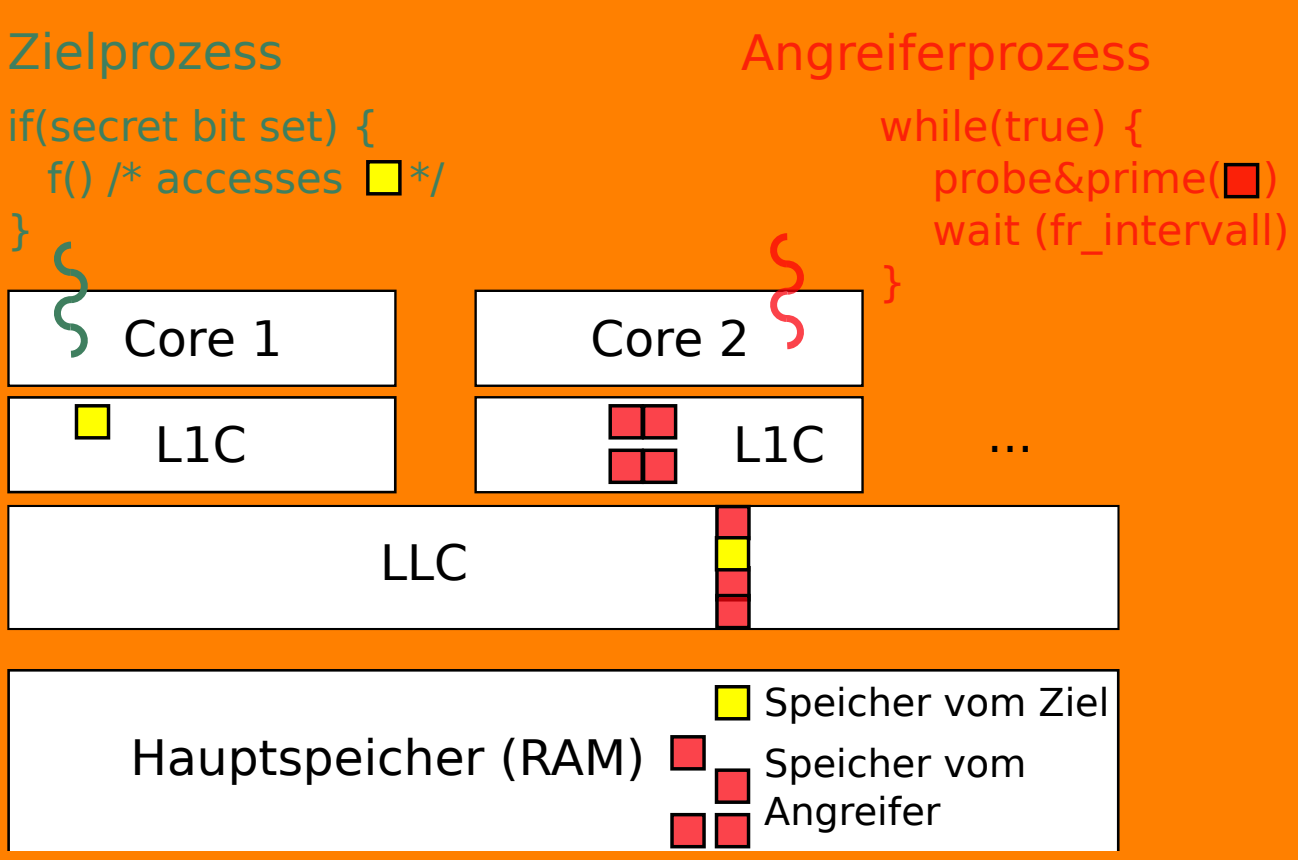
Open C3S
Open Competence Center for Cyber Security

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik



Mikromodul 8002: Cloud-Sicherheit und Bedrohungsmodelle

Autoren:

Prof. Dr. Hans P. Reiser

Noëlle Rakotondravony

Johannes Köstler

Universität Passau

Mikromodul 8002: Cloud-Sicherheit und Bedrohungsmodelle

Autoren:

Prof. Dr. Hans P. Reiser

Noëlle Rakotondravony

Johannes Köstler

---

1. Auflage

Universität Passau

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

AUFSTIEG DURCH BILDUNG >>
OFFENE HOCHSCHULEN

Universität Passau
Fakultät für Informatik und Mathematik
Innstraße 43
94034 Passau

1. Auflage (4. Mai 2017)

Um die Lesbarkeit zu vereinfachen, wird auf die zusätzliche Formulierung der weiblichen Form bei Personenbezeichnungen verzichtet. Wir weisen deshalb darauf hin, dass die Verwendung der männlichen Form explizit als geschlechtsunabhängig verstanden werden soll.

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 16OH12025 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

Inhaltsverzeichnis

Einleitung **4**
I. Abkürzungen der Randsymbole und Farbkodierungen 4
II. Zu den Autoren 5

Mikromodul: Bedrohungsmodelle **7**
1 Lernziele 7
2 Grundlagen der Bedrohungsmodellierung 7
3 Sicherheitsherausforderungen in der Cloud 9
3.1 Sicherheitsvorteile 9
3.2 Sicherheitsnachteile 10
4 Sichere Datenspeicherung und -verarbeitung in der Cloud 11
4.1 Sichere Datenspeicherung 12
4.2 Verarbeitung verschlüsselter Daten 14
5 Bedrohungen durch Koresidenz in der Cloud 16
5.1 Cloud-Kartographie 16
5.2 Koresidenz-Tests 17
5.3 Koresidenz-Angriffe 18
5.4 Neuere Untersuchungen 19
6 Seitenkanalangriffe in der Cloud 19
6.1 Grundbegriffe und Grundlagen 19
6.2 Seitenkanäle bei der Erzeugung von Zufallszahlen 21
6.3 Cachebasierte Seitenkanäle im PaaS-Modell 23
6.4 Cachebasierte Seitenkanäle im IaaS-Modell 26
7 Übungsaufgaben 28

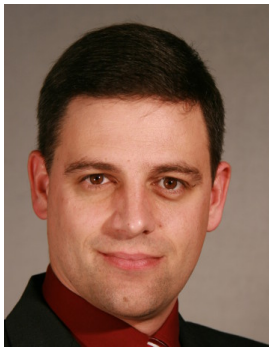
Verzeichnisse **31**
I. Abbildungen 31
II. Definitionen 31
III. Exkurse 31
IV. Kontrollaufgaben 31
V. Tabellen 31
VI. Literatur 31

Einleitung

I. Abkürzungen der Randsymbole und Farbkodierungen

Definition	D
Exkurs	E
Kontrollaufgabe	K
Übung	Ü

II. Zu den Autoren



Hans P. Reiser ist Juniorprofessor für Sicherheit in Informationssystemen an der Universität Passau. Schwerpunkte seiner Arbeitsgruppe sind die Weiterentwicklung von Konzepten und Systemen aus dem Bereich der fehler- und einbruchstoleranten Replikation, die frühzeitliche und umfassende Erkennung von Sicherheitsproblemen und Sicherheitsvorfällen in Cloud-Umgebungen sowie die Erforschung neuartiger Sicherheitskonzepte auf Hypervisorebene.

Noëlle Rakotondravony ist seit September 2015 wissenschaftliche Mitarbeiterin in der Arbeitsgruppe von Prof. Hans P. Reiser an der Universität Passau.

Johannes Köstler ist seit Mai 2015 wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Hans P. Reiser an der Universität Passau.

Mikromodul: Bedrohungsmodelle

1 Lernziele

Nach Abschluss dieses Mikromoduls sind Sie mit den Grundlagen von Bedrohungsmodellierung und Risikomanagement in IT-Systemen vertraut. Auf dieser Grundlage können Sie Sicherheitsrisiken in Cloud-Infrastrukturen beurteilen. Sie sind vertraut mit theoretischen und praktischen Ansätzen zur sicheren Speicherung und Verarbeitung von Daten in der Cloud. Sie können cloudspezifische Angriffe wie Koresidenz und Seitenkanalangriffe bewerten.

2 Grundlagen der Bedrohungsmodellierung

Die Entwicklung sicherer Softwaresysteme stellt eine große Herausforderung dar. Werden im Entwicklungsprozess Sicherheitslücken frühzeitig erkannt, können die Kosten zur Behebung dieser Lücken gering gehalten werden. Eine Voraussetzung für die Entwicklung sicherer Softwaresysteme ist die systematische Analyse von Bedrohungsmodellen und daraus abgeleiteten Sicherheitsanforderungen.

Bedrohungsmodellierung (threat modelling) ermöglicht es, einen Systementwurf methodisch zu überprüfen, um potentielle Schwachstellen zu identifizieren und zu beheben. Am größten ist der Nutzen dann, wenn damit Sicherheitslücken bereits in einer frühen Phase des Entwicklungsprozesses identifiziert werden. Dieser Aspekt ist nach NIST (2002) exemplarisch in Tabelle 1 illustriert. Dargestellt sind Schätzwerte für die relativen Kosten zur Behebung von Schwachstellen abhängig davon, in welcher Phase des Entwicklungszyklus diese erkannt werden.

Anforderungs-analyse	Implementierung/Unit-Tests	System-integration	Beta-Tests	Nach dem Release
1x	5x	10x	15x	30x

Tabelle 1: Relative Kosten der Schwachstellen-Reparatur, abhängig von der Phase des Entwicklungszyklus

Bei der Bedrohungsmodellierung werden auf Grundlage einer Bewertung von Sicherheitslücken mögliche Schutzmechanismen abgeleitet, mit denen das *Risiko* reduziert werden kann. Im Folgenden wollen wir nun diesen und weitere relevante Begriffe näher definieren.

Nach ISO/IEC 21827, einem internationalen Standard basierend auf dem *Systems Security Engineering Capability Maturity Model*, ist der zentrale Begriff *Risiko* wie folgt definiert:

> Definition 1: Risiko nach ISO/IEC 21827
>
> „The potential that a given threat will exploit vulnerabilities of an asset or group of assets to cause loss or damage to the assets."

D

Aus dieser Definition lassen sich die drei Faktoren ablesen, die das Risiko beeinflussen: *Bedrohungen (threats)*, *Schwachstellen (vulnerabilities)* und *Auswirkungen (loss or damage to the assets)*.

Bedrohungen

Für den Begriff *Bedrohung* finden sich in der Literatur viele verschiedene Definitionen. Wir greifen in Definition 2 eine ausführliche Definition auf, die aus „Minimum

Security Requirements for Federal Information and Information Systems“ (FIPS 200) der NIST stammt.

D

Definition 2: Threat nach NIST/FIPS 200 (NIST, 2006)

„Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, or individuals through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. Also, the potential for a threat-source to successfully exploit a particular information system vulnerability.“

Bedrohungen umfassen demnach eine große Bandbreite an Ereignissen oder Umständen, die den Betrieb beeinträchtigen können. Dazu lassen sich sowohl zufällig auftretende Fehler, als auch verschiedenste Arten von gezielten Angriffen (wie unberechtigter Zugang zu Geräten, Schadsoftware, Sabotage, Spionage etc.) zählen.

Schwachstellen (Vulnerabilities)

Auch für den Begriff *Schwachstelle* gibt es in der Literatur eine Vielzahl von ähnlichen Definitionen. Exemplarisch greifen wir an dieser Stelle eine Definition auf, die aus IETF RFC 4949 stammt:

D

Definition 3: Vulnerability nach IETF RFC 4949

„A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy.“

Nach dieser Definition kann es drei unterschiedliche Arten von Schwachstellen geben: zum Ersten Schwachstellen im Systemdesign bzw. in der Spezifikation des Systems, zum Zweiten Schwachstellen in der Systemimplementierung und zum Dritten in geben: zum einen Schwachstellen im Systemdesign bzw. in der Spezifikation des Systems; zum anderen Schwachstellen in der Systemimplementierung; und außerdem in Betrieb und Management des Systems.

Nicht jede Bedrohung führt zu einer schädlichen Auswirkung, ebenso wie nicht jede Schwachstelle zu einer solchen führt. Vielmehr muss die Bedrohung auf eine geeignete Schwachstelle treffen und ggf. vorhandene Schutzmaßnahmen (*„safeguards“*) müssen versagen.

Auswirkungen (Loss or damage to the assets)

Auch der Blick auf die Auswirkungen eines möglicherweise erfolgreichen Angriffs ist ein wesentlicher Bestandteil der Bedrohungsanalyse. Häufig werden Sicherheitsmechanismen eingesetzt, ohne sich über den Grund dafür im Klaren zu sein. Neben der Wahrscheinlichkeit von erfolgreichen Angriffen (abhängig von Bedrohungen und Schwachstellen) ist es jedoch immer angebracht, die Auswirkungen zu betrachten. Was ist an einem System wertvoll? Wie kann dieser Wert verloren gehen? Von der Beantwortung dieser Fragen ist es abhängig, ob es sinnvoll sein kann, geeignete Sicherheitsmechanismen zur Reduktion des Risikos einzusetzen.

3 Sicherheitsherausforderungen in der Cloud

Nach dieser allgemeinen Einführung in die Grundbegriffe der Bedrohungsmodellierung soll dieser Abschnitt näher auf Cloud-Computing-Umgebungen eingehen. Dabei werden angelehnt an ein Dokument der ENISA (Dupré und Haeberle, 2012) im Folgenden sowohl Sicherheitsvorteile von Cloud-Umgebungen betrachtet als auch Sicherheitsrisiken näher diskutiert.

3.1 Sicherheitsvorteile

Cloud Computing bietet ein großes Potential hinsichtlich Vorteile bei IT-Sicherheit. Eine manchmal durch Werbung von Cloud-Anbietern suggerierte Gleichsetzung von Cloud-Nutzung und erhöhter IT-Sicherheit stellt dabei zwar eine zu starke Vereinfachung einer komplexen Fragestellung dar. Nichtsdestotrotz ist es angebracht, sich über die wesentlichen Sicherheitsvorteile durch den Einsatz von Cloud Computing bewusst zu sein.

An erster Stelle ist hier ein *Skalierungsgewinn* zu nennen. Insbesondere große Cloud-Anbieter können grundsätzlich signifikante Skalierungsgewinne erzielen. Angefangen bei der physischen Sicherung der Gebäude (Einbruchsschutz, Zugangskontrolle), über System-Überwachung bis hin zu Wartung und Patch-Management, sinken die Kosten pro Instanz bei einer für viele Instanzen gemeinsam genutzten, großen Infrastruktur. Damit ist oft ein höheres Sicherheitsniveau bei geringeren Kosten erreichbar als bei einer privaten Inhouse-Infrastruktur.

Damit eng verbunden ist die Möglichkeit, *Updates* schneller und effizienter durchzuführen. Insbesondere bei kleineren Organisationen ist es weit verbreitet, dass Koordinierung und Durchführung von Aufgaben des Sicherheitsmanagements von Personal durchgeführt wird, das daneben auch andere Aufgaben zu erledigen hat. Werden Aufgaben wie das Ausrollen von Sicherheitsupdates von dedizierten Verantwortlichen durchgeführt, so reduziert sich die Zeit von Bekanntwerden einer Schwachstelle bis zu deren Beseitigung.

Marktdifferenzierung ist ein gutes Motiv für Cloud-Anbieter, um für ein hohes Sicherheitsniveau zu sorgen. Sicherheit ist für Cloud-Kunden ein zentrales Anliegen und bekannt gewordene Sicherheits-Schwachstellen können eine katastrophale Auswirkung auf die Reputation eines Cloud-Anbieters haben. Daher besteht besonders für große Cloud-Anbieter eine starke Motivation für eine kontinuierliche Verbesserung der IT-Sicherheit und für den Einsatz von gut ausgebildetem, erfahrenem Personal.

Zudem ist hier die *flexible Ressourcenskalierung* zu nennen. Dynamische Ressourcenskalierung ermöglicht es bei Bedarf während des Betriebs weitere Ressourcen zu allozieren. Dieser vor allem mit dem Ziel der Verbesserung der Dienstqualität eingesetzte Mechanismus ist auch aus Blickwinkel der IT-Sicherheit relevant, insbesondere weil sich damit die Auswirkungen von Denial-of-Service-Angriffen entgegenwirken lassen und so die Verfügbarkeit verbessern lässt. Vorstellbar sind aber auch Mechanismen, bei denen bei Verdachtsfällen Dienste zur Vorfallsuntersuchung und Forensik nach Bedarf bereitgestellt werden.

Ein weiterer Aspekt ist der *physische Schutz* vor nicht autorisierten Zugriffen. Bei Rechenzentren großer öffentlicher Cloud-Anbieter sind eine Vielzahl an Maßnahmen zum physischen Schutz nach aktuellem Stand der Technik üblich, wie beispielsweise eingezäunte Gelände, Alarmsysteme, Videoüberwachung und Zugangskontrolle an Türen. Bei kleineren Organisationen sind derartige Mechanismen oft nicht zu finden bzw. dediziert für eine entsprechend kleine IT-Infrastruktur unverhältnismäßig teuer.

Sicherheitsvorteile sind dabei auch abhängig vom *Cloud-Servicemodell* zu betrachten. Am geringsten fallen Vorteile im *IaaS-Modell* aus. Hier ist der Cloud-Kunde für das Sicherheitsmanagement seiner virtuellen Maschine in gleicher Weise selbst verantwortlich, wie er es auch in seiner eigenen privaten Infrastruktur wäre. Nichtsdestotrotz sind hier die Vorteile des im Allgemeinen besseren physischen Schutzes und ggf. auch die Möglichkeit der dynamischen Skalierung zu nennen. Im *PaaS-Modell* weiten sich diese Vorteile auf das Sicherheitsmanagement auf Ebene von Betriebssystem und Plattform aus. Im *SaaS-Modell* wird schließlich die Verantwortung für nahezu alle technischen Sicherheitsaspekte vom Cloud-Kunden zum Cloud-Betreiber verlagert.

3.2 Sicherheitsnachteile

Den genannten Vorteilen stehen in der Praxis allerdings auch einige wichtige Nachteile entgegen. Zunächst sind hier Risiken für die *Verfügbarkeit* zu nennen.

Offensichtlich ist die Abhängigkeit der Nutzung einer öffentlichen Cloud von der *Verfügbarkeit des Kommunikationsnetzes*. Dieser Aspekt ist weniger für IT-Dienste, die der Cloud-Kunde für externe Dritte anbietet relevant, sondern insbesondere für intern selbst genutzte. Fällt das Kommunikationsnetz aus, so sind die in die Cloud ausgelagerten IT-Dienste nicht mehr verfügbar, während bei einer Inhouse-Infrastruktur dieses Risiko nicht gegeben ist bzw. selbst auf Ausfälle reagiert werden kann. Gegenmaßen wie redundante Anbindung über mehrere Netzanbieter bringen wiederum höhere Kosten mit sich.

Ein ähnlicher Aspekt ist der *Vendor Lock-In*, also die Abhängigkeit von dem genutzten Cloud-Anbieter. Während im IaaS-Modell oft eine Migration zu einem Anbieter noch relativ leicht machbar ist, kann dies bei PaaS und insbesondere bei SaaS im Allgemeinem mit sehr hohem Aufwand verbunden sein. Auch kann die Migration von großen Datenmengen zu signifikanten Kosten führen. Daher kann es insbesondere dann, wenn ein Cloud-Anbieter seinen Dienst komplett einstellt, zu hohen Kosten und Einschränkungen der Verfügbarkeit kommen.

Ebenfalls als Nachteil zu nennen ist der *Kontrollverlust*. Aus vertraglichen oder gesetzlichen Gründen kann es für den Cloud-Kunden erforderlich sein, bestimmte *Audits* oder *Zertifizierungen* durchzuführen. Dies wird spätestens dann problematisch, wenn entsprechend notwendige Maßnahmen nicht vom Cloud-Anbieter angeboten werden. Oft können diese Maßnahmen nicht durch den Cloud-Kunden selbst durchgeführt werden. Das gleiche Problem kann entstehen, wenn Sicherheits-Richtlinien des Cloud-Kunden nicht dem entsprechen, was der Cloud-Anbieter bereitstellt, und der Kunde seine Richtlinien nicht durchsetzen bzw. kontrollieren kann.

Neben diesen Nachteilen gibt es noch weitere, cloudspezifische Sicherheitsrisiken, die auch die *Vertraulichkeit* und *Integrität* betreffen können. Nach Ramos (2009) können diese in drei Kategorien eingeteilt werden: zum Ersten Insider-Angriffe, bei denen ein Angriff von der Cloud-Infrastruktur (beispielsweise durch einen Cloud-Administrator als Insider) ausgeht, zum Zweiten Angriffe von außen gegen die Cloud-Infrastruktur und zum Dritten Angriffe die von einer von einem Angreifer kontrollierten virtuellen Maschine ausgehen.

Dass Risiken durch *Insider-Angriffe* beim Cloud-Anbieter praxisrelevant sind, zeigt exemplarisch der Fall von CyberLynk, bei dem ein Mitarbeiter nach seiner Entlassung umfangreiches Datenmaterial eines Kunden löschen konnte (Nemani, 2011). Auch wenn meist davon auszugehen ist, dass Cloud-Anbieter bei der Auswahl und Sicherheitsüberprüfung von Mitarbeitern (insbesondere Administratoren, die

Zugriff auf die Cloud-Infrastruktur haben) angemessene Sorgfalt walten lassen, so sind derartige Vorfälle nicht auszuschließen.

Eine breite Angriffsfläche können auch *Management-Schnittstellen* der Cloud-Infrastruktur bieten. Beispielsweise konnten Somorovsky et al. (2011) in einer Analyse zeigen, dass sowohl die Schnittstellen von Amazons EC2- und S3-Diensten als auch die der verbreiteten Cloud-Verwaltungs-Software Eucalyptus Schwachstellen aufgewiesen haben, die sich durch Cross-Site-Scripting-Angriffe (XSS) ausnutzen ließen. Derartige Schwachstellen erlauben es einem Angreifer, die vollständige Kontrolle über virtuelle Maschinen eines Cloud-Kunden und unbeschränkten Zugriff auf gespeicherte Daten zu erlangen.

Auch von Angreifern, die eine *virtuelle Maschine* kontrollieren, kann ein Risiko für die Infrastruktur und für andere Cloud-Kunden ausgehen. Die Möglichkeit, Zugriff auf virtuelle Maschinen in der Cloud-Infrastruktur zu erhalten, besteht insbesondere in öffentlichen Clouds. Von einer virtuellen Maschine kann zum einen durch *VM-Escape-Angriffe* eine Gefahr ausgehen. Hierbei nutzt der Angreifer Schwachstellen des Hypervisors aus, um die normalerweise von der virtuellen Maschine garantierte Isolation zu durchbrechen und so direkten Zugriff auf die Infrastruktur zu erhalten. Zum anderen gibt es auch *Seitenkanalangriffe*, die direkt auf andere virtuelle Maschinen auf dem selben physischen Rechner zielen. Auf diese Art von Angriffen gehen wir vertieft in Abschnitt 6 ein.

Nur am Rande soll hier erwähnt werden, dass Vorteile von Cloud Computing auch durch Angreifer gezielt genutzt werden können. Neben der direkten Nutzung von Cloud-Ressourcen für Angriffe auf externe Parteien (z. B. DoS-Angriffe oder das Hosten von Malware oder Command&Control-Servern) lassen sich kostengünstig für den Angreifer Cloud-Ressourcen auch für rechenintensive Operationen wie beispielsweise Password Cracking verwenden.

Kontrollaufgabe 1: Bewertung Vor- und Nachteile

Firma A benötigt einen Dienst zur Terminplanung und möchte dafür eine öffentliche Cloud-Infrastruktur nutzen. Als Möglichkeiten stehen der Betrieb eines eigenen Terminplanungsdiensts auf Basis einer existierenden Open-Source-Software in einer IaaS-Cloud oder die Nutzung eines vom Cloud-Provider angebotenen SaaS-Dienst zur Verfügung. Vergleichen Sie diese beiden Möglichkeiten hinsichtlich Sicherheitsfragen.

K

4 Sichere Datenspeicherung und -verarbeitung in der Cloud

Zu den beschriebenen Sicherheitsrisiken gibt es in der Praxis eine Vielzahl an Strategien und Maßnahmen, um diese zu vermeiden bzw. zu verringern. Eine sehr wichtige Rolle spielen hierbei rechtliche und organisatorische Aspekte. Angefangen von der Fragestellung, wann eine Auslagerung von Diensten und Daten in eine Cloud überhaupt rechtskonform möglich ist bzw. welche Rahmenbedingungen dabei eingehalten werden müssen, bis hin zu Schadensersatzforderungen, falls ein Cloud-Nutzer durch vorsätzliches oder fahrlässiges Verhalten des Cloud-Anbieters durch Angriffe zu Schaden kommt, gibt es eine eine große Bandbreite von Einflussfaktoren. Auch eher organisatorische Aspekte wie Zertifizierungen und Audits auf Seiten des Cloud-Anbieters können im Rahmen einer Sicherheitsanalyse relevant sein.

Derartige nicht-technische Fragestellungen liegen aber außerhalb des Fokus dieses Lehrmoduls. An dieser Stelle wollen wir insbesondere den Stand der Forschung

hinsichtlich technischer Maßnahmen betrachten, welche als Schutz gegen verschiedene zuvor betrachtete cloudspezifische Risiken zum Einsatz kommen könnten. Hierzu werden wir zwei verschiedene Kategorien betrachten: die sichere Speicherung von Daten durch clientseitige Verschlüsselung und die sichere Verarbeitung von verschlüsselten (oder anderweitig hinsichtlich Geheimhaltung geschützten) Daten in der Cloud.

4.1 Sichere Datenspeicherung

Clientseitige Verschlüsselung

Wird eine (insbesondere öffentliche) Cloud nur zur Speicherung von Daten genutzt, so bietet clientseitige Verschlüsselung wirksamen Schutz gegen cloudseitige Angriffe. Werden alle Daten durch einen Schlüssel, der nur dem Cloud-Nutzer bekannt ist, bereits auf dem Client-Gerät verschlüsselt und die Daten nur in dieser Form an cloudbasierte Dienste weitergegeben, so wird ein weitreichender Schutz der Vertraulichkeit erreicht.

Ein einfaches Beispiel unter vielen ist die Möglichkeit, ein verschlüsseltes Dateisystem z. B. mittels Veracrypt auf einem cloudbasierten Storage-Dienst wie Dropbox abzulegen. Auch wenn clientseitige Verschlüsselung manchmal als die „eierlegende Wollmilchsau“ des Cloud-Computing gesehen wird, löst diese nicht alle relevanten Probleme.

- *Vertraulichkeit:* Vertraulichkeit der Daten wird uneingeschränkt erreicht, allerdings keine Vertraulichkeit von Metadaten, wie die Tatsache, dass/wann/von wem Daten gespeichert werden.
- *Integrität:* Verschlüsselung alleine bietet noch keine starken Integritätsgarantien, auch wenn eine gezielte Veränderung von Daten auf einen bestimmten Wert erschwert ist. Clientseitige Verschlüsselung lässt sich aber auch leicht mit zusätzlichem Integritätsschutz verbinden (wie Message Authentication Codes oder digitaler Signatur aller gespeicherten Daten). Aber auch dann ist nicht automatisch ein Schutz vor *Rollback-Angriffen* sichergestellt, bei denen der Anbieter einen (korrekt signierten) älteren Zustand als aktuellen präsentiert.
- *Verfügbarkeit:* Verfügbarkeit ist das größte durch Verschlüsselung ungelöste Problem. Werden Daten auf Seite des Anbieters durch Angriffe verändert oder gelöscht, so sind sie verloren.

Verfügbarkeit durch redundante Provider

Während Verschlüsselung eine Lösung für die Dimension der Vertraulichkeit ist, lassen sich die Herausforderungen hinsichtlich Integrität und Verfügbarkeit durch redundante Datenspeicherung auf mehreren Provider angehen.

Eine zentrale Herausforderung ist dabei, festzustellen, ob Daten ohne Veränderung noch verfügbar sind. Der naive Ansatz, Daten redundant z. B. bei mehreren Providern zu speichern, weist dabei einige Nachteile auf. Zum einen ist der hohe Overhead zu nennen. Sowohl bei der Übertragung als auch bei der Speicherung wird eine vielfache Kapazität verwendet, mit den entsprechend höheren Kosten. Des Weiteren wird eine Veränderung oder ein Verlust der Daten erst dann erkannt, wenn auf die Daten zugegriffen wird. Das heißt aber, dass ein Datenverlust bei Daten, auf die nur selten zugegriffen wird (z. B. Backup-Daten), erst sehr spät erkannt wird. Wird diesem Problem dadurch begegnet, dass die Daten periodisch z. B. täglich oder stündlich geprüft werden, entstehen hohe Kosten durch die Datenübertragung.

Eine vorteilhaftere Lösung für diese beiden Herausforderungen muss daher zwei komplementäre Mechanismen mit sich bringen: Als Erstes sind Verfahren notwendig, die einen Datenverlust durch *effiziente redundante Speicherung* ausgleichen können, so dass der Ressourcenbedarf sowohl für die Datenspeicherung als auch für die Übertragung minimiert werden kann. Als Zweites ist es erforderlich, *Datenverlust effizient und frühzeitig zu erkennen*.

Verfahren zur effizienten redundanten Speicherung sind bereits lange bekannt. Das vermutlich bekannteste Beispiel hierbei ist RAID (Redundant Arrays of Independent Disks) aus dem Bereich der Datenspeicherung auf Festplatten (Chen et al., 1994). In der Variante RAID-5 wird zu N Festplatten eine weitere Platte hinzugefügt, und Datenblöcke aus N Festplatten werden mit einem weiteren Paritätsblock versehen. Damit lässt sich der komplette Ausfall einer der $N+1$ Festplatten kompensieren. Der Overhead an Ressourcenbedarf beträgt dabei $1/N$, bei beispielsweise insgesamt 6 Festplatten ($N = 5$) also nur $1/5 = 20\%$. Wird dagegen die Tolerierung eines Totalausfalls einer Platte durch Spiegelung auf einer weiteren Platte sichergestellt, beträgt dieser Overhead 100%.

Neben RAID-5 gibt es weitere Ansätze, die es darüber hinaus ermöglichen, eine größere Anzahl an Ausfällen zu kompensieren. Eine Variante zur Tolerierung von zwei Ausfällen ist als RAID-6 bekannt, bei der zwei zusätzliche Festplatten für Paritätsinformationen verwendet werden. Allgemein lassen sich Erasure Codes und Codierverfahren wie Reed Solomon Codes verwenden, um Verfahren zur effizienten fehlertoleranten Speicherung von Daten zu implementieren. Für einen Überblick zu diesen Verfahren verweisen wir auf einen Artikel von Plank (2013). Derartige Verfahren haben heutzutage eine weite Verbreitung gefunden. Beispielsweise verwendet das Cloud-Storage-System Microsoft Azure Storage eine Variante von Erasure Codes, um Fehlertoleranz bei geringem Overhead zu erreichen (Huang et al., 2012).

Die zweite bereits genannte Herausforderung besteht darin, Datenverlust möglichst effizient und frühzeitig zu erkennen. Während Daten bei lokaler Speicherung – z. B. innerhalb eines Cloud-Rechenzentrums – aufgrund der vorhandenen leistungsfähigen Kommunikationsnetze einfach periodisch vollständig geprüft werden können, ist dies bei Verteilung der Daten auf mehrere Rechenzentren oder Cloud-Anbieter im Allgemeinen mit hohen Kosten verbunden. Einen Lösungsansatz für diese Herausforderung bietet das Forschungsgebiet der *Proofs of Retrievability (PoR)*. Ein PoR ist dabei einfach ausgedrückt ein Nachweis eines Storage-Anbieters gegenüber einem Nutzer, dass der Storage-Anbieter gespeicherte Daten korrekt vorliegen hat und bei Bedarf an den Nutzer übermitteln kann. Einen weiterführenden Überblick über die theoretischen Hintergründe lassen sich in der Arbeit von Bowers et al. (2009a) finden.

Exemplarisch greifen wir an dieser Stelle eine der ersten Forschungsarbeiten (Bowers et al., 2009b) heraus, welche die oben beschriebenen Ansätze zu einem effizienten und fehlertoleranten Cloud-Storage-System über mehrere Anbieter hinweg verbindet. In HAIL („HAIL: A High-availability and Integrity Layer for Cloud Storage“) werden Daten (analog zu RAID-5, das Daten auf mehrere Festplatten verteilt), mit einem auf Reed-Solomon-Codes basierenden *Dispersal Code* auf mehrere Cloud-Storage-Anbieter effizient verteilt. Ein PoR wird verwendet, um zu prüfen, ob die Daten bei allen Anbietern weiterhin verfügbar sind. Wird hierbei ein Datenverlust festgestellt, so werden die Daten des betroffenen Anbieters mit Hilfe des Dispersal Codes an einem neuen Speicherort wiederhergestellt.

Der von HAIL verwendete PoR wiederum ist eine Kombination von zwei Techniken: Zum einen wird die Verfügbarkeit von Daten probabilistisch durch *Spot Checking* geprüft. Dabei werden nicht alle Daten komplett geprüft, sondern nur

ein kleine, zufällig gewählte Teilmenge (für Details wird an dieser Stelle auf die Originalpublikation verwiesen). Dieses Verfahren eignet sich sehr gut, um den Verlust (oder die Verfälschung) einer großen Menge der Daten bei einem Provider zu erkennen. Sind beispielsweise 10% der Daten eines Anbieters verloren gegangen, so kann dies (unabhängig von der Gesamtmenge der Daten) im Mittel durch nur 10 „Spot Checks“ erkannt werden. Natürlich lassen sich durch dieses Verfahren keine Probleme erkennen, die nur einen kleinen Teil der Daten betreffen. Spot Checking wird bei HAIL daher durch eine redundante Codierung mit einem fehlerkorrigierenden Code innerhalb eines Providers kombiniert. Damit lassen sich dann alle „kleinen“ Fehler, die sich mit Spot Checking nur mit geringer Wahrscheinlichkeit erkennen lassen, korrigieren, so dass die Daten bei diesen Fehlern immer noch vollständig verfügbar sind.

4.2 Verarbeitung verschlüsselter Daten

Sollen Daten nicht nur gespeichert, sondern in der Cloud auch verarbeitet werden, so gestalten sich die technischen Schutzmaßnahmen schwieriger. Im Allgemeinen ist es notwendig, dass Software, die Daten in der Cloud verarbeiten soll, auch Zugriff auf die (nicht verschlüsselten) Daten haben muss, und diese daher grundsätzlich einem höheren Risiko insbesondere hinsichtlich Vertraulichkeit ausgesetzt sind.

In den letzten Jahren sind dennoch eine Reihe von Forschungsarbeiten entstanden, die sich mit dem Problem des technischen Schutzes der Vertraulichkeit von Daten, die in der Cloud verarbeitet werden, auseinandersetzen. Eine vertiefte Diskussion der einzelnen Methoden würde den Rahmen dieses Studienbriefes sprengen. Dieses Kapitel soll aber zumindest einen ersten Überblick über grundsätzliche Methoden geben. Diese lassen sich wie folgt grob in drei Kategorien einteilen:

Zunächst gibt es das große Gebiet der Verfahren, bei denen sensitive Informationen versteckt bzw. entfernt werden, bevor diese in eine öffentliche Cloud-Umgebung übertragen werden. Hierzu gehören alle *Anonymisierungs- und Pseudonymisierungstechniken*. Beispielsweise ist hier die statistische Auswertung von medizinischen Daten, bei denen personenbezogene Informationen wie Name und Geburtsdatum entfernt wurden, zu nennen. Auch die Verschleierung von Informationen (z. B. Ersetzen des genauen Geburtsdatums durch eine Alterklasse in 10-Jahres-Schritten oder das Entfernen des letzten Oktetts einer IP-Adresse) sowie die Aggregation von Daten (z. B. Übertragung der feingranularen Stromverbrauchs-Daten an einen Stromlieferanten nicht pro Haushalt, sondern nur als Summe über einen ganzen Straßenzug) ist hier zu nennen. Derartige Ansätze sind in einer Vielzahl von Varianten in der Praxis einsetzbar. Gerade bei Pseudonymisierungsmethoden konnte aber in der Vergangenheit in einigen Fällen gezeigt werden, dass Verfahren Schwächen aufweisen, die eine De-Anonymisierung ermöglichen. Beispielsweise konnten Narayanan und Shmatikov (2008) zeigen, dass sich anonymisierte Filmbewertungen von Netflix-Kunden mit nur wenig Zusatzinformationen über die Nutzer mit hoher Wahrscheinlichkeit deanonymisiert werden können. Eine entsprechende Sorgfalt und Sachverstand bei Verwendung von Anonymisierungstechniken sind daher unbedingt erforderlich.

Die zweite Kategorie sind Verfahren, bei denen Daten verschlüsselt gespeichert werden und ausgewählte Operationen auf diesen Daten ermöglicht werden. Ein sehr einfaches Beispiel ist ein Key-Value-Store, bei dem sowohl der Schlüssel als auch der Wert verschlüsselt gespeichert werden. Durch den Einsatz eines deterministischen Verschlüsselungsverfahrens kann auf diesen Key-Value-Store dann wie bei einem normalen Key-Value-Store zugegriffen werden. Es gibt eine Vielzahl von Forschungsarbeiten, die nach Strategien suchen, wie auf ähnliche Art kom-

plexere Operationen wie z. B. die Suche nach Keys über einen Substring oder eine Bereichsangabe ermöglicht werden können. Hierzu verweisen wird exemplarisch auf die Forschungsarbeit von Yuan et al. (2016).

Als weitere Variante ist die *homomorphe Verschlüsselung* zu nennen. Diese zielt darauf ab, im Idealfall beliebige Rechenoperationen auf verschlüsselten Daten vorzunehmen. Um diese mathematisch anspruchsvollen Varianten zu illustrieren, betrachten wir zunächst die einfachere Variante der partiell homomorphen Verschlüsselung anhand des RSA-Verschlüsselungsverfahrens ohne Padding. Bei RSA gibt es einen privaten Schlüssel der Form (d,n) und einen öffentlichen Schlüssel der Form (e,n). n ist dabei das Produkt zweier großer Primzahlen p,q. e ist eine zufällig gewählte Zahl mit $1 < e < (p-1)(q-1)$ und $e \perp (p-1)(q-1)$. Aus diesen Zahlen muss anfänglich der Entschlüsselungsexponent d berechnet werden. Ein Klartext t wird dann nach Formel 1.1 zu einem Ciphertext c verschlüsselt. Danach kann c durch die Operation nach Formel 1.2 wieder entschlüsselt werden.

$$c := E_{(e,n)}(t) = t^e \pmod{n} \tag{1.1}$$

$$t := D_{(d,n)}(c) = c^d \pmod{n} \tag{1.2}$$

Bei RSA handelt es um einen multiplikativen Homomorphismus, es gilt für zwei Werte t_1, t_2:

$$E(t_1) * E(t_2) = t_1^e * t_2^e \bmod n = (t_1 * t_2)^e \bmod n = E(t_1 * t_2). \tag{1.3}$$

Aus den verschlüsselten Werten von t_1 und t_2 kann damit der verschlüsselte Wert von $t_1 * t_2$ berechnet werden, ohne dass für diese Berechnung t_1 oder t_2 entschlüsselt werden müssen.

Bei der vollständigen homomorphen Verschlüsselung werden Verfahren konstruiert, bei denen zwei unterschiedliche Operationen (+, *) auf den verschlüsselten Daten ermöglicht werden, aus denen sich im Prinzip beliebige Operationen darstellen lassen. Für eine detailliertere Darstellung derartiger Verfahren wird hier lediglich auf die Arbeit von Gentry (2010) verwiesen. Wichtig ist aber festzuhalten, dass selbst nach signifikanten Verbesserungen der Verfahren in den letzten Jahren der Aufwand für diese Verfahren – sowohl für die Durchführung der Berechnung als auch für die Speicherung von Daten und kryptographischen Schlüsseln – immer noch mehrere Größenordnungen größer ist als bei einer Berechnung ohne homomorphe Verschlüsselung[1]. Auch wenn neuere Arbeiten zeigen, dass hier noch eine deutliche Verbesserung möglich ist, eignet sich homomorphe Verschlüsselung nach Stand der Technik nicht zur effizienten und preiswerten Durchführung von Berechnungen auf Cloud-Ressourcen.

Dennoch kann es sinnvolle Anwendungen geben, falls bei diesen die Rechenleistung nicht im Vordergrund liegt, aber dennoch mit Daten, die aus Vertraulichkeitsgründen verschlüsselt werden, eine Berechnung durchgeführt werden soll. Ein Beispiel dazu stellt die Addition von (verschlüsselten) Stimmen bei einem E-Voting-Verfahren dar.

[1] Bei der Arbeit von Gentry und Halevi (2011) hat ein privater Schlüssel mit einem zu 1024-bit-RSA vergleichbaren Sicherheitsniveau eine Größe von 2.25 GB.

5 Bedrohungen durch Koresidenz in der Cloud

Viele Angriffe insbesondere im IaaS-Servicemodell in öffentlichen Cloud-Umgebungen setzen voraus, dass ein Angreifer eine virtuelle Maschine auf dem selben physischen Rechner ausführen kann, auf dem sich auch das Ziel des Angriffs befindet. Neben Denial-of-Service-Angriffen sind hier vor allem Seitenkanalangriffe zu nennen, die im nächsten Kapitel genauer betrachtet werden. Es stellt sich daher die Frage, inwieweit ein Angreifer in der Lage ist, eine derartige Koresidenz einer virtuellen Maschine zu erreichen. Hierzu betrachten wir im Folgenden bekannte Ansätze aus Forschungsarbeiten und aktuelle Gegenmaßnahmen in Cloud-Infrastrukturen.

Immer noch gut geeignet, um das grundsätzliche Vorgehen eines Angreifers zu illustrieren, ist die bekannte Arbeit von Ristenpart et al. (2009), die erstmals die praktische Möglichkeit von gezielten Koresidenz-Angriffen detailliert aufgezeigt hat. Ausgegangen wird im Folgenden von einem externen Angreifer, der keinen Insider-Zugriff auf das Cloud-Management-System hat. Bevor ein solcher Angreifer ausgehend von einer Koresidenz eine andere VM angreifen kann, sind drei grundsätzliche Herausforderungen zu lösen:

- Kann man herausfinden, wo in der Cloud-Infrastruktur eine virtuelle Maschine (das Ziel des Angreifers) ausgeführt wird? *(Cloud-Kartographie)*
- Kann man feststellen, ob zwei virtuelle Maschinen auf dem selben physischen Rechner ausgeführt werden? *(Koresidenz-Test)*
- Kann ein Angreifer virtuelle Maschinen so erzeugen, dass sie auf dem selben physischen Rechner wie sein Ziel ausgeführt werden? *(Koresidenz-Angriff)*

Die folgenden Untersuchungen wurden von Ristenpart et al. in der Amazon-EC2-Infrastruktur durchgeführt. Seit dieser Untersuchung haben sich einige Aspekte dieser Infrastruktur verändert; darunter sind auch gezielte Mechanismen, die derartige Angriffe erschweren. Nichtsdestotrotz ist es von Vorteil, zunächst diese ersten praktikablen Angriffe etwas detaillierter zu betrachten. Auf neuere Entwicklungen gehen wir später ein.

5.1 Cloud-Kartographie

Ziel von Cloud-Kartographie ist es, den Ausführungsort einer virtuellen Maschine so einzugrenzen, dass das Ziel der Koresidenz einfacher erreicht werden kann. Dabei steht nicht unbedingt der exakte Ort im Vordergrund (welcher Einschub in welchem Rack etc.), sondern vielmehr die Frage, welche Parameter, die bei Erzeugung einer VM spezifiziert werden können, eine Koresidenz einfacher bzw. wahrscheinlicher machen.

Im Beispiel von Amazon-EC2 existieren im Wesentlichen drei Parameter, die die Erzeugung der virtuellen Maschine beeinflussen können:

- die Region *(region)*, welche unterschiedlichen Standorten von Rechenzentren entspricht;
- die Verfügbarkeitszone *(availability zone)*, welche unabhängigen, physisch getrennten Zonen innerhalb einer Region entspricht;
- der Instanztyp *(instance type)* der verwendeten virtuellen Maschine.

Ristenpart et al. haben die Tatsache ausgenutzt, dass jede virtuelle Maschine neben einer öffentlichen IP-Adresse auch eine interne private IP-Adresse besitzt, die sich über einen Amazon-internen DNS-Dienst aus der öffentlichen IP-Adresse

ermitteln lässt. Ausgehend von (damals verwendeten) 3 Verfügbarkeitszonen und 5 Instanztypen konnte experimentell ermittelt werden, dass die Adressbereiche, die für private IP-Adressen verwendet werden, von den verwendeten Parametern (sowohl Verfügbarkeitszone als auch Instanztyp) abhängen.

Auf Grundlage einer experimentell ermittelten Korrelation zwischen internen IP-Adressen und Instanz-Parametern kann ein Angreifer damit wie folgt vorgehen:

- Aus der öffentlichen IP-Adresse des Angriffsziels wird die EC2-interne IP-Adresse ermittelt.
- Aus der internen IP-Adresse werden die Parameter (Region, Verfügbarkeitszone, Instanztype) des Angriffsziels ermittelt.
- Die Platzierung der Angriffs-VM wird durch die Wahl der entsprechenden Parameter so beeinflusst, dass die Koresidenz wahrscheinlicher wird.

Das beschriebene Vorgehen war in dieser Form nur möglich, weil die EC2-Infrastruktur hierfür geeignete Mechanismen bereitstellte. Eine derartige Kartographie lässt sich durch Cloud-Provider deutlich erschweren, wenn zum einen die internen IP-Adressen von Cloud-Kunden anderen Kunden nicht zugänglich gemacht werden und zum anderen, wenn keine statische Abbildung von internen IP-Adressen auf Instanztypen verwendet wird.

5.2 Koresidenz-Tests

Unabhängig davon, wie gut sich die Platzierung mit Mechanismen, wie sie gerade beschrieben worden sind, eingrenzen lässt, wird sich eine Koresidenz nicht vollständig deterministisch erreichen lassen. Nach dem Erzeugen einer virtuellen Maschine ist also zu prüfen, ob das Ziel der Koresidenz tatsächlich erreicht wurde.

Drei einfache Metriken werden hierzu von Ristenpart et al. beschrieben:

- Identische IP-Adressen der Dom0-VM
- Niedrige Round-Trip-Zeiten im Netz
- Numerisch eng zusammenliegende interne IP-Adressen

All diese Metriken habe sich als sehr gut geeignet herausgestellt (siehe Exkurs 1 zur Prüfung, ob eine Koresidenz-Check-Metrik korrekt funktioniert).

Nichtsdestotrotz ist anzumerken, dass zu vielen derartigen Tests wirksame Gegenmaßnahmen des Cloud-Providers möglich sind. Insbesondere lassen sich interne IP-Adressen zum Beispiel durch geeignete virtuelle Netze so zu virtuellen Maschinen zuordnen, dass sie keine direkten Rückschlüsse auf Koresidenz ermöglichen. Tests, die auf Laufzeitmessungen im Netz beruhen, sind möglicherweise problematischer, lassen sich aber auch durch erzwungene Nachrichtenverzögerungen verhindern (wobei dies natürlich einen Performance-Nachteil mit sich bringt und daher ggf. nicht erwünscht sein kann). Mit etwas mehr Aufwand lassen sich aber

auch Seitenkanalangriffe nutzen, um Koresidenz zu prüfen. Mehr dazu in Abschnitt 6.

E

Exkurs 1: Validierung von Koresidenz-Checks

Eine Prüfung von Koresidenz-Checks ist mit Covert-Channel-Tests möglich, die nur bei Koresidenz erfolgreich funktionieren. Für weitere Details zu Covert Channels wird auf Kapitel 6 verwiesen. Die von Ristenpart et al. verwendete Methode nutzt dabei den Zugriff auf eine lokale Festplatte aus. Ausgangspunkt sind zwei virtuelle Maschinen, deren Koresidenz geprüft werden soll. Eine virtuelle Maschine greift kontinuierlich lesend auf die lokale Festplatte zu, während die zweite virtuelle Maschine in zeitlichen Abständen nach einem vorgegebenen Muster Schreibzugriffe durchführt. Wenn die erste virtuelle Maschine Änderungen in den Zugriffszeiten beobachtet, die mit dem Schreibmuster korrelieren, so kann auf eine Koresidenz geschlossen werden. Die zuvor beschriebenen Koresidenz-Checks können validiert werden, indem deren Aussage mit der eines Covert-Channel-Tests verglichen wird.

5.3 Koresidenz-Angriffe

Auf Grundlage der bisher beschriebenen Mechanismen zur Cloud-Kartographie und Koresidenz-Prüfung lässt sich zunächst eine einfache Strategie für einen Angreifer ermitteln:

- Der Angreifer ermittelt mit Hilfe der Techniken aus Abschnitt 5.1 für das gewünschte Ziel dessen Verfügbarkeitszone und Instanztyp.
- Der Angreifer erzeugt periodisch neue virtuelle Maschinen mit gleicher Zone und gleichem Instanztyp und prüft jeweils mit Hilfe der Techniken aus Abschnitt 5.2, ob sich die erzeugte virtuelle Maschine auf demselben physischen Host wie das Angriffsziel befindet.
- Falls nicht, wird die neue virtuelle Maschine gelöscht und der vorherige Schritt wiederholt.

Eine praktische Evaluation der Durchführbarkeit eines solchen Angriffs findet sich ebenfalls bei Ristenpart et al. (2009). Für manche Instanztypen (m1.xlarge und c1.xlarge) waren alle Koresidenz-Versuche erfolglos, was ein Indiz sein kann, dass bei xlarge-Instanzen nur eine einzige virtuelle Maschine auf dem physischen Host platziert wird und somit eine Koresidenz nicht möglich ist. Für kleine Instanzen (Instanztyp m1.small) konnten die Autoren dagegen in 18 Tage laufenden Experimenten für 8.4% aller Ziel-Hosts eine Koresidenz erfolgreich erreichen.

Neben dieser sehr primitiven Variante sind auch andere Strategien denkbar. Ristenpart et al. (2009) beschreiben die Beobachtung, dass zwei virtuelle Maschinen, die in zeitlich kurzem Abstand erzeugt werden, mit hoher Wahrscheinlichkeit auf dem selben physischen Rechner platziert werden. Dies lässt sich ausnutzen, wenn sich durch Beobachten des Ziels eine Neuerstellung der virtuellen Maschine bei Wartungsarbeiten erkennen lässt.

Zudem kommen in cloudbasierten Diensten oft Auto-Scaling-Mechanismen zum Einsatz, die bei hoher Last bedarfsgerecht weitere Instanzen einer VM erstellen. Durch Lasterzeugung über eine öffentliche Schnittstelle des Zielsystems kann der Angreifer damit gezielt die Erzeugung einer virtuellen Maschine auslösen. Mit derartigen Optimierungen konnten Ristenpart et al. die Wahrscheinlichkeit, erfolgreich Koresidenz zu erreichen, auf über 40% steigern.

5.4 Neuere Untersuchungen

In einer neueren Forschungsarbeit untersuchen Xu et al. (2015), wie sich derartige Koresidenz-Angriffe heutzutage durchführen lassen. Dabei zeigte sich, dass Amazon seit der Veröffentlichung der Arbeit von Ristenpart et al. (2009) einige Veränderungen an der Infrastruktur vorgenommen hat, die Koresidenz-Angriffe erschweren. So ist beispielsweise die IP-Adresse der Dom0 für virtuelle Maschinen nicht mehr sichtbar. Koresidenz-Prüfungen auf Basis von Dom0-IP-Adresse oder Zahl der Netz-Hops sind damit nicht mehr möglich. Auch werden heutzutage Instanzen von verschiedenen Instanztypen auf physischen Rechnern gemischt, so dass die Maßnahmen aus Abschnitt 5.1 weniger effizient sind.

Nichtsdestotrotz zeigte sich, dass es weiterhin möglich ist, gezielte Koresidenz mit vertretbaren Aufwand zu erreichen. Für Details verweisen wir an dieser Stelle auf die Original-Publikation. Insgesamt lässt sich aber festhalten, dass es mit nicht vernachlässigbarer Wahrscheinlichkeit einem Angreifer in einer öffentlichen Cloud möglich ist, eine virtuelle Maschine auf dem selben Rechner wie ein gewünschtes Ziel zu erzeugen. Die Kosten für den Angreifer (für die Nutzung von Cloud-Ressourcen) beliefen sich dabei nach Xu et al. (2015) im Allgemeinen jeweils auf weniger als 100 US$.

Ausgeschlossen werden können Koresidenz-Angriffe nur dann, wenn der Cloud-Provider Angebote bereitstellt, die eine Ausführung von virtuellen Maschinen auf einer physischen Maschine geteilt mit anderen Kunden ausschließt, wie das beispielsweise bei Amazon EC2 Dedicated Instances oder Dedicated Hosts der Fall ist.

6 Seitenkanalangriffe in der Cloud

Koresidenz stellt die Grundlage für Seitenkanalangriffe in der Cloud dar. Dieses Kapitel wird nun die Thematik der Seitenkanalangriffe vertieft betrachten. Dabei unterscheiden wir zwischen den beiden Konzepten der verdeckten Kanäle und der Seitenkanäle.

6.1 Grundbegriffe und Grundlagen

Eine etablierte Definition von *verdeckten Kanälen* stammt von Lampson, der diese als „those not intended for information transfer at all, such as the service program's effect on the system load“ (Lampson, 1973) beschreibt (siehe auch Definition 4).

Definition 4: Verdeckter Kanal (covert channel)

Bei einem verdeckten Kanal erlangen zwei Prozesse die Fähigkeit, Informationen auf einem nicht dafür vorgesehenen Weg auszutauschen.

D

Bei vielen klassischen Beispielen von verdeckten Kanälen werden Informationen dabei in einen existierenden Kommunikationskanal eingebettet. Hierzu zählt unter anderem das große Gebiet der Steganographie. Ein Beispiel eines verdeckten Kanals zwischen A und B ist die Übertragung eines Bildes, in dem (steganographisch) nicht sichtbar weitere Informationen eingebettet sind. In diesem Fall ist zwar der primäre Informationsaustausch (das Bild) sichtbar, der sekundäre Austausch (die eingebetteten Information) aber unsichtbar.

In diesem Modul betrachten wir vor allem verdeckte Kanäle, die nicht in einem vorhandenen Kommunikationsmechanismus eingebettet sind. Hierzu ein einfaches Beispiel: Auf einem Android-basierten Smartphone könnten Apps miteinander

kommunizieren, indem sie die Lautstärke-Einstellungen verändern und auslesen. Eine maliziöse App, die nicht die Berechtigung besitzt, Daten über das Netz zu versenden (beispielsweise eine einfache Taschenlampen-App, die nur Zugriffsrechte auf die Kamera besitzt, um das Licht zu aktivieren), könnte einen solchen Weg wählen, um so Daten (z. B. das Kamerabild) an eine weitere App weiterzugeben.

Für cloudbasierte Anwendungen sind in der Regel cloudspezifische verdeckte Kanäle praktisch kein relevantes Thema. Solange eine cloudbasierte Anwendung in der Lage ist, über eingehende oder ausgehende Netzverbindungen Informationen auszutauschen, was nahezu immer der Fall sein wird, ist die Verwendung von verdeckten Kanälen für einen Angreifer irrelevant. Erst wenn die Kommunikation der Anwendung strikt reglementiert und überwacht wird, gewinnen verdeckte Kanäle an Praxisrelevanz. Nichtsdestotrotz sind die technischen Möglichkeiten im Fokus von Forschungsarbeiten. Exemplarisch wird hier auf eine neuere Arbeit von Liu et al. (2015) verwiesen, bei der ein verdeckter Kanal zwischen virtuellen Maschinen auf einem Rechner mit einer Kapazität von etwas 1Mbit/s beschrieben wird.

Eine weitere Bedeutung kommt verdeckten Kanäle im Zusammenhang mit den bereits beschriebenen Koresidenz-Angriffen zu. Wie beschrieben erfordern Koresidenz-Angriffe immer eine Heuristik, mit der sich eine erfolgreiche Koresidenz erkennen lässt. Verdeckte Kanäle lassen sich nun nutzen, um zu prüfen, ob eine Koresidenz-Heuristik korrekt funktioniert: Ist zwischen zwei von einem Benutzer erzeugten Instanzen ein hostbasierter verdeckter Kanal möglich, so lässt sich dies als Nachweis der Koresidenz verwenden. Dies kann mit der Aussage der verwendeten Heuristik verglichen werden.

Ein einfacher verdeckter Kanal wird beispielsweise in dem Artikel von Ristenpart et al. (2009) beschrieben: Um ein „1“-bit zu senden, liest der Sender zufällige Blöcke von einem mit dem Empfänger gemeinsam genutzten physischen Speichermedium (z. B. eine lokale Festplatte). Um ein „0“-bit zu senden, macht der Sender nichts. Der Empfänger misst die Zugriffszeit beim Lesen von Daten vom gleichen Medium. Eine längere Zugriffszeit bedeutet „1“, eine kürzere Zugriffszeit „0“. Mit diesem Verfahren wurde von Ristenpart et al. 5 bit Informationen innerhalb 10 Sekunden übertragen (0,2 bps).

Von verdeckten Kanälen zu unterscheiden sind Seitenkanäle, die wir wie folgt definieren:

D

Definition 5: Seitenkanal (side channel)

Bei einem Seitenkanal-Angriff gewinnt ein Prozess über einen zweiten Prozess Informationen, die er nicht erhalten sollte.

Der Unterschied zwischen verdecktem Kanal und Seitenkanal besteht also darin, dass bei ersterem die Informationsquelle aktiv bei der Informationsübertragung mitwirkt, während bei zweiterem Informationen aus einer Quelle extrahiert werden, ohne dass dies der Quelle bewusst sein muss.

Traditionelle Seitenkanalangriffe nutzen zur Extraktionen von Informationen aus dem Zielsystem insbesondere Daten über den Energieverbrauch, elektromagnetische Strahlung sowie das Zeitverhalten.

Energieverbrauch

Energiebasierte Seitenkanalangriffe sind schon seit langem bekannt, um unter anderem kryptographische Schlüssel aus Smartcards zu extrahieren. Beispielsweise beschreibt Mangard (2003) einen Angriff auf den geheimen Schlüssel bei einer in Software implementierten AES-Verschlüsselung auf einer Smartcard. In der Fachliteratur findet sich eine Vielzahl von derartigen Angriffen. Im Bereich von Cloud Computing sind diese in der Regel irrelevant, da ein Angreifer nicht in der Lage ist, den Energieverbrauch eines cloudbasierten Rechners zu ermitteln.

Elektromagnetische Strahlung

Jedes elektronische Gerät erzeugt in mehr oder weniger geringem Umfang elektromagnetische Abstrahlungen. Diese lassen sich ähnlich zum Energieverbrauch ausnutzen, um Information über ausgeführte Operationen zu extrahieren. Dies ist zum einen über die Abstrahlung von Peripheriegeräten und deren Verbindungskabeln möglich (beispielsweise ist schon lange bekannt, dass sich so Tastatureingaben oder der Bildschirminhalt (van Eck, 1985) rekonstruieren lassen), zum anderen auch über Ausstrahlungen, die innerhalb von Geräten erzeugt werden. Beispielsweise zeigen Goller und Sigl (2015), wie sich ein privater RSA-Schlüssel rekonstruieren lässt aus den Abstrahlungen eines Smartphones, wenn dieses eine RSA-Berechnung mittels des Square-and-Multiply-Verfahrens durchführt. Derartige Verfahren erfordern, dass sich der Angreifer in der Nähe (meist cm bis m) des Zielobjekts befindet, und sind daher ebenfalls ungeeignet für cloudbasierte Systeme.

Zeitverhalten

Auch die zeitlichen Abstände zwischen Ereignissen können ungewollt Informationen über durchgeführte Berechnungen o. Ä. preisgeben. Neben Angriffen ähnlich zu den bereits beschriebenen lassen sich hier Angriffe finden, die auch über Kommunikationsnetze durchgeführt werden können. Beispielsweise zeigen Song et al. (2001), wie sich aus dem verschlüsselten Datenaustausch von SSH zeitliche Abstände zwischen Tastendrücken bei Passworteingaben rekonstruieren lassen. Diese Information reicht zwar nicht aus, um ein Passwort direkt zu rekonstruieren, Song et al. zeigten aber, dass sich damit die Länge eines Passworts erraten lässt und durch geschickte Modellierung sich eine Brute-Force-Suche nach dem Passwort um den Faktor 50 beschleunigen lies. Derartige Angriffe sind grundsätzlich auch auf Kommunikation mit cloudbasierten Diensten möglich, sie nutzen aber keine cloudspezifischen Eigenschaften aus.

6.2 Seitenkanäle bei der Erzeugung von Zufallszahlen

In diesem Abschnitt greifen wir zunächst ein einfaches Beispiel auf, das illustriert, wie für ein System, das auf einem dedizierten physischen Rechner als sicher angesehen werden kann, in der Cloud, zusammen mit sehr einfachen Seitenkanälen, neue Gefährdungen entstehen können.

Dazu betrachten wir zunächst, wie in einem Linux-basierten System vom Betriebssystem-Kernel Zufallszahlen generiert werden (siehe Abbildung 1).

Kernstück des Linux-Zufallszahlengenerators ist ein kryptographisch sicherer Pseudozufallszahlengenerator (CS-PRNG), der aus einem Anfangswert (Seed) eine deterministische, aber zufällig erscheinende Bitfolge generiert. Dieser CS-PRNG ist mit einem Entropie-Pool verbunden, der zum einen für den Anfangswert sorgt (insbesondere wird hierfür die Uhrzeit des Systemstarts sowie ein Seed, das vor dem Herunterfahren in einer Datei gespeichert wurde, verwendet), zum anderen aber auch zusätzlich kontinuierlich mit „zufälligen“ Werten angereichert wird.

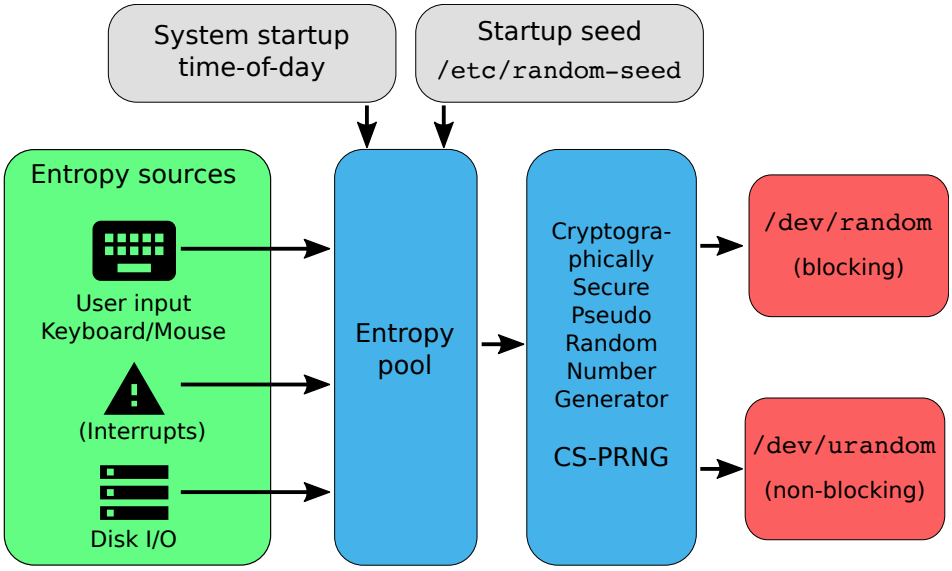


Abb. 1: Erzeugung von Zufallszahlen im Linux-Kernel

In einem dedizierten System kann man davon ausgehen, dass der Zustand des Entropie-Pools nicht vorhersehbar ist. Beim Systemstart wird die Systemzeit mit einem Wert aus der Datei `/etc/random-seed` kombiniert, die beim Herunterfahren des Systems mit einem zufälligen Wert versehen wird. Selbst wenn ein Angreifer in Kenntnis des genauen System-Startzeitpunkts kommt, kann er den Wert aus dieser Datei nicht vorhersehen.

Anders kann es nun aussehen, wenn ein Linux-basiertes System in der Cloud verwendet wird, insbesondere dann, wenn mehrere Instanzen von identischen Virtual-Machine-Images erzeugt werden. Bei den Entropiequellen sind bei einem cloudbasierten Server keine Nutzereingaben (per Maus oder Tastatur) vorhanden; Disk I/O bei mehreren Starts vom gleichen Image sind jeweils sehr ähnlich und daher ggf. bis auf eine sehr kleine Ungewissheit vorhersehbar. Stammt nun das „startup seed“ aus einem öffentlichen VM-Image, welches für mehrere Instanzen der VM verwendet wird, so ist dieses auch jeweils identisch.

Nehmen wir nun an, dass über Seitenkanäle die Information zur Verfügung steht, (a) welches VM-Image verwendet wird und (b) wann die VM gestartet wurde, dann ist der Entropie-Pool direkt nach dem Start der VM (ggf. abgesehen von einer sehr kleinen Restunsicherheit) vollständig vorhersehbar, und damit können auch die vom Pseudo-Zufallszahlengenerator gelieferten Zahlen vorhersehbar sein.

Werden diese für sicherheitsrelevante Zwecke eingesetzt, so kann dies ein Angreifer durchaus leicht ausnutzen. Zum Beispiel kann der Zufallszahlengenerator verwendet werden, um einen Session Key zur symmetrischen Verschlüsselung zu erzeugen. Kann ein Angreifer den mit diesem Session Key verschlüsselten Netzverkehr aufzeichnen und den verwendeten Session Key erraten, so kann er die verschlüsselte Kommunikation mitlesen. Eine kleine Ungewissheit lässt sich durch Probieren mehrerer Möglichkeiten kompensieren.

Ein potentiell noch folgenreicherer Angriff betrifft Authentifizierungsmechanismen in Web-basierten Diensten. Weit verbreitet ist hier ein Mechanismus, mit dem ein Nutzer sein eigenes Passwort automatisiert zurücksetzen kann. Eine Authentisierung des Nutzers erfolgt dabei durch die Zusendung einer E-Mail oder einer Textnachricht (SMS), die eine zufällig generierte Information enthält. Ist ein Nutzer in Kenntnis dieser Information, gilt er als authentisiert und kann sein Passwort zurücksetzen. Die Sicherheit dieses Verfahrens beruht darauf, dass der Angreifer

nicht in der Lage ist, die verwendete geheime Information zu erraten. Oben beschriebene Schwachstelle bei der Zufallszahlen-Erzeugung ermöglicht genau dies, und damit die Übernahme von beliebigen Benutzer-Accounts.

6.3 Cachebasierte Seitenkanäle im PaaS-Modell

Von besonderer Bedeutung in (insbesondere öffentlichen) Cloud-Umgebungen sind in der Praxis cachebasierte Seitenkanalangriffe. Nachdem bereits in Abschnitt 5 diskutiert wurde, dass einem Angreifer Koresidenz-Angriffe mit einer nicht zu vernachlässigenden Erfolgswahrscheinlichkeit möglich sein können, stellt sich nun die Frage, ob sich diese Koresidenz weiter ausnutzen lässt.

Exemplarisch greifen wir hierfür zunächst eine Arbeit von Zhang et al. (2014) heraus, welche dies an mehreren Beispielen für PaaS-Cloud-Umgebungen untersucht. Dabei gehen die Autoren davon aus, dass einzelne Nutzerinstanzen durch Container-Virtualisierung voneinander getrennt werden. Bei Container-Virtualisierung erfolgt die Trennung der Nutzer nicht durch einen Hypervisor, sondern durch das Betriebssystem.

Flush&Reload-Angriffe

Bei Flush&Reload-Angriffen wird davon ausgegangen, dass der Angreifer und das Ziel (Opfer) einen Speicherblock gemeinsam nutzen. Dies ist bei PaaS-Umgebungen oft dann der Fall, wenn Angreifer und Ziel identische Bibliotheken verwenden, die vom zugrunde liegenden Betriebssystem nur einmal in den Speicher geladen werden.

Der gemeinsame Speicherblock, im Folgenden auch *Chunk* genannt, besitzt die Größe einer Cacheline (beispielsweise 64 Byte) und ist entsprechend der Blockgröße im Speicher ausgerichtet (d. h. die Adresse des Blockanfangs liegt bei einem ganzzahlig Vielfachen der Cacheline-Größe). Ein Flush&Reload-Angriff besteht aus den folgenden grundlegenden Funktionsblöcken:

- Flush: Mit Hilfe der CPU-Instruktion `clflush` werden eine bzw. mehrere Chunks aus allen Caches der Cache-Hierarchie einschließlich des Last-Level-Caches (LLC) entfernt.
- Flush-Reload-Intervall: Der Angreifer wartet für ein bestimmtes Zeitintervall, während das Opfersystem weiter ausgeführt wird und ggf. Seiten in den Last-Level-Cache lädt.
- Reload: Der Angreifer misst die Zeit, die notwendig ist, um die zuvor entfernten Chunks zu lesen. Während eine kurze Zugriffszeit darauf hinweist, dass das Ziel seit dem letzten Flush auf diesen Chunk zugegriffen hat (und sich dieser dadurch im LLC befindet), deutet eine lange Zugriffszeit darauf hin, dass dies nicht der Fall ist.

Die beschriebenen Abläufe sind in Abbildung 2 nochmals illustriert. Falls der Zielprozess und der Angreiferprozess auf dem gleichen CPU-Core ausgeführt werden, so kann dieser Angriff nicht nur wie dargestellt auf dem LLC durchgeführt werden, sondern auch auf einem darüber liegenden Cache (L1 oder auch bei mehreren Cache-Ebenen L2).

Grundsätzlich liefert ein Flush&Reload-Angriff dem Angreifer Informationen darüber, ob ein Zielsystem auf ein Chunk (bzw. eine Menge von Chunks) zugegriffen hat. Dabei gibt es allerdings eine Menge an Unsicherheiten: Der Zugriff des Zielsystems kann sich zum Beispiel mit der Reload-Operation oder dem darauffolgenden

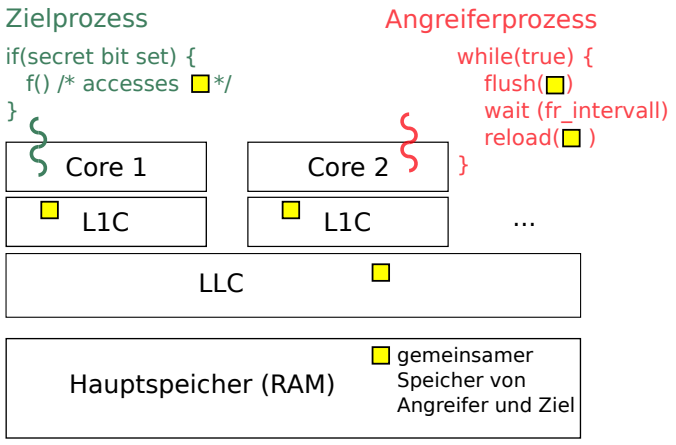


Abb. 2: Illustration der Funktionsweise von Flush&Reload-Seitenkanalangriffen

Flush überlappen und so unerkannt bleiben („false negative“). Der Zugriff kann ggf. auch von einem anderen als dem Zielprozess erfolgt sein („false positive“). Für eine detailliertere Diskussion von derartigen Unsicherheiten wird auf Zhang et al. (2014) verwiesen.

Kontrollfluss-Rekonstruktion

Die Fähigkeit, den Zugriff auf einzelnen Chunks innerhalb eines Zielsystems (trotz einer gewissen Rest-Unsicherheit) nachvollziehen zu können, bietet die Grundlage, um den Kontrollfluss (bzw. zumindest Teile des Kontrollflusses) innerhalb des Zielsystems nachvollziehen zu können und daraus ggf. für den Angreifer interessante Informationen ableiten zu können.

Die Autoren (Zhang et al., 2014) beschreiben ein abstraktes Modell basierend auf Zustandsautomaten, mit denen sich ein bestimmter Ausführungspfad im Zielsystem durch eine Folge von Chunk-Zugriffen nachvollziehen lässt. Ein solches Modell des Verhaltens des Zielsystems lässt sich sowohl allein durch Beobachtung des Verhaltens erstellen, als auch noch einfacher auf Basis einer Quelltext-Analyse des Zielsystems, falls dieser verfügbar ist.

Beispiel 1: Koresidenz-Erkennung

Als einfachstes Beispiel, wie sich Informationen mittels Cache-Angriffe ermitteln lassen, dient eine Koresidenz-Erkennung, d. h. es wird lediglich ermittelt, ob das Ziel tatsächlich auf demselben Rechner ausgeführt wird. Zhang et al. (2014) haben dazu durch manuelle Analyse herausgefunden, dass bei einem fehlgeschlagenem Anmeldeversuch beim Webshop-System Magenta die beiden Funktionen `cmlXPathNodeSetSort` und `php_session_start()` nacheinander ausgeführt werden, während dieses Abfolge von Funktionsaufrufen sonst nicht häufig auftritt.

In Experimenten sendete der Angreifer einen entsprechenden fehlschlagenden Anmeldeversuch an die öffentliche Schnittstelle des Ziels und konnte dann erfolgreich diese Folge von Funktionen tatsächlich per Flush&Reload-Angriff erkennen. Dieses erste Beispiel zeigt grundsätzlich, dass über cachebasierte Seitenkanäle Informationen über das Zielsystem gewonnen werden können. Allerdings stellt die hier gewonnene Information (Koresidenz) für sich noch kein nennenswertes Risiko dar.

Beispiel 2: Sensitive Benutzerdaten

In einem zweiten Beispiel soll nun eine etwas weitergehende Gewinnung von Informationen aus dem Zielsystem betrachtet werden. Wieder dient als Beispiel ein Online-Webshop in einer öffentlichen PaaS-Cloud. Es wird zusätzlich angenommen, dass der Angreifer z. B. durch einen Cross-Site-Angriff einen Request ausgehend von einem angemeldeten Webshop-Nutzer auslösen kann. Als sensitive Information über den Nutzer soll die Anzahl der Elemente im Einkaufswagen ermittelt werden.

Eine Analyse des Shop-Systems ergibt, dass es eine Funktion gibt, die beim Anzeigen des Einkaufswagen pro Element einmal aufgerufen wird. In anderem Kontext wird diese Funktion normalerweise nicht aufgerufen. Es konnte von Zhang et al. (2014) experimentell gezeigt werden, dass der Angreifer durch cachebasierte Nachverfolgung von Aufrufen dieser Funktion die exakte Anzahl von Elementen im Einkaufswagen sehr zuverlässig ermitteln konnte.

Auch in diesem zweiten Beispiel ist sicher anzumerken, dass diese Information (Anzahl der Elemente im Einkaufswagen) im Allgemeinen keinen großen Schaden verursachen wird. Nichtsdestotrotz zeigt das Beispiel eindrucksvoll, dass sich sensitive Daten aus dem Zielsystem mittels cachebasierter Seitenkanäle extrahieren lassen.

Beispiel 3: Passwort-Rücksetzung

Noch eindrucksvoller ist ein drittes Beispiel, das ebenfalls mit der Magenta-Webshop-Software getestet wurde: Um es Nutzern zu ermöglichen, ihr Passwort zurückzusetzen, bietet die Software die Möglichkeit, den Nutzern per E-Mail eine URL zum Zurücksetzen des Passworts zuzusenden. Diese URL enthält eine Zufallszahl, und nur mit deren Kenntnis ist ein Zurücksetzen des Passworts möglich.

Ein ähnliches Szenario wurde bereits zuvor in Abschnitt 6.2 beschrieben. Im nun konkret betrachteten Fall kommen aber geringfügig andere Randbedingungen zu tragen. Es wird nicht der Zufallszahlengenerator des Linux-Kernels verwendet, sondern der einer PHP-Laufzeitumgebung. Diese erzeugt Zufallszahlen mit Hilfe eines Pseudozufallszahlengenerators (PRNG), basierend auf einem Initialwert, der beim Start eines Prozesses aus Prozess-ID (PID) und aktueller Uhrzeit abgeleitet wird.

Ziel des Angriffes ist es nun, den Initialisierungswert des Zufallszahlengenerators sowie die Anzahl der seit Initialisierung erfolgten Zufallszahlen zu verfolgen. Durch den cachebasierten Seitenkanal lässt sich zunächst feststellen, wann genau das Zielsystem den aktuellen Zeitstempel zur Initialisierung ermittelt. Die PID lässt sich zwar nicht über einen Seitenkanal direkt auslesen, aber der Angreifer kann einfach sein eigenes Passwort zurücksetzen, und so auf Basis des bekannten Zeitstempels und dem ihm zugesendeten Link durch eine Brute-Force-Suche die PID ermitteln. Danach kann der Angreifer das Zurücksetzen des Passworts für einen anderen Cloud-Nutzer initiieren, aus Zeitstempel und PID die eigentlich geheime Information, die dem Cloud-Nutzer per E-Mail zugesendet wird, erraten und damit für einen anderen Nutzer das Passwort auf einen vom Angreifer gewählten Wert setzen.

Zusammenfassung Flush&Reload-Angriffe

Die genannten Beispiele zeigen, dass Flush&Reload-Angriffe grundsätzlich ein Risiko darstellen können. Natürlich ist zu berücksichtigen, dass ein erfolgreicher

Angriff einen signifikanten Aufwand erfordert. In den einfachsten Fällen (siehe Beispiel 1 oder 2) gewinnt der Angreifer nur minimale Informationen, so dass es fraglich scheint, ob derartige Angriffe in der Praxis zu finden sind. Kombiniert mit anderen Schwachstellen (siehe Beispiel 3) lassen sich aber auch Angriffe rekonstruieren, mit denen Benutzeraccounts komplett übernommen werden können oder mit denen private Schlüssel des Servers extrahiert werden können – mit entsprechenden Schadenspotential.

6.4 Cachebasierte Seitenkanäle im IaaS-Modell

Im IaaS-Modell sind cachebasierte Seitenkanalangriffe wesentlich schwieriger durchzuführen. Die gerade beschriebene Methode von Flush&Reload-Angriffen ist nur möglich, wenn Angreifer und Ziel über einen gemeinsamen Speicher (wie z. B. eine shared library) verfügen. Bei IaaS ist dies nur in sehr seltenen Fällen der Fall (wenn beispielsweise der Hypervisor eine Hauptspeicher-Deduplizierung zwischen virtuellen Maschinen unterstützt). Im Folgenden soll anhand einer neuen Arbeit von Liu et al. (2015) aufgezeigt werden, dass dennoch Angriffsmöglichkeiten in IaaS-Umgebungen bestehen.

Zunächst lässt sich im IaaS-Modell feststellen, dass im Normalfall davon auszugehen ist, dass einzelne virtuelle Maschinen auf unterschiedlichen CPU-Kernen ausgeführt werden, und damit als Grundlage eines Seitenkanal vor allem der Last-Level-Cache, der von allen CPU-Kernen gemeinsam genutzt wird, in Frage kommt, während Angriffe basierend auf beispielsweise dem L1-Cache kaum zwischen virtuellen Maschinen im IaaS-Modell denkbar sind.

Grundlegender Prime&Probe-Angriff

Grundlage der von Liu et al. beschriebenen Vorgehensweise ist eine modifizierte Variante eines *Prime&Probe*-Angriffs (Osvik et al., 2006). Die Grundidee dabei ist folgende:

- Prime: Der Angreifer füllt ein (oder mehrere) Cache-Set komplett mit eigenen Daten (andere Inhalte des Cache-Sets werden dabei aus dem Cache entfernt).
- Prime-Probe-Intervall: Der Angreifer wartet eine (kurze) Zeitspanne ab, in der das Zielsystem möglicherweise auf den Speicher zugreift (und dadurch Einträge im Cache ersetzt).
- Probe: Der Angreifer greift auf die im ersten Schritt verwendeten Daten erneut zu und misst die Zugriffszeit. Die Probe-Operation stellt dabei zugleich die Prime-Operation für die nächste Iteration dar. Eine erhöhte Zugriffszeit deutet darauf hin, dass seit dem Prime-Vorgang das Zielsystem auf Speicheradressen zugegriffen hat, die auf das entsprechende Cache-Set abgebildet werden.

In Abbildung 3 ist die Funktionsweise eines Prime&Probe-Angriffs nochmals grafisch dargestellt.

Herausforderungen beim Last-Level Cache

Die erfolgreiche Durchführung eines Prime&Probe-Angriff auf dem Last-Level Cache ist keineswegs trivial. Zunächst stellt sich die Frage, ob sich das Verhalten des Zielsystems überhaupt auf Ebene des Last-Level Cache beobachten lässt. In der Praxis werden viele Speicherzugriffe bereits durch davor liegende Cache-Ebenen behandelt (L1/L2). Hier lässt sich allerdings die Eigenschaft eines *inklusiven Cache* –

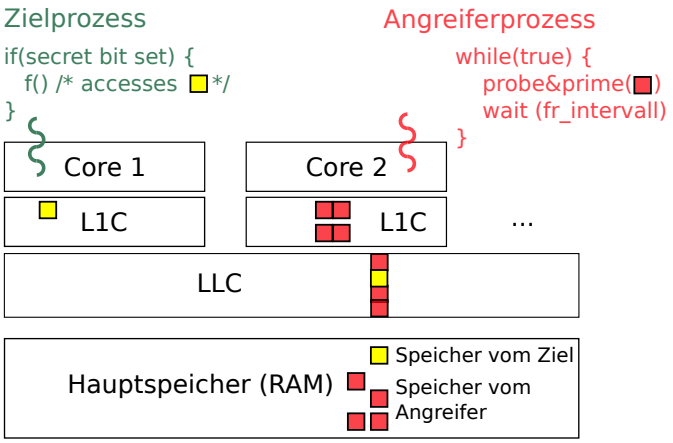


Abb. 3: Illustration der Funktionsweise von Prime&Probe-Seitenkanalangriffen

wie es z. B. auf aktueller Intel-Hardware üblich ist – nutzen: Eine Cache-Line, die in einem L1-Cache vorhanden ist, ist auch im L2 und Last-Level-Cache vorhanden.

Eine Herausforderung bei LLC-basierten Angriffen ist die Größe dieses Caches. Von einigen Autoren wurden bisher Prime&Probe-Angriffe beschrieben, die auf dem L1-Cache arbeiten. Dabei lässt sich bei jeder Iteration aus Prime und Probe der komplette L1-Cache untersuchen. Aufgrund der Größe des LLC (einige MiB statt einige KiB) ist dies mit feiner zeitlicher Auflösung nicht beim LLC durchführbar. Ein Angriff auf den LLC erfordert also zunächst, eine geeignete Menge an Cache-Lines auszuwählen, die von relevanten Zugriffen des Zielsystems verwendet werden.

Die Auswahl von relevanten Cache-Sets ist wiederum eine Herausforderung. Selbst wenn genau bekannt ist, welcher Code vom Zielsystem ausgeführt wird, ist dem Angreifer weder die Lage dieses Codes im virtuellen Adressraums des Zielsystems bekannt, noch die Abbildung dieses virtuellen Adressraums auf den physischen Adressraum. Als Lösung für dieses Problem verwenden Liu et al. (2015) den Ansatz, einzelne Cache-Sets auf für relevante Funktionen charakteristische zeitliche Zugriffsmuster zu untersuchen.

Darüber hinaus ist aber auch bereits die Prime-Operation eine Herausforderung. Während beim L1-Cache die Indizierung der Cache-Zugriffe über virtuelle Adressen erfolgt, geschieht dies beim LLC über physische Adressen. Das heißt, dass der Angreifer bei der Prime-Operation auf eine Menge von Speicherblöcken zugreifen muss, die (auf Basis ihrer physischen Adresse) alle auf das gleiche Cache-Set abgebildet werden. Dabei ist dem Angreifer innerhalb seiner virtuellen Maschine aber die Abbildung auf physische Adressen nicht bekannt.

Für Details, wie diese Herausforderungen gelöst werden können, wird an dieser Stelle auf die Originalpublikation von Liu et al. (2015) verwiesen.

Verwendung als verdeckte Kanäle

Für einen verdeckten Kanal legen wir folgende Vorgehensweise für den Sender fest: Es werden zwei Cache-Sets für den Kanal ausgewählt und der Sender wählt für beide Cache-Sets je eine darin liegende Cache-Line. Um nun ein „0“-bit zu übertragen, greift der Sender für ein bestimmtes Zeitintervall wiederholt auf die eine Cache-Line zu, für ein „1“-bit auf die andere.

Der Empfänger versucht, wechselweise über Prime&Probe zu prüfen, ob der Sender auf die eine oder die andere Cache-Line zugegriffen hat. Damit lässt sich

grundsätzlich die vom Sender erzeugte Bitfolge rekonstruieren. Fehlinterpretationen sind allerdings durch Störungen durch andere Speicherzugriffe, die auf das selbe Cache-Set abgebildet werden, möglich. Je länger das Zeitintervall pro Bit gewählt wird, desto weniger fallen diese Störungen ins Gewicht, desto geringer wird aber auch die Übertragungsrate. So konnten Liu et al. (2015) 1.2Mb/s bei einer Fehlerrate von 22% übertragen. Mit einer Fehlerrate von nur 1% konnte eine Übertragungsrate von 600Kb/s erreicht werden.

Verwendung als Seitenkanäle

Während die Verwendung als verdeckter Kanal meist keine große Praxisrelevanz aufweist, stellen Seitenkanalangriffe ein großes potentielles Risiko dar. Folgendes Beispiel aus Liu et al. (2015) zeigt, dass sich damit beispielsweise der private Schlüssel aus einer virtuellen Maschine extrahieren lässt.

Zunächst muss es hierzu einem Angreifer möglich sein, das Zielsystem dazu zu bringen, eine Berechnung mit dem privaten Schlüssel durchzuführen. Dies ist beispielsweise bei einem Webserver, der den Schlüssel zur Authentisierung bei jeder neuen Verbindung verwendet, trivial möglich.

Eine einfache Möglichkeit, RSA-Operationen mit dem privaten Schlüssel zu berechnen, ist mit Hilfe des Square-and-Multiply-Verfahren. Dabei wird in einer Schleife in jeder Iteration eine Quadrat-Operation (Square) durchgeführt. Zusätzlich erfolgt für jedes Bit des privaten Schlüssels eine Multiplikation (Multiply), falls das Bit den Wert 1 hat. Das heißt, dass das Zielsystem bei der Berechnung eine Folge von Square-Operationen durchführt, deren zeitliche Abstände von den Bits des privaten Schlüssels abhängen.

Zunächst muss der Angreifer bei einem Prime&Probe-Angriff das Cache-Set finden, das bei der Square-Operation verwendet wird. Dies lässt sich einfach durch eine Lernphase realisieren, in der sequentiell jeweils ein Cache-Set beobachtet wird, und dann geprüft wird, ob das charakteristische Muster (regelmäßig wiederkehrende Zugriffe mit jeweils einem von zwei möglichen Abständen zwischen den Zugriffen) vorhanden ist. Sobald dieses charakteristische Muster gefunden ist, lässt sich daraus direkt der Exponent des privaten Schlüssels ablesen.

Zusammenfassung Prime&Probe-Angriffe

Insgesamt zeigt das Beispiel von Liu et al. (2015), dass auch in Cloud-Umgebungen im IaaS-Modell, bei denen einzelne Kunden-VMs durch den Hypervisor voneinander isoliert sind und kein gemeinsamer Speicher vorliegt, cachebasierte Seitenkanäle grundsätzlich möglich sind, auch wenn der erforderliche Aufwand höher ist als bei Flush&Reload-Angriffen.

7 Übungsaufgaben

Ü

Übung 1: Grundbegriffe zu Risiko bei IT-Systemen

Erläutern Sie anhand eines Webservers, der im IaaS-Modell in einer öffentlichen Cloud ausgeführt wird, die Grundbegriffe *Bedrohungen*, *Schwachstellen* und *Auswirkungen* mit jeweils zwei Beispielen.

Übung 2: Sicherheitsvorteile und -nachteile in der Cloud

Vergleichen Sie die drei Service-Modelle IaaS, PaaS und SaaS in einer öffentlichen Cloud hinsichtlich ihrer Sicherheitsvorteile und -nachteile.

Ü

Übung 3: Clientseitige Verschlüsselung

Diskutieren Sie, wie sich clientseitige Verschlüsselung von Daten, die in einer öffentlichen Cloud gespeichert werden, auf die Vertraulichkeit, Integrität und Verfügbarkeit auswirkt.

Ü

Übung 4: Sichere Datenspeicherung und -verarbeitung in der Cloud

Angenommen, Sie erfassen personenbezogene Kundendaten zunächst im eigenen Rechenzentrum und möchten nun zur Weiterverarbeitung von Kundenaufträgen und zur statistischen Analyse der Daten einen Dienst in einer öffentlichen Cloud nutzen und dazu die kundenbezogenen Daten in der öffentlichen Cloud speichern.

Diskutieren Sie, inwieweit hierfür die *verschlüsselte Datenspeicherung*, die *Anonymisierung/Pseudonymisierung der Daten* sowie der Einsatz von *homomorpher Verschlüsselung* geeignet ist, um Vertraulichkeit, Integrität und Verfügbarkeit der Daten zu erreichen.

Ü

Übung 5: Proofs of Retrievability

Erklären Sie, was man unter dem Begriff *Proof of Retrievability (PoR)* versteht und erläutern Sie, wie dieser Mechanismus im System HAIL verwendet wird, um die Verfügbarkeit von Daten, die in öffentlichen Cloud-Infrastrukturen gespeichert werden, zu erhöhen.

Ü

Übung 6: Koresidenz in Cloud-Umgebungen

Erläutern Sie den Begriff *Koresidenz* in öffentlichen IaaS- und PaaS-Clouds und skizzieren Sie Möglichkeiten, wie ein Angreifer Koresidenz zu einem Zielsystem erreichen kann.

Ü

Übung 7: Seitenkanäle und verdeckte Kanäle

Beschreiben Sie den Unterschied zwischen Seitenkanälen und verdeckten Kanälen, und erläutern sie jeweils, inwiefern diese beiden Kanäle zu einem Risiko in einer öffentlichen Cloud führen können.

Ü

Ü

Übung 8: Cachebasierte Seitenkanäle

Beschreiben Sie die grundlegende Idee hinter Prime&Probe- sowie Flush&Reload-Angriffen. Wie unterscheiden sich beide Angriffe hinsichtlich der notwendigen Voraussetzungen für einen erfolgreichen Angriff?

Verzeichnisse

I. Abbildungen

Abb. 1: Erzeugung von Zufallszahlen im Linux-Kernel 22
Abb. 2: Illustration der Funktionsweise von Flush&Reload-Seitenkanalangriffen 24
Abb. 3: Illustration der Funktionsweise von Prime&Probe-Seitenkanalangriffen 27

II. Definitionen

Definition 1: Risiko nach ISO/IEC 21827 7
Definition 2: Threat nach NIST/FIPS 200 (NIST, 2006) 8
Definition 3: Vulnerability nach IETF RFC 4949 8
Definition 4: Verdeckter Kanal (covert channel) 19
Definition 5: Seitenkanal (side channel) 20

III. Exkurse

Exkurs 1: Validierung von Koresidenz-Checks 18

IV. Kontrollaufgaben

Kontrollaufgabe 1: Bewertung Vor- und Nachteile 11

V. Tabellen

Tabelle 1: Relative Kosten der Schwachstellen-Reparatur, abhängig von der Phase des Entwicklungszyklus 7

VI. Literatur

Kevin D. Bowers, Ari Juels, und Alina Oprea. Proofs of Retrievability: Theory and Implementation. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, S. 43–54. ACM, 2009a. ISBN 978-1-60558-784-4.

Kevin D. Bowers, Ari Juels, und Alina Oprea. HAIL: A High-availability and Integrity Layer for Cloud Storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, S. 187–198. ACM, 2009b.

Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, und David A. Patterson. RAID: High-performance, Reliable Secondary Storage. *ACM Comput. Surv.*, 26(2):145–185, Juni 1994. ISSN 0360-0300.

Lionel Dupré und Thomas Haeberle. Cloud Computing: Benefits, risks and recommendations for information security, rev. B. Technical report, ENISA – European Network and Information Security Agency, 2012.

Craig Gentry. Computing Arbitrary Functions of Encrypted Data. *Commun. ACM*, 53(3):97–105, März 2010.

Craig Gentry und Shai Halevi. Implementing Gentry's Fully-homomorphic Encryption Scheme. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, S. 129–148. Springer-Verlag, 2011.

Gabriel Goller und Georg Sigl. Side Channel Attacks on Smartphones and Embedded Devices Using Standard Radio Equipment. In *Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064*, COSADE 2015, S. 255–270. Springer-Verlag New York, Inc., 2015.

Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, und Sergey Yekhanin. Erasure Coding in Windows Azure Storage. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, S. 2–2. USENIX Association, 2012.

Butler W. Lampson. A Note on the Confinement Problem. *Commun. ACM*, 16(10):613–615, Oktober 1973.

Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, und Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*, S. 605–622, 2015.

Stefan Mangard. A Simple Power-analysis (SPA) Attack on Implementations of the AES Key Expansion. In *Proceedings of the 5th International Conference on Information Security and Cryptology*, ICISC'02, S. 343–358. Springer-Verlag, 2003.

Arvind Narayanan und Vitaly Shmatikov. Robust De-anonymization of Large Sparse Datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, S. 111–125. IEEE Computer Society, 2008.

Purna Nemani. Hacker deleted entire season, TV station says. Courthouse news service, online verfügbar unter `http://www.courthousenews.com/2011/03/31/35406.htm` [abgerufen am 2017-01-02], 2011.

NIST. *The Economic Impact of Inadequate Infrastructure for Software Testing*. Planning Report 02-3, National Institute Of Standards & Technology, Mai 2002. URL `http://www.nist.gov/director/planning/upload/report02-3.pdf`.

NIST. *Minimum Security Requirements for Federal Information and Information Systems*. FIPS PUB 200, National Institute Of Standards & Technology, 2006.

Dag Arne Osvik, Adi Shamir, und Eran Tromer. Cache Attacks and Countermeasures: The Case of AES. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, S. 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

James S. Plank. Erasure codes for storage systems: a brief primer. *;link: the Usenix Magazine*, 38(6), Dezember 2013.

João Ramos. Security Challenges with Virtualization. Master Thesis, MSc in Information Security, Departamento de Informatica, Faculdade de Ciências da Universidade de Lisboa / Carnegie Mellon University (USA), 2009.

Thomas Ristenpart, Eran Tromer, Hovav Shacham, und Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, S. 199–212. ACM, 2009.

Juraj Somorovsky, Mario Heiderich, Meiko Jensen, Jörg Schwenk, Nils Gruschka, und Luigi Lo Iacono. All Your Clouds Are Belong to Us: Security Analysis of Cloud Management Interfaces. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, S. 3–14. ACM, 2011.

Dawn Xiaodong Song, David Wagner, und Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, Berkeley, CA, USA, 2001. USENIX Association.

Wim van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Comput. Secur.*, 4(4):269–286, Dezember 1985.

Zhang Xu, Haining Wang, und Zhenyu Wu. A Measurement Study on Co-residence Threat inside the Cloud. In *24th USENIX Security Symposium (USENIX Security 15)*, S. 929–944. USENIX Association, 2015.

Xingliang Yuan, Xinyu Wang, Cong Wang, Chen Qian, und Jianxiong Lin. Building an Encrypted, Distributed, and Searchable Key-value Store. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, S. 547–558, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4233-9. doi: 10.1145/2897845.2897852. URL `http://doi.acm.org/10.1145/2897845.2897852`.

Yinqian Zhang, Ari Juels, Michael K. Reiter, und Thomas Ristenpart. Cross-Tenant Side-Channel Attacks in PaaS Clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, S. 990–1003. ACM, 2014.