2 Spezielle Bedrohungen

Kapitelüberblick

Das Kapitel widmet sich Klassen von Bedrohungen, die Ursache für die in heutigen Systemen am häufigsten auftretenden Sicherheitsprobleme sind. Dazu zählen Bedrohungen durch Pufferüberlauf-Angriffe, Viren, Würmer und Trojanische Pferde sowie mobile Applikationen (Apps). Nicht abgefangene Pufferüberläufe (engl. *Buffer Overflow*) sind noch immer sehr weit verbreitete Schwachstellen, die durch nachlässige Programmierung sowohl in Betriebssystemen als auch in Anwendungsdiensten auftreten. Abschnitt 2.2 erläutert die Buffer-Overflow-Problematik, erklärt die prinzipielle Funktionsweise von Exploits, die solche Schwachstellen ausnutzen, und diskutiert Abwehrmaßnahmen. Abschnitt 2.3 beschäftigt sich dann mit dem allgemeinen Aufbau und der Funktionsweise von Computerviren und gibt Hinweise auf wirksame Maßnahmen zur Virenabwehr. Die charakteristischen Eigenschaften von Würmern zusammen mit möglichen Abwehrmaßnahmen werden in Abschnitt 2.4 behandelt. Abschnitt 2.5 widmet sich in gleicher Weise der Thematik der Trojanischen Pferde. In den letzten Jahren wurden zunehmend Angriffe über sogenannte Bot-Netze durchgeführt. Abschnitt 2.6 erläutert die prinzipielle Arbeitsweise eines Bot-Netzes. Abschnitt 2.7 beleuchtet die Probleme, die mit Apps einhergehen können. Abschließend gehen wir in Abschnitt 2.8 auf die Anfang 2018 bekannt gewordenen speziellen Sicherheitsprobleme Meltdown und Spectre ein, von denen nahezu alle heute im Einsatz befindlichen Prozessoren betroffen sind.

2.1 Einführung

Schadsoftware

Spezielle Klassen von Bedrohungen eines IT-Systems gehen von Viren, Würmern und Trojanischen Pferden aus, die wir als Schadsoftware (engl. *malware* oder *malicious code*) bezeichnen. Es handelt sich hierbei um Programme, die festgelegte, dem Benutzer jedoch verborgene Funktionen ausführen. Die Festlegung der zusätzlichen und beabsichtigten (i.d.R. böswilligen) Funktionalität unterscheidet diese Klasse von Programmen von den so genannten Wanzen (bugs), deren Fehlverhalten meist von nicht beabsichtigten Programmierfehlern verursacht wird. Der Befall von IT-Systemen durch Viren, Würmer und Trojanische Pferde führt jedes Jahr zu erheblichen

https://doi.org/10.1515/978110563900-002

wirtschaftlichen Schäden. So schätzte die US-Beratungsfirma Computer Economics[1] den Schaden, den Schadsoftware bereits 2006 weltweit angerichtet hat, auf über 13 Milliarden Dollar. Im Jahr 2008 hat die Firma Symantec 1 656 227 neue Auftreten von Schadsoftware erkannt. Das bedeutet einen Anstieg um 265 Prozent im Vergleich zu 2007[2]. Diese steigende Tendenz hat über die Jahre angehalten. So hat das Deutsche Bundeskriminalamt (BKA) für das Jahr 2016 fast 100.000 Fälle von Internetkriminalität registriert mit einem Schaden von über 50 Millionen Euro für Privathaushalte, wobei der Graubereich der nicht gemeldeten oder nicht bemerkten Vorfälle sehr hoch ist und in den Statistiken gar nicht auftritt. Im Mai 2017 verursachte allein der Trojaner WannaCry den Ausfall von über 200.000 nicht sorgfältig gepatchte Windows-Systemen weltweit mit Schäden in mehreren hundert Millionen Dollar.

Architektur-Eigenschaften

Basiskonzepte und -eigenschaften heutiger Rechnerarchitekturen erleichtern das erfolgreiche Durchführen von Buffer-Overflow-Angriffen bzw. ermöglichen die Existenz von Trojanischen Pferden und die Verbreitung von Viren und Würmern. Zu nennen ist dabei die gemeinschaftliche (aber nicht notwendigerweise gleichzeitige) Nutzung von Programmen und Daten (engl. *sharing*). Eine solche gemeinsame Nutzung wird u.a. über eine zum Teil sehr freizügige Netzwerkfreigabe oder über öffentliche Verzeichnisse ermöglicht. Die gemeinsame Nutzung von Ressourcen ist ebenfalls ein Charakteristikum von Peer-to-Peer Overlay-Netzen, die sich seit einiger Zeit etablieren. Weit verbreitet sind Musiktauschbörsen und File-Sharing Systeme wie Gnutella, KaZaA oder BitTorrent. Die Eigenschaft der gemeinsamen Ressourcennutzung wird insbesondere von Viren zur Verbreitung ausgenutzt. Eine weitere wesentliche Eigenschaft ist die universelle Interpretierbarkeit von Daten. Sie besagt, dass kein Unterschied zwischen Daten und Programmen besteht. Programme können demnach wie Daten leicht modifiziert werden, indem man sie beispielsweise um einen Schadensteil erweitert. Die universelle Interpretierbarkeit bedeutet aber auch, dass in Speicherbereiche, die eigentlich ausschließlich Daten beinhalten wie beispielsweise die Laufzeitkeller (engl. *stack*) von Prozessen, gezielt ausführbarer Code geladen werden kann. Werden diese binären Daten nun als ausführbarer Code interpretiert, so kann auf diese Weise Schadsoftware zur Ausführung gelangen. Dies wird von den Buffer-Overflow-Angriffen ausgenutzt.

Sharing

universelle Interpretation

Als dritte wesentliche Eigenschaft ist die Transitivität des Informationsflusses zu nennen. Falls eine Information vom Rechner A zum Rechner B fließt

Transitivität

[1] Quelle: Studie von Computer Economics http://www.welt.de/die-welt/article1037708/Computer_werden_zu_Zombies.html

[2] Quelle: Symantec Threat Report, http://www.symantec.com/threatreport/

und es Möglichkeiten gibt, Informationen von B zu einem Rechner C fließen zu lassen, so kann die Information auch von A zu C weitergeleitet werden. Die Transitivität wird insbesondere von Würmern ausgenutzt, um sich auf diese Weise zu verbreiten.

Techniken zur Bedrohungsabwehr

Zur Abwehr der Bedrohungen kann man drei Ansätze verfolgen, die in der Praxis in der Regel in verschiedenen Kombinationen eingesetzt werden. Zu nennen sind an erster Stelle Techniken zur Verhinderung (engl. *prevention*) von Angriffen wie beispielsweise die Verwendung stark typisierter Programmiersprachen, Code-Verifikation oder die Beschränkung von Zugriffen aufgrund von systembestimmten Regeln (z.B. kein Schreiben mit anschließendem Ausführen von Daten auf dem Stack-Segment). Im weitesten Sinn gehören in diese Klasse von Techniken auch Maßnahmen zur Sensibilisierung und Bewusstseinsbildung insbesondere auch von Entwicklern von Software-Systemen. Klar ist, dass Vermeidungstechniken nach dem Motto „Sicherheit per Design“die konzeptuell befriedigenste Lösung darstellen, wir aber heute in der Praxis der System-Konstruktion hiervon noch sehr weit entfernt sind.

Verhinderung

Die zweite Klasse von Ansätzen beschäftigt sich deshalb mit Techniken zur Erkennung und frühzeitigen Abwehr möglicher Bedrohungen. Als Beispiele seien statische Analysen des Programmcodes genannt, um z.B. Code-Stücke, die eine Buffer-Overflow-Schwachstelle aufweisen (u.a. Kopieren von Zeichenfolgen ohne die Längenangaben zu prüfen), zu identifizieren und zu korrigieren. Aber auch die bekannten Viren-Scanner und Intrusion Detection Systeme gehören in diese Klasse von Techniken, die anhand charakteristischer Merkmale versuchen, potentielle Schadsoftware zu erkennen.

Erkennung und Behebung

Zu der dritten Klasse von Ansätzen zählen Techniken, die darauf abzielen, die Auswirkungen von eingetretenen Bedrohungen zu begrenzen (engl. *mitigation*). Dazu zählen Isolierungsansätze wie das Sandboxing, die darauf abzielen, Anwendungsbereiche gegeneinander abzuschotten, so dass ein Schaden, der bei der Ausführung eines Anwendungsprogramms auftritt, sich nicht auf andere Anwendungsbereiche ausbreiten und womöglich das ganze System beeinträchtigen kann.

Schadensbegrenzung

2.2 Buffer-Overflow

Buffer-Overflow-Angriffe nutzen Schwachstellen aus, die sich aus Implementierungsfehlern als Folge einer nachlässigen Programmierung ergeben. Der Ansatzpunkt für entsprechende Angriffe sind Programme, in denen Daten in einen Bereich einer Variablen fester Länge, z.B. in einen String oder ein Feld fester Länge, eingelesen bzw. kopiert werden, ohne dass das

Buffer-Overflow

Programm prüft, ob die kopierte Eingabe überhaupt in den bereitgestellten Bereich passt. Derartige Variablen, in die Daten kopiert werden, sind abstrakt gesehen Pufferbereiche, woraus sich der Name dieses Angriffs ableitet. Durch das Schreiben einer zu großen Datenmenge in den Bereich des Puffers wird dieser zum Überlauf (engl. *overflow*) gebracht.

2.2.1 Einführung

Wie auch bei den Viren, so kann man auch bei den Buffer-Overflow-Angriffen verschiedene zeitliche Entwicklungsphasen unterscheiden. Während die erste Generation der Angriffe vordringlich Operationen zum Kopieren von Zeichenketten für Angriff ausnutzten, werden heute zunehmend auch einzelne Integer-Zahlen zum Überlaufen gebracht oder es werden Anweisungen, wie Schleifen, die nicht korrekt terminieren, für Überlauf-Angriffe missbraucht.

C, C++

Schwachstellen, die für Buffer-Overflow Angriffe ausgenutzt werden können, finden sich häufig in Programmen, die in den Programmiersprachen C bzw. C++ geschrieben sind. Probleme ergeben sich beispielsweise durch die Verwendung von Funktionen zur Verarbeitung von Zeichenketten, wie `strcpy()`. Diese Funktion kopiert Zeichen für Zeichen des Eingabestrings auf den Zielstring, ohne dabei zu prüfen, ob der Bereich des Zielstrings groß genug ist, um die Eingabe zu speichern. Der Kopiervorgang terminiert erst dann, wenn im Eingabestring der ASCII-Wert 0 gelesen wird. Auf diese Weise ist es möglich, gezielt vorhandene Werte von Variablen, die auf dem Speicherbereich abgelegt sind, der sich unmittelbar an den Bereich des Zielstrings anschließt, mit eingeschleusten Werten zu überschreiben.

Häufig sind auch Funktionen wie *sizeof()* Ausgangspunkt für eine Buffer-Overflow Verwundbarkeit. So wird beispielsweise bei der Transformation von Daten in Unicode ein Datenstring einer bestimmten Länge, die durch das Kommando *sizeof()* bestimmt wird, in einen anderen Datenstring kopiert. In der Unicode-Transformation wird die Größenangabe jedoch in Byte und nicht in der Anzahl der Characters berechnet, so dass bei einer falschen Verwendung viel mehr Speicher überschrieben wird als vorgesehen war. Ein weltweit bekanntes Beispiel eines Angriffs, der eine derartige Schwachstelle ausnutzte, war der Code-Red Wurm im Jahre 2001, der einen Pufferüberlauf in Microsofts IIS (Internet Information Service) ausnutzte, um das System zu befallen.

Wie bereits weiter oben kurz angedeutet, werden heutzutage auch zunehmend Schwachstellen ausgenutzt, die sich durch die unsichere Programmierung einer Loop-Schleife ergeben. Häufiges Beispiel ist hier das zeichenweise Kopieren eines Eingabestrings in dieser Loop-Schleife, z.B. das Kopieren einer URL, wobei als Abbruchkriterium der Schleife das Le-

sen eines bestimmten Zeichens festgelegt wird. Fehlt dieses Abbruchzeichen in der Zeichenkette oder taucht es gezielt erst nach einer Menge von anderen Daten auf, so werden alle diese Daten Zeichen für Zeichen kopiert. Der bekannte Blaster-Wurm (siehe Seite 68) hat beispielsweise eine solche Schwachstelle ausgenutzt.

Da unter anderem alle UNIX-Derivate (u.a. BSD-UNIX, Solaris, HP-UX, Linux) und die aktuellen Windows-Versionen in C bzw. C++ geschrieben sind, bilden immer wieder fehlerhaft programmierte Betriebssystemdienste, Serverroutinen oder auch Bibliotheksfunktionen Angriffspunkte für Buffer-Overflows.

Ansatzpunkt

In einem Programmfragment in der Programmiersprache C könnte ein möglicher Ansatzpunkt für Buffer-Overflow Angriffe wie folgt gegeben sein:

```
cmd =lese_aus_netz();
do_something(cmd);
int do_something(char *string){
    char buffer[4]; /* Pufferbereich */
    strcpy(buffer,string);  /* Ansatzpunkt: */
     /* Einlesen von Daten in Pufferbereich */
     .....
    return 0;
}
```

Adressraumverwaltung

virtueller Adressraum

Um die Vorgehensweise und Ziele eines Buffer-Overflow-Angriffs verstehen zu können, muss man sich zunächst einmal klarmachen, auf welche Weise in heutigen Betriebssystemen die Daten eines Prozesses, also eines in Ausführung befindlichen Programms, verwaltet werden. Alle gängigen Betriebssysteme unterstützen heute virtuellen Speicher. Jedem Prozess ist ein virtueller Adressraum zugeordnet, der – wie in Abbildung 2.1 skizziert – vom Betriebs- und Laufzeitsystem verwaltet wird.

Segmente

Typischerweise wird der virtuelle Adressraum in drei logische Bereiche, die Segmente genannt werden, aufgeteilt, wobei das Text- bzw. Codesegment die ausführbaren Befehle des Prozesses enthält. In der Regel wird dieses Segment vom Betriebssystem gegen ein Überschreiben geschützt. Auf dem Halden-Bereich (engl. *heap*) werden zur Programmlaufzeit dynamisch erzeugte Datenobjekte abgelegt, aber auch globale Variablen sowie globale Konstanten. Typische dynamische Objekte sind Zeigerstrukturen oder Objekte in objektorientierten Programmiersprachen. Der Heap-Bereich wächst normalerweise von niedrigen Adressen zu höheren Adressen. In der Regel ist die Trennlinie zwischen dem Heap und dem Stack-Segment fließend.

Heap

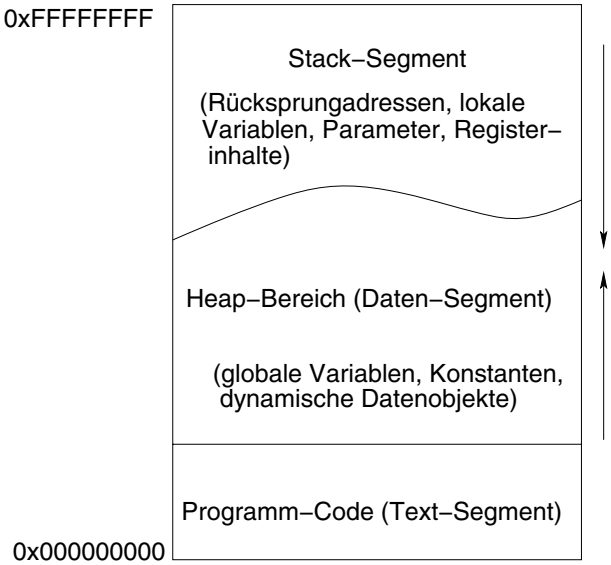

Abbildung 2.1: Segmentierung des virtuellen Prozessadressraums

Stack

Der Laufzeitkeller, das Stacksegment (engl. *stack*), wird LIFO[3]-artig verwaltet und dient zur Verwaltung von Prozeduraufrufen. Auf dem Stack werden u.a. lokale Variablen von Prozeduraufrufen, deren Eingabeparameter sowie die Rücksprungadresse des Aufrufs abgelegt. Der Stack-Bereich wächst von hohen zu niedrigen Adressen; er wächst also dem Heap entgegen. Nach Beendigung eines Prozeduraufrufs werden die lokalen Daten, Parameter und sonstigen Umgebungswerte des Aufrufs wieder vom Stack entfernt. Bei der Programmübersetzung erzeugt der jeweilige Compiler Maschinencode, um diese Daten auf Stack und Heap-Bereichen zu speichern bzw., wie im Falle des Stack-Bereichs, sie auch wieder zu entfernen.

Schutz?

Im Unterschied zum Code-Segment können weder das Stack- noch das Heap-Segment vom Betriebssystem gegen Überschreibungen geschützt werden, da ein schreibender Zugriff ja gerade beim Ablegen von dynamischen lokalen bzw. globalen Daten benötigt wird. Gravierender ist, dass die Daten in diesen Bereichen in der Regel nicht nur gelesen und geschrieben werden dürfen, sondern auch ausführbar sind. Wir werden bei der Diskussion der Gegenmaßnahmen hierauf zurückkommen.

2.2.2 Angriffe

Wie bereits einführend erwähnt, ist es das Ziel eines Buffer-Overflow-Angriffs den Bereich, der für eine Variable im Speicher reserviert ist, zu

[3] Last in first out

überschreiben und in der Regel eben auch zum Überlaufen zu bringen. Derartige Daten können prinzipiell an unterschiedlichen Stellen im Speicher abgelegt werden. Dazu zählt in erster Linie das Stack-Segment, auf dem lokale Variablen sowie Eingabeparameter, die bei Prozedur- bzw. Methodenaufrufen verwendet werden, abgelegt werden. Es kann sich aber ebenso um den Heap-Bereich des Prozess-Adressraums handeln, auf dem die dynamisch erzeugten Datenobjekte liegen, oder aber auch um einen Bereich handeln, der von mehreren Prozessen gemeinsam genutzt und an jeweils unterschiedliche Stellen des virtuellen Adressraums des einzelnen Prozesses eingeblendet wird. Im Folgenden konzentrieren wir die Beschreibung von Angriffen auf die Stack-Bereichs-Überläufe, da sie sehr häufig auftreten. Aber natürlich muss man sich der anderen Angriffspunkte stets bewusst sein.

Buffer-Overflows auf Stack-Segmenten

überschreiben

Ein Buffer-Overflow-Angriff zielt nun darauf ab, über Eingabeparameter bei einem Prozeduraufruf den Speicherbereich derjenigen lokalen Variable (d.h. den Puffer) auf dem Stack, die diese Eingabe aufnehmen soll, zum Überlauf zu bringen. Das Ziel besteht dann meist darin, durch einen gezielt konstruierten Eingabestring die auf dem Stack abgelegte Rücksprungadresse des Prozeduraufrufs zu überschreiben (vgl. Abbildung 2.2). Hierbei wird wesentlich die Organisation des Stacks ausgenutzt, der wie gesagt von hohen zu niedrigen Adressen wächst. Bei einem Prozeduraufruf werden stets zunächst die Verwaltungsdaten wie z.B. die Rücksprungadresse auf dem Stack abgelegt, bevor Speicherplatz für die lokalen Variablen, die zur Prozedurausführung benötigt werden, angelegt wird. Bei der Speicherung von aktuellen Werten in diesen reservierten Speicherbereichen, also z.B. bei der Übernahme der Eingabewerte eines Eingabstrings, werden die Daten beginnend bei den niedrigen Anfangsadressen der lokalen Variablen auf dem Stack abgelegt.

An Bild 2.2 kann man sich unmittelbar klarmachen, dass durch eine zu lange Eingabe die gespeicherten Werte, die auf dem an den für die lokale Variable reservierten Adressbereich unmittelbar angrenzenden Bereich abgelegt sind, überschrieben werden.

Einschleusen von Fremd-Code

Fremd-Code

Nach dem Überschreiben der Rücksprungadresse kann der Fall vorliegen, dass die überschriebenen Speicherzellen keine sinnvolle Adresse mehr enthalten, so dass das Programm bei der Ausführung des Rücksprungbefehls aufgrund eines Speicherzugriffsfehlers (engl. *segmentation fault*) abstürzen wird. Damit hat der Angreifer zwar u.U. bereits großen Schaden verursacht, aber ggf. noch nicht sein eigentliches Ziel, ausführbaren Code auf dem Stack zu platzieren, erfolgreich erreicht. Der Angreifer wird also versuchen, seinen Eingabestring so zu konstruieren, dass über die Rücksprungadres-

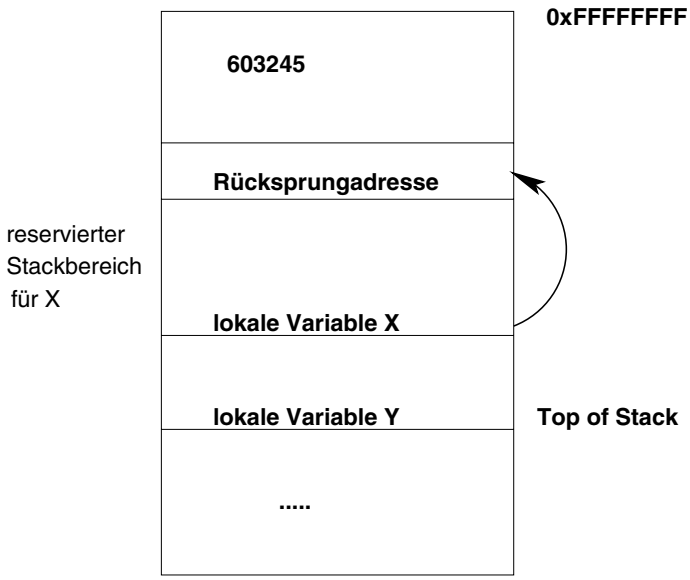

Abbildung 2.2: Buffer-Overflow mit Überschreiben der Rücksprungadresse

se die Anfangsadresse des eingeschleusten Schadcodes oder ein anderer, bereits geladener Code angesprungen und dann ausgeführt wird. Der eingeschleuste Code wird mit den Berechtigungen des Prozesses ausgeführt, dessen Stack durch den Buffer-Overflow manipuliert wurde. Da der Angreifer auf diese Weise seine Zugriffsrechte vermehrt und im Falle von befallenen Systemprozessen auch anhebt, spricht man auch häufig von Elevation of Privileges-Attacken. Ein erfolgreicher Buffer-Overflow-Angriff ist nur ein Beispiel dafür, wie man als Angreifer seine Rechte ausweiten kann.

Das Problem, das sich hierbei ergibt, besteht darin, dass der Angreifer die genaue, d.h. die absolute Adresse seines eingeschleusten Codes auf dem Stackbereich kennen muss, um gezielt dorthin zu springen. Die absolute Adresse kann jedoch nicht unbedingt vorab ermittelt werden, da sie unter anderem davon abhängig ist, wie viele lokale Variablen und Registerinhalte auf dem Stack abgelegt werden. Für dieses Problem gibt es jedoch eine einfache Lösung. In den zu präparierenden Eingabestring kann der Angreifer eine Anzahl von No-Operation Befehlen (NOP) einstreuen und die Rücksprungadresse mit der Adresse eines dieser NOP-Befehle überschreiben. Da Prozessoren NOP-Befehle einfach ignorieren und überspringen, wird automatisch der nächste Befehl angesprungen bis der eingeschleuste Code erreicht ist (vgl. Abbildung 2.3)

NOPs

komplexe Funktionalität

Das nächste Problem, das ein Angreifer nun haben könnte, ist der relativ begrenzte Speicherbereich, der ihm mit dem Stack als Speicherort für seinen eingeschleusten Code zur Verfügung steht. Dies ist aber nur ein kleines

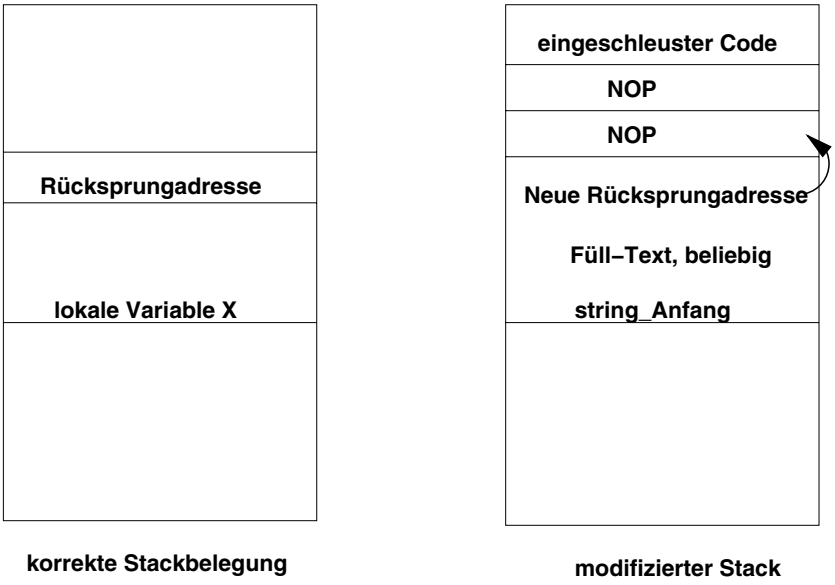

Abbildung 2.3: Einschleusen von Code mit NOPs-Befehlen

Hindernis, da zum einen mit hoch optimiertem Assemblercode komplexe Programme erstellt werden können, die nur einen geringen Speicherbedarf besitzen. Zum anderen stehen dem Angreifer viele Funktionen direkt zur Verfügung, ohne dass er sie selber einschleusen muss. So liegen in Windows-Systemen viele Bibliotheken (DLLs) bereits im Speicher. Ferner stehen dem Angreifer alle diejenigen Funktionen des Windows API zur Verfügung, die bereits zu dem attackierten Programm hinzu gebunden sind. Befindet sich unter diesen gebundenen Funktionen beispielsweise auch die Funktion `LoadLibrary`, so können durch den Exploit sogar beliebige Funktionen dynamisch nachgeladen werden. Da Bibliotheken häufig an gleichen Stellen im Speicher abgelegt werden, kann ein Angreifer diese direkt aufrufen, um beispielsweise TCP/IP-Verbindungen aufzubauen oder Befehle einer Shell auszuführen.

2.2.3 Gegenmaßnahmen

sichere Programmierung

Aus dem Besprochenen wird klar, wie man sich als Software-Entwickler vor Buffer-Overflow-Angriffen schützen kann. Durch sorgfältige Programmierung (u.a. [175]), bei der insbesondere alle Eingabewerte geprüft und Bereichsgrenzen kontrolliert werden, kann man verhindern, Ziel von gezielten Buffer-Overflow-Angriffen zu werden. Wie bereits eingangs erwähnt, treten besonders in Programmen der Programmiersprachen C und C++ Pufferüberlaufprobleme auf. Durch den Übergang auf Programmiersprachen mit einer höheren Programmiersicherheit wie Java, könnten die entspre-

chenden Probleme überwunden werden. So überprüft beispielsweise das Java-Laufzeitsystem die Einhaltung von Bereichsgrenzen automatisch. Aber auch im C bzw. C++ Umfeld gibt es Möglichkeiten, die Programmiersicherheit zu erhöhen. So kann man beispielsweise anstelle der bereits angesprochenen unsicheren Funktion `strcpy()` die Funktion `strncpy` verwenden, die es ermöglicht, die Zahl der zu schreibenden Zeichen zu begrenzen[4]. Da die Funktion `strcpy()` aber nur eine von vielen derartigen problematischen Bibliotheksfunktionen ist, sollte man eine spezielle Bibliothek verwenden, durch die sichere Bibliotheksfunktionen zur Verfügung gestellt werden. Unter Linux leistet dies beispielsweise die Bibliothek `Libsafe`[5], mit der Standard-C Bibliotheksaufrufe gekapselt (engl. *to wrap*) und durch die gesicherte Version ersetzt werden.

Wrapper

Hat man den Quellcode von Programmen zur Verfügung, so kann man diesen unter Zuhilfenahme von speziellen Tools bearbeiten, um Überlaufangriffe zu erkennen. Ein Beispiel hierfür ist das `StackGuard Tool`[6] für UNIX bzw. StackShield für Linux, das im Prinzip einen Patch für den GNU C-Compiler bereitstellt. Der derart modifizierte Compiler sichert bei jedem Funktionsaufruf die Returnadresse und korrigiert sie bei Bedarf. Der Stack-Smashing-Protector, der im GNU C-Compiler seit der Version 4.1 verfügbar ist, fügt ein spezielles Kontrollzeichen, das so genannte Canary[7], direkt hinter die Rücksprungadresse auf den Stack ein. Ferner erzeugt er Code, der vor dem Rücksprung prüft, ob das Kontrollzeichen verändert wurde. Wird eine solche Änderung erkannt, so wird automatisch eine Warnmeldung in die Log-Datei des Systems geschrieben und das Programm abgebrochen. Ein analoges Konzept wurde mit den Stack Cookies von Microsoft u.a. in den Windows Server 2003 und Vista integriert.

Compiler-unterstützung

Natürlich sind solche Maßnahmen unbefriedigend, da damit ja nicht die Ursache des Problems behoben, sondern nur die Auswirkungen begrenzt werden. Da die Absicherung von Systemen gegen Angriffe von außen ein ständiger Wechsel zwischen Aktionen seitens der Angreifer und Reaktionen seitens der Software-Entwickler und Systemadministratoren bedeutet, ist klar, dass die skizzierten Compilermodifikationen keinen nachhaltigen Effekt haben werden. Angreifer werden ihre Angriffe daraufhin verfeinern und dafür sorgen, dass das eingefügte Kontrollzeichen trotz Manipulation erhalten bleibt.

---

[4] Microsoft bietet `strncpy_s` anstelle von `strncpy`.

[5] http://http://www.research.avayalabs.com/gcm/usa/en-us/initiatives/all/nsr.htm

[6] http://immunix.org

[7] In Anlehnung an das frühere Vorgehen in Bergwerken, wo man einen Kanarienvogel unter Tage mitgenommen hat. Sobald dieser aufhörte zu singen bzw. gar starb war klar, dass ein bedrohlicher Sauerstoffmangel herrschte.

Der gerade beschriebene Ansatz setzt voraus, dass die abzusichernden Programme im Quellcode vorliegen, so dass sie mit den entsprechend modifizierten Compilern erneut übersetzt werden können. Eine Neu-Übersetzung der Software ist aber häufig gar nicht möglich, wie zum Beispiel bei proprietären Betriebssystemen oder bei kommerzieller Anwendungssoftware.

proprietärer Code

Mit den Techniken ASLR (Address Space Layout Randomization) und DEP (Data Execution Prevention) stehen in herkömmlichen Betriebssystemen Konzepte zur Verfügung, um den Schaden von erfolgreich durchgeführten Buffer-Overflows zu begrenzen bzw. Angriffe prinzipiell zu erschweren. Mit DEP werden Speicherbereiche als nicht-ausführbar gekennzeichnet, so dass kein Code, insbesondere kein Schadcode in einem solchen Datensegment ausgeführt werden kann. Ein entsprechender Versuch, Code auszuführen, löst eine Speicherschutzverletzung aus.

ASLR, DEP

Die ASLR-Technik erschwert Angriffe, die Kenntnisse über das Speicherlayout nutzen, um den Speicher zu manipulieren. Beispiele für solche Angriffe sind neben den beschriebenen Buffer-Overflows auch Heap Overflows bzw. auch Underflows, Format-String Verwundbarkeiten, oder auch Array Index Overflows. Die randomisierte Speicherung von ausführbarem Binärcode, von Dynamic Link Libraries (DLLs), Text-, Heap- und teilweise auch Stack-Segmenten mittels der ASLR-Technik (vgl. u.a. [162]) verschleiert deren Speicheradresse, die für die genannten Klassen von Angriffen oder auch für Angriffe unter Nutzung von ROP (Return-oriented Programming) notwendig ist, um die gespeicherten Module aufrufen zu können. Die ASLR Technik wurde als erstes im Betriebssystem OpenBSD verwendet. Sie wird seit Windows Vista auch mit Windows-Betriebssystemen angeboten und auch unter MAC-OS, Linux mit PaX ASLR und auch iOS oder auch Android sind ASLR-Varianten im Einsatz.

Beispielsweise kann unter Windows 7 für eine ausführbare Binärdatei (.exe) oder auch eine DLL (.dll) durch das Setzen eines speziellen Bits in dem PE Header des Executables[8] die Verwendung der ASLR Technik angefordert werden. Für das zu ladende Executable wird in diesem Fall ein spezieller Offset-Wert berechnet, um dynamisch, aus Sicht eines Angreifers randomisiert, die Speicherposition zu bestimmen. Der Offset ist ein Wert aus dem Wertebereich von 1 bis 256 und ist an 64 KB ausgerichtet (aligned). Wird ein Programm ausgeführt, das ASLR Bit gesetzt hat, wird auch das Speicherlayout des Prozesses randomisiert, indem seine Stack- und Heap-Segmente randomisiert angelegt werden und auch der initiale Stack-Pointer durch einen randomisierten Wert aus dem Wertebereich 1 bis 16384 dekrementiert wird. Das Speicherlayout der Stack- und Heap-Segmente wird bei jeder Programmausführung erneut randomisiert festgelegt, während das

ASLR bei Windows

[8] PE (Portable Executable) beschreibt ein Binärformat ausführbarer Programme.

Layout des Code- und Datensegments sich nur bei einem Re-Boot ändert. Der Offset von gemeinsam genutzten DLLs wird über eine systemglobale Variable zum Bootzeitpunkt ermittelt und die DLL wird in den gemeinsam genutzten Speicherbereich in den Bereichen zwischen $0x50000000$ und $0x78000000$ abgelegt. Die DLLs werden zudem in randomisierter Reihenfolge geladen.

Trusted OS

Speziell abgesicherte Varianten von Standardbetriebssystemen, die als Trusted Operating Systems vertrieben werden (z.B. Trusted Solaris[9], Trusted HP-UX), führen so genannte Compartments ein, die Systembereiche gegeneinander abschotten, so dass sich die Auswirkungen von Buffer-Overflow-Angriffen auf ein Compartment beschränken lassen.

Abschließend sei noch einmal darauf hingewiesen, dass die Buffer-Overflow-Verwundbarkeit sich nicht auf den Stack beschränkt, sondern Heap-Bereiche in gleicher Weise betroffen sind. Auch beschränken sich die heutigen Angriffe bei Weitem nicht nur auf die gängigen Operationen zum Kopieren von Zeichenketten, sondern nutzen eine Vielzahl weiterer Angriffsflächen aus. Wir haben bereits auf die Loop-Probleme, Integer-Overflows oder auch die Unicode Probleme hingewiesen. Alle diese Schwachstellen rühren von Programmierfehlern her, die man zum Teil frühzeitig durch statische Programmanalysen und durch eine sorgfältige Programmierung in den Griff bekommen könnte. Auf diesem Gebiet sind zurzeit eine Reihe von Tool-Entwicklungen in Arbeit, z.B. auch in den Forschungsbereichen von Microsoft, so dass die Hoffnung besteht, dass Buffer-Overflows in Zukunft nicht mehr länger die „Schwachstellen des Jahrzehnts“sind, wie sie noch in den 90ger Jahren von Bill Gates bezeichnet wurden.

2.3 Computerviren

Ein Virus (deutsch *giftiger Saft*) bezeichnet in der Biologie einen Mikro-Organismus, der auf eine lebende Wirtszelle angewiesen ist, keinen eigenen Stoffwechsel besitzt und fähig ist, sich zu reproduzieren. Diese Eigenschaften sind direkt auf Computerviren übertragbar.

2.3.1 Eigenschaften

Der Begriff des Computervirus wurde zum ersten Mal 1985 durch Arbeiten von F. Cohen (u.a. [43]) der breiten Öffentlichkeit bekannt.

[9] Trusted Solaris wurde 2006 als eigenständiges Produkt eingestellt.

Definition 2.1 (Computervirus)

Virus

Ein Computervirus ist eine Befehlsfolge, die ein Wirtsprogramm zur Ausführung benötigt. Viren sind zur Reproduktion fähig. Dazu wird bei der Ausführung des Virus eine Kopie (Reproduktion) oder eine modifizierte Version des Virus in einen Speicherbereich, der diese Befehlssequenz noch nicht enthält, geschrieben (Infektion). Zusätzlich zur Fähigkeit zur Reproduktion enthalten Viren in der Regel einen Schadensteil. Dieser kann unbedingt oder bedingt durch einen Auslöser aktiviert werden.

□

mutierend

Mit den Festlegungen von Definition 2.1 ist klar, dass ein Computervirus kein selbständig ablauffähiges Programm ist. Falls ein Virus sich nicht identisch reproduziert, also keine identische Kopie erzeugt, sondern die den Virus definierende Befehlssequenz modifiziert, so spricht man von einem mutierenden Virus.

Reproduktionsfähigkeit

Struktur

Zur Codierung der Befehlsfolge eines Virus verwendet man Maschinensprachen ebenso wie Kommandosprachen, Skriptsprachen oder auch Hochsprachen. Potentielle Speicherbereiche für Viren sind der Code ausführbarer Programme von Benutzern, Bereiche des Betriebssystems oder auch Sektoren eines Hintergrundspeichermediums. Der allgemeine Aufbau eines Virus ist in Abbildung 2.4 skizziert. Ein Virus besteht aus einer Viren-Kennung sowie optional aus einem Infektions-, einem Schadens- und einem Sprungteil. Durch die Ausführung des Infektionsteils kopiert sich ein Virus in einen Speicherbereich. Handelt es sich dabei um den in einer Datei gespeicherten Code eines ausführbaren Programms, so werden i.d.R. die Strukturinformationen des infizierten Programms durch die Angabe der neuen Dateilänge sowie durch die Veränderung der Einsprungadresse angeglichen. Die neue Einsprungadresse entspricht der Anfangsadresse des Virus-Codes. Soll nach der Ausführung des Virus doch noch das Programm mit der ursprünglichen Funktionalität ausgeführt werden, so enthält der Virus-Code im optionalen Sprungteil eine Rücksprungadresse, die die Einsprungadresse in das ursprüngliche Programm ist.

Kennung

Anhand einer speziellen Viren-Kennung (z.B. des Wertes `4711` in der Abbildung 2.4), kann ein Virus in seiner Infektionsphase erkennen, ob ein Programm schon vom selben Virus befallen ist. Dies verhindert das wiederholte Infizieren von Programmen. Diese Viren-Kennung (oder eine andere typische Zeichenfolge) dient im Gegenzug Virenerkennungsprogrammen, den Viren-Scannern, zur Aufdeckung von infizierten Programmen.

Bedingung

Die Ausführung des Schadensteils kann von Randbedingungen abhängig gemacht werden. Im Beispiel von Abbildung 2.4 wird auf ein spezielles Da-

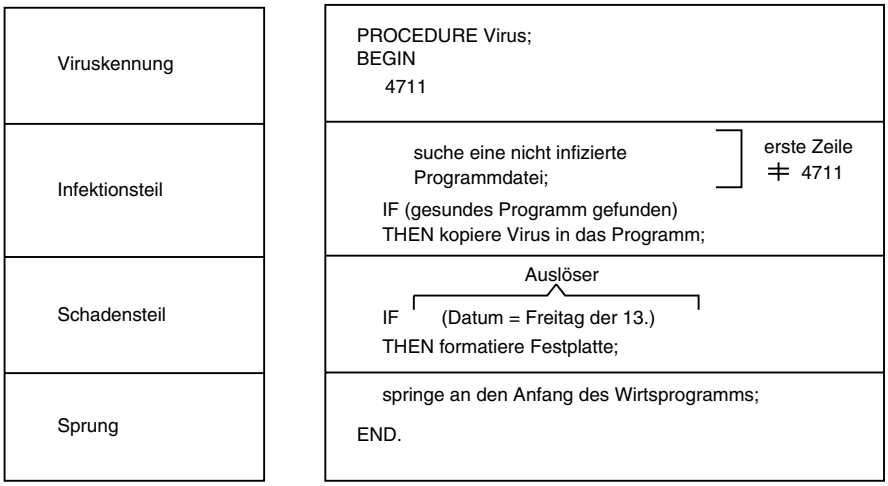

Abbildung 2.4: Allgemeiner Aufbau eines Virus und Beispiel

tum, nämlich Freitag den 13ten, gewartet. Ist zum Zeitpunkt der Ausführung des Wirtsprogramms die Bedingung erfüllt, so wird die angegebene Schadensfunktion wie die Neuformatierung der Festplatte ausgeführt. Soll nach der Abarbeitung der Befehlssequenz des Virus das ursprüngliche Programm ausgeführt werden, so wird ein Sprungbefehl eingefügt, mit dem an die Anfangsadresse des Programms gesprungen wird.

Durch Viren treten in erster Linie Bedrohungen der Integrität und Vertraulichkeit auf.

2.3.2 Viren-Typen

Erste Generation

Mit dem Einzug des Personal Computers (PCs) in den 80er Jahren in Behörden, Firmen und in Privathaushalte erfolgte eine Abkehr von zentral, meist noch manuell administrierten Großrechenanlagen (engl. *mainframe*) hin zu einer dezentralen Verwaltung isoliert betriebener PCs durch deren Benutzer. Das heißt, die entsprechenden Benutzer übernahmen Aufgaben der Systemadministration und waren berechtigt, Software auf ihren PCs zu installieren. Zusammen mit der steigenden Anzahl gemeinsam genutzter Programme, insbesondere Spielprogrammen und Shareware, war damit der Boden zur Verbreitung von Viren geebnet. Die Viren der ersten Generation verbreiteten sich in erster Linie über das Kopieren von Daten von Diskette zu Diskette sowie über die meist noch manuelle Installation von Programmen auf einzelnen PCs.

Viren-Verbreitung

Isolierung

Durch ein Nachbilden der ursprünglichen, zentralistischen Mainframe-Administration kann jedoch in solchen Umgebungen die Verbreitung von

Viren relativ einfach und wirkungsvoll eingegrenzt werden. Dazu ist ein zentraler Server-Rechner so zu konfigurieren, dass jedes neue Softwareprodukt zunächst auf diesem Rechner installiert und getestet wird. Der Server stellt somit eine Art „Quarantänestation“ für Software dar. Das systematische Testen und Kontrollieren von Software muss jedoch einem ausgewählten, vertrauenswürdigen Personenkreis vorbehalten bleiben, damit die Kontrolle nicht unterlaufen wird bzw. nicht durch eine unsachgemäße Behandlung Sicherheitslücken bestehen bleiben. Bemerkenswerter Weise gehört das Einrichten von Quarantäne-Netzen und Quarantäne-Rechnern heute wieder zu den häufig verwendeten Schutz-Maßnahmen in Unternehmen, um in einer dezentralen IT-Landschaft über zentral administrierte Komponenten neue Software zunächst in einem beherrschten und kontrollierbaren Umfeld zu evaluieren, bevor sie auf Rechner im Unternehmen verteilt werden.

Die häufigsten Viren, die in dieser Zeit auftraten, waren die so genannten Programm- und die Bootsektor-Viren.

Programm-Viren

Link-Virus

Ein Programm-Virus (auch bekannt als Link-Virus) kopiert sich in eine ausführbare Datei. Er wird durch den Aufruf des Programms verbreitet, da mit der Programmausführung auch der Virus-Code ausgeführt wird. In Abbildung 2.5 ist die Struktur eines infizierten Programms skizziert. Nach der Infektion liegt ein ausführfähiges Programm vor, das beim Aufruf zunächst den Virus und dann erst das eigentliche Programm ausführt. Zur Verschleierung des Virus wurden auch die Strukturdaten der infizierten Datei verändert.

Boot-Viren

Boot-Virus

Boot- oder Bootsektor-Viren befallen die Bereiche einer Diskette oder einer Festplatte, deren Daten beim Hochfahren des Rechners, also beim Systemstart, gelesen und in den Hauptspeicher geladen werden. Beim Booten (vgl. auch Seite 594) eines Rechners initialisiert sich zunächst die CPU und startet einen Code zur Überprüfung der Hardware. Dieser Code befindet sich im ROM BIOS (Basic Input/Output System) auf dem Motherboard. Durch die Ausführung des BIOS-Codes werden im nächsten Schritt die Befehle geladen, die im Bootsektor der Diskette bzw. Festplatte abgelegt sind. Der Bootsektor enthält das Ladeprogramm, durch dessen Ausführung das Betriebssystem in den Hauptspeicher geladen und ausgeführt wird. Ein Boot-Virus wird meist vor dem Bootsektorprogramm in den Speicher geschrieben, so dass bei jedem Hochfahren des Systems zunächst der Virus ausgeführt wird. Bootsektor-Viren sind resident im Hauptspeicher geladen. Die residente Speicherung von Daten bedeutet, dass diese nicht auf den Hintergrundspeicher ausgelagert werden, sondern während der gesamten

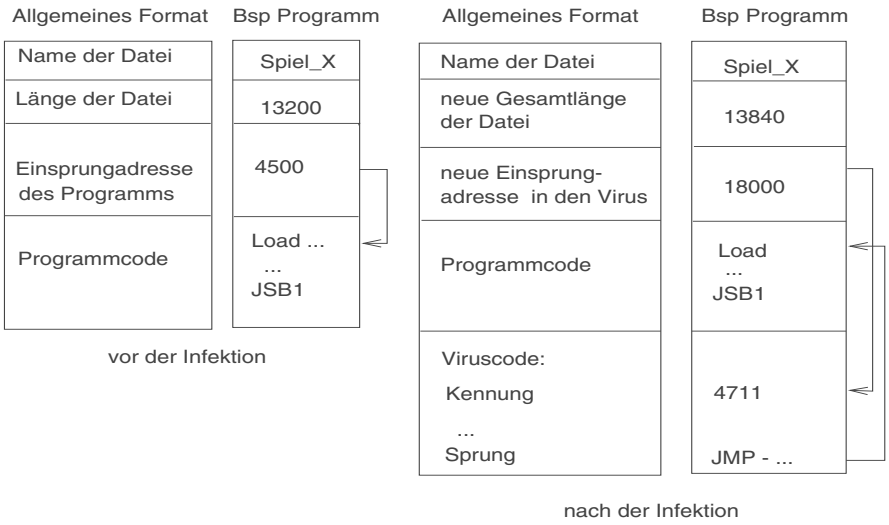

Abbildung 2.5: Infiziertes Programm

Arbeitsphase des Systems im Hauptspeicher eingelagert sind. Abbildung 2.6 skizziert einen infizierten Bootsektor. Ein Boot-Virus kann beispielsweise die Interrupt-Handler-Tabelle verändern, die u.a. die Anfangsadresse der Treiberroutinen enthält, die bei einem Geräte-Interrupt auszuführen sind. Konnte beispielsweise anstelle der Anfangsadresse des Treiber-Codes für die Maus der Anfangscode des Virus platziert werden, würde das bedeuten, dass bei einem durch die Maus ausgelösten Geräte-Interrupt stets der Virus-Code anstelle des Handlers für die Maus gestartet würde. Obwohl sie großen Schaden verursachen können, spielen Boot-Viren heutzutage keine nennenswerte Rolle mehr, da sich die Verbreitungswege von Viren und deren Funktionsweise geändert haben.

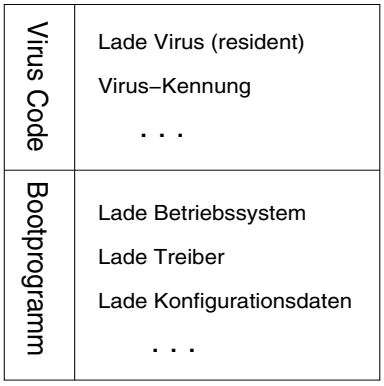

Abbildung 2.6: Infizierter Bootsektor

Viren der zweiten Generation

Durch Übergang zu vernetzten, offenen Systemen eröffnet sich eine Vielzahl von Kanälen, über die fremder Code auf den lokalen Rechner gelangen kann. Beispiele für solche in der Praxis häufig anzutreffende Kanäle sind E-Mails mit Anhang (engl. *attachment*), Java-Applets, elektronische Dokumente oder auch Bild-Dateien. Auch die Verbesserung heutiger Software-Werkzeuge trägt erheblich zur Verbreitung von Viren bei. Diese zunächst wenig einleuchtende These lässt sich jedoch einfach mit der zunehmenden Automatisierung von Vorgängen bei der Informationsverarbeitung begründen. Wesentliches Ziel der entwickelten Werkzeuge ist es nämlich, möglichst viele Aktivitäten automatisch und transparent für den Benutzer durchzuführen. Diese eigentlich höchst wünschenswerte Eigenschaft hat jedoch für den Virenbereich fatale Konsequenzen. Unter Nutzung von Werkzeugen wie Postscript- und PDF-Interpretern, Textverarbeitungsprogrammen (u.a. Word-Prozessor) oder MIME-Interpretern werden Befehle, die in fremden Dokumenten eingebettet sein können, automatisch ausgeführt. Auf diese Weise werden dann die in den Befehlen enthaltenen Viren transparent für den Benutzer, also ohne dessen explizites Eingreifen, in Umlauf gebracht.

Werkzeuge

Die Verbreitung von Viren über Rechnernetze hinweg hat darüber hinaus auch zu einer Veränderung der Funktionalität von Viren geführt. Während die Viren der ersten Generation vordringlich destruktiv ausgerichtet waren, findet man unter den neueren Viren zunehmend solche, die gezielt einen weiteren Angriff vorbereiten, indem sie sicherheitsrelevante Informationen des Zielrechners sammeln (u.a. abgelegte Passworte), oder indem sie gezielt Konfigurations- und Systemdateien modifizieren. Mit dem Wandel der Schadfunktionen von Viren und deren geänderten Verbreitungswegen verschwimmen auch zunehmend die konzeptuellen Unterschiede zwischen Viren und Würmern (vgl. Abschnitt 2.4). Häufig werden die Begriffe Virus und Wurm deshalb synonym verwendet.

Funktionalität

Makro- oder Daten-Viren

Makros sind in der Regel kleine Code-Teile, die in Daten-Dokumenten eingebettet werden. Solche Makros können beispielsweise bei der Erstellung von Word-Dokumenten oder Tabellenkalkulationen mittels Excel einem Daten-Dokument hinzugefügt werden. Damit erhalten diese Dateien einen ausführbaren Teil, so dass ein Dokument das Programm, durch das sie erstellt wurden, wie z.B. den Word-Prozessor, über die Ausführung eines solchen Makros kontrollieren und das Verhalten des Programms an die spezifischen Anforderungen des Dokuments anpassen kann. Beispielsweise kann man über ein Makro, das einem Word-Dokument hinzugefügt wird, dafür sorgen, dass bei jedem Abspeichern des Dokuments automatisch ein

Makros

Programm zur Prüfung der Rechtschreibung aufgerufen wird. Makros enthalten Steuerbefehle, die in einer Makrosprache wie zum Beispiel Microsofts Visual Basic for Applications (VBA), formuliert sind und interpretativ ausgeführt werden. Die Erweiterung von reinen Daten-Dateien zu Objekten mit ausführbaren Bestandteilen stellt einen gravierenden Schritt für die Virenproblematik dar. Nun sind auch Daten-Dateien, die bislang nur gelesen und nicht ausgeführt wurden, potentielle Wirte für Viren. Dadurch ergibt sich ein sehr großes Potential neuer Bedrohungen. Vom Standpunkt eines Benutzers aus betrachtet reicht jetzt bereits ein lesender Zugriff auf ein Dokument aus, z.B. das Öffnen eines Word-Dokuments, um eine Virus-Verbreitung zu initiieren. Dass sich hinter dem Lese-Zugriff die Ausführung von Makro-Kommandos und damit das Starten eines Virus verbergen kann, ist für den Benutzer nicht unbedingt ersichtlich.

Makro-Virus

Seit 1995 sind die ersten Makro-Viren bekannt. Sie stellen bereits heute die häufigste Ursache für einen Virenangriff dar. Die ersten Vertreter dieser Viren-Klasse traten vorzugsweise in Dateien der Programme Word für Windows (WinWord) sowie in Excel-Programmen auf. Makro-Viren sind in der Regel unabhängig von der verwendeten Rechnerplattform, da sie interpretiert ausgeführt und nicht in Maschinencode übersetzt werden. Diese Plattformunabhängigkeit von Makro-Viren stellt eine weitere gravierende Verschärfung der Viren-Problematik dar, da auf diese Weise die „natürlichen“ Barrieren für die Verbreitung von Viren in einer heterogenen Umwelt entfallen. Die WinWord-Viren haben sich beispielsweise auf PCs mit unterschiedlichen Betriebssystemen u.a. Windows 3.xx, Windows NT oder Windows 95 verbreitet.

Attachment

Daten-Viren treten häufig auch in Postscript-Dateien und in Anhängen (engl. *attachment*) von MIME (Multipurpose Internet Mail Extension) Mails auf. MIME ist eine Erweiterung des SMTP-Protokolls zur Nachrichtenkommunikation, das neben der Übertragung einfacher ASCII-Dateien auch die Übertragung von Grafiken, Sprache etc. ermöglicht. Aus Sicherheitssicht problematisch ist, dass MIME-Nachrichten Befehle beinhalten können, die beim Empfänger der Nachricht ausgeführt werden. So ist es zum Beispiel möglich, beim Empfänger einen Datei-Transfer mittels FTP zu initiieren.

Postscript

Für Viren, die sich in Postscript-Dateien (ps-Dateien) verstecken, gilt ganz Analoges. Postscript ist eine Beschreibungssprache zur Darstellung von Grafikdaten und formatierten Texten. Postscript-Dateien (.ps) werden von einem Interpreter ausgeführt, der bei Postscript-fähigen Druckern direkt in den Drucker integriert ist oder als Programm zur Verfügung stehen muss, um die Darstellung von Postscript-Dateien auf dem Bildschirm zu ermöglichen (z.B. mittels *ghostview*). Die Postscript-Sprache enthält einige Befehle, die durch den Interpreter ausgeführt und für Angriffe missbraucht werden kön-

nen. Dazu gehören hauptsächlich Befehle zum Löschen oder Umbenennen von Dateien oder zur Erzeugung einer Kommunikationsverbindung, über die Daten ausgetauscht werden können.

ani-Viren

Varianten der Makro-Viren sind Viren, die in Dateien eingebettet werden, die eigentlich keinen ausführbaren Code enthalten, wie beispielsweise .gif, .wmf oder .ani-Dateien[10]. Die entsprechenden Dateien werden in der Regel über Webseiten oder e-Mails verbreitet. Eine Schwachstelle in den Routinen zur Verarbeitung von Dateien zur Definition des Cursors (.cur), animierter Cursors (.ani) und Icons (.ico) kann von einem Angreifer dazu ausgenutzt werden, beliebigen Programmcode einzuschleusen und mit System-Privilegien auf dem Opfer-Rechner auszuführen. Die Schwachstelle besteht darin, dass die Routine zum Laden der Cursor keine Bereichsprüfung durchführt, sondern der Längenangabe, die die Größe des Headers angibt, vertraut. Die Routinen werden z.B. vom Internet Explorer oder von Outlook aufgerufen, wenn sie .ani-Dateien in einer Webseite oder einer HTML e-Mail empfangen. Um die Schwachstelle erfolgreich auszunutzen, ist es also erforderlich, dass das Opfer eine entsprechende Datei auf einer Webseite ansieht bzw. herunter lädt oder eine entsprechende E-Mail-Nachricht öffnet.

Beispiel 2.1 (Sobig Virus)

Sobig

An dem Sobig-Virus, der sich Anfang 2003 rasant ausbreitete, lässt sich die Funktionsweise klassischer Viren sehr gut verdeutlichen. Der Virus nutzte auf schon fast klassische Weise Social Engineering Vorgehen aus, um sich via E-Mail-Attachments zu verbreiten. In der Subjekt-Zeile der Mail traten unverfängliche, die Neugier des Empfängers anregende Bezeichnungen auf, wie *Re: Sample, Re: Document* oder *Re: Movies*. Die eigentliche Mail enthielt eine Nachricht der Art *Please see the attached file for Details*, die den Empfänger dazu animieren sollte, das beigefügte Attachment, das eine mit einem Virus infizierte pif-Datei (z.B. Sample.pif) enthielt, zu öffnen. Durch einen Doppelklick auf den Anhang wurde dessen Infizierungs- und Schadfunktion ausgelöst.

Social Engineering

Infektionsteil

Der aktivierte Virus führte seinen Infektionsteil aus, der zunächst eine Kopie von sich selbst unter dem Namen *Winmgm32.exe* im Windows-Verzeichnis ablegte und folgende zwei Einträge (Registry-Keys) in der Windows-Registry eintrug, so dass der Virus beim Booten automatisch gestartet wird:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows
\CurrentVersion\Run
"WindowsMGM"=C:\WINDOWS\winmgm32.exe
```

[10] Animated Cursor Handling

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
\CurrentVersion\Run
"WindowsMGM" = C:\WINDOWS\winmgm32.exe
```

Ferner wurde anhand der Netzwerkfreigabe aktiv nach möglichen Opfersystemen gesucht, um den Virus auf diese Systeme zu kopieren. Durchsucht wurden folgende Verzeichnisse:

```
\WINDOWS\ALL USERS\START MENU\PROGRAMS\STARTUP   oder
 \DOCUMENTS AND SETTINGS\ALL USERS\START MENU\PROGRAMS
                                                \STARTUP
```

Schadensteil

Im Schadensteil führte der Virus verschiedene Teilschritte durch. In einem ersten Schritt verschickte er sich selber in einer E-Mail an alle E-Mailadressen, die u.a. in Adressbüchern des befallenen Rechners zu finden waren. Des Weiteren öffnete er eine jpeg-Datei, die übrigens zu allem Überfluss auch noch ein Pornobild enthielt, und startete damit eine weitere Schadfunktion, die versuchte, ein Trojanisches Pferd (vgl. Abschnitt 2.5) aus dem Netz zu laden. War dies erfolgreich, so wurde das Trojaner-Programm gestartet, das u.a. einen Key-logger (*key-logger.dll*) auf dem befallenen Rechner installierte, um die Passwort-Eingaben von Benutzern abzufangen. Weiterhin spähte das Trojaner-Programm Registry-Einträge aus, um Informationen über die System-Konfiguration des befallenen Rechners zu sammeln. Über eine TCP-Verbindung, die der Trojaner zu der Seite *geocities.com* in periodischen Abständen aufbaute, erhielt er eine vom Angreifer vorab präparierte URL, an die die gesammelten Informationen (u.a. IP-Adresse des befallenen Rechners, Benutzerkennung(en) und Passwort(e)) übergeben wurden. Außerdem versuchte der Trojaner den Port 1180 auf dem befallenen Rechner zu öffnen, wodurch dem Angreifer ein entfernter Zugriff auf den Rechner ermöglicht wurde.

Trojaner

Fazit

Zusammenfassend ist festzuhalten, dass der Sobig-Virus viele klassische Charakteristika aufweist. Seine schnelle und große Verbreitung erfolgte über E-Mail-Attachments, die von sorglosen Benutzern gutgläubig geöffnet wurden (Social Engineering), da die Mail angeblich von einer bekannten E-Mail-Adresse stammte, eine unverdächtige Subjekt-Zeile aufwies und der Mail-Körper auf Details im Anhang verwies und damit die Neugierde des Empfängers weckte. Das Ablegen von Einträgen in der Windows-Registry, um ein automatisches Starten des Virus beim Booten zu erreichen, wie auch die Installation von Trojaner-Programmen auf den befallenen Rechnern, zählen zu den üblichen Schadensroutinen heutiger Viren.

▲

2.3.3 Gegenmaßnahmen

Wir können die Menge der möglichen Gegenmaßnahmen grob in zwei Klasse einteilen, nämlich der präventiven Maßnahmen zur Virenabwehr der Werkzeuge zur Erkennung eines Virenbefalls.

Präventive Maßnahmen

Zu dieser Klasse von Abwehrmaßnahmen zählen wir das Beschränken von Schreibberechtigungen, das Verschlüsseln von Daten, die Verwendung digitaler Fingerabdrücke sowie administrative Maßnahmen zur Abwehr von Makro-Viren. Auf diese Bereiche wird im Folgenden kurz eingegangen.

Da heutzutage nahezu alle im Einsatz befindlichen Rechner an das Internet angebunden sind, ist jedes dieser Geräte potentiell gefährdet und erfordert den Einsatz von Überwachungswerkzeugen und Viren-Scannern. Die beschränkte Vergabe von Schreibrechten an Programme ist eine wirksame Abwehrmaßnahme gegen Programmviren, da dadurch das Modifizieren bestehender Programme erschwert und damit die Verbreitung von Viren eingedämmt wird. So benötigen zum Beispiel viele Spielprogramme höchstens Schreibzugriffe auf temporär anzulegende Dateien, so dass auch nur für das `TMP`-Verzeichnis entsprechende Berechtigungen erteilt werden müssen.

Rechtebeschränkung

Durch die Verschlüsselung von gespeicherten Programmen kann man verhindern, dass ein Virus gezielt in der in Abbildung 2.5 skizzierten Weise in ein Programm integriert wird. Ein einfaches Überschreiben des verschlüsselten Codes durch einen Virus führt nach der Entschlüsselung zu nicht ausführbarem Code. Das Überschreiben zerstört zwar die Quelldatei, aber diese Vorgehensweise unterbindet die Viren-Verbreitung sowie die Ausführung von Schadensteilen von Viren. Die Verschlüsselung von Dateien wird zurzeit bereits durch eine Reihe von Produkten wie dem Encrypting File System (EFS) als Bestandteil von Windows (vgl. Abschnitt 13.2.5) unterstützt.

Verschlüsseln

Eine weitere Möglichkeit, durch präventive Maßnahmen eine Vireninfektion vor Ausführung eines Programms erkennen zu können, besteht darin, einen digitalen Fingerabdruck des Programms (vgl. Abschnitt 8.1) zu berechnen. Ein digitaler Fingerabdruck (engl. *digest*) ist ein Bitstring fester Länge, der das Programm eindeutig beschreibt. Diese Bitstrings sind schreibgeschützt zu verwalten, so dass vor der Ausführung eines Programms dessen Fingerabdruck zunächst erneut berechnet und mit dem ursprünglichen Wert verglichen werden kann. Da der Fingerabdruck einen Programmcode eindeutig beschreibt, führt die Veränderung des Codes nach der Integration von Virus-Code zu einem neuen Bitstring als Fingerabdruck des modifizierten Programms. Diese Abweichung zwischen dem ursprünglichen und dem aktuellen Fingerabdruck ist vor einer Programmausführung erkennbar. Auf diese Weise kann die Verbreitung des Virus unterbunden, der erstmalige Befall jedoch nicht verhindert werden. Zur Behebung eines Virenbefalls

Hashfunktion

ist eine gesicherte Version von einem Backup-Medium wiedereinzuspielen, wobei wiederum die digitalen Fingerabdrücke überprüft werden müssen, um sicherzustellen, dass keine infizierte Version eingespielt wird.

Quarantäne

Zur Abwehr von Makro-Viren ist zu verhindern, dass diese beim Lesen eines Dokumentes durch die zum Lesen verwendeten Werkzeuge (u.a. Word-Prozessor, Postscript-Interpreter) automatisch ausgeführt werden. Da eine automatische Meldung, ob ein zu ladendes Dokument Makros enthält, nicht immer erfolgt, sollte in Umgebungen, die keine automatisch ablaufenden Makros erfordern, die Ausführung solcher Makros unterbunden werden.

Isolierung

Die Kontrolle von erweiterten E-Mails oder anderer über das Netz ausgetauschter Dokumente ist nicht immer möglich. Mails bzw. Dokumente von unbekannten Absendern sollten zunächst unter Nutzung von Viren-Scannern überprüft und im Zweifelsfall ungelesen gelöscht oder nur in einer isolierten Ausführungsumgebung gelesen werden. So stehen beispielsweise unter Windows zur automatischen Überprüfung von Dokumenten Suchprogramme zur Verfügung, die periodisch im Hintergrund ablaufen und somit die laufende Anwendung kaum behindern.

Werkzeuge zum Antivirenmanagement

Scanner

Die am häufigsten eingesetzten Werkzeuge gegen Viren sind die Viren-Scanner. Bekannte Viren besitzen eine Viren-Kennung oder enthalten spezifische Bytemuster bzw. Codesequenzen, woran sie erkannt werden können. Viren-Scanner verwalten umfangreiche Datenbanken mit Kennungen bekannter Viren sowie mit Regeln, die u.a. festlegen, welche Dateien bzw. Dateitypen (z.B. alle ausführbaren Programme, u.a. .exe oder .com-Dateien) auf Viren zu überprüfen sind. Bei einem Viren-Scan werden die gespeicherten Dateien auf das Vorhandensein von Viren-Kennungen untersucht und es werden die infizierten Objekte sowie die gefunden Viren angezeigt. Auf diese Weise können natürlich nur bereits bekannte Viren erkannt werden. Mutierende Viren oder neue Viren schlüpfen in der Regel durch das Netz dieser Scanner.

Heuristik

Aus diesem Grund versuchen einige Erkennungsprogramme anhand von Heuristiken auch unbekannte Viren bzw. Abwandlungen bekannter Viren zu entdecken. Die Heuristiken suchen nach verdächtigen Codesequenzen in Programmen, die auf Virenaktivitäten hindeuten, wie zum Beispiel Befehle, die dem Suchen nach neuen Opfern und dem Kopieren/Replizieren des Virus entsprechen. Die anhaltenden Probleme mit Viren-verseuchten Rechnern verdeutlichen jedoch, dass diese Heuristiken unzureichend sind, da die Anzahl der möglichen Veränderungen existierender Viren sehr groß ist und durch Heuristiken unmöglich alle Mutationen bekannter Viren abgedeckt werden können. In Anbetracht der rasanten Entwicklung und Verbreitung

neuer Viren ist deshalb auf jeden Fall eine regelmäßige Aktualisierung der Datenbanken mit Viren-Kennungen unerlässlich.

Aktivitätskontrolle

Ein anderer Ansatz zur Bekämpfung von Viren wird durch Werkzeuge verfolgt, die eine Aktivitätskontrolle durchführen. Dazu werden Programme während ihrer Ausführung überwacht und auf Verhaltensweisen untersucht, die für einen Virenbefall typisch sind. Verdächtige Aktionen umfassen den wiederholten, modifizierenden Zugriff auf ausführbare Dateien, das Suchen nach ausführbaren Dateien oder Versuche, direkt auf externe Speichermedien zuzugreifen. Derartige Tools können als sehr einfache Varianten von Einbruchserkennungs-Systemen (engl. *intrusion detection system IDS*) aufgefasst werden.

Monitoring

Als dritte Klasse von Werkzeugen sind die Monitoring-Tools zu nennen, die Dateien (im Wesentlichen handelt es sich dabei um wichtige Systemdateien) überwachen und durchgeführte Modifikationen anhand geänderter Strukturinformationen oder anhand von Abweichungen bei berechneten digitalen Fingerabdrücken erkennen.

Haftung

Auf Maßnahmen zur Abwehr von Viren wird auch in Zukunft ein Augenmerk bei der Sicherheit von IT-Systemen liegen müssen. Solange jedoch Softwarehersteller und Benutzer nicht in die Pflicht dafür genommen werden können, dass sie ihre Software mit mangelhaften Sicherheitsniveaus ausstatten bzw. ihre Systeme unsicher vorkonfigurieren und fahrlässig mit potentiellen Schadprogrammen umgehen, werden weltweite Virenvorfälle immer wieder auftreten. Erste Ansätze, unachtsamen Anwendern, von deren PCs sich Viren verbreitet haben, mit Klagen auf Schadensersatz wegen der für die Beseitigung der Schäden verursachten Kosten zu drohen, haben aber zurzeit wohl noch keine sehr große Aussicht auf Erfolg. Trotzdem erscheint das Vorgehen, über Fahrlässigkeitsklagen Haftungsansprüche durchzusetzen und Schaden sowie Verantwortung zu teilen, sehr sinnvoll, um in Zukunft von Unternehmen und Einzelpersonen eine größere Verantwortung gegenüber Dritten erwarten zu können. Eine allgemeine rechtliche Verpflichtung zur Vorsorge mit entsprechenden Haftungsansprüchen erscheint angesichts der großen Verbreitung von Viren durchaus möglich. Beim heutigen Kenntnisstand über die Virenproblematik muss es deshalb schon fast als Fahrlässigkeit angesehen werden, wenn auf einen Virenschutz verzichtet wird.

2.4 Würmer

Im Gegensatz zu einem Virus ist ein Wurm ein eigenständiges, ablauffähiges Programm. Das bedeutet, dass ein Wurm kein Wirtsprogramm zur Ausführung benötigt.

Definition 2.2 (Wurm)

Wurm

Ein Wurm ist ein ablauffähiges Programm mit der Fähigkeit zur Reproduktion. Ein Wurm-Programm besteht in der Regel aus mehreren Programmteilen, den Wurm-Segmenten. Die Vervielfältigung erfolgt selbstständig meist unter Kommunikation mit anderen Wurm-Segmenten.

□

Verbreitung

Die Verbreitung von Würmern erfolgt insbesondere über ein Netzwerk, indem sich der Wurm auf andere Rechner innerhalb des Netzes kopiert. Ausgangspunkt für einen Wurmangriff sind häufig Systemprozesse, die ständig rechenbereit sind oder in regelmäßigen Abständen aktiviert werden. Würmer bedienen sich dieser Prozesse, um beispielsweise über eine Buffer-Overflow-Attacke in das Opfersystem zu gelangen. So nutzte im Jahr 2001 der Code Red Wurm einen Pufferüberlauf im Microsoft Internet Information Service (IIS) aus, um sich zu verbreiten. Die ausführbaren Programmteile eines Wurms können den Quellcode eines Programms beinhalten, der auf dem zu befallenden Rechner übersetzt werden muss, sie können aber auch direkt in ausführbarer Maschinensprache oder in interpretierbarer Sprache, z.B. Shell-Kommandos, geschrieben sein. Bei einem Pufferüberlauf-Angriff auf einen Systemprozess werden diese Befehle dann sogar im privilegierten Modus des Betriebssystemkerns ausgeführt, d.h. der Angreifer erlangt unbeschränkte Zugriffsberechtigungen.

Bedrohungen

Durch Würmer treten ebenso wie durch Viren Bedrohungen der Integrität und Vertraulichkeit, aber auch Bedrohungen der Verfügbarkeit auf, die so genannten Denial-of-Service Angriffe. Würmer beanspruchen in der Regel viele Ressourcen, da sie bei ihrer Verbreitung über das Netzwerk häufig eine sehr hohe Netzlast erzeugen und durch einen möglichen Mehrfachbefall von Rechnern deren Speicherressourcen ausschöpfen können.

Die ersten Wurm-Programme wurden 1979 im Xerox Palo Alto Research Center als sinnvolle und nicht bedrohliche Anwendungen entwickelt, um eine verteilte Berechnung wie zum Beispiel das Komprimieren von Daten durchzuführen. Einer der ersten und bekanntesten Würmer mit einer bedrohlichen Funktionalität war der so genannte Internet-Wurm.

Beispiel 2.2 (Internet-Wurm)

Internet-Wurm

Am Abend des 2. November 1988 wurde der Internet-Wurm (u.a. RFC1135) gestartet. Der Wurm verbreitete sich im Verlauf der Nacht über das Internet, und als er am nächsten Morgen gestoppt wurde, hatte er bereits 6000 Rechner befallen.

Der Wurm nutzte einige bekannte Fehler und Eigenheiten von Unix aus, die mittlerweile in den meisten Systemen behoben sein sollten. Zu den allgemeinen, auch in heutigen Systemen noch vorhandenen Schwachstellen, die indirekt zur Verbreitung des Wurms beitrugen, zählt, dass Dienstprogramme und ihre Hintergrundprozesse keinem bestimmten Benutzer zugewiesen werden, so dass sich ein Angreifer ihrer bedienen kann.

genutzte Schwachstellen

Angriff

Das Ziel des Internet-Wurms war es, auf einem fremden Ziel-Rechner eine Shell zu starten, um damit das Wurmprogramm auf diesem Rechner zu laden, dort zu übersetzen und auszuführen. Um eine solche Shell zu starten, wurden drei verschiedene Angriffsversuche durchgeführt: (1) Mit einem „Brute force“ Angriff wurde versucht, Passworte auf dem Ziel-Rechner zu knacken (Remote Shell Angriff).

remote shell

(2) Falls der Ziel-Rechner eine bekannte Sicherheitslücke im `sendmail` Programm enthielt, wurde diese ausgenutzt, um ein Programm zu senden, das auf dem Zielrechner ausgeführt wurde, um eine Shell zu starten. Die Entwickler des `sendmail` Kommandos hatten zum Debuggen einen speziellen Debug-Modus eingeführt, durch den eine Nachricht zu einem Programm und nicht zum Mailer gesendet wurde. Falls der Ziel-Rechner es ermöglichte, diesen Modus einzustellen, wurde er vom Wurm durch das Versenden einer dediziert konstruierten Befehlsfolge ausgenutzt. Die Ausführung dieser Befehlssequenz bewirkte, dass auf dem Ziel-Rechner eine Shell gestartet wurde, mit der das Wurmprogramm übersetzt und ausgeführt wurde. Da mit dem Debug-Modus Superuser-Rechte verknüpft waren, konnte sogar eine Shell mit Root-Rechten erzeugt werden.

sendmail

(3) Als Drittes wurde versucht, einen Implementierungsfehler im `fingered` Programm auszunutzen. Mit diesem Programm kann man Informationen über Benutzer eines Systems erfragen. Das `fingered` Programm nutzt die Systemroutine `gets()`, die eine Eingabe in einen String kopiert. In alten Versionen dieses Programms fehlte eine Bereichsprüfung, so dass mit einer Buffer-Overflow Attacke der Puffer der `gets`-Routine durch einen zu langen Eingabestring zum Überlaufen gebracht werden konnte. Der Wurm nutzte diese Lücke aus, indem er gezielt einen langen Eingabestring erzeugte und Maschinenbefehle auf dem Systemkeller ablegte, durch deren Ausführung eine Shell auf dem befallenen Rechner gestartet wurde. Damit konnte der Angreifer auf dem fremden Rechner Kommandos ausführen.

fingered

▲

Als Reaktion auf den Wurm gründete die U.S. Defense Advanced Research Projects Agency (DARPA) das erste Computer Emergency Response Team

CERT

(CERT), das die Aufgabe hat, sich mit Sicherheitsfragen rund um das Internet zu beschäftigen.

ILOVEYOU

Im Mai 2000 sorgte der ILOVEYOU-Wurm für einiges Aufsehen, da er sich in einer bis dahin nicht bekannten Geschwindigkeit verbreitete und zum Zusammenbruch der elektronischen Datenverarbeitung in vielen europäischen und amerikanischen Firmen und Behörden führte. Man schätzt den verursachten Schaden durch Datenverluste und Produktivitätsausfall weltweit auf über 8.76 Millionen US Dollar.

Beispiel 2.3 (ILOVEYOU)

Bei dem ILOVEYOU-Wurm handelte es sich um einen Wurm, der als .vbs-Datei (Visual Basic Script) über E-Mail Attachments verbreitet wurde, in deren Betreff-Zeile ILOVEYOU stand. Die Verbreitung des Wurms setzte eine Microsoft Windows-Umgebung voraus, da der Wurm sich der Daten von Windows Outlook bediente, um eine Mail mit dem Wurm als Attachment an die in der Adressdatei gespeicherten Mailadressen zu senden. Der Wurm zerstörte gezielt Dateien des lokalen Dateisystems, die u.a. JPEG-Bilder, MP2- oder MP3 Musik-Daten oder Videodateien enthielten. Schließlich durchsuchte der Wurm auch die lokale Festplatte nach Passworten und versuchte, eine Verbindung aufzubauen, um diese Daten an den Programmierer des Wurms zu übermitteln.
▲

Nahezu wöchentlich treten neue Wurm-Varianten in Erscheinung, die häufig nach einem ähnlichen Angriffsschema ablaufen. Als Beispiel hierzu wird abschließend kurz auf den Blaster- bzw. Lovesan-Wurm eingegangen.

Beispiel 2.4 (Lovesan- bzw. Blaster-Wurm)

Lovesan

Im Sommer 2003 verbreitete sich der so genannte Lovesan-Wurm[11] äußerst rasant und richtete beträchtliche Schäden an. Das Besondere an diesem Wurm war, dass er nicht nur Server-Rechner sondern insbesondere auch Heim-PCs zum Angriffsziel hatte. Sein Angriffsvorgehen basiert jedoch wiederum auf Standardangriffstechniken, die im Folgenden kurz erläutert werden.

Angriffsziel

Das eigentliche Angriffsziel war der Server *windowsupdate.com*, der Patches für Windows-Systeme bereitstellt, die von Windows-Benutzern via Windows-Update auf den eigenen Rechner herunter geladen werden können. Der Blaster-Wurm verursachte einen Distributed Denial of Service (DDoS) Angriff auf diese Web-Site. Mit diesem Angriff wurden systematisch Opfer-Systeme vorbereitet, die nach dem Stichtag 16.08.2003 SYN-Anfragen an

[11] auch unter dem Namen Blaster-Wurm bekannt

den Port 80 des Update-Servers senden sollten. Es handelte sich hierbei um eine SYN-Flood Attacke mit gespooften Absenderinformationen, wie sie auf der Seite 122 beschrieben wird. Die gespooften Absenderadressen waren hier zufällig gewählte IP-Adressen eines Klasse B Netzes. Durch den DDoS-Angriff sollte der Update-Server so überlastet werden, dass die Benutzer keine Patches mehr herunterladen konnten. Um dies zu vermeiden hatte Microsoft, nach bekanntwerden des drohenden DDoS-Angriffs vorsorglich den DNS-Eintrag *windowsupdate.com* entfernen lassen.

Buffer-Overflow

Zur Angriffsvorbereitung nutzte der Wurm eine bekannte Schwachstelle aus, nämlich eine Buffer-Overflow Verwundbarkeit (vgl. Abschnitt 2.2) im DCOM RPC-Dienst an TCP-Port 135 unter dem Windows-Betriebssystem. Um Systeme zu finden, die einen derartigen Angriffspunkt aufweisen, scannte der Wurm entweder das lokale Klasse C-Subnetz oder wählte einen beliebigen IP-Adressbereich aus, um in diesem Adressbereich Systeme mit geöffnetem Port 135 (oder auch Port 139, 445 oder 593) zu identifizieren. An den Port 135 eines derart als mögliches Opfer identifizierten Rechners sendete er sodann ein präpariertes TCP-Paket. Ausgenutzt wurde dabei, dass TCP-Pakete, die an Port 135 empfangen werden, ungeprüft an den DCOM-RPC Dienst weitergeleitet wurden. Das präparierte Datenpaket verursachte einen Buffer-Overflow, wodurch ausführbarer Code auf das Opfersystem gelangte und gleichzeitig den RPC-Dienst zum Absturz brachte sowie einen Reboot des Systems verursachte.

remote Shell

Der gezielte Buffer-Overflow Angriff gewährte dem Angreifer Superuser-Rechte bei der Ausführung des eingeschleusten Codes. Diese wurden genutzt, um eine entfernte Shell (remote Shell) an TCP-Port 444 des Opfer-Rechners zu starten. Über diese war es dann in weiteren Schritten möglich, eine TFTP-Verbindung über den UDP-Port 69 zum Ausgangsrechner der Wurm-Attacke aufzubauen sowie die Datei *msblast.exe* in das Windows-Systemverzeichnis des Opferrechners herunter zu laden und zu starten. Mit der Ausführung von *msblast.exe* wurde ein Windows-Registry-Eintrag beispielsweise der folgenden Art

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
\CurrentVersion\Run
windows auto update= msblast.exe I just want to say
                                  LOVE YOU SAN!! bill
```

erzeugt[12]. Der Blaster-Code prüfte das aktuelle Datum des Opfersystems und versendete beim Erreichen bzw. Überschreiten des Datums 16.08.03 die oben bereits angesprochenen 40 Byte SYN-Pakete mit einer gefälschten Absender-IP-Adresse an den Rechner *windowsupdate.com*.

▲

[12] Daher rührt der Name Lovesan-Wurm.

Fazit

Der Lovesan- bzw. Blaster-Wurm ist ein typisches Beispiel eines Wurmes, der einen Programmierfehler in einem Betriebssystemdienst ausnutzt (Buffer-Overflow), um mit Superuser-Privilegien einen Kommandointerpreter (Shell) auf dem Opfersystem zu starten, TCP- und/oder UDP-Verbindungen aufzubauen und darüber beliebigen weiteren Code auf das Opfersystem herunter zuladen. In diesem speziellen Fall wurden die Opferrechner „nur“ für einen DDoS-Angriff instrumentalisiert, aber natürlich könnten sich auf diesem Weg genauso auch Angriffe auf die Integrität und Vertraulichkeit der Daten auf dem Opferrechner selber ergeben. Einmal mehr zeigt das Beispiel, dass eine sorglose Konfiguration mit z.B. zu vielen geöffneten Ports leicht auszunutzende Angriffsflächen bietet.

Das erfolgreiche Wirken von Würmern wie dem Blaster-Wurm wird durch Design- und Implementierungsfehler in Systemdiensten ermöglicht. Davon sind Open Source Betriebssysteme ebenso betroffen wie proprietäre Betriebssysteme, wie die Systeme der Windows-Familie. Letztere stehen aber stärker im Visier, da sie viel größere Verbreitung besitzen, so dass die Betriebssystem-Monokultur auch die Ausbreitung von Würmern fördert. Mangelnde Sicherheitsmaßnahmen in diesen Systemen eröffnet Programmen nahezu unbeschränkte Zugriffe auf Daten der lokalen Festplatte. Daneben verdeutlichen die rasante Verbreitung von Würmern und die sehr hohe Wiederholungsrate (Auftreten von Wurmvarianten) erneut die mangelnden Kenntnisse und das mangelnde Sicherheitsbewusstsein heutiger Benutzer.

Gegenmaßnahmen

Patch

Im Gegensatz zu Viren, die sich meist auf „legalem“ Weg verbreiten, versuchen Würmer Bugs und Lücken in System- und Anwendungssoftware auszunutzen. Da auf den einschlägigen WWW-Seiten (siehe u.a. http://www.securityfocus.com) und Mailinglisten sehr schnell aktuelle Informationen über ausgenutzte Sicherheitsschwachstellen veröffentlicht und auch Patches zum Beheben der Schwachstellen angeboten werden, sollte sich jeder zuständige Systemadministrator hierüber informieren und seine Systeme absichern. Die Lücken in den `sendmail` bzw. `fingered` Programmen waren beispielsweise schon lange bekannt, bevor der Internet-Wurm sie ausnutzte. Aktuelle Informationen über Lücken und Problembereiche werden regelmäßig von den CERTs veröffentlicht.

Vielfach lassen sich mögliche Einfallstore für Würmer bereits durch eine geeignete Konfigurierung mit einem eingeschränkten Diensteangebot (z.B. Schließen von nicht benötigten Ports) verkleinern oder schließen.

minimale Rechte

Durch eine restriktive Vergabe von Zugriffsberechtigungen, insbesondere auch von Leseberechtigungen, lässt sich ein unerlaubtes Akquirieren von Informationen und das Einbringen von fremdem Code beschränken. Beson-

ders Passwortdaten sind so zu schützen, dass nicht jeder Benutzer Leserecht darauf erhält. Generell sind differenzierte und restriktive Rechtefestlegungen und Kontrollen notwendig, insbesondere bei Zugriffen von entfernten Rechnern. Die fehlende differenzierte Zugriffskontrolle in den Windows Betriebssystemen (Windows 95, 98) hat ursächlich dazu beigetragen, dass sich der *ILOVEYOU*-Wurm in einer bis dahin noch nicht bekannten Geschwindigkeit ausbreiten konnte.

2.5 Trojanisches Pferd

Sage

Der Begriff des Trojanischen Pferdes geht zurück auf die Sage vom Kampf um die Stadt Troja, in der die Griechen nach 10-jährigem Kampf als Zeichen des Rückzuges den Trojanern ein großes, hölzernes Pferd zum Geschenk vor die Tore der belagerten Stadt stellten. Die siegestrunkenen Trojaner zogen das Pferd in das Innere der Stadtmauern, um sich an ihrem Geschenk zu erfreuen und um ihren vermeintlichen Sieg zu feiern. Im Verlauf der Nacht offenbarte das Pferd sein verstecktes Geheimnis; es verbarg griechische Soldaten in seinem Inneren, die die Tore der Stadt öffneten, so dass sie von den Griechen eingenommen werden konnte.

2.5.1 Eigenschaften

Das Trojanische Pferd in der Sage beschreibt die Charakteristika von Programmen, die als Trojanische Pferde bezeichnet werden, sehr genau: Es wird eine Funktionalität vorgetäuscht, die Vertrauen erweckt, die aber durch eine verborgene Schadens-Funktionalität ergänzt wird.

Definition 2.3 (Trojanisches Pferd)

Trojanisches Pferd

Ein Trojanisches Pferd (auch Trojaner genannt) ist ein Programm, dessen implementierte Ist-Funktionalität nicht mit der angegebenen Soll-Funktionalität übereinstimmt. Es erfüllt zwar diese Soll-Funktionalität, besitzt jedoch eine darüber hinausgehende, beabsichtigte zusätzliche, verborgene Funktionalität.

□

Unter die Definition 2.3 fallen somit keine Programme, die infolge von Programmierfehlern ein von der Soll-Funktionalität abweichendes Verhalten zeigen, sondern nur solche, für die diese Erweiterungen absichtlich integriert wurden. Ein Trojanisches Pferd besitzt verborgene Eigenschaften, um z.B. in ein System einzudringen, um Daten aufzuzeichnen oder zu manipulieren. Auf Benutzerebene wird dabei korrektes Verhalten vorgetäuscht. Trojanische Pferde können mit dem Programmstart aktiviert oder durch spezielle Aus-

logische Bombe

löser, wie beispielsweise das Eintreten eines bestimmten Datums, gestartet werden. Man spricht hierbei auch häufig von einer logischen Bombe.

Trojanische Pferde können in ganz unterschiedlichen Bereichen eines Systems auftreten. Beispiele dafür sind Editoren oder Textverarbeitungsprogramme, die die Inhalte edierter Dateien unbemerkt und unautorisiert kopieren. Weitere Beispiele sind manipulierte Datenbanken, durch die sensible Informationen zum Angreifer durchsickern können, oder auch manipulierte Betriebssystemfunktionen, durch die der Angreifer beispielsweise zusätzliche Zugriffsrechte erlangt.

Beispiel 2.5 (Trojanische Pferde)

Zinsberechnung

Eines der ersten bekannt gewordenen Trojanischen Pferde trat in einer Bank auf. Ein Angestellter hatte den Auftrag, ein Programm zu schreiben, das Zinsabrechnungen auf drei Stellen genau durchführt. Der Programmierer erfüllte diese Aufgabe und zusätzlich sorgte er dafür, dass Restbeträge, die beim Abrunden anfielen, auf sein Konto gutgeschrieben wurden. Innerhalb kürzester Zeit sammelte er dadurch einen großen Geldbetrag an.

CAD-Demo

Ein weiteres bekanntes Trojanisches Pferd, das 1992 für einiges Aufsehen gesorgt hatte, stammte von der Firma CadSoft, einer Firma, die CAD-Software entwickelt. Diese Firma verteilte eine Demoversion ihres Produkts. Bei der Ausführung dieses Demoprogramms wurde auch ein Formular erzeugt, das man zur Bestellung eines Handbuches verwenden konnte. Die zusätzliche, verborgene Funktionalität des Demoprogramms bestand darin, dass die Festplatte des Benutzer-PCs nach Programmen der Firma CadSoft durchsucht wurde und auf dem Bestellformular Art und Anzahl der gefundenen Programme codiert wurden. Da sich Benutzer, die im Besitz von Raubkopien der CAD-Software waren, bevorzugt das Handbuch zuschicken ließen, gelang es, viele widerrechtliche Benutzer zu entlarven. Anzumerken bleibt jedoch, dass das Ausschnüffeln der Festplatte von Benutzern mittels Trojanern rechtlich nicht zulässig ist und im vorliegenden Fall dann auch entsprechend geahndet wurde.

▲

Trend

Heutige Trojanische Pferd-Programme bieten den Angreifern eine Vielzahl von Möglichkeiten, den befallenen Rechner zu kontrollieren, gespeicherte Daten auszuspähen, oder aber auch Tastatureingaben, wie zum Beispiel Passwörter, sowie Bildschirmausgaben aufzuzeichnen und über eine Netzwerkverbindung zu versenden. Häufig laden sie automatisch Updates und weiteren Schadcode aus dem Internet auf den befallenen Rechner herunter.

Aus dem Lagebericht des BSI zur IT-Sicherheit in Deutschland[13] ist zu entnehmen, dass bereits seit 2007 der Anteil der Trojanischen Pferde unter den Schadprogrammen den Anteil der Viren und Würmer deutlich überholt hat.

2.5.2 Gegenmaßnahmen

minimale Rechte

Sensible Daten, wie Passworte, PINs und TANs sollten wenn möglich auf sicheren externen Medien, wie einer Smartcard oder einem verschlüsselten USB-Stick abgelegt werden. Ist eine Speicherung auf der Festplatte unumgänglich, so sind die Daten unbedingt verschlüsselt abzulegen, da ein Zugriffsschutz über das Betriebssystem nur einen unzureichenden Schutz gewährt. Ist ein Angreifer im physischen Besitz einer Festplatte, z.B. durch Diebstahl eines Notebooks, so ist es sehr leicht, das installierte Betriebssystem durch Booten eines unsicheren Betriebssystems zu umgehen und direkt auf den Speicher zuzugreifen. Werden dennoch sensible Daten gespeichert, so sind starke kryptografische Verfahren zu deren Verschlüsselung einzusetzen und der Zugriff auf die verschlüsselten Daten ist zu beschränken. Zugriffsrechte für Quellcode-Daten sollten nur besonders privilegierte Benutzer erhalten, um das gezielte Einschleusen von Trojanischen Pferden zu beschränken.

Zur Abwehr der besprochenen Bedrohungen ist es wichtig, die Rechte von Benutzern zur Ausführung von Betriebssystemdiensten so zu beschränken, dass jeder Benutzer nur diejenigen Dienste nutzen darf, die er zur Erledigung seiner Aufgaben benötigt (Prinzip der minimalen Rechte). Die Basis dafür ist eine Charakterisierung und Klassifizierung von Betriebssystemdiensten, die für manipulatorische Zwecke missbraucht werden können. Beispiele für solche Betriebssystemfunktionen sind:

kritische Dienste

- Befehle zur Änderung von Schutzattributen an Dateien,
- Funktionen zum Lesen und Bearbeiten von Passwortdateien,
- Anweisungen, um Netzverbindungen zu anderen Rechnern zu öffnen, und
- Anweisungen, die einen direkten Zugriff auf Speicherbereiche zulassen.

Modellierung

Beschränkt man die Zugriffsrechte eines Benutzers, so sind auch seine Manipulationsmöglichkeiten entsprechend beschränkt. Um Zugriffsrechte nach dem Prinzip der minimalen Rechte systematisch vergeben zu können, sind genaue Kenntnisse über die Funktionalität der einzelnen Systemdienste erforderlich. Man muss die Aufgabenprofile der einzelnen Benutzer präzise

[13] Den jeweils aktuellen Lagebericht des BSI findet man unter https://www.bsi.bund.de/DE/Publikationen/Lageberichte/lageberichte_node.html

erfassen können und die Zusammenhänge kennen, die zwischen den Systemkomponenten bestehen. Des Weiteren ist es notwendig, die Auswirkungen von Rechtewahrnehmungen zu kennen. Da die betrachteten Systeme, insbesondere Betriebssysteme, komplexe Programme sind, kann man einen entsprechenden Überblick nur dadurch erhalten, dass man das System auf einer geeigneten Abstraktionsebene durch ein Modell beschreibt und die Funktionalität der Systembausteine semi-formal oder formal spezifiziert und verifiziert. Das Modell muss die zu schützenden Objekte und die agierenden Subjekte geeignet erfassen, Festlegungen für die in dem System zu vergebenden Zugriffsrechte treffen sowie die Sicherheitsstrategie des Systems spezifizieren. Auf der Basis des Sicherheitsmodells können Sicherheitseigenschaften überprüft werden. Die bekanntesten und wichtigsten Klassen von Sicherheitsmodellen werden in Kapitel 6 vorgestellt.

Signieren

Weiterhin kann man Programme mit der digitalen Unterschrift des Erzeugers versehen und vor der Programmausführung die Korrektheit der Signatur überprüfen. Die digitale Unterschrift sollte für den eindeutigen, digitalen Fingerabdruck, also einen kryptografischen Hashwert des Programms erstellt werden, so dass auf dieser Basis auch die Unveränderlichkeit des Codes überprüft werden kann (vgl. Kapitel 8). Signierter Code bietet natürlich keine Gewähr vor Trojanischen Pferden. Die Vorgehensweise basiert darauf, dass der Empfänger des signierten Codes dem Signierer vertraut. Das heißt, dass der Empfänger darauf vertraut, dass der Signierer Code ohne Schadsoftware-Anteile erzeugt; eine Verifikation, ob dies tatsächlich der Fall ist, findet beim Validieren der Signatur aber nicht statt. Zur Überprüfung von Signaturen benötigt man ein Zertifikat des Signierers und eine Infrastruktur zur Verteilung der Zertifikate, eine so genannte Public-Key Infrastruktur (vgl. Kapitel 9.1). Die Aussagekraft einer digitalen Signatur hängt in hohem Maß von der Vertrauenswürdigkeit der Zertifikate ab.

Trojanische Pferde werden häufig über Viren, Würmer und Buffer-Overflows in ein System eingeschleust. Dabei wird beispielsweise versucht, ein bereits vorhandenes ausführbares Programm bzw. einen Systemdienst durch modifizierten ausführbaren Code zu ersetzen. Eine andere Möglichkeit des Einschleusens besteht darin, den Quellcode vorhandener Dienste bzw. Programme direkt zu manipulieren und durch eine erneute Übersetzung den ausführbaren Objektcode auf der Maschine zu erzeugen. Auf diese Weise vermeidet der Angreifer, dass sein Code aufgrund von Portierungsproblemen auf dem Zielsystem nicht ausführbar ist. Modifikationen des Quellcodes lassen sich durch Quellcodeüberprüfung bzw. Verifikation aufdecken. Dieser Ansatz ist jedoch bei der Komplexität der heutigen Programme und der mangelnden automatischen Unterstützung mit erheblichem Aufwand verbunden und in der Regel nicht praktikabel. Zur Verhinderung von Quellcodemodifi-

Code-Inspektion

kationen ist deshalb dafür zu sorgen, dass der Code nicht direkt zugreifbar im Speicher liegt, also durch Zugriffsrechte und Verschlüsselungen gesichert wird.

2.6 Bot-Netze und Spam

Bot-Netze und Spam sind in heutigen IT-Systemen nahezu allgegenwärtig. Bot-Netze nutzen gezielt verbreitete Schadsoftware, wie Viren, Würmer und Trojaner, um insbesondere verteilte Angriffe unter Mitwirkung einer großen Anzahl von Rechnern durchzuführen, wobei das Bedrohungspotential dieser Angriffe durch die gezielte Koordinierung tausender befallener Rechner erheblicher größer ist, als durch einen infizierten Rechner.

Spam zählt in der Regel nicht direkt zu Schadsoftware, da Spam-Mails meist lediglich unerwünschte Nachrichten enthalten. Da aber zunehmend Trojaner über Massen-Mails wie Spam verbreitet werden, widmet sich ein Unterabschnitt dem Spam-Problem.

2.6.1 Bot-Netze

Bot

Ein Bot-Netz ist ein Verbund von infizierten Rechnern, den so genannten Bot-Rechnern, die miteinander kommunizieren und meist durch einen zentralen Server kontrolliert und ferngesteuert werden. Angreifer missbrauchen hierzu sehr häufig den IRC (Internet Relay Chat, vgl. (RFC 2810)), das Protokolle zum Austausch von Text-Nachrichten in Realzeit, das Chatten, bereit stellt. Das Kunstwort *Bot* ist eine Ableitung von dem Wort Robot, das wiederum vom tschechischen Wort Robota (deutsch *Arbeit*) abstammt. Allgemein ist ein Bot ein Programm, das ohne menschliche Eingriffe Aktionen ausführt. Im Kontext der IT-Sicherheit verstehen wir unter einem Bot ein Programm, das von einem Angreifer gezielt ferngesteuert wird. Bots werden herkömmlicher weise über Schadsoftware wie Viren, Würmer und Trojaner verbreitet. Die in einem Bot-Netz verbundenen Rechner führen ihre Angriffe in der Regel völlig transparent für den Besitzer des jeweiligen Rechners aus.

Angriffe

Bot-Netze können aus mehreren tausend Rechnern bestehen. Hauptangriffsziele sind verteilte Denial-of-Service Angriffe (DDoS) auf Anbieter von Internetdiensten und das massenhafte, über verteilte Absender (die Bots) initiierte Versenden von Spam-Nachrichten. Durch die ferngesteuerte Kontrolle von tausenden manipulierter Rechner steht einem Angreifer eine Bandbreite zur Durchführung eines Angriffs zur Verfügung, die die der meisten herkömmlichen Internetzugänge um Größenordnungen übertrifft. Mit einem ausreichend großem Bot-Netz ist es somit möglich, über das Versenden von großen Datenmengen attackierte Serviceanbieter zu überlasten;

also einen klassischen Denial-of-Service herbei zu führen. Neben diesen Angriffen, für die die Bot-Rechner als Zombies instrumentalisiert werden, können Bot-Programme natürlich auch Angriffe auf die befallenen Bot-Rechner selber durchführen und analog zu klassischen Viren und Trojanern als Key-logger agieren, Netzwerkverbindungen zum Download von Malware aufbauen oder aber auch Viren-Scanner auf dem befallenen Rechner deaktivieren.

IRC

Wie bereits erwähnt, wird nach wie vor zur zentral koordinierten Fernsteuerung eines Bot-Netzes zumeist das IRC verwendet. Abbildung 2.7 veranschaulicht die allgemeine Struktur eines derartigen Bot-Netzes. Nach der Infizierung werden die Bot-Programme aktiviert und verbinden sich mit einem IRC-Server. Der IRC-Server dient als Relaisstation (daher Internet Relay Chat - IRC), damit die Bot-Programme miteinander über einen gemeinsamen Kanal, dem sie explizit beitreten, kommunizieren können.

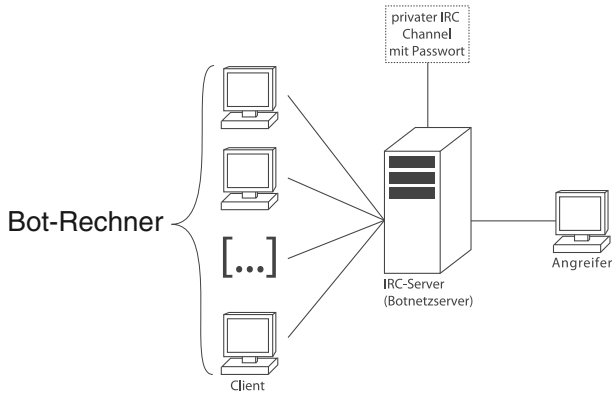

Abbildung 2.7: IRC-kontrolliertes Bot-Netz

Um die Mitgliedschaft in einem Bot-Netz zu kontrollieren, ist ein solcher Kanal zumeist Passwort-geschützt und das jeweilige Passwort wird nur den Bot-Programmen bekannt gegeben. Die Fernsteuerung der Bot-Rechner erfolgt dann durch einen Administrator des IRC-Servers. Beispiele für Kommandos, die ferngesteuert an die Bots weitergeleitet werden, sind die Aufforderung, nach weiteren Systemen, die infiziert werden können, zu scannen, einen DoS-Angriff auszuführen oder aber auch einen Download durchzuführen und das geladene Programm anschließend zu starten.

Abwehr

Da der IRC-Server bei diesen Angriffen die zentrale Kontrolleinheit ist, kann man das Bot-Netz deaktivieren, wenn es gelingt, diesen IRC-Server aus dem Netz zu entfernen. Um zu erkennen, ob ein IRC-Server ein Bot-Netz kontrolliert, wird in der Regel der Datenverkehr überwacht und auf verdächtige Nachrichten hin gescannt. Da zur Fernsteuerung von Bot-Rechnern der

IRC-Server, wie oben beschrieben, Steuerbefehle zur Initiierung von DDoS-Angriffen, zum Download von Malware bzw. zur Durchführung von Scans übermittelt, kann mittels eines Netzwerk Intrusion Detection Systems gezielt nach den Signaturen derartiger Steuerbefehle, also deren syntaktischer Struktur, gesucht werden. Da die Kommunikation in einem IRC-basierten Bot-Netz im Klartext erfolgt, ist eine derartige Überwachung möglich.

Trends

Die nächste Generation der Bot-Netze bedient sich jedoch zunehmend dezentraler Peer-to-Peer (P2P)-Strukturen und verwendet eine verschlüsselte Kommunikation. Dadurch wird sowohl das Erkennen eines Bot-Netzes als auch dessen Deaktivierung erheblich erschwert, da es ja gerade ein Charakteristikum eines dezentralen P2P-Netzes ist, dass es auch nach dem Ausfall eines Peers seine Funktionalität weiter aufrecht erhalten kann und sich selbst-organisierend rekonfiguriert.
Ein weiterer Trend, der auch vom BSI beobachtet wird, ist die Zunahme an Bot-Netzen, die vermehrt zu gezielten, kriminellen Aktivitäten eingesetzt werden. So können Bot-Netze von Kriminellen gemietet werden, um gezielt zum Beispiel einen Service-Provider mittels eines verteilten Denial-of-Service Angriffs zu blockieren und erst nach Zahlung eines Geldbetrages diese Blockade wieder aufzuheben.

2.6.2 Spam

Spam

Neben Viren, Würmern und Trojanern, die erheblichen finanziellen Schaden in heutigen Systemen bewirken können, zählt die zunehmende Flut der Spam-Mails zu einem großen Übel, das Unternehmen erhebliche Ressourcen kostet, um ihrer Herr zu werden. Unter Spam versteht man eine Massen-E-Mail, häufig Werbesendungen, die im Internet verbreitet wird. Diese Werbung wird unaufgefordert an Millionen von E-Mail-Adressen versendet. Die Adressen stammen von Adresshändlern, oder werden durch spezielle Programme im Internet gesucht. Ursprünglich handelt es sich bei Spam um Pressfleisch in Dosen, Spiced Porc And Meat. Der Begriff Spam-Mail geht angeblich auf einen Sketch von Monty Python zurück, in dem ein Restaurantgast nur Gerichte bestellen konnte, wenn er gleichzeitig auch Spam geordert hat. In Übertragung auf das Internet bedeutet das, dass ein Benutzer die Vorzüge des Internets und insbesondere des sehr kostengünstigen Versendens von E-Mails nur nutzen kann, wenn auch bereit ist, Spam-Müll in Kauf zu nehmen.

Schäden

Untersuchungen z.B. von Nucleus Research (USA) gehen davon aus, dass jeder Mitarbeiter in einem Unternehmen im Durchschnitt täglich 13,3 Spam-Mails erhält, für die er 6,5 Minuten benötigt, um diese zu lesen und danach zu löschen. Das summiert sich jährlich auf etwa 25 Arbeitsstunden pro Kopf und macht etwa 1,4 Prozent der gesamten Arbeitszeit aus. Insgesamt

ergeben sich durch die Spam-Mails erhebliche Verluste an Produktivität, die zu sehr hohen gesamtwirtschaftlichen Kosten führen. So kommt 2004 das US-Forschungsinstitut Ferris Research in einer Studie über Spam-Mails zu dem Schluss, dass allein in den USA Einbußen bis zu zehn Milliarden Dollar jährlich aufgrund spamverseuchter E-Mails anfallen. Neben den Kosten, die direkt durch diese Mails verursacht werden, müssen auch die Kosten betrachtet werden, die für die Entwicklung und Aufrechterhaltung von Antispamsystemen aufzubringen sind. Ferris Research schätzt, dass US-Unternehmen allein im Jahr 2004 dafür bis zu 120 Millionen Dollar aufbringen mussten.

Abwehr

Zur Abwehr der Spam-Flut werden Spam-Filterungen eingesetzt. Zur Filterung von Spam stehen im Internet frei verfügbare Filterprogramme zur Verfügung, wie beispielsweise SpamProbe (vgl. http://spamprobe.sourceforge.net/) für Unix-Derivate oder auch für Windows-Systeme unter der Cygwin-Umgebung. In Unternehmen erfolgt die Spam-Filterung häufig über zentral administrierte Mailrelays, die neben einer Spam-Filterung auch in der Regel eine Virenerkennung durchführen. Neben der Spam-Erkennung durch Spam-Filterungsprogramme muss über Regelwerke festgelegt werden, wie mit Mails, die unter Spam-Verdacht stehen, umgegangen wird. Häufig beschränken sich die Filter darauf, in der Subject-Zeile der Mail diese als mögliche Spam zu markieren und ggf. nicht an den Empfänger weiterzuleiten, sondern sie in einer Spam-Quarantäne-Datei für den Empfänger zu sammeln und regelmäßig, z.B. einmal wöchentlich, dem Empfänger eine Mail zu schicken, in der die Subject-Zeilen und Absenderadressen aller als Spam eingestufter Mails zusammengefasst werden. Manche Unternehmen gehen jedoch auch soweit, Spam-Mails direkt zu löschen.

Rechtslage

Beide Vorgehen, also das Ändern der Subject-Zeile und das Löschen sind jedoch rechtlich bedenklich. Gemäß §303a des Strafgesetzbuches (StGB) macht sich jemand strafbar, der rechtswidrig Daten löscht, unterdrückt, unbrauchbar macht oder verändert. Ob die Subject-Zeile als geschützter Bestandteil der Mail anzusehen ist, wird in juristischen Kreisen noch stark diskutiert. Die DFN Forschungsstelle Recht in Münster rät davon ab, Subject-Zeilen zu manipulieren. Das Löschen von Spam ist laut dieser Forschungsstelle ebenfalls rechtlich nicht unbedenklich, falls in Unternehmen die private Nutzung der E-Mails zugelassen oder geduldet wird. Das Fernmeldegeheimnis ist zu beachten, und gemäß §206 Abs. 2 Nr. 2 des Strafgesetzbuches müssen grundsätzlich alle Mails ohne Verzögerung zugestellt werden. Ein Nutzer kann jedoch sein Einverständnis zum Löschen von Spam erklären.

Das Filtern von Spam-Mails ist aber auch problematisch, weil dadurch in das Fernmeldegeheimnis eingegriffen wird, das nicht nur den Inhalt, sondern

auch die Umstände der Telekommunikation schützt. Nach §206 StGB kann derjenige mit einer Freiheitsstrafe bis zu fünf Jahren bedroht werden, der „unbefugt einer anderen Person eine Mitteilung über Tatsachen macht, die dem Post- und Fernmeldegeheimnis unterliegen und die ihm als Inhaber oder Beschäftigtem eines Unternehmens bekannt geworden sind, das geschäftsmäßig Post- oder Telekommunikationsdienste erbringt."Auch hier muss also für das Filtern die Einwilligung des E-Mail Empfängers eingeholt werden.

Fazit

Nach wie vor steigen die Angriffe durch Schadsoftware weiter an. Im Gegensatz zu früher ist hierbei jedoch der Anteil an Viren und Würmer erheblich zurückgegangen und der Anteil an Trojanischen Pferden liegt bei über 70 Prozent. E-Mails sind noch immer häufig genutzte Verbreitungswege für Schadsoftware. Als Ausgangspunkt für die Verbreitung dienen aber zunehmend URLs und gefälschte Web-Seiten, die vermeintlich interessante Downloads anbieten, so dass sich Benutzer sehr häufig auf diesem Weg einen Trojaner auf ihrem System installieren. Spam-Mails oder aber Massen-Emails mit verseuchtem Anhang könnten durch die konsequente Nutzung von Signaturen eingedämmt werden. Es ist kritisch zu fragen, ob hier nicht auch die Politik und die Gesetzgebung gefordert sind, um über Regelungen zur Eindämmung der Flut nachzudenken.

2.7 Mobile Apps

Im Zusammenhang mit Smartphones sind mobile Apps in den letzten Jahren äußerst populär geworden. Die Frage der Sicherheit tritt dabei verstärkt zu Tage, da vermehrt Apps auf mobilen Endgeräten in Umlauf gebracht werden, die Schadcode mit sich führen.

Apps

Mobile Apps[14] sind bereits heute im Consumer-Bereich sehr weit verbreitet. Derzeit wird noch ein großer Teil der Apps als rein lokale Anwendung auf mobilen Geräten ausgeführt, jedoch ist die Entwicklung hin zu smarten mobilen Apps deutlich zu sehen. Kennzeichnend für diese fortgeschrittenen Apps ist ihre Eigenschaft, Dienste zu nutzen, die über das Internet oder ein privates Netzwerk bereitgestellt werden. Viele dieser Dienste werden in einer Cloud betrieben und ermöglichen es dem Nutzer somit, einen konsistenten Datenbestand auf unterschiedlichen mobilen und stationären Geräten zu führen. Eine App stellt dabei die Client-Seite eines Cloud-Dienstes dar, so dass es für den Nutzer keine Rolle mehr spielt, von wo und mit welchem Gerät der Zugriff auf seine Daten erfolgt. Mit diesen Eigenschaften werden smarte Ap-

[14] Der Abschnitt basiert auf einem Teil eines Buchkapitels, das die Autorin zusammen mit Christian Schneider verfasst hat. Das Buch ist unter dem Titel *Smart Mobile Apps* im Springer Verlag erschienen.

ps zunehmend auch für den Geschäftsbereich attraktiv. Als Business-Apps werden sie ein integraler Bestandteil von Geschäftsprozessen werden.

2.7.1 Sicherheitsbedrohungen

Smartphones und Tablet-PCs werden von Nutzern als eine Art persönliche Informationszentralen verwendet. Die Geräte sind *always on* und werden von vielen Benutzern stets mitgeführt. Von zentraler Bedeutung für die Sicherheit der Anwendungen, und damit natürlich auch der Apps, ist die Systemsicherheit der mobilen Plattform, auf der die Anwendungen zur Ausführung gelangen. Der zweite sicherheitskritische Bereich wird durch die Anwendungsebene definiert. Sie bildet die direkte Ausführungsumgebung für die Apps.

Bedrohungen der Mobilen Plattformen

Plattform

Mobile Plattformen unterliegen einer höheren Gefährdung als stationäre. Sie werden an verschiedenen Orten und damit in unterschiedlich vertrauenswürdigen Umgebungen eingesetzt. Sie gehen schneller verloren oder werden gestohlen als etwa ein PC. Gleichzeitig kann sich ein Unbefugter Zugang zu einem mobilen Gerät verschaffen. Die kritischste Bedrohung für mobile Plattformen ist jedoch, dass solche personalisierten Geräte oftmals nicht oder nur unzureichend von einer unternehmensweiten Sicherheitsstrategie erfasst werden. Meistens werden sie von den Benutzern selbst administriert. Die Erfahrung aus der Welt der Desktop-Betriebssysteme lehrt jedoch, dass noch immer vielen Anwendern ein hinreichendes Sicherheitsbewusstsein fehlt: Zugriffssperren und Passwortschutz werden deaktiviert, da sie als störend empfunden werden, Sicherheitswarnungen werden ignoriert, Software-Updates werden, wenn überhaupt, sehr unregelmäßig eingespielt. In solchen Fällen vergrößert sich entsprechend das bereits vorhandene Bedrohungspotential mit einer unnötig großen Angriffsfläche der mobilen Plattform.

Update-Problematik

Viele mobile Betriebssysteme müssen für das Einspielen von Updates mit einem Rechner verbunden werden. Sogenannte *Over-the-Air* (OTA)-Updates ermöglichen Firmware-Updates ohne PC-Verbindung. Mit den schnellen Release-Zyklen von neuen Versionen mobiler Plattformen wie Android, können viele Hersteller von mobilen Geräten jedoch nicht mithalten. Sie stellen deshalb nur für einen kurzen Zeitraum Firmware-Updates für ihre Produkte zur Verfügung, und das auch nur mit deutlicher Verzögerung zur Android-Version. Erschwerend kommt hinzu, dass die Mobilfunkbetreiber viele Smartphones mit speziellen Anpassungen vertreiben. Diese benötigen jedoch ihrerseits speziell angepasst Firmware-Updates, die in der Regel wiederum noch später als die Updates der Hardware-Hersteller zur Verfügung gestellt werden. Die Konsequenz ist, dass auf den meisten Android-Geräten

veralteter System-Software installiert ist, die möglicherweise bekannte aber noch nicht geschlossene Sicherheitslücken enthält.

Bedrohungen der Anwendungsebene

Malware-App

Die meisten Apps werden über einen speziellen App-Marktplatz bekannt gemacht und vertrieben. Für die mobilen Endgeräte von Apple ist Apple's AppStore sogar der einzige Weg, um neue Anwendungen zu beziehen und auf den Geräten zu installieren. Der Nutzer muss also der Qualitätskontrolle der Marktplatzbetreiber vertrauen. D.h. der Nutzer muss darauf vertrauen, dass die über den Marktplatz angebotenen Apps keine Sicherheitslücken oder Schad-Funktionen aufweisen. Da die entsprechenden Kontrollverfahren nicht offengelegt sind und Apps keine standardisierte Zertifizierungsprozedur durchlaufen müssen, kann über die Qualität der durchgeführten Kontrollen keine verbindliche Aussage getroffen werden. So musste Apple in der Vergangenheit wiederholt Apps aus dem Store zurückrufen, da Sicherheitslücken entdeckt wurden. Für den offenen Android-Marktplatz sieht die Situation nicht besser aus.

Die Problematik des Vertrauens in Closed-Source-Software besteht zwar im Prinzip auch bei Software, die auf Desktop-Betriebssystemen installiert wird. Auf einer mobilen Plattform wird dieses Problem jedoch dadurch verschärft, dass beispielsweise Smartphones viele verschiedene Datenquellen aggregieren und diese über eine einheitliche Schnittstelle leicht für andere Anwendungen zugänglich machen. Darüber hinaus können schadhafte Apps ihrem Besitzer direkt beträchtliche Kosten verursachen, in dem etwa SMS-Nachrichten an Mehrwertdienste gesendet oder Sonderrufnummern angerufen werden. Wenn sich das Smartphone durch die NFC-Technologie auch noch als mobile Geldbörse etabliert, wird sich diese Problematik voraussichtlich weiter verschärfen.

Unsichere Umgebung

Eine App kann aber umgekehrt auch nicht zuverlässig feststellen, ob sie in einer sicheren, nicht modifizieren Umgebung ausgeführt wird. Eine durch Schadcode manipulierte Ausführungsumgebung könnte die App missbrauchen, um gezielt Daten auszuspähen, zu manipulieren oder über die App Zugriffe auf die Daten anderen Benutzer, andere Geschäftsprozesse etc. zu gelangen. Es stellt sich die Frage, inwieweit die Daten einer Anwendung vor unberechtigten Zugriffen durch eine andere Anwendung geschützt sind. Hier ist zunächst die Zugriffskontrolle des mobilen Betriebssystems zu betrachten. Je nach Hersteller sind hier sehr starke Unterschiede zu verzeichnen. Die Größe der zu schützenden Einheit, also die Granularität der Kontrolle, spielt dabei eine wesentliche Rolle. Je geringer die verfügbare Granularität, desto gröber sind die zu schützenden Einheiten und desto mehr Zugriffsrechte erhält eine App, auch wenn sie diese nicht im vollen Umfang benötigen würde. Da neue Software von unbekannten Autoren heutzutage auf einfachste Weise

aus dem App-Marktplatz auf die mobilen Geräte gelangt, sollte die Isolation der Apps untereinander und die Beschränkung der Rechte auf das Nötigste oberstes Gebot sein.

Aber auch legitime Zugriffe können missbraucht werden und damit unerwünschte Folgen haben. Beispielsweise gleicht die offizielle Facebook-App, die für alle populären mobilen Plattformen verfügbar ist, die Kontaktliste auf Smartphones mit den Freunden auf Facebook ab, wozu die App natürlich Zugriff auf die Kontakte des Geräts benötigt. Allerdings lädt die App auch alle Kontakte in das Phonebook hoch, die nicht bei Facebook sind, ohne dass der Benutzer dies unterbinden kann. Auf diese Weise gelangt Facebook an viele Kontaktdaten von Nicht-Mitgliedern und kann diese über die Telefonnummern als Identifikatoren verknüpfen, um das soziale Netzwerk auch über die Mitglieder hinaus zu erweitern. Dieses Beispiel verdeutlicht, warum die Aggregation verschiedener Datenquellen auf einem Gerät und die Verfügbarkeit dieser Daten über eine systemweite Schnittstelle eine besondere Bedrohung darstellt.

2.7.2 Gegenmaßnahmen

Die Hersteller von mobilen Betriebssystemen, Endgeräten und klassischen Sicherheitslösungen sind sich der zuvor beschriebenen Bedrohungen natürlich bewusst und versuchen diesen auf unterschiedliche Weise zu begegnen. Im Folgenden wird auf einige Maßnahmen exemplarisch eingegangen.

Android

Google folgt auf seiner Android-Plattform der Philosophie, dass den Apps und ihren Autoren grundsätzlich nicht vertraut werden kann. Entsprechend restriktiv sind hier die Vorgaben für die Zugriffsrechte, so dass jede App zunächst nur ihre eigenen Daten lesen und schreiben darf. Um auf systemweite Daten, wie etwa die Kontakte oder den Kalender, zuzugreifen, muss eine App die dafür vorgesehene Schnittstelle zum Android-System verwenden. Diese Schnittstellen unterliegen einem speziellen Kontrollmechanismus. Damit der App diese Zugriffe vom System gewährt werden, muss sie schon bei der Installation angegeben haben, dass sie diese Berechtigung erwünscht. Die dem Vorgehen zugrundeliegenden allgemeinen Prinzipien des *default deny* und des *need-to-know* sind im Grundsatz sehr positiv. Allerdings definiert Android annähernd 200 verschiedene Rechte, die den Zugriff auf die persönlichen Daten regeln, oder aber auch auf Geräte wie Mikrofon, Kamera und GPS-Empfänger. Daneben vergibt Android Rechte zur Verwendung von Systemfunktionen, wie das Senden von SMS-Nachrichten und das Initiieren von Telefongesprächen. Diese Berechtigungen muss der Benutzer bestätigen, bevor die App installiert wird. Dieses Vorgehen schafft eine gewisse Transparenz, verlagert aber damit das Problem auf den Benutzer, der ein entsprechendes Problembewusstsein haben muss, um eine sinnvolle Entscheidung treffen zu können.

iOS

Apples iOS, siehe dazu auch Abschnitt 13.1, verfügt nicht über solche feingranularen Berechtigungen, sondern verfolgt eine andere Strategie zur Zugriffskontrolle. Anders als auf der Android-Plattform erlaubt Apple die Installation von Apps ausschließlich über den Apple AppStore. Dadurch gibt es für iOS-Geräte einen zentralen Punkt, an dem Apple regulierend eingreifen kann. Alle Programme, die in den AppStore eingestellt werden, müssen sich zunächst einem Review-Prozess unterziehen. Dabei analysiert Apple die verwendeten Systemfunktionen und testet das Programm rudimentär. Verwendet eine App Systemfunktionen, die sie zur Erbringung ihrer Funktionalität aber gar nicht benötigt, wird das Programm abgelehnt und nicht im AppStore aufgenommen. Hier übernimmt also der Hersteller in gewisser Weise die Verantwortung, dass eine App nur die Zugriffsrechte erhält, die sie benötigt.

Die skizzierten Maßnahmen wie Zugriffskontrollen und Transparenz verhindern jedoch nicht, dass Apps mit Schadfunktionen auf ein mobiles Endgerät gelangen können. Das haben auch die Hersteller von Sicherheits-Software erkannt und bieten mobile Varianten ihrer Security-Suiten an. Der Funktionsumfang dieser Produkte reicht vom obligatorischen Viren-Scanner über ein entferntes Löschen der Daten (Remote-Wipe) bei Verlust oder Diebstahl des Endgeräts bis hin zu Rufnummernsperrlisten und SMS-Spam-Filtern.

2.8 Meltdown- und Spectre-Angriffsklassen

2.8.1 Einführung

Am 3. Januar 2018 wurden erstmalig durch Google[15] in der breiten Öffentlichkeit gravierende Sicherheitsprobleme bekannt, die nahezu alle gängigen Mikroprozessor-Architekturen betreffen. Die Probleme wurden bereits im Juli 2017 in Form eines Responsible Disclosure von Google an die Chiphersteller Intel, AMD und ARM gemeldet. Die Probleme können von Angreifern ausgenutzt werden, um die Speicherisolierung der meisten heutigen Prozessoren zu umgehen und unautorisiert Daten aus dem geschützten Bereich des Betriebssystems oder aus Speicherbereichen anderer Anwendungs-Prozesse auszulesen. Auf diese Weise können Angreifer in den Besitz von zum Beispiel Passworten, Schlüsselmaterialien oder auch anderen sensitiven Daten gelangen.

Verwundbare Systeme

Die Sicherheitsprobleme stellen Bedrohungen für alle Rechner und Plattformen dar, auf denen es möglich ist, Programme (Code) von Dritten zur Ausführung zu bringen. Das betrifft in der Regel alle End-Kunden-Geräte,

[15] https://googleprojectzero.blogspot.de/2018/01/reading-privileged-memory-with-side.html

auf denen ein Web-Browser läuft, in dem JavaScript aktiviert ist, so dass Fremdcode im Browserprozess ausgeführt werden kann (eine heute sehr übliche Web-Nutzung). Aber auch Systeme, auf denen unterschiedliche Anwendungen von Dritten ausgeführt werden (Hosting-Systeme), virtualisierte Umgebungen und Cloud-Plattformen, auf denen Code Dritter ausgeführt wird, sind durch die Lücke gefährdet. In den Anfang Januar 2018 veröffentlichten Papieren werden Proof-of-Concept (PoC) Angriffe vorgestellt, die die Ausnutzbarkeit der Verwundbarkeiten in drei Angriffsvarianten, dem Meltdown und den Spectre-Angriffen demonstrieren. Die Angriffe sind technisch zum Teil sehr anspruchsvoll und hinterlassen nach ihrer Durchführung keine Spuren auf den attackierten Systemen. Derzeit - Stand erstes Quartal 2018 - gibt es keine gesicherten Erkenntnisse, ob die PoC-Angriffe bereits aufgegriffen und Systeme systematisch gehackt wurden. Da jedoch die Meltdown-Angriffe mit leistbarem Aufwand durchführbar sind, ist nicht auszuschließen, dass zumindest Meltdown-artige Angriffe bereits in der realen Welt durchgeführt wurden.

Abwehr

Die Ursache der Problematik liegt in der Hardware der genutzten Prozessoren, so dass reine Software-Patches zur Lösung nicht ausreichen sondern Microcode-Patches zusammen mit Betriebssysteme-Updates erforderlich sind, um die Probleme in den Griff zu bekommen. Diese Lösungen haben zum Teil erhebliche Leistungseinbußen zur Folge, da z.B. zusätzliche Kontextwechsel eingeführt oder die Möglichkeiten zur Parallelverarbeitung eingeschränkt werden. Das Problem wird langfristig nur durch Änderungen bei den Prozessorarchitekturen zu lösen sein.

Meltdown und Spectre im Überblick

Meltdown

Das Meltdown-Problem [109] wurde von Forscherteams vom Google Project Zero, Cyberus Technology sowie der TU Graz unabhängig voneinander aufgedeckt. Meltdown-Angriffe nutzen aus, dass insbesondere bei Intel-Prozessoren aus Performanzgründen der Speicherbereich des Betriebssystems (Kernelbereich) vollständig in den virtuellen Adressraum eines jeden Nutzerprozesses abgebildet ist. Das heißt, dass die Seitentabelle eines jeden Nutzerprozesses auch die Speicherseiten des Betriebssystems umfasst, so dass bei Zugriffen auf den Kernelbereich, zum Beispiel bei Systemcalls, keine aufwändigen Kontextwechsel mit Adressraumwechsel erforderlich sind. Diese Seiten sind als Kernel-Pages markiert, so dass normalerweise bei einem unberechtigten Zugriffsversuch aus dem User-Bereich auf den Kernel-Bereich eine Zugriffsverletzung erkannt und der Zugriff verweigert wird. Genau dieser Zugriffsschutz wird aber in gängigen Prozessoren aus Performanzgründen ganz bewusst temporär umgangen, wenn eine spekulative oder Out-of-Order Ausführung (siehe unten) durchgeführt wird. Dabei erfolgen Speicherzugriffe zunächst ohne Kontrolle. Eine Prüfung findet erst

statt, wenn klar ist, dass die betroffenen Befehle wirklich benötigt werden. Wird dabei eine Zugriffsverletzung erkannt, so werden die Ergebnisse der spekulativen Ausführung verworfen. Solche Ergebnisse sind beispielsweise Registerinhalte. Wie weiter unten genauer ausgeführt, werden durch das spekulative Vorgehen Daten unkontrolliert aus dem Adressraum des Kernels in den L1 Cache des Prozessors geladen und können über gezielte Cache-Seitenkanal-Angriffe (vgl. Seite 11.1.3) im User-Bereich sichtbar gemacht werden.

Spectre

Die Spectre [104] Angriffe gefährden insbesondere auch virtualisierte Umgebungen und Cloud-Plattformen, die von unterschiedlichen Nutzern gemeinsam genutzt werden. Die nachfolgend geschilderten Familien von Angriffen ermöglichen den Zugriff auf Speicherbereiche anderer Nutzerprozesse, die durch die genutzten Virtualisierungstechniken eigentlich voneinander abgeschottet sein sollten. Die Familie von Angriffen, die einen unerlaubten Zugriff auf Speicherbereiche im gleichen Prozess ermöglichen, stellen insbesondere eine Bedrohung für Web-Anwendungen dar, falls JavaScript auf dem Opfersystem aktiviert ist. Gelingt es einem Angreifer, schadhaften Javascript-Code auf dem Opferrechner einzuschleusen und zur Ausführung zu bringen, dann kann dieser Schadcode auf den Speicherbereich seines Host-Prozesses, das ist der Browser-Prozess, zugreifen. Da der Browser häufig beispielsweise Zugangspassworte zwischenspeichert, kann über einen Timing-Angriff versucht werden, diese Daten auszulesen.

2.8.2 Background

Die Meltdown und Spectre-Angriffe nutzen aus, dass heutige und auch die meisten älteren Prozessoren zur Steigerung der Leistung der Prozessoren verschiedene Techniken einsetzen. Dazu gehören (implicit) Caching, Out-of-Order Execution, Speculative Execution und Branch Prediction. Durch eine Kombination verschiedener dieser Techniken entstehen Sicherheitsprobleme, auf die wir im Folgenden näher eingehen. Die eigentlichen Angriffe basieren auf wohlbekannten Techniken der Seitenkanalanalysen von Caches (u.a. [82, 135]).

Spekulative Ausführung

Das Ziel der spekulativen Ausführung von Befehlen besteht darin, Wartezeiten, die sich bei der sequentiellen Bearbeitung von Befehlen ergeben können, zu nutzen, um Befehle auszuführen, für die der Prozessor spekuliert, dass sie als nächstes benötigt werden. Mögliche Wartezeiten bei der Befehlsausführung ergeben sich dadurch, dass für die Ausführung je nach dem, auf welche Speicher zur Bearbeitung des Befehls zugegriffen wird, sehr viele Taktzyklen erforderlich sind, da ein elektronisches Signal pro Takt nur eine geringe Distanz zurücklegt. Je größer der physische Abstand zwischen der ALU und dem Speicher, aus dem Daten gelesen werden, ist,

desto länger dauert der Zugriff. Durch den Aufbau von Speicherhierarchien mit verschiedenen Schnellzugriffsspeichern, den L1, L2 und L3 Caches, die deutlich geringere Zugriffslatenzen aufweisen als der Hauptspeicher, reduzieren Mikroprozessoren die erforderlichen Zugriffszeiten. Die Cachegrößen und Zugriffslatenzen beispielsweise eines Intel i7-4770 (Haswell) Prozessors[16] sind in der Tabelle 2.1 zusammengestellt.

Typ	Größe	Latenz
L1 Data	32 KB, 64 B/line	4-5 cycles
L1 Instruction	32 KB, 64 B/line	(bei misprediction) 18-20 cycles
L2	256 KB, 64 B/line	12 cycles
L3	8 MB, 64 B/line	36 cycles
RAM	32 GB	36 cycles + 57 ns (0.29ns / cycle), d.h. ca 230 cycles

Tabelle 2.1: Cachegrößen und Zugriffslatenzen eines Intel i7-4770 Prozessors

implizites Cachen

Zur Erhöhung der Performanz lädt der Prozessor durch seine Prefetcher-Komponente Daten frühzeitig in Caches. Beispielsweise wählt der Prozessor bei einem bedingten oder indirekten Sprung einen Ausführungspfad spekulativ aus (Branch Prefetching). Die zur Ausführung der spekulativ ausgeführten Maschinenbefehle erforderlichen Daten werden in den L1 Cache geladen. Dies nennt man auch implizites Caching, da Daten geladen werden, noch bevor auf sie gemäß der sequentiellen von Neumann-Befehlsbearbeitung zugegriffen wird. Sollten in den folgenden Instruktionen die Daten benötigt werden, so kann der Speicherzugriff sehr schnell in wenigen Zyklen erfolgen, da die Daten bereits im L1 Cache vorhanden sind. Der Prozessor führt die Befehle auf diesem Pfad solange aus, bis die Sprungbedingung ausgewertet und das Ziel des Sprungs berechnet ist. Erst dann ist für den Prozessor klar, ob die bereits ausgeführten Befehle wirklich benötigt werden oder deren Ergebnisse zu verwerfen sind.

BTB

Um möglichst wenig fehlerhafte Vorhersagen (misprediction) bei einem Branch Prefetching zu erhalten, nutzen Prozessoren so genannte Branch Target Buffer (BTB). Dazu wird ein spezieller Cache, der Branch Target Buffer verwaltet, der die Zieladressen für Verzweigungen protokolliert.

Out-of-Order Execution

Ein weiteres Feature, um die Leistung heutiger Prozessoren zu erhöhen, ist die Out-of-Order Execution. Hierbei nutzt der Prozessor die Möglichkeit zur Parallelverarbeitung (superskalare Prozessoren), die durch die vorhandenen Subsysteme, wie z.B. Arithmetik-Einheit, Speicher, Vektor Logik gegeben ist. Das heißt, Instruktionen werden parallel ausgeführt, aber erst

[16] http://www.7-cpu.com/cpu/Haswell.html

dann als gültig gesetzt, wenn alle vorherigen Instruktionen fehlerfrei abgearbeitet sind. Spekulative und Out-of-Order Ausführung führen zu sehr großen Leistungsverbesserungen. Werden jedoch im realen Programmablauf die vorausschauend bereitgestellten Daten und berechneten Ergebnisse doch nicht benötigt, beispielsweise weil das Programm an eine andere Stelle springt, so werden die Ergebnisse verworfen. Da die Berechnungen in den Wartezeiten des Prozessors durchgeführt wurden, ist das kein Performanznachteil. Wenn die fälschlich voraus berechneten Befehle völlig seiteneffektfrei zurück gerollt würden, würden keine Probleme auftreten. Diese ergeben sich jedoch gerade daraus, dass die Befehlsausführungen bemerkbare Zustandsänderungen nach sich ziehen.

Basis für Angriffsmöglichkeiten

Dieses skizzierte prinzipielle Verhalten von Prozessoren birgt folgende Probleme. Zum einen werden Daten von spekulativ ausgeführten Befehlen ohne Prüfung der Zugriffsrechte in die L1 Caches geladen. Auf diese Weise können beispielsweise Daten des Betriebssystemkerns geladen werden und sind unter Umgehung der Rechtekontrolle im User-Bereich zugreifbar. Ein entsprechender Zugriff aus einem nicht privilegierten Bereich auf einen privilegierten, nämlich den Betriebssystembereich, würde normalerweise über eine Zugriffsverletzung erkannt und unterbunden.

Zudem werden die Daten nicht aus dem L1 Cache gelöscht, auch wenn die spekulativ ausgeführten Befehle nicht benötigt und die spekulativen Zwischenergebnisse aus den Registern verworfen werden. Das heißt, dass die spekulative Ausführung den Systemzustand verändert. Basierend auf gelesenen Daten können weitere Berechnungen durchgeführt und auch weitere Daten aus dem Hauptspeicher (RAM) gelesen werden. Alle diese Daten verbleiben auch dann im L1 Cache, wenn der Prozessor erkannt hat, dass die spekulative Berechnung zu verwerfen ist.

Beispiel 2.6 (Spekulative Ausführung)

Das nachfolgende Beispiel ist aus dem Google Project Zero Papier[17] entnommen.

```
struct array {
unsigned long length;
unsigned char data[];
};
struct array *arr1 = ...;
unsigned long untrusted_offset_from_caller = ...;
```

[17] https://googleprojectzero.blogspot.de/2018/01/reading-privileged-memory-with-side.html

```
if (untrusted_offset_from_caller < arr1->length) {
unsigned char value =
arr1->data[untrusted_offset_from_caller];
...
}
```

Betrachten wir den Code im Beispiel. Der Prozessor weiß nicht, dass *untrusted_offset_from_caller* größer als *arr1->length* ist. Er wird die nächsten Instruktionen spekulativ ausführen. Das heißt, aus dem Hauptspeicher wird der Wert in *arr1->data[untrusted_offset_from_caller]* gelesen. Dieser Zugriff außerhalb der Array-Grenzen (out-of-bounds) führt aber nicht zu einer Fehlersituation, da der Prozessor im Laufe der Programmausführung feststellen wird, dass dieser Verzweigungspfad nicht beschritten wird und die Befehlsausführungen verwirft, d.h. die Registerwerte und Speicherwerte werden dem Programm nicht sichtbar gemacht. Dennoch liegen die gelesenen Daten im L1 Cache vor.

▲

Angriffsklassen wie Meltdown und Spectre nutzen dieses Verhalten von leistungsfähigen Prozessor gezielt aus, um unberechtigten Zugriff auf Daten zu erhalten. Wenn beispielsweise abhängig von einem spekulativ geladenen Datum weitere Befehle geladen werden, so kann man anhand der ausgeführten Befehle Rückschlüsse auf den Wert des geladenen Datums ziehen. Dazu werden wohlbekannte Techniken, die Cache-Timing-Angriffe, verwendet. Die Angriffe erfordern jeweils sehr exakte Zeitmessungen, was ein Ansatzpunkt sein könnte, um die Timing-Angriffe abzuwehren.

2.8.3 Angriffsklassen

Die im Januar 2018 veröffentlichten Angriffe beschreiben drei Angriffsfamilien:

- Variante 1: Bounds Check Bypass: CVE-2017-5753
- Variante 2: Branch Target Injection: CVE-2017-5715
- Variante 3: Rogue Data Cache Load: CVE-2017-5754

Der vergröberte Ablauf ist bei allen drei Familien ähnlich. Um jeweils ein Bit an Information wieder herzustellen, sind folgende Schritte durchzuführen:

1. Trainieren des Branch Target Buffers (BTB).
2. Spekulative Ausführung des Codes, der den Speicher liest.
3. Messen der Cache-Zugriffszeiten, um die Daten aus Schritt 2 zu lesen.

Variante 1 Bounds Check Bypass

Das Angriffsziel ist es, solche Speicherinhalte des eigenen Prozess-Adressraums zu lesen, die im User-Bereich normalerweise nicht sichtbar sind. Der Angriff verläuft analog zu dem in Beispiel 2.6 skizzierten Vorgehen, indem der Angreifer dafür sorgt, dass die gewünschten Daten spekulativ in den L1 Cache geladen werden. Der Angreifer kann nun versuchen, den Cache-Speicher bitweise zu lesen. Nachfolgend wird das Beispiel aus dem Google-Papier[18] zur Erläuterung aufgegriffen.

Beispiel 2.7 (Bounds Check Bypass)

Sei der unten angegebene Code gegeben. Im L1 Cache seien alle Daten außer arr1->length, arr2->data[0x200] und arr2->data[0x300] bereits vorhanden. Für die if-Anweisung wird die Vorhersage getroffen (branch prediction), dass die Bedingung zu *true* ausgewertet wird. Der Prozessor führt deshalb solange spekulativ die Befehle des True-Zweiges aus, bis arr1->length geladen und die spekulative Ausführung beendet wird. Die spekulative Befehlsausführung umfasst damit folgende Schritte:

- Lade value = arr1->data[untrusted_offset_from_caller].
- Lade index2-Wert.
- Lade, abhängig vom Wert von index2 den Wert aus arr2, also arr2->data[index2], in den L1 Cache.

```
struct array {
unsigned long length;
unsigned char data[];
};
struct array *arr1 = ...; /* small array */
struct array *arr2 = ...; /* array of size 0x400 */
                          /* >0x400 (OUT OF BOUNDS!) */
unsigned long untrusted_offset_from_caller = ...;
if (untrusted_offset_from_caller < arr1->length) {
unsigned char value =
arr1->data[untrusted_offset_from_caller];
unsigned long index2 = ((value&1)*0x100)+0x200;
if (index2 < arr2->length) {
unsigned char value2 = arr2->data[index2];
}
}
```

[18] https://googleprojectzero.blogspot.de/2018/01/reading-privileged-memory-with-side.html

Nachdem der Prozessor erkannt hat, dass der Wert untrusted_offset_from_caller größer ist als der Wert arr1->length, wird der spekulative Pfad verworfen, jedoch bleibt der Wert arr2->data[index2] weiterhin im L1 Cache. Dies ermöglicht nun folgenden Seitenkanal. Ein Angreifer kann die Zeit messen, die erforderlich ist, um die Datenwerte arr2->data[0x200] und arr2->data[0x300] zu laden. Der Angreifer kann daraus ableiten, ob der Wert von index2 während der spekulativen Ausführung den Wert 0x200 oder 0x300 hatte. Dies wiederum erlaubt den Rückschluss, ob arr1->data[untrusted_offset_from_caller]&1 den Wert 0 oder 1 besitzt. Ein solcher Timing-Angriff enthüllt also einen 1-bit Wert.

Timing-Attack

Den Papieren von Google ist zu entnehmen, dass mit einer Datenrate von ca. 2000 Byte pro Sekunde Daten aus dem Speicher über solche Angriffe Bit für Bit erschlossen werden können. In den Papieren wird ein Proof-of-Concept erläutert, in dem unter einer Debian-Linux-Distribution aus dem nicht privilegierten User-Bereich beliebige Zugriffe auf 4GB Speicher des Betriebssystemkerns möglich waren.

▲

Um dieses Verhalten des Prozessors gezielt für einen Angriff ausnutzen zu können, muss es einem Angreifer möglich sein, dafür zu sorgen, dass ein solcher verwundbarer Code im Kontext des Opferprozesses mit einem Indexwert außerhalb der Grenzen (out-of-bounds) ausgeführt wird. Der Code, der spekulativ ausgeführt wird, muss dazu bereits auf dem Rechner vorhanden sein. Darum sind für diese Angriffsszenarien JIT (just in Time) Compiler von besonderem Interesse, da sie genutzt werden können, um verwundbaren Code zu generieren. Auch JavaScript mit Exploits, die Daten aus der Browser Sandbox extrahieren, kann genutzt werden. Mit Techniken wie eBPF[19] kann, wie der Proof-of-Concept von Google auch gezeigt hat, sogar Kernel-Speicher gelesen werden, da der eBPF Interpreter direkt im Kernel ausgeführt wird und aus dem User-Bereich nutzbar ist.

Abwehr

Derartige Timing-Angriffe sind zu verhindern, wenn keine Daten mehr spekulativ geladen werden, deren Adressen von anderen spekulativ geladenen Daten abhängen, oder wenn sichergestellt wird, dass solche Daten nicht im Cache verbleiben, wenn sie nicht vorher schon dort waren. Patches, wie der für die C/C++-Compiler von Microsoft, stellen eine Switch (Qspectre) zur Verfügung. Damit wird durch den Compiler automatisch eine so genannte spekulative Barriere in den übersetzten Code eingefügt, wenn er eine Instanz

[19] Extended Berkeley Packet Filter. Paketfilter, der in der virtuellen Maschine im Kernel ausgeführt wird, um Netzwerkpakete frühzeitig zu filtern. Um BPF Bytecode in Assemblercode der Gastmaschine zu transformieren, kann der in den Kernel integrierte JIT Compiler genutzt werden.

der Variante 1 Problematik erkennt, so dass mit den noch nicht geprüften Werten keine vorausschauenden Ladeoperationen durchgeführt werden.

Variante 2 Branch Target Injection

Ausgangspukt für diese Angriffsvariante ist der Branch Target Buffer (BTB), der über Prozessgrenzen hinweg gültig ist. Bislang waren nur Angriffe bekannt, durch die der Angreifer über diesen gemeinsamen Buffer Informationen darüber erhalten hat, an welcher Adresse Code im Opferprozess ausgeführt wird. In den neuen Angriffen versucht der Angreifer den Opferprozess direkt zu beeinflussen, indem er den BTB von anderen Prozessen trainiert. Dieses Training hat zur Konsequenz, dass für den Angriff nutzbarer Code spekulativ durch den Opferprozess ausgeführt wird. Konzeptuell besitzt der Angriff eine Ähnlichkeit mit ROP-Angriffen (Return-oriented Programming), wobei anders als bei ROP die genutzten Gadget(s) nicht sauber enden, dafür aber einen Seitenkanal aufbauen müssen. Laut dem zugrundeliegenden Papier von Google ist eine Datenrate von ca. 1500 Byte pro Sekunde möglich.

Indirekter Sprung

Die Basis für einen Angriff ist Code, der einen indirekten Sprung enthält[20]. Der Angriff sorgt durch das Flushen von Caches dafür, dass das Sprungziel aus dem Speicher geladen werden muss. Wenn die CPU eine solche indirekte Verzweigung ausführt, kann das Sprungziel erst berechnet werden, nachdem die Cacheline wieder geladen ist. Dieser Ladevorgang ist zeitaufwändig und erfordert i.d.R. über hundert Taktzyklen. Die CPU kann diese Zeit nutzen, um spekulativ Befehle basierend auf der Branch Prediction auszuführen. Die Angriffsidee ist nun, diese Vorhersage zu beeinflussen und dafür zu sorgen, dass die CPU Befehle ausführt, die der Angreifer gezielt über dedizierte Sprungzieladressen in den BTB injiziert hat. Dafür ist es notwendig, das genaue Verhalten der Branch Prediction von Prozessoren zu verstehen. Dies ist jedoch nicht einfach, da die Hersteller keine vollständige Spezifikation offen legen.

vergröberter Ablauf

In dem Google Papier ist ein Proof-of-Concept Angriff erläutert, bei dem u.a. ausgenutzt wird, dass der BTB nicht die vollständige Zieladresse sondern nur die unteren 32 Byte abspeichert. Weiterhin wird der Branch History Buffer (BHB) für den Angriff ausgenutzt. Der BHB speichert Informationen über die letzten Verzweigungen und deren Ziele und stellt damit einen Ausschnitt des Kontrollflusses dar. Diese Information ist für den Predictor hilfreich, wenn es für eine Verzweigung mehrere mögliche Ziele gibt. In dem

[20] Im Gegensatz zu einem direkten Sprung spezifiziert ein indirekter Sprung in seinem Argument nicht die Zieladresse des nächsten Befehls sondern eine Adresse, unter der die Adresse des nächsten Befehls zu finden ist (Indirektion). Die Zieladresse des nächsten Befehls kann also erst bestimmt werden, wenn der Sprungbefehl ausgeführt wird

KVM-Angriff

PoC Angriff aus dem Project Zero Papier wird durch gezieltes Herbeiführen von Mispredictions dafür gesorgt, dass aus den im BHB gespeicherten Informationen die Adresse eines interessierenden Speicherbereichs hergeleitet werden kann.

Um Daten aus dem Speicher zu extrahieren, wird wie oben bereits gesagt, der Cash gezielt geflusht, wozu die Operationen Flush+Reload genutzt werden. Dies erfordert jedoch die Kenntnis einer virtuellen Adresse des Kernelbereichs des Hosts im virtuellen Speicher des KVM-Gasts. Da der gesamte physikalische Speicher des Kernels des Hosts in einen bestimmten Bereich des Gastes (physmap) abgebildet wird, dieser Bereich aber den Gastprozessen zufällig zugewiesen wird, würde ein Bruteforce-Angriff mit Durchsuchen des möglichen Adressraums sehr aufwändig (aber möglich) sein. Durch einen BTB-Injection Angriff kann die Angriffsdauer auf wenige Minuten verkürzt werden. Hierzu wird der BTB gezielt mit Daten injiziert, so dass Daten geladen werden, die vom Angreifer kontrolliert werden und der Angreifer kann damit testen, ob eine Seite des Gast-Adressraums geladen wird. Um schließlich die eigentlichen Daten aus dem Speicherbereich der Kernels des Hosts zu lesen, wird ein ROP-artiger Ansatz verfolgt. Dazu wird wiederum der eBPF Interpreter ausgenutzt, der im Kernelbereich des Hosts integriert ist. Normalerweise ist es jedoch nicht möglich, diesen Interpreter aus einer Gast-VM aufzurufen. Da jedoch der Code des Interpreters im Code-Segment des Kernels liegt, kann dieser Code als Gadget für einen ROP Angriff genutzt und ungeprüfter eBPF-Bytecode kann spekulativ ausgeführt werden,

BTP Injection

Gegenmaßnahmen

Zur Abwehr derartiger Angriffe wurden Maßnahmen vorgestellt, die teilweise jedoch zu kurz greifen oder sogar kontraproduktiv wirken können. Da die Ursache für Angriffe der Variante 2 indirekte Sprünge sind, wurde die Maßnahme namens Retpoline[21] vorgeschlagen, um solche Sprünge und Verzweigungen zu vermeiden. Dazu wurden indirekte Sprünge durch Return-Befehle ersetzt, wobei die Return-Adresse beim Aufruf der Funktion auf den Stack geschrieben wird und der Stack beim Aufruf von Return im Cache vorliegt. Dadurch wird erreicht, dass der Prozessor keine Befehle spekulative ausführt. Da jedoch in den letzten Jahren im Zuge der ROP-Angriffsklassen die Return-Befehle durch Sprung-Befehle ersetzt wurden, um ROP-Verwundbarkeiten zu mitigieren, wurde dieser Vorschlag bereits nach wenigen Tagen wieder verworfen.

Da die Angriffsvariante 2 den BTB gezielt durch Injektionen nutzt, ist das Leeren des Caches bei jedem Kontextwechsel eine wirksame Abwehrmaßnahme. Durch Microcode-Updates für die betroffenen Prozessoren wird

[21] https://support.google.com/faqs/answer/7625886

hierfür eine Lösung geliefert, die von den jeweiligen Betriebssystemen genutzt werden kann.

Variante 3 Rogue Data Cache Load

Die dritte Variante der Angriffe ist unter dem Namen Meltdown bekannt.

Adressraumorganisation

Konzepte der Adressraum- und Speicherisolierung zählen zu den zentralen Schutzkonzepten heutiger Betriebssysteme. Damit wird sichergestellt, dass Prozesse im User-Space nicht wechselseitig auf ihre jeweiligen Speicherbereiche zugreifen können und dass aus der Anwendungsebene, dem User-Space, kein direkter Lese- oder Schreibzugriff auf Speicherbereiche des Betriebssystem-Kerns möglich ist. Wie bereits auf Seite 84 angesprochen, wird die Isolation zwischen Kernel- und User-Space-Speicherbereichen hardwareseitig durch das Setzen eines Supervisor-Bits für die Speicherseiten des Kernels realisiert. Dadurch können in heutigen Architekturen die Speicherbereiche des Kernels in den Adressraum eines jeden Prozesses abgebildet werden, da die Zulässigkeit von Zugriffen anhand des Bits kontrolliert wird. Dies ermöglicht eine effiziente Verwaltung, da beispielsweise bei einem Syscall, der einen Wechsel aus dem User-Space in den Kernel-Space nach sich zieht, kein aufwändiger Adressraumwechsel mit dem Laden der Seitentabelle des Kernel-Bereichs erforderlich ist. Die Seiten des Kernels sind in der Seitentabelle des Anwendungsprozesses vorhanden und können nach dem Wechsel in den Kernel-Modus direkt genutzt werden.

Meltdown-Angriff

Bei einem Meltdown-Angriff wird dieses Isolations- und Kontrollkonzept umgangen. Mittels eines Meltdown-Angriffs kann aus dem User-Space heraus Information aus Speicher des Kernels ausgelesen, bzw. präziser ausgedrückt, diese Information kann mittels Seitenkanalanalysen rekonstruiert werden. Für den Angriff wird keine Software-Schwachstelle ausgenutzt, sondern analog zu den zuvor beschriebenen Vorgehensweisen wird das Verhalten der Out-of-Order-Execution gängiger Prozessoren genutzt, um über Seitenkanalangriffe die geschützte Information zu extrahieren. Der Angriff basiert damit auf der gleichen Idee wie die Angriffe der Variante 1. Mit einem Meltdown-Angriff ist es möglich, den gesamten Adressbereich des Kernels in einer Folge von Out-of-Order Ausführungen auszulesen und über verdeckte Kanäle der zugrundeliegenden Microarchitektur beispielsweise mittels Flush+Reload in Bereiche zur transferieren, in denen die Registerwerte dann mittels der Seitenkanalanalysen rekonstruiert werden können. Neben Zugriffen auf den Adressbereich des Kernels kann über Meltdown-Angriffe auch auf physischen Speicher direkt zugegriffen werden, der ggf. Daten anderer Prozesse enthält. Wird der Kernel-Bereich gemeinsam genutzt, wie dies durch Sandboxing mit Docker oder auch Xen mit Paravirtualisierung oder auch durch Hypervisor-Implementierungen der Fall

ist, so kann aus dem User-Space auch auf Daten anderer Prozesse, z.B. in einer Cloud-Umgebung, zugegriffen werden. Meltdown stellt somit auch ein gravierendes Problem für Cloud-Anbieter dar, besonders wenn deren Gastsysteme nicht vollständig virtualisiert sind und u.a. aus Performanzgründen mit gemeinsamen Kernel-Bereichen arbeiten.

Da beim Meltdown-Angriff jedoch kein Adressraumwechsel erforderlich ist, kann der Speicher mit hoher Datenrate von ca. 500 *kb/s* gelesen werden und Angriffe sind damit ca. 300 mal schneller durchführbar, als die ähnlichen Angriffe der Spectre-Variante 1.

Gegenmaßnahmen

KAISER

Ein Schutz gegen die Meltdown-Angriffe bietet das unter dem Namen KAISER [75] bekannte Konzept, das ursprünglich eingeführt wurde, um einen Schutz gegen Seitenkanalangriffe auf KASLR (Kernel Address Space Layout Randomization) zu bieten. Die naheliegende Lösungsidee von KAISER besteht darin, die Adressbereiche des Kerns und des User-Space strikt zu trennen, so dass der Kernel-Adressbereich nicht mehr im virtuellen Adressbereich eines Nutzerprozesses vorhanden ist. Bis auf wenige Teile, die von der x86 Architektur benötigt werden, wie beispielsweise Interrupt Handler, kann durch die Aufteilung von Kernel- und User-Space-Adressraum sichergestellt werden, dass während der Ausführung im User-Mode in der Hardware keine Daten aus dem Kernel-Bereich verfügbar sind. Beim Wechsel zwischen Kernel- und User-Space wird der TLB geflushed, so dass eine strikte Isolierung umgesetzt wird. KAISER erfordert eine Integration der entsprechenden Maßnahmen in die zugrundeliegenden Betriebssysteme. Unter der Bezeichnung Kernel Page-Table Isolation (KPTI) wurde das KAISER-Konzept in den Linux-Kernel 4.15 integriert und auch frühere Linux-Versionen, wie der Linux-Kernel 4.4.110, 4.9.75 und 4.14.11 wurden gepatcht. Zu beachten ist, dass diese mit KPTI einhergehende Trennung der Seitentabellen und damit der Adressräume von Kernel und User-Space kein Schutz vor Angriffen der Spectre Varianten 1 und 2 darstellt. Die zusätzliche Isolierung bringt jedoch erhebliche Performanzeinbußen mit sich; Messungen schwanken zwischen 5 und bis zu 30 Prozent, abhängig von der Anzahl der auszuführenden Systemaufrufe.

KPTI

Abschließend bleibt festzuhalten, dass die Meltdown und Spectre-Lücken keine Software-Schwachstellen darstellen und deshalb nicht allein durch Software-Patches zu reparieren sind, sondern zumindest Microcode-Updates erfordern. Die auf den ersten Blick einfachste Lösung wäre natürlich, bei den Prozessoren auf die Out-of-order Execution oder auf die spekulative Ausführung zu verzichten. Da dies aber mit nicht tragbaren Performanzverminderungen verbunden wäre, ist dies keine akzeptable Lösung. Für das Meltdown-Problem könnte eine physische Aufteilung des User- und Kernel-

physische Aufteilung

Speicherbereichs eine Lösung sein. dies würde das Setzen eines neuen Hardwarebits in einem CPU Kontrollregister erfordern, wie beispielsweise CR4. Bei einer solche physischen Aufteilung könnte der Kernel-Bereich im oberen Adressbereich und der User-Space im unteren Adressbereich abgelegt werden. Eine mögliche Zugriffsverletzung könnte dann direkt aus der virtuellen Adresse abgeleitet werden, wodurch eine sehr effiziente Kontrolle möglich ist. Eine solche Maßnahme schützt jedoch nicht vor den Spectre-Angriffen.

Microcode-Updates

Als Schutzmaßnahme gegen alle drei Angriffsvarianten stehen Microcode-Updates der Prozessor-Hersteller zur Verfügung und werden mittlerweile von fast allen gängigen Betriebssystem-Herstellern über Betriebssystem- oder BIOS-Updates den Nutzern zur Verfügung gestellt. So sorgt beispielsweise die neue Funktion *Indirect Branch Prediction Barriers (ibpb)* in den Intel-Prozessoren dafür, dass eine Vorhersage für einen indirekten Sprung nicht mehr von vorher ausgeführten Anweisungen abhängig sein kann. Es bleibt abzuwarten, wie in zukünftigen Prozessorarchitekturen Fragen der IT-Sicherheit und der Performanz mit neuen Architektur-Designs einheitlich abgedeckt werden; die Spectre-Schwachstellen haben den Bedarf für neue Architektur-Ansätze klar aufgezeigt.