einführung

Die zunehmende Abhängigkeit gesellschaftlicher Prozesse von IT-Infrastrukturen, sowie deren zunehmende technologische Komplexität implizieren ein Bedrohungspotential, das diese Prozesse gefährdet. Um derartigen Gefahren zu begegnen, kommt auf technologischer Ebene, neben der Ergreifung *präventiver* Sicherheitsmaßnahmen, dem Feststellen aufgetretener IT-Sicherheitsverletzungen als Grundlage *reaktiver* Mechanismen durch Intrusion-Detection-Systeme (IDS) eine besondere Bedeutung zu. In Ergänzung präventiver Sicherheitsmechanismen erlauben sie eine automatische Erkennung von IT-Sicherheitsverletzungen und ggf. auch die Einleitung von Abwehrmaßnahmen. Signaturbasierte Analyseverfahren untersuchen Protokoll- bzw. Auditdaten nach Mustern bekannter Sicherheitsverletzungen, den sog. Signaturen. Die Wirksamkeit dieser Verfahren hängt entscheidend von der Präzision der verwendeten Signaturen ab. Unpräzise Signaturen schränken die Erkennungsfähigkeit der Intrusion-Detection-Systeme stark ein und führen u. a. zu Fehlalarmen. Die Ursachen der Erkennungsunsicherheit sind nur in eingeschränktem Maß qualitativen Einschränkungen der Audit-Funktionen zuzuschreiben.

Sie sind vielmehr auf der Ebene der Signaturableitung zu suchen. Insbesondere die Ableitung von Signaturen aus Angriffsprogrammen, sog. Exploits, erweist sich häufig als Schwachpunkt. Ihre Ableitung erfolgt zumeist empirisch auf der Grundlage von Expertenwissen. Es gibt bisher kaum Methodiken für eine systematische Vorgehensweise. Ebenso existieren keine automatisierbaren Ansätze, um Entwurfs- bzw. Modellierungsentscheidungen bereits spezifizierter Signaturen wieder zu verwenden. Damit verbunden sind relativ lange Entwicklungszeiten für Signaturen, die zu ungünstigen Verwundbarkeitsintervallen führen (vgl. [8]).

Ähnlich wie in der Softwaretechnik könnte durch den Einsatz von Mechanismen zur Wiederverwendung von Entwurfs- bzw. Implementierungsentscheidungen aus bereits spezifizierten Signaturen die Entwicklungszeit gesenkt werden. Allerdings wurden bisher nur wenige Ansätze publiziert, die sich dieser Thematik annehmen. Cheung et al. schlagen vor, den Signaturentwurf durch Verwendung von "Attack Models" zu vereinfachen [1]. Dieser Ansatz entspricht den Entwurfsmustern der Softwaretechnik [2]. Die vorgestellten Modelle ermöglichen eine Wiederverwendung von architekturellen Entwurfsentscheidungen, allerdings ist eine Wiederverwendung von bereits spezifizierten Signaturen bzw. Signaturfragmenten nicht vorgesehen. Rubin et al. geben an, wie zu einem gegebenen Angriff Mutanten generiert werden können [3]. Mutanten nutzen die gleichen Schwachstellen wie die Ausgangsattacken, ohne allerdings dieselben sicherheitskritischen Aktionen auszuführen. Soll zu einem Attackenmutanten eine Signatur entwickelt werden, so könnte die Signatur des Ausgangsangriffs, falls vorhanden, zur Wiederverwendung bzw. Weiterentwicklung genutzt werden. Das Verfahren von Rubin et al. könnte durch die Ableitung der abstrahierten Instanz einer Attacke zu diesem Zweck genutzt werden. Allerdings sind die dazu benötigten Transformationsregeln (ausgenommen einfache Transformationen, wie IP-Fragmentierung) stark vom jeweiligen Angriff abhängig. Ein allgemeingültiges Vorgehen für alle Angriffe ist mit diesem Ansatz somit nicht realisierbar. Rubin et al. beschreiben aufbauend auf zwei formale Sprachen die Weiterentwicklung bzw. Verfeinerung von Signaturen [4]. Dieser Ansatz unterstützt den Signaturmodellierer bei der Beseitigung von Fehlalarmen, die durch unpräzise Signaturen ausgelöst werden. Allerdings setzt dieses Verfahren eine bereits weitgehend fehlerfreie Referenz-Signatur voraus. Larson et al. [10] präsentieren ein Werkzeug, das die Extrahierung von signifikanten Attacken-Ereignissen aus Auditdaten unterstützt. Dazu wird der Angriff ausgeführt und die entstehenden Auditdaten werden protokolliert. Anschließend wird die Differenz zwischen diesen Auditdaten und einer attackenfreien Audit-Trail ermittelt. Das Problem aus dieser Differenz eine Signatur abzuleiten, bleibt jedoch offen.

In diesem Beitrag wird ein Ansatz vorgestellt, der dem Signaturmodellierer Hinweise zur Wiederverwendung von Signaturenscheidungen auf der Grundlage bereits spezifizierter Signaturen gibt. Hierbei liegt der Fokus auf der Entwicklung von Signaturen für *Multi-Step-Attacken*, d.h. Sicherheitsverletzungen zu deren Erkennung mehrere Protokolleinträge in Zusammenhang gebracht werden müssen. Ist für einen neuen Angriff eine Signatur zu entwickeln, so sollen für den Signaturmodellierer aus der Menge der bereits spezifizierten Signaturen diejenigen abstrahiert und selektiert werden, die zur Erkennung des neuen Angriffs am besten geeignet sind. Anhand dieser Signaturen kann sich der

Modellierer dann bei der Spezifizierung der neuen Signatur orientieren bzw. Entwurfsentscheidungen sowie Erkennungsmechanismen wiederverwenden.

Der Beitrag ist wie folgt gegliedert. In Abschnitt 2 wird zunächst der neue Ansatz motiviert, bevor Abschnitt 3 eine Sprache zur Modellierung und Beschreibung komplexer Multi-Step-Signaturen skizziert. Aufbauend auf dieser Beschreibung werden in Abschnitt 4 verschiedene Transformationen zur Abstraktion von Signaturen vorgestellt, um anschließend in Abschnitt 5 ein Selektionsverfahren sowie die zugehörige Metrik für Signaturähnlichkeit zum Zweck der Wiederverwendung zu beschreiben. Abschnitt 6 demonstriert die praktische Anwendung des neuen Ansatzes. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick.

2 Ansatz

Der Entwicklungsprozess einer Signatur für eine neue Attacke kann folgendermaßen gegliedert werden: (1) Ausführung des Angriffes auf einem dedizierten System, um die beim Angriff entstehenden Spuren aufzuzeichnen. Spuren sind einzelne sicherheitsrelevante Aktionen (Basisereignisse), die durch Sensoren protokolliert werden. (2) Der Signaturmodellierer untersucht die Basisereignisse, identifiziert die wesentlichen und entwickelt (3) mit Hinblick auf die Strategie der Attacke von Grund auf schrittweise eine neue Signatur. Ein wesentlicher Aspekt dabei ist das Erkennen von Mustern, die das Aufdecken von Spuren dieser Attacke durch ein IDS ermöglichen. Nachdem die Signatur spezifiziert ist, wird (4) in der Testphase die Korrektheit bzw. Präzision überprüft und die Signatur gegebenenfalls korrigiert. Dieser Entwicklungsprozess ist aufwendig und wird durch die verwendeten Sensortypen bzw. Signatursprachen beeinflusst. Demzufolge ist der Wiederverwendung von Signaturen bzw. Signaturfragmenten ein hoher Stellenwert zuzuordnen. Neben der Verkürzung der Entwurfsphase könnte durch die Wiederverwendung von bewährten, d.h. validierten, Signaturfragmenten die aufwendige Test- bzw. Korrekturphase entscheidend verkürzt werden.

Im Folgenden wird gezeigt, wie automatisiert aus einer Menge existierender Signaturen, wiederverwendbare Signaturen selektiert werden können. Hierbei liegt die Annahme zugrunde, dass sich bestimmte Bestandteile in den Signaturen zur Erkennung ähnlicher Angriffe gleichen. Nach einer Auswahl und Abstraktion von Signaturen, die für den neuen Angriff relevant sind, kann der Signaturmodellierer relevante Parallelen erkennen und für die neue Signatur übernehmen bzw. anpassen. Relevante Signaturen zu einem neuen Angriff sind abstrahierbar und identifizierbar, indem die bekannten Signaturen sukzessive solange abstrahiert werden bis sie die Spuren des Angriffs beschreiben. Dies kann leicht überprüft werden, indem ein IDS die Spuren des Angriffs im Zusammenhang mit der abstrahierten Signatur überprüft. Die Abstraktion von Signaturen erfolgt durch Transformationen. Jede Transformationsart ist durch eine Metrik gewichtet. Durch die Gewichtung der Transformationen ist es möglich, jeder abstrahierten Signatur ein Ähnlichkeitsmaß in Bezug zur Ausgangssignatur zuzuordnen. Der Signaturmodellierer sollte nun all jene Signaturen näher untersuchen, die am schwächsten abstrahiert werden mussten, um wiederverwendbare Fragmente zu identifizieren.

Eine Signatur kann durch Entfernen bzw. Verändern von semantischen Merkmalen abstrahiert werden. Diese semantischen Abstraktionen sind zwangsläufig stark mit der verwendeten Signaturbeschreibungssprache gekoppelt. Hier wird die Petrinetz-basierte Signaturbeschreibungssprache EDL (Event Description Language) [9] als Basis für die Transformationen verwendet. Diese Sprache kann im Kontext komplexer, aus mehreren Schritten bestehender Attacken, alle von Meier identifizierten semantischen Aspekte einer Signatur spezifizieren [5] und bietet bei hoher Ausdrucksmächtigkeit dennoch eine effiziente Berechnungskomplexität [7]. Bevor die Transformationen und die Auswahl von relevanten Signaturen näher beschrieben werden, sollen kurz die grundlegenden und zum Verständnis der weiteren Ausführungen notwendigen Aspekte der Sprache EDL eingeführt werden. Eine ausführliche EDL-Beschreibung findet sich in [6].

3 Signaturbeschreibungen mit EDL

In EDL bestehen Signaturbeschreibungen aus *Plätzen* und *Transitionen*, die mit gerichteten *Kanten* verbunden sind. Plätze beschreiben Teilzustände des Systems, die eine entsprechende Attacke durchläuft. Transitionen repräsentieren Zustandsübergänge und stellen die einzelnen Ereignisse, z.B. sicherheitskritische Aktionen, dar, aus denen eine Attacke besteht. *Token* sind die dynamischen Elemente von EDL-Signaturen. Zur Laufzeit repräsentieren Token konkrete Signaturinstanzen. Dazu werden Token, analog zu gefärbten Petrinetzen, mit Werten belegt.

Plätze repräsentieren während einer Attacke die relevanten Teilzustände des Systems. Plätze, von denen eine Kante zu einer Transition *t* führt, stellen die *Vorplätze* dieser Transition *t* dar. Die Plätze, zu denen eine gerichtete Kante von der Transition *t* führt, sind *Nachplätze* der Transition *t.* EDL unterscheidet zwischen den vier Platztypen *Initial-*, *Interior-*, *Escape-* und *Exit-Platz*. *Inital-Plätze* sind die Startplätze einer Signatur und sind zum Analysebeginn mit einem initialen Token belegt. Jede EDL-Signatur besitzt genau einen *Exit-Platz*. Wenn ein Token diesen Platz erreicht, wurde durch die Signatur eine Manifestation der entsprechenden Attacke im Audit-Datenstrom erkannt. *Escape-Plätze* beschreiben den Abbruch der Verfolgung einer Attackeninstanz. Sie werden erreicht, wenn Ereignisse auftreten, die eine Vervollständigung der Attackeninstanz unmöglich machen. Token, die einen Escape-Platz erreichen, werden gelöscht. Alle anderen Plätze sind vom Typ *Interior*. Zur Illustration zeigt Abb. 1 eine einfache Signatur mit den Plätzen P_1 bis P_4.

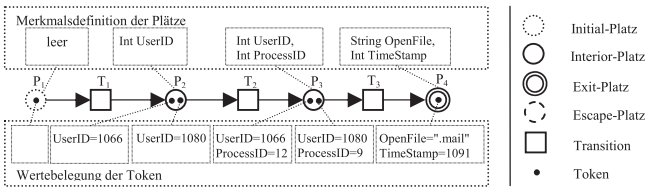

Abb. 1: Merkmale von Plätzen

Transitionen. Diese repräsentieren die Ereignisse, die Zustandsübergänge der Signaturinstanzen bewirken. Eine Transition wird charakterisiert durch Vorplätze, Nachplätze, Ereignistyp, Bedingungen, Merkmalsbindungen, Konsumtivität und Aktionen. Die *Vorplätze* einer Transition beschreiben den erforderlichen Zustand des Systems, bevor die Transition ausgelöst werden kann. Die *Nachplätze* charakterisieren den angepassten Systemzustand, der durch das sicherheitsrelevante Ereignis entsteht, das von der Transition repräsentiert wird. Jede Transition ist mit einem *Ereignistyp* assoziiert. Desweiteren kann für einen Zustandsübergang die Erfüllung zusätzlicher *Bedingungen* erforderlich sein, mit denen einerseits weitere Eigenschaften des Ereignisses oder bestimmte Zusammenhänge zwischen dem aktuellen Zustand und dem Ereignis gefordert werden. Das heißt, es können Bedingungen formuliert werden, die fordern, dass bestimmte Merkmale des Ereignisses (z.B. Benutzername) bestimmte Werte (z.B. *root*) aufweisen müssen. Außerdem können über Bedingungen Beziehungen zwischen Ereignismerkmalen und Merkmalen von Token auf Vorplätzen (z.B. gleicher Wert) beschrieben werden.

Durch Schalten einer Transition entstehen auf den Nachplätzen neue Token, die den neuen Systemzustand kennzeichnen. Um die Merkmalswerte der neuen Token auf den Nachplätzen zu definieren, besitzt eine Transition *Merkmalsbindungen*. Diese können einfache Zuweisungen oder komplexe Ausdrücke, parametrisiert mit Konstanten oder Referenzen auf Ereignismerkmale bzw. Vorplatzmerkmale, sein. Die *Konsum-Eigenschaft* einer Transition (vgl. [6]) steuert das Verbleiben der transitionsaktivierenden Token auf den Vorplätzen nach dem Schalten der Transition. Diese Eigenschaft wird für verschiedene Vorplätze separat definiert und fiktiv als Eigenschaft der verbindenden Kante definiert. Nur im *consuming*-Fall wird das Token beim Schalten der Transition vom Vorplatz entfernt.

Abb. 2 illustriert die Eigenschaften von Transitionen. T_1 besitzt zwei Bedingungen. Die erste fordert, dass Ereignis *E* das Merkmal *Typ* mit dem Wert *FileCreate* belegt. Die zweite Bedingung bezieht sich mit $P_1.UserID$ auf das Merkmal *UserID* des Platzes P_1 und *EUserID* referenziert das gleichnamige Merkmal des Ereignistyps *E*. Diese Bedingung fordert, dass der Wert des Tokenmerkmals *UserID* auf P_1 mit dem Wert des Ereignismerkmals *EUserID* übereinstimmt. T_1 besitzt zwei Merkmalsbindungen: Dem Merkmal *UserID* des neuen Tokens auf P_2 wird der Wert des Merkmals *UserID* des auslösenden Tokens von P_1 zugewiesen. Das Merkmal *Name* des neuen Tokens auf P_2 wird mit dem Wert des Ereignismerkmals *EName* belegt.

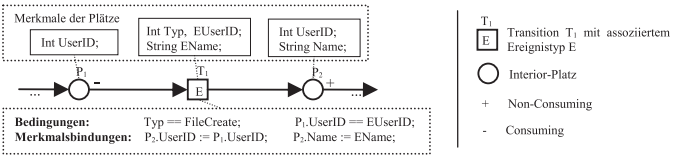

Abb. 2: Eigenschaften von Transitionen

4 Transformationen zur Abstraktion von Signaturen

Unter der Annahme, dass ähnliche Angriffe ähnliche Spuren hinterlassen und somit mittels ähnlicher Signaturen entdeckbar sind, kann die Signatur zu einem gegebenen Angriff abstrahiert werden, um ähnliche Angriffe zu entdecken. Dieser Abschnitt stellt Transformationsregeln zur Abstraktion von Signaturen vor. Eine Signatur kann durch das Entfernen bzw. Verändern von semantischen Merkmalen abstrahiert werden. Eine Signatur spezifiziert eine Menge von Manifestationen eines Angriffs. Unter einer *Manifestation* (Spur) einer Attacke werden die bei der Durchführung einer Sicherheitsverletzung generierten Audit-Datensätze bzw. Ereignisse verstanden. Seien M_S und M_{AS} die Mengen der von den Signaturen S und AS erkannten Manifestationen. Eine Transformation abstrahiert die Signatur von S zu AS, wenn $M_S \subseteq M_{AS}$ gilt. Im Folgenden werden die Transformationen näher vorgestellt, die an grundlegenden Modellierungselementen von EDL ansetzen.

Transformation 1 *(Ausweitung der Ereigniskorrelation):* Existiert in einer EDL-Signatur eine einlaufende Transitionskante mit der Konsum-Eigenschaft *consuming*, so wird diese Kante in *non-consuming* gewandelt.

Die Konsum-Eigenschaft einer Transition bestimmt, welche Ereignisse im Audit-Datenstrom miteinander korreliert werden sollen. Durch die Verwendung des *consuming*-Modus kann die Menge der in Zusammenhang betrachteten Ereignisse eingeschränkt werden (s. T_1 in Abb. 2). Das Konsumieren von Token annulliert Informationen über bereits auftretende Vorgänge. Somit vergrößert die Umwandlung einer konsumierenden Kante einer Transition in eine nicht-konsumierende Kante die Menge der zu betrachtenden Ereignispaare und damit ggf. die Menge der von der Signatur erkannten Manifestationen. Mittels Transformation 1 kann somit eine Signatur abstrahiert werden.

Transformation 2 *(Aufhebung statischer Einschränkungen):* Existiert in der EDL-Signatur eine Kontext-Bedingung, die lediglich Wertebelegungen von Merkmalen des assoziierten Ereignisses fordert, so wird diese einzelne Bedingung entfernt.

Eine solche Bedingung stellt eine *statische* Einschränkung der transitionsauslösenden Ereignisse dar. Durch das Entfernen einer solchen Bedingung vergrößert sich die Menge der Ereignisse, welche die zugehörige Transition auslösen können. Somit kann durch die Anwendung von Transformation 2 eine Signatur abstrahiert werden. Die beschriebene Transformation würde die erste Bedingung (*Typ == FileCreate*) der Transition T_1 in Abb. 2 entfernen.

Transformation 3 *(Aufhebung dynamischer Einschränkungen):* Existiert in der EDL-Signatur eine Kontext-Bedingung an einer Transition, die bestimmte Wertebelegungen des mit der Transition assoziierten Ereignisses in Abhängigkeit von einer Signaturinstanz (Token) fordert, so wird diese Bedingung entfernt.

Diese Bedingungen realisieren eine dynamische Einschränkung der Ereignisse, die die Transition auslösen können. Die zweite Bedingung ($P_1.UserID == EUserID$) in Abb. 2 stellt eine solche Bedingung dar. Dabei werden Token-Merkmale mit Ereignis-

Merkmalen korreliert. Somit wird die Menge der transitionsauslösenden Ereignisse für eine einzelne Signaturinstanz (Token) eingeschränkt, weil Zusammenhänge zwischen derzeitig auftretenden und bereits aufgetretenen Ereignissen gefordert werden. Durch das Entfernen einer solchen Bedingung wird diese Beschränkung aufgehoben und die Menge der auslösenden Ereignisse vergrößert. Demzufolge vergrößert sich die Menge der erkannten Manifestationen und die Transformation abstrahiert die Signatur.

Transformation 4 *(Entfernen von Vorbedingungen):* Existiert eine Transition *T*, die als Vorplätze nur Initial-Plätze der EDL-Signatur aufweist, so wird diese entfernt. Vorplätze von *T*, die nicht Vorplätze von anderen Transitionen sind, werden entfernt. Die Nachplätze von *T* (falls nicht soeben, aufgrund einer Schleife entfernt, Vorplatz==Nachplatz) bilden neue Initial-Plätze der abstrahierten EDL-Signatur.

Wird diese Transformation auf die in Abb. 3 dargestellte Signatur angewendet, so wird die Transition T_1 entfernt und es entsteht die Signatur in Abb. 4.

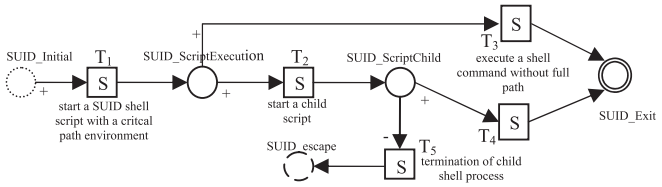

Abb. 3: Ausgangssignatur vor der Transformation

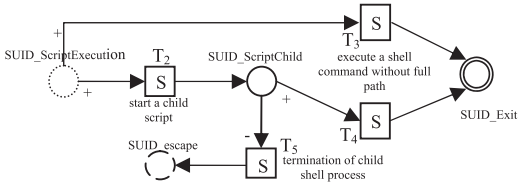

Abb. 4: Abstraktion der Ausgangssignatur durch Transformation 4

Durch das Entfernen einer Transition, die nur Initial-Plätze als Vorplätze aufweist, wird auf das Auftreten dieses zeitlich zuerst geforderten Ereignisses verzichtet. Somit ist dieses Ereignis zum Auslösen der abstrahierten Signatur nicht mehr notwendig. Durch diese Abstraktion werden bestimmte Vorbedingungen der Ausgangssignatur ignoriert. Dementsprechend vergrößert sich die Menge der erkannten Manifestationen und die Transformation abstrahiert die Signatur.

Transformation 5 *(Entfernen von Abschlussbedingungen):* Existiert eine Transition *T*, die als Nachplätze nur Exit-Plätze der EDL-Signatur aufweist, so wird diese entfernt. Nachplätze von *T*, die nicht Nachplätze von anderen Transition sind, werden ebenfalls entfernt. Die Vorplätze von *T* bilden neue Exit-Plätze der abstrahierten EDL-Signatur. Abb. 5 zeigt exemplarisch die abstrahierte Signatur aus Abb. 3, nachdem durch die Transformation die Transition T_4 entfernt wurde.

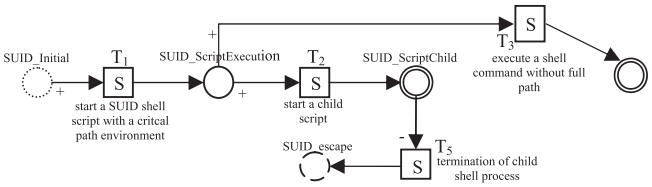

Abb. 5: Abstraktion der Ausgangssignatur durch Transformation 5

Durch diese Transformation wird das zeitlich zuletzt geforderte, transitionsauslösende Ereignis unnötig. Auch ohne dieses Ereignis wird die abstrahierte Signatur ausgelöst. Durch diese Transformation werden bestimmte Abschlussbedingungen der Signatur ignoriert und daher stellt sie eine Abstraktion der Ausgangssignatur dar.

Transformation 6 *(Entfernen von Abbruchbedingungen):* Entfernen einer Transition *T*, die als Nachplatz einen Escape-Platz der Signatur aufweist. Ferner wird der Escape-Platz der Transition *T* entfernt, falls er an keiner weiteren Transition mehr referenziert wird. Transitionen, die als Nachplatz einen Escape-Platz aufweisen, repräsentieren Abbruchkriterien bei der Attackenerkennung. Durch das Entfernen einer solchen Transition werden ggf. auch die Signaturinstanzen weiterverfolgt, die ansonsten verworfen werden. Demzufolge werden durch diese Transformation Abbruchbedingungen ignoriert und somit kann die Menge der erkannten Manifestationen gegenüber der Ausgangssignatur vergrößert werden, so dass die Transformation die Signatur abstrahiert.

Rahmenbedingungen für die Transformationsanwendungen: Die angegebenen Transformationen abstrahieren eine Signatur *S* nur, wenn für die resultierende Signatur *AS*, die Ausgangssignatur *S* und die zugehörigen Mengen M_S bzw. M_{AS} der Manifestation gilt, dass $M_S \subseteq M_{AS}$ ist. Dies ist nicht bei allen Transformationen zwangsläufig gegeben. Durch geeignete Vorbedingungen ist dies sicherzustellen, ebenso wie die syntaktische Korrektheit der resultierenden Signatur. Des Weiteren ist die Vollständigkeit und Minimalität der Menge der Transformationen zu untersuchen.

5 Grad der Wiederverwendbarkeit von Signaturen

Die in Abschnitt 4 vorgestellten Transformationen abstrahieren Signaturen. Um dem Signaturmodellierer Signaturen vorzuschlagen, die sich zur Wiederverwendung eignen könnten, müssen die bekannten Signaturen sukzessive abstrahiert werden. Anschließend werden die abstrahierten Signaturen im Zusammenhang mit den Spuren des neuen Angriffs evaluiert. Ferner wird die Ähnlichkeit der Signaturen zu der jeweiligen Ausgangssignatur bewertet. Durch das Ähnlichkeitsmaß können diejenigen Signaturen zur Wiederverwendung selektiert werden, die am wenigsten abstrahiert wurden, um die Spuren des Angriffs zu erkennen. Dies erfolgt unter der Annahme, dass diese Signaturen Angriffe beschreiben, die eine Ähnlichkeit zu der neuen Attacke aufweisen. Zunächst wird der Abstraktionsprozess der Signaturen erläutert, um anschließend eine Bemessungsgrundlage für den Grad der Abstraktion zu veranschaulichen. Danach wird der Selektionsprozess erläutert.

5.1 Abstraktionsgraphen von Signaturen

Durch die induktive Anwendung von Transformationen auf die Ausgangssignatur *S* aus der Menge *K* der bekannten Signaturen können alle Abstraktionen von *S* konstruiert werden. Die abstrahierten Signaturen können bezüglich ihrer Konstruktion in Relation gestellt werden. Zwei Signaturen *A* und *B* stehen in Relation, falls sich *B* durch Anwendung der angegebenen Transformationen aus *A* erzeugen lässt. Die Relationen zwischen der Ausgangssignatur *S* und ihren Abstraktionen lassen sich durch einen Baum darstellen, wobei *S* die Wurzel des Baumes repräsentiert. Direkte Kinder von *S* sind die Abstraktionen von *S*, die sich durch die einmalige Anwendung einer Transformation erzeugen lassen. Der Abstraktionsbaum einer Signatur und die zugehörigen Abstraktionen können somit sukzessive generiert werden.

Signaturen definieren die Menge der erkannten Manifestationen der zugehörigen Attacke. Aufgrund der verwendeten Transformationen beschreibt die einem Knoten zugeordnete Signatur unter anderem alle Manifestationen der Vatersignatur. Demzufolge stellt der Baum von Signaturen eine *Enthaltensein*-Beziehung über die Menge der definierten Manifestationen auf. Respektive beschreibt jede abstrahierte Signatur mindestens die Manifestationen der Ausgangssignatur *S*.

Abb. 6 stellt exemplarisch einen Konstruktionsbaum der Abstraktionen aus S dar. Beispielweise wurde Abstraktion AS_1 mittels der Transformation 1 direkt aus *S* erzeugt. Ferner wurde die Signatur AS_7 aus AS_1 durch Transformation 3 erzeugt. Demzufolge gilt für die Manifestationen M_S, M_{AS1}, M_{AS7} der Signaturen *S*, AS_1 und AS_7: $M_S \subseteq M_{AS1} \subseteq M_{AS7}$.

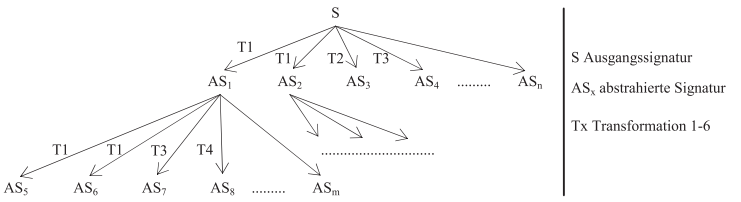

Abb. 6: Konstruktionsbaum und Manifestationsmengen

Der aus dem Abstraktionsvorgang resultierende Baum enthält gegebenenfalls identische Knoten. Zwei Knoten werden als identisch bezeichnet, falls die zugehörigen Signaturen identisch sind. Bei der Zusammenfassung von identischen Knoten entsteht ein gerichteter azyklischer Graph (DAG). Dieser Graph vereinheitlicht für identische Signaturen die Information über die Reihenfolge der Transformationsanwendungen. Diese Information ist für die spätere Bemessung des Abstraktionsgrades einer Signatur irrelevant, da eine unterschiedliche Reihenfolge von Transformationsanwendungen in der gleichen Signatur resultieren kann. Nachdem der DAG vorliegt, werden die generierten Signaturen im Zusammenhang mit den Spuren *T* des neuen Angriffs durch ein IDS automatisiert getestet. Der Test erfolgt durch eine Breitensuche und endet, falls eine abstrahierte Signatur gefunden wurde, welche die Spuren *T* erkennt und keine Signatur im DAG ungetestet ist, die einen größeren Grad der Ähnlichkeit bzgl. der Ausgangssignatur aufweist. Die Be-

messung der Ähnlichkeit wird im folgenden Abschnitt erörtert. Der Test unter Zuhilfenahme der DAG-Repräsentation verhindert doppelte Tests gegenüber der Baum-Repräsentation. Ferner sollten die Konstruktion und die Evaluierung der Signaturen mit den Spuren des neuen Angriffs parallel ablaufen, so dass nur die zum Test notwendigen Signaturen generiert werden.

5.2 Bemessung der Ähnlichkeit von Signaturen

Die Kanten des im Abschnitt 5.1 konstruierten DAG werden im Folgenden gewichtet. Dies erfolgt mittels einer Metrik δ. Jeder in Abschnitt 4 beschriebenen Transformation wird durch δ ein Wert zugeordnet. Falls die Signatur *AS* aus der Signatur *S* unter Anwendung der Transformation *X* erzeugt wurde, ist die Kante zwischen den zu *S* und *AS* gehörenden Knoten *P* und *P'* mit dem Metrikwert *A* der Transformation *X* zu gewichten. Nachdem die Kanten im Graph bewertet wurden, kann die Ähnlichkeit unterschiedlicher abstrahierter Signaturen gegenüber der Ausgangssignatur auf der Grundlage der Metrik δ verglichen werden. Dazu werden die Kantengewichte auf dem Pfad vom zur Signatur gehörenden Knoten bis zur Wurzel summiert. Diese Summe bestimmt den Abstraktionsgrad der Signatur.

5.3 Allgemeine Vorgehensweise

Im Folgenden wird die generelle Arbeitsweise der Selektion von Signaturen aus einer Menge *K* beschrieben. Die Spuren *T*, die der neue Angriff auf einem System hinterlässt, sind aufzuzeichnen und zu isolieren [10]. Anschließend wird jeder Signatur *S* aus *K* ein Abstraktionsgrad zugeordnet, wobei eine von S abstrahierte Signatur die Spuren *T* erkennt. Dazu sind die nachfolgenden fünf Schritte für jede Signatur *S* aus *K* auszuführen. (1) Konstruktion aller verschiedenen abstrahierten Signaturen, die sich aus *S* mittels der Transformationen ableiten lassen. (2) Generierung des zugehörenden Abstraktionsgraphen. (3) Bewertung der Kanten im DAG mittels einer Metrik δ. (4) Analyse der Spuren *T* im Zusammenhang mit allen aus *S* abgeleiteten Signaturen durch ein IDS. Signaturen, welche die Spuren *T* erkennen, werden markiert. (5) Aus der Menge der markierten Signaturen bzw. Knoten im Abstraktionsgraphen wird das geringste Kantengewicht zur Wurzel bestimmt. Dieser Abstraktionsgrad wird zur Signatur *S* vermerkt. Nachdem diese Schritte für alle Signaturen aus *K* durchgeführt wurden, liegt für jede Signatur ein Metrikwert der nötigen Abstraktion vor. Die Signaturen aus *K*, die am wenigsten abstrahiert werden müssen, um die Spuren *T* zu erkennen, werden dem Signaturmodellierer zur Wiederverwendung vorgeschlagen. Anhand dieser Signaturen kann sich der Signaturmodellierer beim Entwurf der neuen Signatur orientieren. Die für jede Signatur aus *K* durchzuführenden Schritte 1-5 können wie in Abschnitt 5.1 erläutert parallelisiert bzw. optimiert werden.

6 Anwendungsbeispiel

Um einen Eindruck von der Eignung des vorgestellten Verfahrens zu erhalten, wird der Mechanismus exemplarisch angewandt. Dabei werden die Vorgehensweise, Resultate und mögliche Verbesserungen erläutert.

Die Grundlage für die Menge der bekannten Signaturen bilden u. a. die in [7] beschriebenen Signaturen einer Suid-Skript-, einer Link-Shell- und einer Failed-Login-Attacke. Als neuer Angriff wird eine Attacke verwendet, die starke Ähnlichkeit zur Suid-Skript-Attacke aufweist. Beide Angriffe nutzen die Umgebungsvariable PATH und den Suid-Mechanismus aus. Durch die PATH-Variable kann ein Nutzer den Suchpfad für ausführbare Dateien festlegen. Bei der Suid-Skript-Attacke wird vorausgesetzt, dass ein Verzeichnis (*Dir*) existiert, das vom Angreifer beschrieben werden kann, und dass weiterhin ein Suid-Root-Skript existiert, welches ein Kommando (*Cmd*) ohne Angabe des vollständigen Pfades aufruft. Dann kann vom Angreifer ein zu *Cmd* gleichnamiges Skript im Verzeichnis *Dir* angelegt werden, welches dann bei der Ausführung des Suid-Root-Skriptes mit den Rechten von *root* gestartet wird. Abb. 3 zeigt den schematischen Aufbau der zugehörigen EDL-Signatur. Der neue Angriff verwendet fast den gleichen Mechanismus und erzeugt auch fast die gleichen Spuren, allerdings wird hier anstelle des Suid-Root-Skripts eine Anwendung verwendet. Der Aufruf von Anwendungen wird im Gegensatz zu Skripten durch andere Ereignisse bzw. andere Wertebelegungen protokolliert.

Auf Grundlage der aufgezeichneten Audit-Trail des neuen Angriffs und den Signaturen wird das Verfahren durchgeführt. Zur Bemessung der Ähnlichkeit wird eine Metrik δ verwendet, die jeder Transformation den Wert 1 zuordnet. Abb. 7 zeigt die Resultate. Es werden die angewendeten Transformationsarten, deren Häufigkeiten, der Abstraktionsgrad, sowie die Anzahl der Plätze bzw. Transitionen der vier am besten bewerteten Ausgangssignaturen dargestellt.

Signaturname	Abstraktionsgrad (bzgl. $\delta(x)=1$)	angewandte Transformationen	Anzahl Transitionen	Anzahl Plätze
Suid-Skript	1	1*Transform. 2	5	5
JoinMailFile	3	3*Transform. 2	7	7
Link-Shell	4	3*Transform. 2 1*Transform. 3	9	5
Failed-Login	6	6*Transform. 2	7	5

Abb. 7: Signatur-Ranking

Als Signatur für den neuen Angriff kann die SUID-Skript-Signatur verwendet werden. Es ist lediglich eine Bedingung der Transition T_1 anzupassen. Selbst wenn der Angriff dahingehend abgewandelt wird, dass statt dem zu *Cmd* gleichnamigen Skript eine Anwendung im Verzeichnis *Dir* abgelegt ist, wird die Suid-Skript-Signatur aus der Menge der gegebenen Signaturen als geeignetste Signatur zur Wiederverwendung vorgeschlagen. Für die Erkennung eines solchen Angriffs müssten zusätzlich die Bedingungen der Transitionen T_2, T_3 und T_4 angepasst werden. Die eigentliche Signaturcharakteristik, welche die Verfolgung von Kindprozessen realisiert, bleibt dabei bestehen.

7 Schlussbemerkungen

Durch die Wiederverwendung von Signaturentwurfsentscheidungen bzw. Fragmenten aus bereits spezifizierten Signaturen kann der Entwurfsprozess von neuen Signaturen unterstützt werden. Wiederverwendung von bewährten Strukturen kann nicht nur den Aufwand für den Entwurfsprozess der neuen Signatur reduzieren, sondern auch die aufwendige Test- bzw. Korrekturphase kann möglicherweise entscheidend verkürzt werden. Dieser Beitrag skizzierte ein automatisierbares Verfahren mit dessen Hilfe aus einem Menge von bestehenden Signaturen, diejenigen Signaturen selektiert werden können, die Attacken beschreiben, welche Ähnlichkeit zu dem neuen Angriff aufweisen. Unter Zuhilfenahme dieser Signaturen sollte der Signaturmodellierer die neue Signatur spezifizieren. In weiteren Arbeiten sollen die Vollständigkeit und Minimalität der Menge der Transformationen untersucht werden. Ferner sind aussagekräftige Ähnlichkeitsmetriken zu bestimmen. Dies soll durch Evaluierung des Verfahrens bzw. der Metriken im Zusammenhang mit unterschiedlichen Angriffstypen und größeren Signaturmengen unterstützt werden.

8 Literaturverzeichnis

[1] Cheung S.; Lindqvist U.; Fong M.: Modeling Multistep Cyber Attacks for Scenario Recognition. In Proc.: DARPA Information Survivability Conference and Exposition 2003.

[2] Gamma E., Helm R., Johnson E. R., "Design Patterns – Elements of Reusable Object-Oriented Software", Addison-Wesley Professional, 1997.

[3] Rubin S., Jha S., Miller B.: Automatic Generation and Analysis of NIDS Attacks. In Proc.: 20th Annual Computer Security Applications Conference, Tucson, Arizona, 2004.

[4] Rubin S.; Jha S.;Miller P. B.: Language-based generation and evaluation of NIDS signatures. In Proc.: IEEE Symposium on Security and Privacy, Oakland, CA, May 2005.

[5] Meier M.: A Model for the Semantics of Attack Signatures in Misuse Detection Systems. In: Proc. of 7th ISC 2004, LNCS 3225, Springer, 2004.

[6] Schmerl S.: Entwurf und Entwicklung einer effizienten Analyseeinheit für Intrusion-Detection-Systeme. Diplomarbeit, Lehrstuhl Rechnernetze, BTU Cottbus, 2004.

[7] Meier M.; Schmerl S.: Improving the Efficiency of Misuse Detection. In Proc. of 2nd GI-onference DIMVA 2005, LNCS 3548, Springer, 2005.

[8] Lippmann, R.; Webster, S.; Stetson, D.: The Effect of Identifying Vulnerabilities and Patching Software on the Utility of Network Intrusion Detection. In Proc. of 5th RAID Symposium, LNCS 2516, Springer, 2002.

[9] Meier M., Schmerl S.: Effiziente Analyseverfahren für Intrusion-Detection-Systeme. In Proc. of the 2nd Conference on "Sicherheit - Schutz und Zuverlässigkeit", LNI P-62, Köllen, 2005.

[10] Larson U., Lundin Barse E., Jonsson E.: METAL - A Tool for Extracting Attack Manifestations. In Proc. of 2nd GI-Conference DIMVA 2005, LNCS 3548, Springer, 2005.