Technische Universität Ilmenau 27. Juni 2024

Vorlesung

Schutz von Kommunikationsinfrastrukturen

Prof. Dr.-Ing. Günter Schäfer

NeXTeX

Vorlesungsmitschrift von
Adrian Schollmeyer

Inhaltsverzeichnis

1 Introduction **6**
- 1.1 Motivation . . . 6
- 1.2 Threats and Security Goals . . . 6
- 1.3 Security Analysis of Layered Protocol Architectures . . . 7
- 1.4 Security Analysis of Communication Infrastructure . . . 7
- 1.5 Towards Systematic Threat Analysis . . . 8
- 1.6 System Security Engineering Process . . . 8
- 1.7 A High Level Model for Internet-based IT-Infrastructure . . . 9
- 1.8 Countering Attacks . . . 9
  - 1.8.1 Principle Classes of Action . . . 9
  - 1.8.2 Safeguards Against Information Security . . . 10
- 1.9 Terminology . . . 10
- 1.10 Security Services . . . 11

2 Security Aware System Design and Implementation **12**
- 2.1 Problems of Practical System Security . . . 12
- 2.2 Origin of Attacks . . . 12
- 2.3 Unix Access Control . . . 13
- 2.4 Buffer Overflows . . . 13
  - 2.4.1 Stack Smashing . . . 14
  - 2.4.2 Angriffe auf dem Heap . . . 15
  - 2.4.3 Exploits mit Buffer Overflow . . . 15
  - 2.4.4 Verhinderung von Buffer-Overflow-Angriffen . . . 16
- 2.5 Format String Attacks . . . 17
- 2.6 Race Conditions . . . 17
- 2.7 SQL Injections und Cross-Site-Scripting (XSS) . . . 18
- 2.8 Malware . . . 18
- 2.9 Exploiting Trust . . . 19
  - 2.9.1 Reflections on Trusting Trust . . . 19
  - 2.9.2 Security of Open Source Software . . . 20
- 2.10 Trating Input in Secure Programs . . . 20
- 2.11 Countermeasures . . . 21

3 Denial Of Service **22**
- 3.1 Reaction According to Protocol Specification . . . 22
- 3.2 TCP-SYN Flood Attack . . . 22
- 3.3 Abusing ICMP for Malicious Activities . . . 22

3.4 TCP Bang Attack . . . 22
3.5 DNS and NTP Amplification . . . 23
3.6 Resource Depletion with Distributed DoS . . . 23
3.7 Defense Techniques Against DoS Attacks . . . 23
3.8 Authentication . . . 25
3.9 Countering CPU Exhaustion with Client Puzzles . . . 25
3.10 Countering Memory Exhaustion: Stateless Protocol Design . . . 26
3.11 Remaining Issues . . . 27
3.12 Possible Solutions to DDoS Attacks . . . 27
3.12.1 Ingress Filtering (RFC 2827) . . . 27
3.13 Identifying Malicious Nodes . . . 27
3.13.1 Requirements / Evaluation Metrics . . . 27
3.13.2 Number of Packets Needed to Track Source . . . 27
3.13.3 Resource Overhead . . . 27
3.13.4 Logging Approaches . . . 27
3.13.5 Source Path Identification . . . 28
3.13.6 ICMP Traceback . . . 28
3.13.7 Probabilistic Packet Marking . . . 29
3.13.8 PPM Marking . . . 29
3.14 Related Techniques for Mitigation / Avoidance . . . 30
3.14.1 Hop Count Filtering . . . 30
3.14.2 Aggregate Based Congestion Control . . . 30
3.14.3 Secure Overlay Services / Onion Routing . . . 30

4 Routing Security 32
4.1 General Threats . . . 32
4.2 Inter-AS Routing Threats in the Internet . . . 33
4.3 Securing BGP Operation . . . 34
4.3.1 Verifying Peer Messages . . . 34
4.3.2 Problems Beyond Simple Peer-to-Peer BGP Security . . . 34
4.3.3 Secure Border Gateway Protocol . . . 35
4.3.4 S-A (Signature Amortization) . . . 36
4.3.5 Secure Path Vector Protocol – Lamport Signatures . . . 37
4.3.6 Secure Path Vector Protocol – HORS Signature . . . 37
4.3.7 Secure Path Vector Protocol – Implementation . . . 37
4.3.8 Interdomain Route Validation . . . 38
4.3.9 Secure Origin BGP . . . 38
4.4 BGPSEC and RPKI . . . 39
4.5 Securing BGP by State Observation . . . 39
4.5.1 Pretty Good BGP: Cautiously Adopting Routes . . . 39
4.5.2 Topology-Based Analysis . . . 39
4.5.3 Stable Route Information Objects . . . 40
4.5.4 Monitoring TCP Flows . . . 40

5 Secure Name Resolution **41**
5.1 Security of the Domain Name System . . . 41
5.2 DNS Structure . . . 41
5.3 DNS Security Objectives . . . 42
5.4 Denial of Service . . . 42
5.5 Threats to Date Integrity and Authentication . . . 42
5.5.1 Data Corruption / Cache Poisoning . . . 43
5.5.2 Split-Horizon DNS . . . 44
5.6 Robustness Towards Data Corruption: Data Integrity . . . 44
5.7 DNSSEC . . . 45
5.7.1 Means of Securing RRSets . . . 45
5.7.2 Authority Delegation and Trust Chaining . . . 46
5.7.3 DNSSEC Deployment . . . 46
5.7.4 DNSSEC Resource Records . . . 46
5.7.5 DNSSEC Issues . . . 47
5.7.6 Alternatives to DNSSEC . . . 48

6 Internet Firewalls **49**
6.1 Introduction . . . 49
6.2 Fundamental Approaches Regarding Firewall Policy . . . 49
6.3 Protocol Fields Important for Firewalls . . . 50
6.4 Firewall Terminology and Building Blocks . . . 50
6.5 Firewall Architectures . . . 51
6.6 Packet Filtering . . . 52
6.7 Bastion Hosts . . . 53
6.7.1 Proxy Services . . . 53
6.7.2 Aspects of modern Firewall Systems . . . 54

7 Intrusion Detection Systems **55**
7.1 Introduction . . . 55
7.2 Intrusion Detection Systems . . . 56
7.3 Tasks of an Intrusion Detection System . . . 56
7.4 Requirements of Intrusion Detection Systems . . . 56
7.5 Classification of IDS . . . 56
7.6 Host Intrusion Detection Systems . . . 57
7.7 Network Intrusion Detection Systems . . . 57
7.8 Signature-based Detection . . . 58
7.9 Detection of Abnormal Behavior . . . 59
7.10 Automatic Anomaly Detection . . . 59
7.10.1 Übersicht . . . 59
7.10.2 Systemmodell . . . 60
7.10.3 Klassifikationskriterien . . . 60
7.10.4 Anomalietypen . . . 60
7.10.5 Erkennungstypen . . . 60

7.10.6 Eigenschaften 61
7.11 Testing and Benchmarking of IDS 61

Stichwortverzeichnis **62**

1 Introduction

1.1 Motivation

Moderne Infrastruktur ist zunehmend abhängig von funktionierenden Kommunikationsnetzen. Verfügbarkeit und Sicherheit beim Betrieb von solchen Kommunikationsnetzen und daran angeschlossenen Teilnehmern wird immer wichtiger. Dabei ist insbesondere nicht nur die Verfügbarkeit wichtig, sondern auch die Sicherheit, da Angriffe auch die Verfügbarkeit und Stabilität des Systems beeinträchtigen können.

1.2 Threats and Security Goals

Eine **Bedrohung** ist ein mögliches Ereignis oder eine Sequenz von Ereignissen, die zu einer Verletzung von mindestens einem Sicherheitsziel führen kann. Im Unterschied zu einem **Angriff**, der tatsächlich passiert, liegt eine Bedrohung bereits vor, wenn es nur möglich ist, einen Angriff zu fahren. Beim Schutz von Kommunikationsinfrastruktur sollte nicht auf Angriffe reagiert, sondern Bedrohungen im Vorfeld vermieden werden.

Sicherheitsziele können abhängig von verschiedenen Faktoren definiert werden, beispielsweise anhand der Anwendungsumgebung:

Telekommunikationsanbieter Privatsphäre der Nutzer, Zugriffsbeschränkung für administrative Funktionen, Ausfallsicherheit

Firmen-/Privatnetze Privatsphäre, Verschluss von Firmengeheimnissen, Authentizität von Nachrichten, Ausfallsicherheit

Alle Netzwerke Schutz vor Eindringlingen von außen

Technisch definiert wird sich auf fünf Ziele beschränkt:

Vertraulichkeit (von Daten, Identitäten, etc.)

Integrität (Sicherstellung, dass Daten auf dem Kommunikationsweg nicht verändert worden sein können; benötigt auch Authentizität der Nachricht)

Accountability (Nachweisbarkeit, welche Entität für welches Ereignis verantwortlich ist)

Kontrollierter Zugriff (Durchsetzung von Regeln, welche Entitäten welche Rechte im System haben)

Verfügbarkeit (Systeme sollten verfügbar sein, korrekt funktionieren und in ihrer Leistungsfähigkeit nicht beeinträchtigt werden)

Angriffe lassen sich auch in Kategorien einteilen:

Masquerade (der Angreifer kann sich als jemand anderes ausgeben)

Eavesdropping (Abhörung von übermittelten Nachrichten)

Authorization Violation (ein Angreifer macht etwas, wofür er eigentlich nicht die nötigen Berechtigungen haben sollte)

Verlust oder Veränderung von Daten

Repudiation (Abstreiten von Kommunikationsvorgängen, erfordert meist anwendungsspezifisches Wissen)

Fälschung von Informationen (Angreifer erzeugt neue Informationen im Namen einer andere Entität)

Sabotage

Verschiedene Angriffe bedrohen Sicherheitsziele in verschiedener Weise. Es ist jedoch immer möglich, dass die Verletzung eines Sicherheitsziels dazu führen kann, dass andere, gefährlichere Angriffe gefahren werden können (bspw. kann mit Eavesdropping ein Root-Passwort ausgelesen werden, wodurch Maskerade möglich ist).

1.3 Security Analysis of Layered Protocol Architectures

Die erste Frage bei der Analyse ist, an welchen Stellen ein Angreifer welche Angriffe fahren kann. Je nach Ort kann ein Angreifer dann an verschiedenen Schichten der Protokollarchitektur angreifen. Je nach Schicht sind wiederum verschieden mächtige Angriffe möglich.

Auf der Nachrichtenebene (sprich, wo einzelne Pakete der eingesetzten Protokolle verschickt werden), lässt sich analysieren, welche Angriffe auf die **PDUs** (Protocol Data Units) gefahren werden können. Für einen erfolgreichen Angriff darf es keine Nebeneffekt auf andere Verbindungen, idealerweise auch nicht auf die betroffene Verbindung geben. Ansonsten besteht das Risiko, dass der Angriff scheitert oder sogar vom Opfer erkannt wird.

1.4 Security Analysis of Communication Infrastructure

Neben den Angriffen auf die Informationsübertragung sind auch Angriffe auf die Systeme, die Teil des Kommunikationsnetzes sind, wichtig, u. A.

- Endsysteme
- Router
- wichtige Infrastrukturdienste (z. B. DNS, E-Mail, Webserver, etc.)

Dabei erweitert sich die Analyse deutlich und wird viel anwendungs- und systemspezifischer.

1.5 Towards Systematic Threat Analysis

Die Erstellung beliebiger Listen möglicher Angriffe ist keine besonders zielführende Methode. Dabei ist es nur schwer möglich, die Vollständigkeit der identifizierten Angriffe zu zeigen.

Ein **Bedrohungsbaum** beschreibt Bedorhungen auf verschiedenen Eben mit verschiedenen Leveln an Abstraktion. An den obersten Knoten werden die generellen Ziele definiert, während beim Absteigen in den Baum immer konkretere Angriffsszenarien und Bedrohungen notiert werden. Irgendwann erhält man Blattknoten, die sehr detaillierte Bedrohungen beschreiben und die in einer (weniger beliebige) Bedrohungsliste aufgenommen werden können.

Bei der Erstellung von Bedrohungsbäumen sollte darauf geachtet werden, auf jeder Analyse vollständig zu arbeiten, bspw. in dem binäre Entscheidungen in Knoten repräsentiert werden (sodass alle möglichen Fälle abgedeckt werden). Weiterhin können Knoten auch in verschiedenen logischen Beziehungen (und, oder) stehen, je nachdem ob ein Angreifer bspw. nur eine von mehreren oder auch mehrere Hürden auf einmal überwinden können muss, um einen Angriff durchzuführen (vergleiche: Stellen, um in ein Haus einzubrechen vs. Sicherheitsmaßnahmen wie Zäune, Wachhunde, etc.). Sind nun Angriffsszenarien bekannt, kann der Aufwand (die Kosten), Angriffe durchzuführen, abgeschätzt werden, um zu analysieren, welche Angriffe am wahrscheinlichsten sind. Für oder-verknüpfte Knoten muss dabei das Minimum genommen werden (angenommen der Angreifer kennt den Aufwand, wird er den geringsten Aufwand wählen), während bei und-verknüpften Knoten das Maximum gewählt werden muss (er muss mindestens diesen Aufwand betreiben, um den Angriff durchzuführen).

Daneben stellt sich auch die Frage, wie viele Personen zu einem Bestimmten Angriff bereit sind. Das ist vor allem eine Frage dessen, was geschützt werden soll (vgl. Staatsgeheimnisse vs. eine Privatwohnung) und wer als Angreifer infrage kommt (und welche Möglichkeiten die Angreifer dann potentiell haben). Dies lässt sich im Bedrohungsbaum nicht direkt modellieren. Stattdessen muss die resultierende Angriffsliste auf Kosten und Gewinn analysiert werden und die Frage ist, welche Angreifer dann noch zu welchen Angriffen bereit sind.

1.6 System Security Engineering Process

Dies erlaubt ein etwas systematischeres Vorgehen:

- Spezifikation der Systemarchitektur
- Identifikation von Bedrohungen, Schwachstellen und Angriffstechniken
- Abschätzung von Risiken (zusätzliche Attribute am Bedrohungsbaum)
- Priorisierung von Schwachstellen
- Identifikation und Installation von Sicherungsmaßnahmen (für Schwachstellen mit hoher Priorität)

- Neuanalyse/Iteration der o. g. Schritte

Es kann dabei prinzipiell auch möglich sein, dass der Angreifer damit rechnet, dass bestimmte Sicherungsmaßnahmen ergriffen werden. Es besteht also grundsätzlich die Gefahr, dass durch die Installation von solche Maßnahmen auch neue Bedrohungen entstehen, deren Ausnutzung potentiell schon vom Angreifer geplant sind. Es kann also auch Sicherungsmaßnahmen geben, die tatsächlich kontraproduktiv sind.

1.7 A High Level Model for Internet-based IT-Infrastructure

Die Hauptunterscheidung ist zwischen private Netzen (d. h. Zeug, was keine Dienste anbietet, sondern mit Diensten spricht; bspw. Heimnetze, Sensornetze, etc.), Support-Infrastruktur (Transportnetze wie das Internet) und ISP-Netzwerke (Bereitstellung von Diensten, Cloud-Hosts, Rechenzentren, Mobilkommunikationsnetze).

Grundsätzlich dürfen physikalische Bedrohungen (bspw. Zerstörung eines Rechenzentrums) nicht missachtet werden und physische Redundanz ist immer wichtig, um Verfügbarkeit zu garantieren. Dennoch sind solche Bedrohungen für diese Vorlesung weniger relevant und werden nicht wirklich behandelt.

Link-basierte Bedrohungen auf physischer Ebene sind zwar theoretisch möglich, lohnen sich aber meist weniger als Angriffe auf dem Data Link Layer. Die Untersuchung und Behandlung solcher Bedrohungen ist Teil der Veranstaltung „Network Security“.

Beim Network Layer werden hier auch Dienste hinzugezogen, die nicht direkt auf dem Network Layer laufen, aber für den Betrieb des Netzes an sich notwendig sind (bspw. DNS). Die Anwendungsschicht wird in diesem Bedrohungsbaum vereinfacht und ist nicht vollständig. Dieser Bereich ist auch sehr stark davon abhängig, welche Anwendungen im Netz betrieben werden.

1.8 Countering Attacks

1.8.1 Principle Classes of Action

Prävention Umfasst alle Maßnahmen, um Angreifer davon abzuhalten, erfolgreiche Angriffe durchzuführen (bspw. Verschlüsselung, Signaturen, Firewalls) Dies muss grundsätzlich passieren, *bevor* der Angriff stattfindet.

Erkennung Umfasst alle Maßnahmen, um laufende oder vergangene Angriffe zu erkennen. Dazu gehören Audit Trails, Traffic Monitoring (idealerweise on-the-fly), etc. Wird ein laufender Angriff erkannt, sollten Maßnahmen ergriffen werden, um den Angriff entweder zu unterbinden oder zunächst zu entscheiden, ob der Angriff zu echten Schäden führen oder der Angreifer ermittelt werden kann.

Eine Variante der Erkennung von potentiellen Angreifern ist der Betrieb eines **Honeypots**. Diese Systeme sind am Netz erreichbar mit dem Ziel, dass Angreifer diese angreifen. Auf solchen Systemen sollten keine echten Dienste laufen, sodass der Zugriff auf das System direkt als Angriff erkannt werden kann. Wichtig ist

hierbei jedoch auch wieder, dass der Honeypot selbst kein Einfallstor in echte Kommunikationsinfrastruktur darstellt.

Reaktion Umfasst alle Maßnahmen, die in Reaktion auf vergangene oder laufende Angriffe unternommen wurden.

1.8.2 Safeguards Against Information Security

- Physical Security (Beschränkung von physischem Zugang zu Servern u. Ä.)
- Personnel Security (Überprüfung von Personal, welches Zugriff zu Servern u. Ä. hat)
- Administrative Security (Ausbildung von Personal, Einbringung von Fremdsoftware, Prozeduren und Workflows zur Untersuchung von Sicherheitsvorfällen, Review von Audit Trails etc.)
- Emanations Security (Abstrahlung von Geräten, etc.)
- Media Security (Kontrolle darüber, wie sensible Informationen (sicher) reproduziert und zerstört werden können, Sicherhung der Speichermedien, Virenscans, etc.)
- Lifecycle Controls (Kontrolle des Softwareentwicklungsprozesses, Programmierstandards, etc.)
- Computer/System Security (Schutz der verarbeiteten Informationen und verarbeitenden Geräte)
- Kommunikationssicherheit (Schutz der Informationen auf dem Transportweg, Schutz der Kommunikationsinfrastruktur)

1.9 Terminology

Security Service Abstrakter Dienst, welcher eine bestimmte Sicherheitseigenschaft garantieren soll. Kann mithilfe von Kryptographie, Protokollen, aber auch konventionellen Mitteln (z. B. Ablegen des Datenträgers in einem Tresor) garantiert werden. Meist werden mehrere Dienste miteinander kombiniert.

Cryptographic Algorithm

cryptographic algorithm

Cryptographic Protocol

cryptographic protocol Regeln, wer wann welche Berechnungen durchführen und welche Nachrichten verschiken muss, um bestimmte Sicherheitsziele zu erreichen (z. B. für Authentisierung, Schlüsselaustausch, etc.).

1.10 Security Services

Authentisierung

authentication

Integrität

integrity

Vertraulichkeit

confidentiality

Zugriffskontrolle

access control

Nichtabstreitbarkeit

non-repudiability Es soll sichergestellt werden, dass kein Kommunikationspartner später abstreiten kann, an einer Kommunikation teilgenommen zu haben. Dies lässt sich eigentlich nur durch Zertifizierung durch eine unabhängige dritte Stelle sicherstellen.

2 Security Aware System Design and Implementation

2.1 Problems of Practical System Security

Es ist unmöglich, die Sicherheit eines moderat komplexen Systems zu zeigen. Allgemein ist es schwierig, die Abwesenheit einer Eigenschaft (hier: Bedrohung) zu zeigen. Daher ist es wichtig, möglichst einfache Systeme einzusetzen, wenn Sicherheit garantiert werden soll. Zusätzlich müssen die Quellen beherrscht werden, von denen die Systeme kommen. Meist ist dabei Software das Problem, die durch schlechtes Softwaredesign oder schlechte Implementierung Sicherheitsprobleme haben. Systemadministratoren müssen dann häufig Sicherheitspatches einpflegen, was zu zusätzlicher Last (und potentieller Nachlässigkeit) bei der Wartung führt, wodurch wieder Fehler gemacht werden können.

Weitere Probleme sind der Einsatz von Low-Level-Programmiersprachen, die einfache Angriffe ermöglichen (bspw. Buffer Overflow), schlechte Standardkonfigurationen von Software, Erweiterbarkeit von Software durch Updates und dynamisch ladbare Erweiterungen (e. g. Plugins) und der Mangel an Diversität in beliebten Rechenumgebungen (bspw. Windows auf PCs, Linux auf Servern, Cisco IOS für Router, etc.). Durch die immer steigende Geschwindigkeit bei der Softwareentwicklung kommt es zu immer kürzeren Entwicklungszyklen mit schlechter Spezifikation von Anforderugen, sofern diese überhaupt existieren. Gerade letztes führt dazu, dass Sicherheitsaspekte nicht oder nur unzureichend beachtet werden.

2.2 Origin of Attacks

Zunächst werden Angriffe nach ihrem Ursprung unterschieden. Entfernte Angriffe werden typischerweise unter Ausnutzung geklauter/schwacher Passwörter oder offener Schwachstellen in der Software gefahren, um Zugriff auf ein System zu erhalten. Lokale Angriffe arbeiten mit einem bestehenden Zugang und haben zum Ziel, die eigenen Berechtigungen auszuweiten. Unter Linux ist dies beispielsweise ein interessantes Ziel, da normale Service-Benutzer (also eigens für den Betrieb eines Dienstes eingerichtete Benutzerkonten) meist stark eingeschränkte Berechtigungen haben. Echte Benutzeraccounts hingegen sind noch interesssanter als Ziel, da diese zwar nicht durchgehend aktiv in Benutzung sind, aber dafür über Tools wie sudo Root-Zugriff haben.

Am interessantestn sind von den beiden Arten die Remote-Angriffe, da der Angreifer in aller Regel keinen direkten Zugang zum System hat und dies in so einem Fall also immer eine Voraussetzung für einen lokalen Angriff ist.

2.3 Unix Access Control

Im Server-Bereich sind unixoide Systeme wie Linux sehr prävalent. Auch Router-Betriebssysteme haben häufig Linux als Grundlage. Dies hat auch zur Folge, dass die meisten lokalen Exploits von der Ausnutzung unixoider Zugriffskontrollen abhängen. Die Zugriffskontrolle unter Unix basiert auf Benutzern, die in Gruppen sein können, sowie Berechtigungen auf Dateien, die für Besitzer der Datei, Gruppe der Datei und beliebige andere Benutzer festgelegt werden können. Hinzu kommt der root-Benutzer (UID 0), der immer alle Berechtigungen hat und Berechtigungen nach Belieben ändern kann. Bei der Rechteprüfung wird für einen Prozess die effektive Benutzer-ID, effektive Gruppen-ID und die Berechtigungseinstellungen der Datei miteinander verglichen, um zu entscheiden, ob eine bestimmte Aktion (Lesen, Schreiben, Ausführen) zulässig ist. Zusätzlich können Dateien mit dem *setuid*-Bit versehen, mit dem bei Ausführung der Datei der Prozess mit dem Besitzer der Datei als effektive User-ID ausführt wird. Ähnlich kann mit *setgid* auch die effektive Gruppen-ID verändert werden.

Die Benutzung potentiell gefährlicher Operationen wie setuid oder setgid sollten nur sparsam eingesetzt werden. Grundsätzlich ist es keine gute Idee, mit mehr Rechten zu arbeiten als nötig, da nicht nur die Gefahr besteht, dass (versehentlich) ausgeführter Schadcode zu Sicherheitsproblemem führt, sondern schon allein menschliche Fehler (z.B. Tippfehler) können zu Schäden am System führen. Prozesse, die mit hohen Rechten (bspw. durch setuid) laufen, kann eine Ausnutzung von Schwachstellen/Programmierfehlern benutzt werden, um (Voll-)Zugriff auf das System zu erhalten. Solche Zugriffe können möglicherweise erlauben, zusätzliche Zugänge z. B. durch Manipulation von `/etc/passwd` zu erhalten.

Ein Workaround, um hohe Berechtigungen für notwendige Operationen (bspw. Binden von Ports < 1024) zu machen, ohne dauerhaft mit hohen Privilegien zu laufen, besteht darin, alles, was hohe Berechtigungen benötigt, zum Start des Dienstes auszuführen (das ist meist ausrechend) und danach die Berechtigungen auf die eines weniger privilegierten Benutzers zu reduzieren, sodass spätere Angriffe weniger Schaden anrichten können.

2.4 Buffer Overflows

Diese Angriffsart betrifft vor allem low-level Programmiersprachen wie C oder C++, die manuelle Speicherverwaltung benutzen. Speicher kann dort auf dem Stack oder auf dem Heap allokiert werden. Auf dem Stack werden dabei die lokalen Variablen, Rückgabewerte, Sprungadressen und Funktionsargumente gespeichert. Dieser Bereich wird vom Programm automatisch verwaltet. Auf dem Heap hingegen wird Speicher grundsätzlich manuell, z. B. durch `malloc` und `free` angefordert und freigegeben.

Buffer-Overflow-Angriffe beruhen darauf, über Speichergrenzen hinweg zu schreiben. In Sprachen wie C und C++ wird dabei ausgenutzt, dass es keine automatisch Prüfung von Speichergrenzen bspw. von Arrays gibt. So können Programmierfehler dazu führen, dass in Speicherbereiche geschrieben wird, die gar nicht zu dem Objekt gehören, was eigentlich geschrieben werden sollte. Dies kann von Angreifern ausgenutzt

werden, um bspw. Sprungadressen zu überschreiben und damit andere Funktionen aufzurufen. Ein übliches Problem dabei ist bspw., wenn in einen Puffer Daten geschrieben werden sollen, die länger als der Puffer selbst sind. Wurde dabei dann kein ordentliches Bounds-Checking implementiert, kann in Speicherbereiche hinter dem Puffer geschrieben werden.

Die Konsequenzen eines Buffer Overflows sind nicht fest definiert. Je nachdem, welcher Speicher sich dahinter befindet, verhält sich das Programm komisch (Daten wurden falsch geschrieben und jetzt befindet sich das Programm in einem falschen Zustand), stürzt ab (z. B. bei Zugriff auf einen nicht allokierten Speicherbereich) oder verhält sich wie zuvor (z. B. wenn in Padding-Bereiche geschrieben wurde). Es hängt also immer davon ab, wie viele Daten außerhalb der Grenzen geschrieben wurden und welche Speicherbereiche sich dort befinden.

Ein Angreifer kann je nach Möglichkeiten Buffer Overflows für verschiedene Zwecke nutzen:

- Manipulation von Rücksprungadressen, um anderen (z. B. eingeschleusten) Code auszuführen
- Eskalation von Privilegien, indem bspw. Programme mit setuid ausgenutzt werden
- etc.

Besonders interessant sind dabei immer Angriffe auf dem Stack, da nur hier Rücksprungadressen manipuliert werden können. Beliebt dabei sind viele Funktionen der C-Standardbibliothek wie bspw. `gets`, `strcpy`, `scanf`, die kein Bounds-Checking unterstützen. Aber auch Funktionen mit Bounds-Checking (z. B. `strncpy`) können ausgenutzt werden, wenn dort Strings ohne Nullbyte kopiert werden, ohne nach der Operation ein Nullbyte ans Ende zu setzen.

2.4.1 Stack Smashing

Ein wichtiges Problem bei Buffer Overflows ist das Speicherlayout, vor allem das des Stacks. Hier legt der Compiler für jede Funktion bestimmte Werte auf den Stack, u. A. Funktionsparameter, Rücksprungadresse und der sogenannte **Stack Frame**, der den Beginn des Speicherbereichs markiert, ab dem die Variablen gespeichert werden, die auf dem Stack gespeichert sind. Ein übliches Layout eines Stack Frames:

- Funktionsargumente…
- Rücksprungadresse
- Alter wert ebp (Mitte des Stack Frames, ab wo die Variablen beginnen)
- Stack-Variablen (Variablen in der Funktion, die nicht auf dem Heap allokiert sind)

Der Pointer ebp wird dabei auf die Anfangsadresse der „Mitte“ des Stack Frames gesetzt, also da, wo die erste Variable der Funktion steht. Ab hier bis zum Ende des Stacks liegen

alle lokalen Variablen, die die Funktion benutzt. Problem ist jedoch, dass der Stack von oben nach unten aufgebaut wird. Die Rücksprungadresse wird also in einer höheren Adresse auf dem Stack gespeichert als die Variablen auf dem Stack. Dennoch werden bspw. Arrays so auf dem Stack gespeichert, dass das erste Element (Offset 0) an der niedrigsten Adresse des Puffers (also ganz unten im Stack) liegt, während das letzte Element an der höchsten Adresse des Puffers (also ganz oben im Stack, z. B. kurz vor der Rücksprungadresse) liegt. Wenn man also als Stack-Variable einen Puffer stehen hat und zu weit schreibt, kommt man schnell in die Situation, bei der die Rücksprungadresse überschrieben werden kann. Das gezielte Überschreiben der Rücksprungadresse durch einen Buffer Overflow wird auch als **Stack Smashing** bezeichnet.

2.4.2 Angriffe auf dem Heap

Das Speicherlayout auf dem Heap sind stark davon abhängig, wann wie viel Speicher allokiert wurde und wann welche anderen Prozesse speicher allokiert haben. Hier ist es sehr schwer vorherzusagen, welche Speicherbereiche wo stehen. Durch dynamisches Linken ist es auch schwer vorhersehbar, welche Version einer Bibliothek vom Programm benutzt wird, was das Vorhersagen des Speicherlayouts weiter erschwert.

Dennoch ist es möglich, potentiell kritische Variablen zu finden und auszunutzen. Zudem muss dann aber auch noch für einen Buffer-Overflow-Angriff eine Möglichkeit gefunden werden, einen Puffer so zu überschreiben, dass solch eine Variable manipuliert werden kann.

Allgemein sind Stack Overflows einfacher durchzuführen als Heap Overflows.

2.4.3 Exploits mit Buffer Overflow

Ein mögliches Ziel ist das Erlangen einer Shell, damit der Angreifer beliebig Befehle auf dem System ausführen kann. Üblicherweise wird der boshafte Code (z. B. Start einer Shell) vom Angreifer kompiliert und dann versucht, diesen Code irgendwo in den Speicher zu schreiben, um ihn Anschließend durch Stack Smashing ausführen zu können. Beim Schreiben von solchem Exploit-Code muss häufig darauf geachtet werden, dass der Code keine Null-Bytes enthält, da diese bei den Funktionen, die Strings verarbeiten, häufig als Ende des Strings angesehen werden. Stattdessen werden dabei dann Tricks wie XOR angewendet, die Nullbytes erzeugen können, ohne dass Nullbytes in den Programmcode geschrieben werden müssen. Dies erfordert jedoch manchmal auch Anpassung des kompilierten Schadcodes.

Manchmal ist es auch schwierig, zu genau der richten Adresse zu springen. Eine Technik ist dabei, viele NOPs an vor den Schadcode zu schreiben, sodass es nur irgendwie möglich gemacht werden muss, an eine der NOP-Instruktionen zu springen. Dies wird als **NOP-Slide** bezeichnet. Solche NOP Slides können bspw. durch Heap Spraying (Allokation von ganz viel Speicher gefüllt mit NOPs) erstellt werden.

2.4.4 Verhinderung von Buffer-Overflow-Angriffen

Es ist quasi unmöglich, vollkommen fehlerfreien und nicht angreifbaren Code zu schreiben. Programmierer dazu zu motivieren, besonders vorsichtig beim Programmieren zu sein, ist nur bedingt möglich. Stattdessen werden Methoden diskutiert, Fehler technisch zu verhindern oder besser zu erkennen. Dazu gibt es verschiedene Ansätze.

Der **Stackguard** Ansatz fügt eine gewisse Menge zufälliger Daten (**canary**) an das Ende von Stack-allozierten Daten. Später wird dann geprüft, ob dieser Wert immer noch dort steht. Wenn nicht, dann scheint ein Buffer Overflow aufgetreten zu sein. Dieser Ansatz ist allerdings nur bedingt nützlich, da Variablen im Stack weiter überschrieben werden können (aber nicht mehr gesprungen werden kann). Heap Overflows sind damit allerdings weiter möglich. Außerdem hat das Problem leichte Performanceeinbußen.

Memory Integrity Checking mit Tools wie Valgrind [9] oder LLVM Address Sanitizer [1] kann während der Entwicklung Speicherprobleme aufzeigen. Allerdings sind diese sehr langsam und werden daher in Produktiv-Builds nicht eingesetzt. Daher können Angreifer Buffer Overflows, die nicht bei der Entwicklung erkannt wurden, weiterhin ausnutzen. Außerdem können solche Tools auch nur beschränkt gegen Probleme schützen, die in Drittbibliotheken auftreten.

Eine weitere Möglichkeit besteht darin, die Speicherseiten in ausführbare und nicht ausführbare Seiten zu unterscheiden. Damit ist es möglich, zu verhindern, dass Code auf Seiten ausgeführt wird. Wird dann der Stack als nicht ausführbar markiert, dann kann dort eingeschleuster Programmcode nicht ausgeführt werden. Stattdessen würde das Programm abstürzen. Leider unterstützen nicht alle Programme (bspw. solche mit JIT Compilern) dies nicht. Es könnte allerdings weiter möglich sein, geschickt an ausführbare Stellen zu springen, die die gewünschten Operationen ausführen. Man ist zwar weiterhin stark in den Operationen eingeschränkt, die man ausführen kann, aber diese können möglicherweise gut kombiniert werden, um Schadcode zu erstellen. Diese Art der Programmierung heißt **Return-Oriented Programming** (ROP).

Eine Gegenmaßnahme zu ROP ist **Address Space Layout Randomization** (ASLR). Ziel hierbei ist, es dem Angreifer schwer zu machen, die richtigen Sprungadressen zu erraten. Dazu werden Bibliotheken beim Laden an zufällige, aber aufeinanderfolgende Adressen geladen. Dadurch können die Adressen für ROP nicht mehr geraten werden. Dies erfordert jedoch Unterstützung vom Betriebssystem und Programm. Wirklich effektiv ist dies auch nur, wenn Heap und Stack als nicht ausführbar markiert sind, da sonst Heap Spraying benutzt werden kann. Außerdem erfordert dies einen großen Adressraum (also z. B. reichen 32 Bit nicht), um genug Entropie haben (damit die Trefferwahrscheinlichkeit ausreichend klein ist). Diese Maßnahme führt zu leichten Performanceeinbußen, da Sprünge immer indirekt über eine Sprungtabelle laufen.

Dies sind alles nur Gegenmaßnahmen, die das Symptom, jedoch nicht die Ursache bekämpfen. Das grundsätzliche Problem, dass Buffer Overflows auftreten, sollte besser von Anfang an verhindert werden können, bspw. durch Wahl einer entsprechenden Programmiersprache, die dies verhindert.

2.5 Format String Attacks

Format-Strings werden in diversen Programmiersprachen (z. B. C) benutzt, um einen String auszugeben oder zu erzeugen, der aus einer Vorlage (dem **Format-String**) und darin an geeigneten Stellen eingesetzten variablen Werten besteht. Im Format-String werden Platzhalter für Variablen gesetzt (z. B. `%s` im String `Hallo, ich bin %s!` kann später durch einen beliebigen anderen String ersetzt werden). Ein Beispiel dafür ist die Funktion `printf` [4] aus der C-Standardbibliothek. Diese arbeitet so, dass manche C-Funktionen beliebig viele Parameter übernehmen kann. Diese sucht sich aus dem Stack an den richten Positionen die Parameter der Funktion raus. Allerdings gibt es keine Sprachkonstrukte, die sicherstellen, dass an den richtigen Stellen gesucht und die richtigen Datentypen benutzt werden.

Gefährlich werden solche Funktionen, wenn ein vom Benutzer angegebener String über eine solche Funktion ausgegeben werden soll. Hierbei können Benutzer Format-Strings in den auszugebenden String einfügen, wodurch die Funktion versucht, weitere Daten aus dem Stack auszugeben. Dies ermöglicht das Auslesen des genauen Stack-Inhalts durch aneinanderreihen von Format-Platzhaltern. Weitere Möglichkeiten ergeben sich dadurch, dass es Platzhalter gibt, um auch Variablen mit Informationen über den Format-String (bspw. die Länge des erzeugten Strings) zu befüllen. So ist es möglich, durch manipulierte Format-Strings auch gezielt Werte in den Speicher zu schreiben. Damit lässt sich beispielsweise auch die Return-Adresse manipulieren. Natürlich muss dafür auch ein Format-String beliebig lang gemacht werden können. Dazu können Format-Optionen genutzt werden, um bspw. die Stelligkeit oder Präzision einer Zahl festzulegen. Allerdings erfordert das Zählen der Stringlänge auch, dass der gesamte String erzeugt wird. Damit ist man auch durch den verfügbaren Speicher begrenzt. Dennoch gibt es auch dafür geschicktere Angriffe, die jeweils nur Teilworte schreiben und so mit weniger Speicherverbrauch beliebige Sprungadressen festlegen können.

Format-Strings sind allgemein viel gefährlicher als Buffer-Overflow-Attacken. Es ist möglich, präzise bestimmte Speicherbereiche zu überschreiben (und so bspw. einen Canary nicht zu modifizieren) und auch Daten gezielt auszulesen. Außerdem fuktionieren sie selbst mit Bounds-Checks. Dennoch lassen sie sich durch Verwendung anderer Konstrukte (z. B. C++ `std::cout`) gänzlich vermieden werden.

2.6 Race Conditions

Allgemein treten **Race Conditions** auf, wenn eine Annahme vor Ausführung von Code geprüft wird, es aber zwischen der Prüfung und der Ausführung möglich ist, die geprüfte Bedingung zu verändern. Dadurch ist es möglich, dass der Code anschließend auf falschen Annahmen ausgeführt wird und sich dann möglicherweise anders verhält. Diese Art von Angriffen erfordern, dass mehrere Threads auf dem System verfügbar sind (was jedoch auf allen modernen Systemen der Fall ist).

Leider ist diese Sorte von Bedrohung nur schwer erkennbar und schwer zu beheben. Übliche Quellen von Race Conditions sind Datesystemzugriffe oder Rechtemanagement.

Allerdings erfordert bspw. die Ausnutzung von Dateisystem Race Conditions auch lokalen Zugriff, bspw. um Dateien zu modifizieren. Diese Art von Angriff lässt sich lösen, indem soweit möglich mit Dateideskriptoren statt Dateinamen gearbeitet wird. Dies verhindert, dass sich die Datei ändert, mit der gearbeitet wird.

2.7 SQL Injections und Cross-Site-Scripting (XSS)

Diese Sorte von Angriffen wird meist in Sprachen wie Java, PHP, Perl, Python gefahren. Diese machen Angriffe wie Buffer Overflows und Format String Angriffe bereits per Design extrem schwer.

Allerdings gibt es hier andere Angriffe wie **SQL Injection**. Diese treten auf, wenn dynamsich SQL-Queries aus Strings erzeugt werden, schlimmstenfalls durch Nutzereingaben. Dabei ist es bei schlechtem Design möglich, dass ein Angreifer die eigentliche SQL-Query um Steuerzeichen (z. B. Anführungszeichen für Stringende, Semikolon für Ende der Query) zu erweitern, die es ihm ermöglichen, im Anschluss eine beliebige (oder abgewandelte) SQL-Query auszuführen.

Diese Art von Angriffen ist eine ganze Familie von Angriffen, die sich nicht nur auf SQL beschränkt, sondern auch andere Systeme mit beinhaltet, z. B. LDAP Injections, CRLF Injections oder auch **Cross-Site-Scripting** (XSS), bei der eine Möglichkeit ausgenutzt wird, aus der Ferne irgendwelchen Programmcode im Browser des Opfers auszuführen, sodass dieser Code mit den Rechten des Benutzers ausgeführt wird (z. B. mit Rechten eines Administrators).

2.8 Malware

Eines der größten Probleme heutzutage ist boshafte Software (**Malware**), die vom Opfer aufgrund von Vertrauen heruntergeladen und ausgeführt wird, aber im Hintergrund Schadcode ausführt, um bspw. Nutzer auszuspionieren oder Daten zu zerstören (z. B. Ransomware). In so einem Fall wird also nicht irgendeine Schwachstelle in laufender Software ausgenutzt, sondern Menschen werden manipuliert, Schadcode auszuführen. Bekannte Quellen für Schadcode sind bspw. fremde USB-Sticks, Code aus E-Mails, gehackten Websites oder auch fehlerhaften Betriebssystem-Updates. Ein ganz wichtiges Einfallstor für Malware sind E-Mails. Angreifer können hier Schadcode in HTML-Nachrichten einbetten, bösartige Links einfügen oder Makros in Word-Dokumenten verstecken, die beim Anschauen ausgeführt werden und Schadcode nachladen.

Malware wird in mehrere Klassen unterschieden:

Backdoors sind (teilweise unerwünschte) Features wie Vendor Logins, die den Zugriff auf das System erlauben. Diese breiten sich nicht aus.

Trojanische Pferde werden genutzt, um Computer fernzusteuern und wird häufig in normale Software eingebettet, um sich zu verstecken.

Rootkits sind spezielle trojanische Pferde, die sich tief im Betriebssystem verstecken. Diese können teilweise auch das ursprüngliche Betriebssystem als VM emulieren (**Blue Pill**) oder sich in Mikrocontrollern verstecken.

Viren verbreiten sich durch Kopieren auf Wechselmedien (z. B. USB-Sticks).

Übliche Gegenmaßnahmen sind meist schwierig. Gegen bekannte Malware sind Virenscanner und Netzwerküberwachung im Einsatz. Regelmäßige Updates sind zudem wichtig, um Schwachstellen, die das Einschleusen von Malware ermöglichen können, verhindern. Bei der Installation von Software sollte nach Möglichkeit auf signierte Software gesetzt werden, die von vertrauenswürdigen Quellen stammt (was nur schwer möglich ist). Software sollte zudem regelmäßig durch Audits und System Call Monitoring auf mögliche Backdoors geprüft werden.

Besonders wichtig ist vor allem, Benutzer darauf zu trainieren, vorsichtig zu agieren. Dabei sollte das Prinzip der geringstmöglichen Privilegien durchgesetzt werden. Dies bedeutet, dass Software niemals mehr Berechtigungen bekommt, als tatsächlich für die Arbeit notwendig ist.

Allerdings ist auch das im zweifelsfall nicht hilfreich.

2.9 Exploiting Trust

2.9.1 Reflections on Trusting Trust

Vertrauen kann immer wieder missbraucht werden. Vertrauensannahmen werden teilweise durch Programmierer implizit getroffen und nicht näher dokumentiert oder definiert. Beispielsweise wird implizit angenommen, dass Compiler den Code korrekt kompilieren. Bspw. könnte jedoch eine Backdoor im Compiler prüfen, ob gerade ein bestimmtes Programm kompiliert wird, und genau dann in den Programmcode eine Backdoor reinkompilieren, die im Quellcode des ursprünglichen Programms nicht existiert. So war es bspw. Ken Thomposn möglich, das `login`-Programm auf Unix-Systemen so zu modifizieren, dass (nur) er sich auf beliebigen Systemen als beliebiger Nutzer mit einem besonderen Super-Passwort einloggen konnte.

Durch Einbau dieser Backdoor mit einem selbst reproduzierendem Programm (*Quine*) ist es möglich, dass der Compiler immer wieder die Backdoor einkompiliert, selbst wenn der Bug aus dem Quellcode des Compilers entfernt wird. Somit wäre der Compiler (und damit die betroffenen zu kompilierenden Programme) selbst dann betroffen, wenn die Backdoor erkannt und entfernt wird.

Vertrauen ist genau genommen sogar notwendig für die Programme, mit denen andere Programme auf Schwachstellen analysiert werden, die Hardware, auf denen (Mikro-)Code läuft und selbst der Software, die die Chips designt. Denn theoretisch wäre es nämlich sogar möglich, dass ein Angreifer die Software mit einer Backdoor versieht, die den Chip designt, auf dem der Code läuft, der angegriffen werden soll. Wirklich vertrauen kann man Code (und Hardware) nur, wenn man den kompletten Produktionsprozess selbst durchgeführt hat.

2.9.2 Security of Open Source Software

Eine beliebte Annahme ist, dass Open Source Software ja sicher sei, weil jeder den Quellcode lesen und Schwachstellen finden könnte. Ein Beispiel für Sicherheitsprobleme ist CVE-2008-0166 [6], bei der eine Funktion, die angeblich uninitialisierte Daten zum PRNG hinzufügt, auskommentiert wurde. Dies war aber diejenige Funktion, die dem PRNG Entropie zugeführt hat. Durch Entfernung dieser Funktion wurde der PRNG vorhersehbar, was gravierende Auswirkungen auf die Vertraulichkeit erzeugter Schlüssel u. Ä. hatte, da nur noch 2^{18} unterschiedliche Schlüssel erzeugt werden konnten.

Diese Schwachstelle betraf alle SSL- und SSH-Schlüssel, die mit OpenSSL auf Debian-basierten Systemen erzeugt wurden. Allerdings waren selbst sichere Schlüssel betroffen, die für den Nachrichtenaustausch mit einem kompromittierten System genutzt wurden, wenn dort DSA-Schlüssel eingesetzt wurden. Dies ist ein Problem, da sich bei DSA der Schlüssel rekonstruieren lässt, wenn zum Ciphertext auch die (jetzt leicht vorhersagbare) Nonce bekannt ist.

Wenn die generierten Schlüssel kompromittiert sind, ist es auch möglich, Passwörter auszulesen und sich auf fremden Systemen einzuloggen. Damit hätte ein Angreifer also bspw. auch die Debian Paketserver angreifen und dort Malware einschleusen können. Durch die o. g. gezeigte Methode, Schadcode über Compiler einzuschleusen, wäre es so möglich gewesen, Schwachstellen für lange Zeit für Debian-Systeme zu verteilen.

Diese Schwachstelle wurde nicht durch Analyse des Quellcodes gefunden, sondern durch den Umstand, dass eine Person „zufällig“ auf zwei Systemen gleiche Schlüssel erzeugt hat. Auch bei der Lücke bei xz-Utils [3, 7] wurde das Problem zufällig gefunden, weil jemand Verzögerungen bei OpenSSH gefunden hat.

Es lässt sich also festhalten, dass der Mythos, dass Open Source Software regelmäßig auf Schwachstellen studiert wird, nicht wahr ist.

2.10 Trating Input in Secure Programs

Eine Grundannahme bei Eingaben ist, dass möglichst alles nicht vertrauenswürdig ist. Vertrauen sollte immer nur dann gegeben werden, wenn es absolut notwendig ist, z. B.

- Benutzereingaben sollten immer validiert werden.
- Ein Cloud-Anbieter, bei dem Dienste gehostet werden, muss vertraut werden, dass die Eingaben, die man bspw. per WebKVM macht, nicht mitgelesen werden.
- Zugriffe von Diensten, die auf eigenen Servern laufen, sollten nie mehr Berechtigungen auf dem Server haben als notwendig.
- Bei sämtlichen Eingaben/Übertragungen von/über Netzwerke(n) muss angenommen werden, dass ein Angreifer boshafte Daten eingeschleust oder die übermittelten Daten mitgelesen hat.

Wichtig ist auch, eigene Programme so weit wie möglich einzuschränken. Dies beginnt bei der korrekten Konfiguration von Signal-Handling (explizit default-Verhalten einstellen, damit Einstellungen vom Eltern-Programm rückgängig gemacht werden) und geht

bis hin zu freiwilligen Ressourcen- (rlimit) und Zugriffsbeschränkungen (z. B. Landlock LSM).

2.11 Countermeasures

Verschiedene Gegenmaßnahmen für Bedrohungen in Systemen sind möglich. Diese sind unterschiedlich gut. Die weit verbreitete Maßnahme ist „*Penetrate-and-Patch*“, bei der eine Bedrohung erst bekämpft wird, nachdem sie bereits ausgerollt wurde. Im günstigsten Fall wurde eine Bedrohung von einem Sicherheitsforscher gefunden und mit einer Frist gemeldet, zu der die Bedrohung veröffentlicht wird. Patches für solche Bedrohungen werden dann häufig unter Zeitdruck geschrieben und behandeln häufig nur Symptome. Diese Patches werden dann teilweise aber auch nur langsam oder gar nicht ausgerollt, weil Systeme manuell gewartet, mit neuer Firmware bespielt oder zertifiziert werden müssen. In dieser Zeit ist es also möglich, die Bedrohung aktiv auszunutzen. Dieser Ansatz ist der teuerste und unsicherste von allen.

Stattdessen sollte (unabhängig vom Softwareentwicklungsprozess) Sicherheit bereits im Softwareentwicklungsprozess berücksichtigt werden. Idealerweise ist eine Person dafür zuständig, die Sicherheitsaspekte der Software zu prüfen. Solches Personal benötigt aber nicht nur ein Verständnis von Sicherheit, sondern auch ein tiefes Verständnis des Softwareentwicklungsprozesses und muss als Ansprechpartner für Entwickler und Softwarearchitekten zur Verfügung stehen. Solche Personen sind nur schwer zu finden und kostet viel Geld.

Eine weitere wichtige Maßnahme ist die Erstellung von *Sicherheitsanforderungen*, die genau spezifizieren, welche Sicherheitsmaßnahmen umgesetzt werden, welche Daten und Vorgänge schützenswert sind und wie sie geschützt werden sollen. Spezifiziert werden muss dabei, was das System können muss und was es nicht tun darf, warum das System sich in einer bestimmten Art und Weise verhalten soll und wie Informationen geschützt werden sollen. Dabei sollte insbesondere auch berücksichtigt werden, wie schützenswert Daten sind und wie lange sie geschützt sein müssen.

Es ist ein gängiges Mantra, dass Security tief im Softwareentwicklungsprozess berücksichtigt werden muss. Dies wird aber realistisch fast nur für Software gemacht, die zertifiziert werden muss. Sicherheitspersonal sollte schon beim Product Design mitwirken und sich auf Implikationen von Designentscheidungen für die Sicherheit beschäftigen, u. A. wie Daten zwischen Komponenten fließen, wie Berechtigungen durchgesetzt werden und welche Komponenten wie anderen vertrauen.

3 Denial Of Service

3.1 Reaction According to Protocol Specification

Für einen DoS-Angriff wird immer das Protokoll irgendwie ausgenutzt, um möglichst viel Last auf dem Opfersystem zu erreichen. Dafür wird häufig das Verhalten von Protokollen ausgenutzt, wenn mit diesen nicht wie erwartet interagiert wird. Beispielsweise reagiert TCP auf Verbindungsversuche auf geschlossene Ports mit TCP RST.

Mit diesem und anderen Verhaltensweise lässt sich provozieren, dass die Gegenseite Pakete generiert. Dies kann geschickt ausgenutzt werden, um DoS-Angriffe zu fahren.

3.2 TCP-SYN Flood Attack

Grundidee ist, dass der Angreifer sehr viele TCP SYN-Pakete an das Opfersystem schickt. Um Erkennung zu umgehen, kann der Angreifer versuchen, gefälschte Quelladressen in die SYN-Pakete zu schreiben. Das Opfersystem antwortet dann mit SYN ACK an die Rechner (**Backscatter**), denen die gefälschten Quelladressen tatsächlich gehören. Da diese jedoch die Verbindung nicht kennen, antworten diese mit TCP RST, wodurch sich der Angriff zusätzlich verstärkt.

Tatsächlich kann man den Backscatter von solchen Angriffen auch lokal beobachten, indem man einfach mitschreibt, wann TCP SYNACK-Pakete eintreffen, zu denen man selbst keine Verbindung aufgebaut hat.

3.3 Abusing ICMP for Malicious Activities

Ziel des **Smurf**-Angriffes ist, sich selbst als der Opferrechner auszugeben und Broadcast-Pings an das Netzwerk zu schicken. Die Geräte im Netzwerk antworten dann auf diesen Broadcast-Ping, sodass das Netzwerk den Angriff um einen großen Faktor verstärkt. So ein Angriff ist Beispiel für einen **Reflection**-Angriff, bei denen das Ziel ist, dass andere Hosts an das Opfersystem Pakete schicken.

Die Gegenmaßnahme gegen den Smurf-Angriff ist, Broadcast-Pakete aus dem Internet zu sperren. Es gibt ohnehin auch keinen sinnvollen Anwendungsfall für Broadcast-Traffic im Internet.

3.4 TCP Bang Attack

Der Angreifer fälscht IP-Quelladressen in TCP SYN-Paketen. Die Reflektoren antworten darauf dann mit TCP SYNACK-Paketen. Wenn das Opfer bspw. durch Überlast oder

eine falsch konfigurierte Firewall nicht (mehr) mit TCP RST antworten kann, um den Reflektoren mitzuteilen, dass die Verbindung ungültig ist, senden die Reflektoren für eine gewisse Zeit weiter TCP SYNACK, da sie keine Antwort erhalten haben.

3.5 DNS and NTP Amplification

Auch dies sind Reflection-Angriffe, bei denen die Protokolle DNS und NTP ausgenutzt werden, um möglichst große Antworten der DNS-Server an das Opfersystem zu provozieren (bspw. indem Signaturen oder Schlüssel in der Anfrage angefordert werden). Ähnliche Angriffe lassen sich in vielen Protokollen (bspw. auch bei Memcached) fahren, womit sich sehr große Verstärkungsfaktoren erzielen lassen.

3.6 Resource Depletion with Distributed DoS

Im Internet gibt es viele leicht angreifbare Systeme, die jedoch an sich wenige wertvolle Möglichkeiten bieten. Allerdings kann man dort trotzdem Root Kits installieren, die dem Angreifer die Ausführung beliebiger Kommandos auf dem infizierten System erlauben und sich auf dem System bspw. als Treiber verstecken, um vom Administrator nicht gefunden zu werden. Diese infizierten System können jetzt instruiert werden, das eigentliche Opfer anzugreifen. Solche Angriffe werden als **Distributed DoS** (DDoS) bezeichnet.

Dies hilft zum Einen dem Angreifer, nicht erkannt zu werden, verstärkt ganz leicht den Angrifff und macht den Angriff schwerer zu unterbinden, da das Opfer viele Hosts als Angreifer erkennen und sperren müsste. Um noch besser verborgen zu bleiben, werden einige der infnizierten System benutzt, um andere Server zu kontrollieren (Master/Slave). Die Server, die die anderen koordinieren, werden häufig als **Command And Control Server** bezeichnet und können unabhängig vom eigentlichen Angreifer die Angriffe steuern und machen es schwierig, den kompletten DoS durch Ausschalten eines System zu unterbinden.

DDoS-Angriffe können auch wieder mit Reflection-Angriffen kombiniert werden, bei denen die Slaves Reflection nutzen, um noch mehr Last beim Opfer zu erzeugen.

Moderne DDoS-Netze haben eine vermaschte Topologie, wo die Slaves untereinander Nachrichten austauschen, sodass sich der Master nur zu einzelnen Slaves verbinden und Befehle einleiten müssen, damit das gesamte Netz den Befehl ausführt. Die Netze werden mit Verschlüsselung und Signaturen gesichert, sodass das Netz schwerer erkannt wird und bei Erkennung auch schwerer gestört werden kann.

3.7 Defense Techniques Against DoS Attacks

DoS-Angriffe erzielen immer, die Verfügbarkeit des Systems einzuschränken. Verteidigung besteht darin, die Systeme gut zu administrieren, gute Implementierungen zu nutzen und Protokolle so zu designen, dass sie möglichst unanfällig sind. Das beinhaltet regelmäßige Updates von laufenden Diensten, Reviews und Tests von Software (also

z. B. Simulation von DoS-Angriffen) und Fehlertolerantes Design von Protokollen. Weitere Maßnahmen sind Intrusion Detection Systeme und Fehler-Logging. Diese haben aber auch wieder das Problem, dass sie entweder ein Bottleneck bilden können (z. B. durch tiefergehende Analyse von Paketen) oder ausgenutzt werden können, um das System lahmzulegen (bspw. indem durch Logging die Festplatte vollgeschrieben wird).

Eine weitere Maßnahme ist **Ratenkontrolle**. Die Idee ist, dass nur eine gewisse Datenrate verarbeitet werden darf. Damit kann es zwar einfacher sein, das System lahmzulegen, aber es bleiben Ressource frei, sodass ein Administrator bspw. per SSH auf dem System eingreifen kann.

Teure Operationen können, soweit möglich, bspw. erst nach erfolgter Authentifizierung duchgeführt werden, um Last durch „zufällige“ Requests möglichst klein zu halten. Generell sollen teure Operationen nur von bereits bekannten Hosts angefordert werden können. Speicherallokation und Reservierung von CPU-Ressourcen für unbekannte Verbindungen sind immer ein Problem, da ein Angreifer so recht schnell Ressourcen des Systems vom Angreifer blockiert werden können.

Eine weitere Gegenmaßnahme ist, den Angreifer zum Lösen schwerer Aufgaben (sprich: Investition von Rechenzeit) zu zwingen, um auf dem Server teure Operationen auszuführen. So müssen Angreifer mehr Rechenzeit investieren, um auf dem Server wenig Last zu erzeugen.

All diese Techniken helfen jedoch nicht dagegen, wenn der Angreifer das System mit zu viel Traffic überlastet. Dann kann es bspw. auch schon Probleme geben, wenn ein Router überlastet wird. Kommen solche Angriffe nur von wenigen IPs, können die Abuse-Kontakte der dazugehörigen ISPs kontaktiert werden, um den Traffic nah an der Quelle zu stoppen. Außerdem kann im eigenen Netz bereits versucht werden, Adressbereiche zu sperren, die ohnehin nie mit dem eigenen Servern kommunizieren sollen. Das hilft jedoch auch nur begrenzt viel, wenn IP-Adressen gespooft werden. Dafür müssten ISPs eigentlich auf deren Ebene Filter einrichten, sodass Angreifer keine Pakete mit gefälschten Adressen ins Netz leiten können.

Weitere Maßnahmen können sein, die Quelle von Traffic bspw. mit Cookies zu „verifizieren“ oder zu versuchen, die echte Quelle der Pakete zurückzuverfolgen (was schwer ist). Besonderes wichtige Infrastruktur wie DNS Root Server setzen *Anycast* [2] ein, wo mehrere Server die gleiche IP-Adresse bedienen und per Routing-Protokoll announcieren, sodass in verschiedenen Teilen vom Netzwerk der jeweils nächste Server durch Shortes-Path Routing angefragt wird. Ein Angriff auf einen dieser Hosts verhindert dann natürlich nicht, dass die anderen Server weiter in Betrieb bleiben. In moderner Infrastruktur werden auch häufig *Content Delivery Networks* [10] eingesetzt, die die Last auf viele Rechenzentren auf der Welt verteilen, sodass ein Angreifer alle Rechenzentren angreifen müsste, um den Dienst vom Netz zu nehmen. Weitere Maßnahmen können sein, Clients bspw. zur korrekten Ausführung von JavaScript zu zwingen, um Bots herauszufiltern, die nur ganz stumpf Traffic erzeugen.

3.8 Authentication

Moderne Dienste verwenden Authentisierungsverfahren, um sicherzustellen, dass die Gegenseite auch die ist, für die sie sich ausgibt. Dabei werden in Kommunikationsnetzen kryptographische Protokolle eingesetzt, die zusätzlich auch sicherstellen, dass Angriffe auf den Nachrichtenaustausch (z. B. Replay-Angriffe, MITM-Angriffe) nicht möglich sind. Zusätzlich umfassen solche Protokolle manchmal auch einen Schlüsselaustausch. Insgesamt können solche Protokolle zu viel Rechenaufwand auf Servern führen.

Angreifer können versuchen, die Authentisierungsverfahren zu nutzen, um auf dem Server viel Last durch fehlgeschlagene Authentisierungsvorgänge zu erzeugen. Da die Rechenoperationen kryptographischer Protokolle sehr teuer sind, können Angreifer ohne Ratenbeschränkung schnell Überlast erzeugen, indem viele Authentisierungsvorgänge gestartet werden.

Durch Nutzung von Hardware-Sicherheitsmodulen können zwar die meisten Vorgänge nicht nur gegen das Auslesen von Schlüsseln gesichert, sondern auch beschleunigt werden. Jedoch hat Hardware wieder das Problem, dass diese inkompatibel mit neuen Protokollen ist. Verwendet ein Angreifer also die neueste Protokollversion (z. B. bei TLS), kann möglicherweise keine Hardwarebeschleunigung benutzt werden und Angreifer können mehr Last erzeugen.

3.9 Countering CPU Exhaustion with Client Puzzles

Statt jedoch Ratenlimitierung zu benutzen (was problematisch mit CG-NAT ist), können Server (in Situationen hoher Last) Clients eine Aufgabe stellen, die diese zunächst (unter Nutzung von ein wenig CPU-Last) lösen müssen. Die Aufgaben sollten sich leicht vom Server generieren und lösen lassen und idealerweise lassen sich solche Aufgaben nach der Last des Servers in der Schwierigkeit skalieren. Ein Problem dabei ist, dass die Aufgabe immer noch leicht genug zu gestalten, dass einfache (Kunden-)Endgeräte die Aufgabe noch lösen können.

Ein Beispielschema wäre, dass der Server zwei zufällige Zahlen generiert und einen kryptographischen Hash über diese Zahlen berechnet. Der Server gibt den Client dann eine der beiden Zahlen und eine gewisse Anzahl Bits vom Hashwert. Der Client muss nun eine Zahl finden, zusammen mit der übermittelten Zahl eine partielle Hashkollision erzeugt. Bei k bit Ähnlichkeit muss der Client dann im Schnitt 2^{k-1} Berechnungen durchführen, womit sich das Problem exponentiell skalieren lässt. Der Server kann das Ergebnis jedoch mit nur einer Hashberechnung verifizieren.

Eine vereinfachte Version hiervon wäre, keinen Hashwert serverseitig zu berechnen, sondern einfache eine gewisse Anzahl 0-bits am Anfang des Hashes zu fordern.

Grundlegende Eigenschaften sollten solche Verfahren immer erfüllen:

- Die Erstellung und Prüfung der Aufgabe muss wenig Rechenaufwand haben.
- Die Kosten der Aufgabe sollten anpassbar sein.
- Die Aufgabe sollte auf möglichst sämtlicher Client-Hardware lösbar sein.

- Die Vorberechung von Lösungen ist nicht möglich.
- Der Server muss keine Client-spezifischen Daten speichern, um die Aufgabe zu verifizieren.
- …

Aura et. al. haben entsprechend dieser Kriterien ein Verfahren entwickelt.
In den Hash wird dann immer gepackt:

- Ein Client-Identifier (z. B. IP) C
- Nonce des Servers N_s
- Nonce des Clients N_c
- Lösung X

Die Aufgabe ist dann immer, einen Hash mit k 0-bits am Anfang zu generieren.

k und N_s werden periodisch bestimmt und mit Timestamp und Signatur an den Client übermittelt. Der Client verifiziert den Timestamp, generiert N_c, löst die Aufgabe und übermittelt die Lösung samt N_s, N_c, C und Timestamp an den Server. Der Server prüft, ob N_s noch aktuell ist, C und N_c zuvor nicht benutzt wurden, der Timestamp und die Signatur passen und die Lösung korrekt ist.

3.10 Countering Memory Exhaustion: Stateless Protocol Design

Bei vielen Protokollen muss der Server Daten zu Anfragen von Clients speichern (z. B. Sequenznummern in TCP). Dies kann jedoch für DoS-Angriffe genutzt werden, da viele Verbindungsaufbauten erfordern, dass der Server sehr viel Speicher allokiert. Stattdessen kann man den State immer in den Nachrichten zwischen Server und Client mitgeben. Ein Beispiel dafür sind Cookies im WWW. Um hierbei aber weiter sicherstellen zu können, muss der State mit einem *Message Authentication Code* (MAC) gesichert werden, damit der Client die Daten nicht verändern kann. Allerdings erlaubt dies auch den Client, alten State zu übermitteln oder neuen State zu verwerfen.

Solche Verfahren lassen sich in hybrider Weise verweden, bspw. um erst dann State serverseitig zu speichern, nachdem der Client erfolgreich authentisiert wurde. So kann bspw. ein SYN Flood Angriff erschwert werden, indem in das SYNACK-Paket ein (aus den Verbindungsdaten und einem Geheimnis gebildeten) Cookie als Sequenznummer genutzt und bei eingehenden ACK-Pakete verifiziert wird.

Ein solcher Cookie-basierter Ansatz wurde bspw. auch in IKEv2 implementiert.

3.11 Remaining Issues

Die zuvor genannten Maßnahmen können zwar intensiven Rechenaufwand bei einzelnen Verbindungen verringern, schützen aber weiterhin nicht davor, wenn ein Angreifer sehr viel Traffic erzeugt, um einfach den Link oder Server durch viele Pakete zu überlasten.

3.12 Possible Solutions to DDoS Attacks

DDoS-Angriffe lassen sich nur begrenzt eindämmen. Zwar sind Maßnahmen wie die Sperrung von Broadcast-Pings und die Verhinderung von IP-Adress-Spoofing wichtig, aber ansonsten bleibt bei Angriffe nur die Erkennung des Angriffs und Identifizerung des Angreifers.

3.12.1 Ingress Filtering (RFC 2827)

Um IP-Adress-Spoofing zu umgehen, muss an Routern geprüft werden, ob bestimmte Quelladressen überhaupt an bestimmten Links (z. B. Endkundenanschlüssen) aufschlagen können. Im Backbone ist dies allerdings so gut wie unmöglich, sondern muss an Access Links gemacht werden. Zudem muss für IPv6 auf Präfix-Ebene statt für einzelne IP-Adressen die Prüfung durchgeführt werden, da bei IPv6 die Geräte Adressen aus Subnetzen frei wählen können. Das hat nicht nur das Problem, dass es zu viel Rechenaufwand an den Access Routern führt, sondern kann bei Fehlkonfiguration zu Ausfällen bei den Endkunden führen und benötigt zusätzlich ein weites Deployment auf allen Access Routern aller ISPs.

3.13 Identifying Malicious Nodes

Bei einem **DDoS Attack Tree** wird versucht, einen Baum mit dem Opfer als Wurzel aufzubauen, der die Routing-Pfade aufzeigt, die die Pakete von Angreifer-Knoten zum Opfer zurücklegen. Das Ziel dabei ist, den Ort des Angreifers zu identifizieren. Das Problem ist jedoch sehr schwer zu lösen.

Stattdessen wird versucht, verdächtige Pakete direkt zu mit ihrem Pfad loggen, um später leichter Traffic rückverfolgen zu können.

3.13.1 Requirements / Evaluation Metrics

3.13.2 Number of Packets Needed to Track Source

3.13.3 Resource Overhead

3.13.4 Logging Approaches

Die Idee ist zunächst, Pakete zu speichern. Das skaliert jedoch sehr schlecht. Eine Verbesserung dem gegenüber ist Netflow, was das Logging pro Flow macht. Dies hilft jedoch

nicht unbedingt, DDoS zu erkennen, da im Zweifel jedes Paket des Angreifers ein neuer Flow ist.

Ein Ansatz ist, an den Edge Routern Regeln zu schreiben, die verdächtige Pakete über einen Mirror-Port an einen zentralen Tracking Router zu schicken, der zentral „interessanten“ Traffic von den Edge Routern analysieren kann. Dies lässt sich auch in Hardware implementieren und erzeugt damit nur wenig Overhead. So lassen sich die groben Ursprünge von DDoS-Angriffen identifizieren, skaliert aber nicht ganz so gut, da ein zentraler Tracking Router wieder ein Bottleneck darstellt. Im Falle eines DDoS-Angriffes, würde ein solcher Tracking Router viel Traffic empfangen. Dennoch funktioniert das Netz auch dann weiter, wenn der Tracking Router nicht geht.

3.13.5 Source Path Identification

Auf den Routern läuft eine Anwendung, die Daten erfasst und zu einem SCAR zum Sammeln schickt. Ein Traceback Manager kann dann zentral auf die Daten der SCARs zugreifen. Die konstanten Teile des IP Headers (und die ersten 8 Byte der Payload) werden dabei gehasht, um Platz zu sparen und Pakete zu identifizieren.

In einem *Bloom Filter* wird das Paket mit k Funktionen gehasht. An den Stellen im Speicher des Bloom Filters, auf die die Hashfunktionen zeigen, werden jetzt 1en gesetzt. Bei der späteren Frage, ob ein Paket schon einmal gesehen wurde, kann nun wieder gehasht werden, um zu prüfen, ob das Paket schon einmal gesehen wurde. Sind an allen k Stellen die Bits 1, wurde das Paket wahrscheinlich schon einmal gesehen (oder es gab eine Hashkollision). Ist jedoch irgedwo ein Bit auf 0, wurde das Paket definitiv noch nicht gesehen.

Ist der Bloom Filter mindestens 70 % gefüllt, wird dieser an einen SCAR mittelfristigen Speicherung geschickt. So kann auch von vergangenen Daten geprüft werden, ob ein Paket bereits gesehen wurde.

3.13.6 ICMP Traceback

Eine Methode zur Identifikation ist **ICMP Traceback**. Dabei sendet der Router für jedes 20 000ste IP-Paket ein ICMP ITRACE-Paket an das Ziel. Dieses enthält Timestamp, Adresse des Routers, Ingress und Egress Ports sowie eine Kopie der Payload des Pakets zur besseren Identifikation. Bei einem DoS-Angriff kann dann geprüft werden, entlang welches Pfades die meisten ITRACE-Pakete erzeugt werden. Von diesem müssen dann (egal, was in der Quelladresse steht) die DoS-Pakete kommen.

Diese Methode erzeugt zusätzlichen Verkehr und benötigt eine große Menge Paketen, um den Pfad zu rekonstruieren. Außerdem muss das Opfer des DoS-Angriffs auch in der Lage sein, eine Große Menge ITRACE-Pakete zu analysieren. Die größten Probleme kommen jedoch einerseits davon, dass Firewalls ICMP-Pakete gerne droppen und auch Angreifer in der Lage sind, gefälschte ITRACE-Pakete zu erzeugen. Man könnte jetzt zwar überlegen, ITRACE-Nachrichten zu signieren und mit einer PKI zu validieren. Allerdings ist der Aufwand dafür sehr hoch und würde das angegriffene System nur noch stärker belasten.

3.13.7 Probabilistic Packet Marking

Der Ansatz von ICMP Traceback wird hierbei abgewandelt, indem probabilistisch nicht neue Pakete zu erzeugen, sondern den Router im zu übertragenden IP-Paket zu markieren. Dies wird auch als **IP Traceback** bezeichnet. Das Opfer kann jetzt wieder solche Markierungen prüfen und hat dabei sogar die Möglichkeit, den kompletten Angriffspfad in jedem empfangenen Paket zu erhalten.

Problem ist hier, dass die Bandbreite steigt, insbesondere für kleinere Pakete, da jetzt noch zusätzliche Daten an IP-Pakete angehängt werden. Dies kann bei großen Pakete zu Fragmentierung führen bzw. könnte ein Angreifer auch fragmentierte Pakete schicken, sodass ein Router keine Daten mehr anhängen kann (da man Pakete nicht zweimal fragmentieren kann).

3.13.8 PPM Marking

In jedem Paket kann ein einzelner Router hinterlegt werden. Insbesondere ist es auch möglich, dass Router bestehende Markierungen überschreiben. Jeder Router schreibt mit einer gewissen Wahrscheinlichkeit sich selbst in das Paket. Das Opfer führt eine Tabelle darüber, welche Router wie häufig Markierungen eingefügt haben.

Durch das Überscheiben von Markierungen ist es für dem Opfer nahe Router deutlich wahrscheinlicher, dass der Zählerwert sehr hoch wird. Dadurch weiß man, dass Router mit hohen Zähler wahrscheinlich näher am Opfer liegen, was bei der Rekonstruktion des Pfades hilft. Außerdem können so Angreifer eine Rückverfolgung nicht verhindern, indem sie einfach irgendwelche Werte in das Router-Feld des Pakets schreiben.

Zusätzlich zum Router wird der Router sich selbst als Start-Router mit Distanz 0 eintragen. Empfängt ein Router ein Paket mit Distanz 0, ist er der Next Hop des Start-Routers und trägt sich als End-Router ein. Außerdem wird immer die Distanz um 1 erhöht, wenn der Router nicht Start-Router ist. So empfängt das Opfer in der Markierung einen Link und die Distanz zu diesem Link in Hops. Durch Sammlung vieler solcher Informationen lässt sich ein Angriffsbaum konstruieren. Um gefälschte Einträge zu erkennen, können Router gepingt werden, um Abweichungen in der Distanz zu finden.

Es gibt jetzt noch das Problem, dass IP-Pakete keine Headerfelder für so etwas vorgesehen haben. Die Forscher haben sich hier überlegt, bestehende Headerfelder wieder zu benutzen. Da die Zieladresse eines Links (irgendwann) bekannt ist, wird die Kante als Start XOR End markiert. So kann später, sobald die Endadresse bekannt ist (beim Last Hop ist diese immer bekannt), die Startadresse durch XOR rekonstruiert werden. Das klappt in der Praxis aber nicht ohne weiteres, da Router häufig mehrere IP-Adressen (verschiedene Adressen an verschiedenen Links) haben, weshalb die Endadresse nicht unbedingt bekannt ist.

Ein zweiter Ansatz sieht vor, nur einen Teil der Information zu übertragen. Das Opfer kann dann (durch Aufzeichnung vergangener Edge-IDs) die Informationen rekonstruieren. Ein Problem hierbei ist aber nach wie vor, dass das verwendete ID-Feld des IP-Pakets im Falle von Fragmentierung weiter benötigt wird. In so einem Fall können Downstream ICMP Errors für PMTU-Discovery geschickt werden. Upstream kann aber auch durch

setzen des „don't fragment“ Flags Fragmentierung verhindert werden.

Auch, wenn dieser Ansatz Zeit kostet, funktioniert er recht stabil. Dabei wird zusätzlicher Bandbreiten-Overhead vermieden. Angreifer können sich zwar durch Manipulation „näher“ an das Opfer schummeln, jedoch niemals weiter weg (was nötig wäre, um eine Rekonstruktion zu erschweren).

Allerdings funktioniert das Verfahren nur bei Bandwith Exhaustion Attacks, da viele Pakete für eine Rekonstruktion notwendig sind. Unter Ausnutzung irgendwelcher Programmierfehler (z. B. Teardrop-Angriff) können DoS-Angriffer mit diesem Verfahren weiterhin nicht rückverfolgt werden. Fragmentierte Pakete können gar nicht richtig nachverfolgt werden. Außerdem benötigt das Opfer viel Speicher zur Rückverfolgung und muss viel Rechnen. Hauptproblem ist jedoch, dass alle Router dieses Verfahren mitmachen müssen, da an jedem Router die Distanz erhöht werden muss.

3.14 Related Techniques for Mitigation / Avoidance

3.14.1 Hop Count Filtering

Die Frage ist zunächst, ob sich Verkehr an anhand der gesendeten Daten als Angreifer-Verkehr identifizieren lässt? An sich ist dies nur zu Beginn des Angriffspfades möglich. Allerdings lässt sich die TTL nicht fälschen, da Pakete sonst nicht am System ankommen. Dazu könnte man jetzt versuchen zu schätzen, ob die TTl von Paketen sinnvoll ist. Dazu muss bekannt sein, wie weit das Gerät tatsächlich entfernt ist und wie die TTL bei einem „echten“ System zu Beginn gesetzt worden sein müsste.

Damit soll jetzt u. A. ein Reflektor-Angriff verhindert werden, indem die Reflektor-Nodes zunächst prüfen, ob die TTL von Absendern tatsächlich zu dem passt, was zu erwarten ist (Sanity Check). Allerdings ist es auch hier weiter möglich, dass der Angreifer eine „sinnvolle“ TTL einträgt.

3.14.2 Aggregate Based Congestion Control

Die Kernidee ist hier, Congestion Control im Backbone durchzuführen, um Angreiffsverkehr einzuschränken. Da Verkehr im Backbone sehr divers ist, sollte das große Verkehrsvolumen eines Angriffs als Aggregat durch Analyse des lokal gedroppten Traffics erkannt werden. Wenn jetzt also eine Zieladresse sehr viele Drops hat (z. B. wegen voller Output Queue), kann man sich dazu entschieden, diesen Traffic gänzlich zu unterbinden bzw. den Upstream-Router des Quellports zu identifizieren (sofern sich ein einzelner Upstream-Port identifizieren lässt).

Hauptprobleme hierbei sind nicht nur, dass dabei auch potentiell „guter“ Traffic mit weggeworfen wird, sondern auch dass Router dies wieder unterstützen müssen.

3.14.3 Secure Overlay Services / Onion Routing

Die Idee ist hierbei, Angriffe zu verhindern, in dem der schwache Anschluss des Dienstes „versteckt“ wird, sodass Angreifer die Position des Systems im Netzwerk (die IP-Adresse)

nicht herausfinden können. Da jedoch echte Anfragen weiter den Server erreichen können müssen, müssen davor Schichten liegen, die den Zugriff ermöglichen. Diese Stellen haben dann viel Leistung/Bandbreite und können eine große Verkehrsmenge entgegennehmen und filtern, damit Authentisierung bzw. DoS-Erkennung noch vor dem eigentlichen Anwendungsserver durchgeführt werden kann. So erreicht möglichst nur legitimer Traffic den eigentlichen Anwendungsserver.

4 Routing Security

4.1 General Threats

Routing im Internet ist angreifbar. Dabei können beispielsweise mithilfe manipulierter Links oder manipulierter Router (Routing-)Informationen erspäht, andere Router getäuscht (bspw. mit falsche Routing-Informationen), der Betrieb anderer Router gestört oder auch Traffic im Internet umgeleitet werden, um Traffic zu „stehlen“. Unterschieden wird, ob sich die Störung nur auf einzelne Knoten, Teile des Netzes oder das gesamte Internet erstreckt und ob die Konsequenzen sich nur während des Angriffs oder möglicherweise auch noch danach auswirken.

Konsequenzen für das Netz lassen sich auch gliedern.

- Stau kann auftreten, wenn Traffic plötzlich andere Wege nimmt.
- Wenn Pakete „verschwinden“, ist von einem **Blackhole** die Rede. Dies kann beispielsweise ein Router sein, der Pakete empfängt und einfach verwirft oder eine Reihe von Routern schicken Pakete im Kreis (**Looping**).
- Das Netzwerk kann durch Störungen partitioniert werden. Bei einem Routingangriff können sich die Geräte zwar physisch noch erreichen, aber durch manipuliertes Routing gibt es keine Route mehr zwischen den Teilen der Partition.
- Ständige Routen-Änderungen können das Distanzvektorprotokoll (z. B. BGP) dazu bewegen, viele Routing-Pakete zu verschicken und damit Router zu belasten. Das kann zu Instabilität und langsamer Konvergenz im Routingprotokoll führen und Routing-Prozessoren der Router zu überlasten.
- Delay und Jitter können sich verschlechtern, wenn schlechtere (stärker belastete) Routen genutzt werden.
- Durch Partitionierung können Teile des Netzes abgeschnitten werden.
- Traffic kann in einen Teil des Netzes geleitet werden, der diesen gar nicht (mehr) weiterleiten kann.
- Wenn Traffic zum Angreifer geroutet wird, kann dieser den Traffic mitlesen und möglicherweise analysieren.
- Bei **Controlled Delivery** bzw. **Greyhole-Angriffen** wird der Traffic durch das Angreifernetz geeitet, sodass der Angreifer aussuchen kann, welcher Traffic weitergeleitet und welcher verworfen wird.

Routing-Informationen lassen sich durch Angriffe ermitteln bzw. veröffentlichen, um bspw. geheime Informationen über Netzwerke offenzulegen. Durch Eavesdropping von Routing-Paketen lassen sich solche Informationen ermitteln. Auch durch die Analyse von weitergeleitetem Traffic kann sich erahnen lassen, welche Pakete wie geroutet werden.

Angreifer können vortäuschen, ein Router zu sein, um weitere Angriffe zu fahren. Durch die Verhinderung des Austauschs von Routing-Informationen zwischen Routern kann das Routing gestört werden. Durch bereitstellung falscher Routing-Informationen (bspw. durch Erstellung dieser oder Fälschung weiterzuleitender Informationen) lässt sich das Routing bspw. durch Änderung von Kosten modifizieren:

- Bei **Overclaiming** behauptet man, selbst eine bessere Route für Verkehr zu haben, den zu dem man eigentlich keine gute Route hat. Dadurch wird Traffic angezogen und dann kann manipuliert bzw. mitgelesen werden.
- Bei **Underclaiming** behauptet man, man hätte schlechtere Routen. Damit kann man zwar sich selbst weniger Traffic senden lassen, aber das bringt wenig. Stattdesse kann man aber auch die Routen zu einem anderen Router teurer erscheinen lassen, um bspw. Verkehr aus bestimmten Regionen umzuleiten, wo sich dieser besser manipulieren/analyisieren lässt, um bspw. das Netz zu stören oder zu überlasten.

Bei *Resource Exhaustion* ist das Ziel, möglichst häufig Routen zu aktualisieren, damit viel Last vom Routing-Protokoll erzeugt wird. Das Ziel hierbei ist, den Betrieb des Routing-Protokolls zu stören.

Außerdem können Ressourcen im Netz zerstört werden, bspw. durch Zerstörung von Links (Kabel durchschneiden) oder Knoten (z. B. Router durch Softwarefehler zum Absturz bringen).

Beim **Sinkhole Angriff** wird versucht, bspw. durch Overclaiming Traffic anzuziehen, um dann mit weiteren Angriffen wie Greyhole-Angriffen den Traffic zu stören.

Bei einem **Wormhole Angriff** wird nicht das Routing-Protokoll direkt manipuliert, sondern bspw. durch Tunnelung die Topologie bösartig verzerrt.

Im Folgenden wird sich vor allem auf Inter-AS-Routing konzentriert.

4.2 Inter-AS Routing Threats in the Internet

Im Internet gibt es verschiedene Angriffsszenarien, von der einfachen Störung bis zum Multi-Homing. Ziele können sein, Alternative Pfade zu/von bestimmten ASen zu erstellen oder auch Traffic in ein „Blackhole“ umzuleiten, statt ihn korrekt weiterzuleiten.

Dazu können u. A. IP-Adressen announciert werden, die dem eigenen AS nicht gehören, nicht autorisierte Präfixe in Routingtabellen eingetragen werden, für die gar keine zulässigen Routen existieren, Routing-Nachrichten zwischen Routern manipuliert werden oder auch Ressourcen zerstört werden (also bspw. Kabel zerstören).

Solche Angriffsformen sind schon in der Praxis realisiert worden. Beispielsweise hat 2013 ein belarussischer Provider Traffic von GlobalOneBel über einen Uplink nach Moskau gesendet. Der Traffic hat weiterhin funktioniert, aber nur einen anderen Delay ge-

habt, da er nach der Umleitung wieder nach Frankfurt geleitet wurde. Dieser Angriff konnte über einen Monat im Internet unerkannt bleiben.

Regierungen haben beispielsweise auch schon Routingprotokolle manipuliert, um Traffic zu bestimmten Internetplattformen (z. B. YouTube) in ein Blackhole zu leiten, damit diese Dienste nicht im eigenen Land erreichbar sind. Da diese Route fehlerhafterweise auch an ASe außerhalb des Landes weitergeleitet wurden, wurde der Dienst für (netztopologisch) die halbe Welt unerreichbar gemacht, da der Traffic der anderen ASe auch zum Blackhole geroutet wurde. Die andere (dem Dienst netztopologisch nähere) Hälfte des Internets war nicht betroffen, da dort kürzere Routen zum echten Dienst vorlagen.

4.3 Securing BGP Operation

4.3.1 Verifying Peer Messages

Eine Idee, BGP abzusichern, ist zunächst, BGP-Pakete mit sehr geringer TTL zu verschicken. Eine TTL von 1 würde beispielsweise erzwingen, dass Router direkt miteinander verbunden sein müssen. Damit lassen sich selbst bei einem kompromittierten Router als Absender keine BGP-Verbindungen zu entfernten Routern aufbauen, da BGP über TCP läuft und entsprechend die Gegenseite auch antworten können muss. Da die Gegenseite aber nur Pakete mit TTL 1 sendet, können die Antwortpakete der TCP-Verbindung nicht beim Angreifer-Router ankommen.

Eine alternative, generalisierte Idee dazu ist, die TTL maximal hoch zu setzen und empfängerseitig zu prüfen, ob das eingehende IP-Paket noch eine TTL hat, die mindestens einen gewissen Wert hat. So kommt das Paket unabhängig von der Einstellung der Gegenseite an, aber die Parteien können trotzdem validieren, dass die Gegenstelle nur eine Maximalzahl an Hops entfernt ist.

Eine bessere Alternative ist, Routing-Nachrichten zwischen Peers zu authentisieren. Dazu gibt es eine Methode, BGP-Sessions über die TCP MD5 Signaturoption abzusichern (RFC 2385 [5]). Dabei wird ein MD5-Hash über jedes TCP-Segment und ein der Gegenseite bekanntes Geheimnis berechnet. Da allerdings keine automatische Schlüsselaushandlung oder Updateprozeduren bereitstehen, ist das Deployment schwierig.

Dagegen ist eine bessere Idee, eine bestehende Sicherheitsarchitektur wie IPsec zu nutzen, um Nachrichten abzusichern und sogar zu verschlüsseln. Dabei gibt es jedoch wieder das Problem des Key Exchange, da gerade Backbone-Router sehr viele Schlüssel austauschen (und dabei teure asymmetrische Kryptographie anwenden), bis überhaupt die BGP-Sessions ausgehandelt werden können. Hier besteht aber auch die Gefahr, ein Henne-Ei-Problem zu bauen, da die Kryptographie teilweise auch von funktionierendem Routing abhängen könnte (z. B. bei Certificate Revocation Lists).

4.3.2 Problems Beyond Simple Peer-to-Peer BGP Security

Nur einen Kommunikationskanal für Manipulation zu sichern, ist nicht ausreichend. Es ist weiter möglich, dass ein bösartiger Router falsche Routing-Informationen sendet. Um

dies zu verhindern, müssen mehrere Vorgänge durchgeführt werden, um die Authentizität der Routen sicherzustellen.

Im Bereich der *Secure Origin Authentication* muss sichergestellt werden, dass nur IP-Präfixe announciert werden dürfen, die einem AS auch zugewiesen sind, dass Router auch tatsächlich die von ihnen angegebene AS-Nummer benutzen dürfen und dass die Router auch zu dem AS gehören, was sie announcieren.

Im Bereich der *Path Authentication* wird sichergestellt, dass announcierte Routen valide sind. Dazu muss sichergestellt werden, dass bestimmte Adress-Ranges/Routen auch announciert werden dürfen. Ein noch ungelöstes Ziel ist die Authentisierung, dass bestimmte Routen zurückgezogen werden (z. B. wenn ein Router ausgeschaltet wird).

4.3.3 Secure Border Gateway Protocol

S-BGP ist eine Methode, BGP abzusichern. Grundsätzlich wird hierbei IPsec eingesetzt, um den Kommunikationskanal abzusichern. eine PKI stellt sicher, dass BGP-Partner, Besitzer von ASen und deren Adressblöcke sicher identifiziert werden können. Es kann attestiert werden, dass bestimmte Subjekte bestimmte Adressblöcke announcieren dürfen und BGP UPDATEs werden validiert. Weiterhin werden CRLs geführt, um bspw. geklaute Schlüssel zu invalidieren.

Im Internet wird die vergabe von IP-Adressen und AS-Nummern hierarchisch durchgeführt. An der Spitze verteilt die *Internet Assigned Numbers Authoritiy* (IANA) größere IP-Adressblöcke an regionale Registries (RIRs), die wiederum hierarchisch kleinere Blöcke an lokale Registries oder Provider vergeben. So kann auch die PKI so aufgebaut werden, dass die IANA die Autorität hat, alle IP-Adressen zu vergeben. Bei der Vergabe von Adressblöcken an RIRs werden Zertifikate erstellt, dass diese IP-Adressblöcke von den RIRs weiterverteilt werden dürfen. So können diese wiederum zertifizieren, dass kleinere Adressblöcke dieser größeren Blöcke bestimmten ISPs/LIRs/ASen genutzt bzw. weitervergeben werden dürfen. Ähnlich muss auch registriert werden, wer welche AS-Nummer verwenden darf. Der Admin eines AS muss außerdem festlegen, welche Router Teil des AS sind.

So kann jetzt ein Router UPDATEs vom eigenen Präfix mit Signatur announcieren. Nachbarn können diese validieren und sich selbst in den Pfad einfügen. Dafür wird ein neues (optionales) Argument hinzugefügt, in der die Route mit einem bestimmten Next Hop AS autorisiert wird. Der Besitzer eines AS- (oder der Router-) Zertifikats generiert dafür Attestierungen. So wird die Route in jedem Schritt um weitere Attestierungen verlängert, sodass sich eine signierte Sequenz von Attestierungen für jede Route ergibt, die validiert werden kann. Zur Validierung der Attestierungen reicht natürlich nicht nur die Validierung der Subscriber-Zertifikate. Stattdessen muss die komplette Zertifikatskette bis zum Root-Zertifikat ähnlich wie bei TLS hierarchisch validiert werden. Ähnlich müssen Router auch zeritifizieren, dass sie zum AS gehören. Dieses Zertifikat muss auch wieder vom dazugehörigen AS signiert sein, was wiederum von der RIR signiert sein muss, damit sichergestellt werden kann, dass der Router das AS überhaupt beanspruchen darf. Das Zertifikat der RIR über den AS-Block wiederum muss von der IANA signiert sein, um zu attestieren, dass die RIR die Befugnis hat, AS-Nummern aus diesem

Block weiterzuverteilen.

Zu beachten ist bei den UPDATEs, dass mit attestiert wird, an welche Gegenstelle (welches AS) das UPDATE verschickt wird, damit sichergestellt werden kann, dass das korrekte AS dann auch die Route weitergibt (und nicht jemand anderes ein signiertes UPDATE klaut und anderswo daraus eine scheinbar valide Route baut).

Zur Validierung müssen Zertifikate und CRLs so verteilt werden, dass Router diese abfragen können. Dazu werden Server in der Nähe der Router bereitgestellt, die redundant ausgelegt sind und skaliert werden können. Dabei könnte man jetzt entweder die kompletten Datenbanken für Zertifikate/AAs/CRLs herunterladen oder einfach immer Anfragen on demand stellen. Das erfordert allerdings auch wieder, dass diesen Servern vertraut wird.

Das zweite Problem ist der Ressourcenverbrauch. Einerseits müssen Zertifikate gespeichert und validiert werden. Andererseits benötigen Attestierungen Speicherplatz und CPU-Ressourcen für die Signierung und Validierung von Attestierungen. Außerdem müssen diese übertragen werden, was weitere Ressourcen erfordert. Im Jahr 1999 waren die Routing-Tabellen deutlich kleiner. Bereits damals war eine Abschätzung, dass eine Datenbank von Routing-Zertifikaten mit ca. 26 Megabyte. Dabei wurden CRLs nicht einberechnet.

Weitere Probleme sind die fehlende Möglichkeit, Routen zurückzuziehen, vorzeitige Wieder-Announcierung zurückgezogener Routen, falsche Weiterleitung von Traffic oder auch falsche Topologieänderungen (z. B. Wormholes). S-BGP wurde bis heute nicht im Internet eingesetzt. Dennoch zeigt S-BGP, welche Aufgaben notwendig sind, um BGP abzusichern und gibt eine Sicht darauf, wie viel Aufwand dafür notwendig ist.

4.3.4 S-A (Signature Amortization)

Da Nachrichten deutlich häufiger validiert als signiert werden, ist die Idee von S-A, RSA als Signaturalgorithmus einzusetzen, da hier die Validierung von Nachrichten im Vergleich zur Signatur recht schnell geht. Allerdings benötigt RSA einen höheren Aufwand zum Signieren, da größere Schlüssel notwendig sind.

S-A-P (Signature Amortization Across Peers)

Statt jedem Nachbarn individuelle BGP UPDATEs zu senden, wird ein Update mit mehreren Nachbarn (Targets) versehen, einmal signiert und an alle betroffenen Nachbarn verschickt. Dadurch können viele Signaturen eingespart werden.

Zur Validierung von vielen Daten werden Merkle Hash Trees eingesetzt. **Merkle Hash Trees** sind Baumstrukturen, bei denen Blätter Daten enthalten und innere Knoten Hashes aus den Kindknoten bilden. So sichert der Wurzelknoten mit einem Hash alle Blätter ab. Wird also dieser Hash signiert, sind alle Informationen an den Blättern signiert.

S-A-B (Signature Amortization Across Buffers)

Eine Beobachtung zeigt, dass BGP-Router neue Nachrichten erst nach einem Intervall akzeptieren, um Instabilitäten zu vermeiden (das *Minimum Route Advertisement Interval*). Während dieser Pufferung können Updates in einem Merkle Hash Tree zusammengefasst werden, sodass bei der Weitergabe der Route nur die Wurzel des Baums signiert werden muss. So spart nicht nur der Signierer Aufwand bei der Signatur, sondern der Validierer kann später die Updates einfacher validieren.

4.3.5 Secure Path Vector Protocol – Lamport Signatures

Da digitale Signaturen sehr rechenintensiv sind, ist die Kernidee jetzt, nur kryptographische Hashes einzusetzen. Dazu werden sog. *Lamport Signatures* eingesetzt.

Im ersten Schritt werden dabei Tupel von Zufallszahlen als Secret key erzeugt. Der Public Key ist dann ein Tupel aus Hashes $H\left(s_{i,1/0}\right)$ der Zufallszahlen. Lücke

Eine Verbesserung besteht darin, dass der Public Key die Wurzel eines Merkle Hash Trees ist. Der Secret Key ist dann die Menge aller n Blätter. Eine Singatur kann berechnet werden, indem der Hash $\mod n$ genommen wird. s_i sowie alle für die Berechnung der Wurzel nötigen Hashwerte werden veröffentlicht. Mit den Hashwerten lässt sich die Wurzel berechnen, um Authentisierung durchzuführen. Durch den Vergleich der Position von s_i mit $H(m) \mod n$ kann Integrität sichergestellt werden.

Vorteil ist, dass der Public Key und die Signatur weniger Platz benötigen. Allerdings muss n jetzt sehr groß gewählt werden, wodurch es praktisch nicht berechenbar ist.

4.3.6 Secure Path Vector Protocol – HORS Signature

Die Idee ist, mehr als einen Teil des privaten Schlüssels zu veröffentlichen. Das HORS-Signaturschema wählt m von den n Teilen des privaten Schlüssels. Dabei werden nicht nur die letzten Bits genutzt, um einen einzelnen Private Key zu identifizieren, sondern $m \log_2 n$ Bits. Dadurch entstehen sehr viele Kombinationen von Teilen des Schlüssels, wodurch die Chance, eine zweite (für den Angreifer nützliche) Nachricht zu finden, die zu einer existieren Signatur passt, sehr gering wird.

Dies gilt allerdings nur für die erste Signatur. Wird ein Schlüssel erneut benutzt, reduziert sich die Sicherheit, da mehr Teile des privaten Schlüssels bekannt werden. Schon nach wenigen Runden ist die Sicherheit nicht mehr ausreichend.

4.3.7 Secure Path Vector Protocol – Implementation

Um möglichst wenige Signaturen erzeugen zu müssen, sollen nur so wenige Daten wie möglich signiert werden. SPV benutzt dafür modifzierte HORS-Signatur. Es wird ein Abbildungsscheama mit sogenannten „AS-Path Protektoren“ gebildet. Dies war aber so kompliziert, dass schwere Probleme aufgetreten sind und Angreifer sogar bei bestimmten Topologien AS-Pfade modifizieren konnten. Die benötigten Hashoperationen haben das Verfahren auch nicht schneller als S-A gemacht. Dafür hat man aber lernen können, dass

ein komplexes System mit vielen schnellen Operationen nicht unbedingt schneller ist als ein einfaches System mit wenigen langsamen Operationen.

4.3.8 Interdomain Route Validation

S-BGP funktioniert nicht wirklich, wenn ein Zwischenhop kein S-BGP spricht und Routing-Informationen nur weitergibt. Ein alternatives Verfahren ist *Interdomain Route Validation* (IRV), ein dezentrales System mit Fokus auf Interoperabilität. Statt die Router die Validierung durchführen zu lassen, gibt es in einigen ASen einen zentralen IRV-Server, die Routen validieren können. Die IRV-Server können sich auch gegenseitig zur Authentizität von Routen befragen und so falsche Routen erkennen.

Jeder IRV-Server repräsentiert dabei ein einzelnes AS. Das AS validiert eingehende BGP UPDATE Nachrichten, indem die IRV-Server entlang des AS-Pfades der Reihe nach gefragt werden. Die zu befragenden IRV-Server werden in Feldern in den BGP-Nachrichten eingebettet. Das Risiko ist hier weiterhin, dass ein IRV-Server einen anderen IRV-Server belügen kann (auch, wenn Nachrichten mit PKI abgesichert sind). Aber da sich die Prüfung bis zur Quelle der Route durchzieht, wird es schwieriger, Manipulationen zu verstecken. Ein weiteres Problem tritt auf, wenn ein Totalaufall auftritt. In so einem Fall muss eine Sonderbehandlung gemacht werden, da IRV-Server möglicherweise nicht erreicht werden können.

4.3.9 Secure Origin BGP

soBGP ist ein IETF Draft, der nicht weiter realisiert wurde. soBGP baut auf S-BGP auf, aber versucht, auf PKI zu verzichten, kein funktionierendes Internet-Routing vorauszusetzen und auch ein inkrementelles Deployment zu ermöglichen.

Für den Zertifikatstransport werden jetzt SECURITY Nachrichten eingeführt, um bspw. Route Attestations zu senden. UPDATE Nachrichten werden nicht modifiziert, um Abwärtskompatibilität zu gewährleisten. Für die Validierung und Zertifikatsaustausch gibt es verschiedene Setups, entweder direkt auf Routern, über einen zentralen Server oder als Mischoptions.

Anstatt einer PKI soll ähnlich zu PGP ein *Web of Trust* etabliert werden. Hier ist natürlich fragwürdig, ob das indirekte Vertrauen über das Web of Trust auch tatsächlich gut ist. Für soBGP wird jetzt aber jedenfalls angenommen, dass man ASen traut, denen die eigenen Nachbarn trauen. Man kann jetzt über die „Distanz“ eines AS zu einem vertrauenswürdigen Knoten auch eine Tiefe für das Vertrauen konfiguriert werden.

Sicherheit ist bei soBGP nicht bewiesen, da es keine Trusted Third Party gibt. Angreifer können zusammenarbeiten, um in PolicyCerts zu lügen. Bei inkrementellem Deployment lässt sich außerdem kein Vertrauen zu stellen aufbauen, die kein soBGP unterstützen.

4.4 BGPSEC and RPKI

BGPSEC und *RPKI* bauen auf S-BGP auf. BGPSEC und Resource Public Key Infrastructure behandeln Routing Attestations und stellen ein Verzeichnis für Secure Origin Authentication sicher. Zentrale Root-Zertifikate werden dabei nicht benutzt.

RPKI wird bereits seit 2010 schrittweise im Internet ausgerollt. Seit 2014 (nach einigen Angriffen auf die Internet-Infrastruktur) wird RPKI zunehmend deployt.

4.5 Securing BGP by State Observation

Kryptographische Ansätze haben das Problem, dass sie viele Rechen- und Kommunikationsaufwand haben. Dafür sind üblicherweise PKI und zentrale Datenbanken notwendig. Inkrementelles Deployment liefert außerdem nur wenig Sicherheitsgewinn. Außerdem hilft all dies nicht gegen Instabilitäten, bspw. durch Angreifer.

Stattdessen wurde andere Ansätze ausprobiert, die ohne Kryptographie auskommen. Dafür wird der Zustand von BGP und der Routing-Tabellen überwacht, um mit Heuristiken Angriffe erkennen zu können.

4.5.1 Pretty Good BGP: Cautiously Adopting Routes

Viele falsch Origin/Prefix-Assoziationen existieren weniger als 24 Stunden im Netz. Die Kernidee ist also, neue unbekannte Routen nur vorsichtig zu importieren. Eine History über früher schon einmal gesehen Routen wird gebildet und für einige Tage gecached. Neue Routen werden mit der Datenbank abgeglichen und für eine gewisse Zeit als „verdächtig“ markiert. Nachdem diese Zeit vergangen ist, werden die Routen aufgenommen, sollten sie noch announciert werden. Alternativ können neue Routen auch eine geringe Präferenz bekommen. So werden zunächst „alte“ Routen präferiert und der Router wählt die beste „vertrauenswürdige“ Route. Falls diese nicht existiert, wird eine nicht vertrauenswürdige Route gewählt.

Allerdings können Probleme auftreten, wenn Sub-Präfixe eingeführt (bzw. vom Angreifer generiert) werden. In dem Fall wird zunächst die bekannte Route mit der größeren Präfixlänge gewählt. Außerdem haben alle Angreifer Erfolg, die länger als die Wartezeit warten.

4.5.2 Topology-Based Analysis

Die Topologie des Internets wurde analysiert. Dabei hat sich ergeben, dass Core-Router im Backbone sehr dicht miteinander verbunden sind. Daran angebunden sind *Periphery Nodes*, die eher am Rand sind und zum Backbone sowie einigen wenigen Nachbarn verbunden sind.

Jetzt werden die Router nach geographischen Informationen (z. B. aus der whois-Datenbank) geclustert. Die Durchmesser der meisten Cluster sind dann sehr gering. Gültige Routen gehen höchstwahrscheinlich von einem Cluster nur einmal durch das Backbone und erreichen dann ein anderes Cluster. Liegt jetzt ein AS einer Router im

Randbereich, wird der Pfad zweimal durch das Backbone gehen, was auf einen gefälschten Pfad hinweist.

Nachteile umfassen unter anderem, dass Angreifer innerhalb eines Cluster weiter Traffic anziehen können. Außerdem ist es schwierig, zuverlässige geographische Informationen zu bekommen. Die Authentizität und Aktualität von whois-Daten ist außerdem auch fraglich und es gibt keine Algorithmen, um dynamisch den Konnektivitätsgraphen zu aktualisieren und ASe zu clustern.

4.5.3 Stable Route Information Objects

SRIO ist ähnlich zu Pretty Good BGP, arbeiten damit, dass sich die direkten Links zwischen Nachbarn und Prefix/Origin Associations im Normalfall nicht (häufig) ändern. Daraus wird eine Datenbank gebildet, gegen die BGP UPDATEs geprüft werden können.

Gibt es in einem UPDATE einen neuen gerichteten Link, wird dieser zunächst als verdächtig eingestuft. In dem Fall wird wieder abgewartet, ob die Route bestehen bleibt. Dies lässt sich noch mit ein paar Heuristiken verbessern, indem beim Erfüllen bestimmter Kritieren Routen sofort aufgenommen werden, wenn sich die lokale Routingentscheidung dadurch nicht ändert.

Weitere Heuristiken können bspw. aufgrund der Ähnlichkeit der IP-Adresse bilden lassen, führen aber zu weiteren falsch negativen Ergebnissen. Hier ist eine Abwägung notwendig, um einen gute Balance zu schaffen.

4.5.4 Monitoring TCP Flows

Die Grundidee ist hier, zu beobachten, ob TCP Traffic über diese Route funktioniert. Da jedoch Routen unsymmetrisch sein können, kann man nicht garantieren, dass auch Antwortpakete gesehen werden können.

Bei TCP kann man also zwar nicht das SYNACK-Paket als Antwort auf SYN sehen, aber höchstwahrscheinlich kann ein ACK-Paket vom Sender des SYN-Paketes gesehen werden. Passiert dies innerhalb einer gewissen Zeit, wird dies als Zeichen angesehen, dass eine Route funktioniert. Wird eine gewissen Zeit beobachtet, welche Routen funktionieren, kann man nicht funktionierende Routen als verdächtig einstufen.

Dies funktioniert nur bedingt, da nicht zwangsläufig in alle Richtungen Verbindungen aufgebaut werden. Außerdem können Angreifer dieses Verhalten ausnutzen, um bspw. mit deinem SYN Flood Angriff eine Route als nicht funktionierend erscheinen zu lassen. Dann schlägt möglicherweise der Erkennungsmechanismus an und sperrt die Route.

5 Secure Name Resolution

Bevor Pakete überhaupt an einen Server geschickt werden, werden normalerweise zunächst bei Nameservern anfragen gestellen, um die IP-Adresse einer Gegenstelle vor dem eigentlichen Kommunikationsvorgang zu ermitteln. DNS ist also eine Schlüsseltechnologie im Internet, deren Störung große Folgen haben kann.

5.1 Security of the Domain Name System

DNS bietet schon einen einfachen Schutzmechanismus, da gefährlich aussehende DNS-Namen von Nutzern (hoffentlich) nicht angesurft werden. Hingegen können natürlich auch vertrauenswürdig aussehende Namen von Angreifern benutzt werden. Nicht zuletzt ist dies möglich, sollte es ein Angreifer schaffen, dass das DNS seine IP-Adressen ausliefert.

DNS bietet keine Integrität, keine Data Origin Authentication und Vertraulichkeit. Zu den möglichen Bedrohungen zählen Angriffe auf Authentizität und Integrität sowie Denial of Service.

5.2 DNS Structure

DNS ist als hierarchische verteilte Datenbank strukturiert, Bei der Root-Domain-Server (Zone `.`) Informationen bereitstellen, welche Nameserver darunterliegende Zonen (`com.`, `de.`, etc.) haben, welche wiederum Angaben machen, welche Nameserver darunterliegende Zonen (`web.de.`, `google.com.`, etc.) haben. Diese (und potentiell weitere nachgelagerte Nameserver) liefern dann die eigentlichen autoritativen DNS-Informationen (also z. B. welche IP-Adresse hinter einem Hostnamen steckt) aus. Diese Struktur gibt autoritative Antworten aus der Datenbank, die als objektive Wahrheit angesehen werden. Autoritative Server lösen jedoch in der Regel nicht alle DNS-Anfragen auf, sondern beantworten nur Anfragen zu DNS-Anfragen für die Zonen, die sie verwalten.

Bei der Namensauflösung machen Clients in der Regel Anfragen bei einem Recursive (Caching) Server, einem Nameserver, der stellvertretend für Clients DNS-Anfragen bei den autoritativen Servern auflöst bzw. aus dem lokalen Cache in der Vergangenheit durchgeführte Anfragen direkt beantwortet. Anfragen werden von Resolvern (Software auf Endgeräten) gestellt, die die Ergebnisse der DNS-Anfragen dann an den jeweiligen Prozess übermitteln, der auf dem Endgerät eine Adresse auflösen möchte.

5.3 DNS Security Objectives

DNS kann auf verschiedene Weisen angegriffen werden. Neben Angriffen auf die Verfügbarkeit können DNS-Server für Amplification benutzt werden. Außerdem lässt sich insbesondere auch die Integrität bspw. durch Cache Poisoning angreifen. Gegenmaßnahmen, um Integrität sicherzustellen sind TSIG Records und DNSSEC.

5.4 Denial of Service

DNS als fast überall genutzter Dienst ist ein gutes Ziel für DoS-Angriffe. Viele Internetdienste funktionieren ohne funktionierendes DNS nicht. So können beispielsweise DNS-Server mit Anfragen geflutet werden. Durch die Nutzung von UDP ist der Angriff nur schwer nachverfolgbar. „Gute“ Ziele dafür können insbesondere autoritative Server für TLD-Server, da so direkt die Namensauflösung aller darunterliegenden Domains unmöglich wird.

Um möglichst viel Last bei Anfragen zu erzeugen, können bewusst „Tippfehler“ in die Anfragen eingefügt werden, damit die Resolver immer neue Anfragen stellen müssen. DNS-Server können außerdem für Amplification genutzt werden, da kleine Anfragen große Antworten generieren können. Durch die UDP-basierte Kommunikation ist IP Address Spoofing leicht umzusetzen. Dabei können Recursor sowohl genutzt werden, um andere DNS-Server zu überlasten, aber auch, um Anschlüssel mit Paketen zu fluten.

Eine generelle Absicherung besteht in der Verwendung „sicherer“ DNS-Server, d. h. mit Firewalls, Updates und geeigneter (gepflegter) Server-Software. Gegen Ausfälle und für Lastbalancierung können DNS-Server redundant ausgelegt werden. Hierfür kann *Anycast* eingesetzt werden, wo in unterschiedlichen ASen der gleiche IP-Adressbereich (nämlich der der DNS-Server) announciert wird, sodass an verschiedenen Stellen des Internet die verschiedenen Server benutzt werden. Dafür muss natürlich sichergestellt werden, dass die Server über eine zweite, nicht per Anycast verteile IP-Adresse für Managment-Traffic erreichbar sind und alle Server denselben Datenstand haben. Beispielsweise werden so die DNS Root-Server redundant per Anycast zur Verfügung gestellt.

5.5 Threats to Date Integrity and Authentication

Üblicherweise werden Anfragen über einen Caching Server gestellt. Dieser löst für den Resolver die Anfragen auf und speichert die Ergebnisse zwischen. Die Anfragen stellt der Caching Server gegenüber autoritativen Servern. Diese sind mit Backup-Servern redundant ausgelegt. In dem ganzen System gibt es verschiedene Stellen, um DNS anzugreifen.

So könnte das Zone-File eines autoritativen DNS-Server verändert werden, damit dieser falsche Ausgaben liefert. Bei dynamischen DNS-Einträgen (bspw. DynDNS von DHCP) können unautorisierte Refreshes provoziert werden. Gegenüber den Backup-Servern könnte sich ein Angreifer als Master ausgeben, sodass der Angreifer zumindest den Backup-Servern falsche Daten bereitstellen kann. Alternativ kann der Angreifer direkt versuchen, sich als Master oder Backup-Servern gegenüber dem Caching Server aus-

geben. Schlussendlich kann der Angreifer auch versuchen, den DNS-Verkehr des Caching Servers anzuziehen, um sich selbst als Caching Server auszugeben und falsche Adressen zu liefern.

5.5.1 Data Corruption / Cache Poisoning

Dieser Angriff ist der im Internet am weitesten verbreitete Angriff auf DNS. Ausgenutzt werden Caching-Eigenschaften der DNS-Server. Bei den lokalen DNS-Servern werden alle aufgelösten RRs für eine gewisse Zeit im Cache gehalten. Autoritative DNS-Slave-Server replizieren und cachen ihre Zonendaten vom Master. Anfragen werden üblicherweise als UDP-Pakete gesendet. Für jede Anfrage wird eine Transaktions-ID vergeben, um später die Antwort wieder der richtigen Anfrage zuordnen zu können, da DNS selbst auf UDP basiert und keinen Verbindungsaufbau hat. Ziel des Angriffes ist es, falsche Informationen in die DNS-Caches zu bringen, um Opfer auf einen falschen Server zu leiten (bspw. für Phising-Angriffe).

DNS-Transaktionen werden nur durch IP-Adresse und Port des autoritativen Servers sowie die Transaktions-ID identifiziert. Ist also das Wissen über die genutzte Transaktions-ID bekannt (die Adresse und der Port des autoritativen Servers ist öffentlich im DNS bekannt), kann ein Angreifer durch schnelleres Beantworten von Anfragen falsche Informationen in den DNS-Cache einfügen. Dafür ist jedoch weiter nötig, die Quell-Portnummer des DNS-Caches zu kennen. Diese änderte sich jedoch zumindest in alten Versionen von DNS-Servern meist nicht, solange der Server läuft. Ein Problem war außerdem, dass Server früher die Transaktions-IDs nicht zufällig, sondern sequentielle gewählt haben.

Indem der Angreifer eine Anfrage an den Caching Server stellt, lässt sich einerseits (indem er selbst ein autoritativer Server der Anfrage ist) die Transaktions-ID und andererseits auch der verwendete Quellport ermitteln. Anschließend kann eine Anfrage des Opfers schneller als der autoritative DNS-Server beantwortet werden.

Für diese Art von Angriffen gibt es bereits Gegenmaßnahmen. Der weit verbreitete DNS-Server BIND benutzt seit Version 8 für jedes ausgehende Paket einen Pseudozufallszahlgenerator, um Quellport und Transaktions-ID auszuwählen. Dieser war allerdings noch vorhersehbar, sodass in Version 9 ein schwerer vorhersehbarer PRNG benutzt wurde. Ein weiteres Problem für Angreifer besteht darin, dass Angriffe erst gefahren werden können, wenn Cache-Einträge aus dem Cache gealtert sind. Der Angreifer kann nur dann den Cache vergiften, wenn der entsprechende Eintrag nicht bereits im Cache ist. Durch längere TTLs (z. B. 2 Tage) ist das Zeitfenster, in dem Angriffe gefahren werden können, recht gering (vor allem bei häufig aufgerufenen Seiten). Dadurch sind Angreife zwar schwieriger zu fahren, sind jedoch nicht gänzlich verhinderbar.

Eine weitere Maßnahme des Angreifers könnte ein „Brute Force“-Angriff sein, bei dem der Angreifer den DNS Cache mit vielen Anfragen und Antworten flutet, um mit Glück eine passende Transaktions-ID und Quellportnummer zu erraten und dadurch hoffentlich den Cache mit einer falschen Antwort zu vergiften. Auch hier sinkt durch hohe TTLs die Wahrscheinlichkeit für Erfolg, da nur wenige Anfragen gestellt werden und genau dann zufälligerweise eine der gefälschten Antworten des Angreifers passen muss.

Ein weiterer Trick besteht jetzt darin, nicht den eigentlichen Server im ersten Request

anzufragen, sondern stattdessen einen nicht existierenden Eintrag der Subdomain. Da niemand zuvor diese Anfrage gestellt hat, fragt der Cache Server immer bei autoritativen Server nach. Der Angreifer kann jetzt damit antworten, dass die Anfrage nicht beantwortet werden kann und dabei direkt den (falschen) Nameserver mitliefern, der die Anfrage beantworten kann. Dadurch hat sich der DNS Cache gemerkt, dass der Angreifer-Server der autoritative Server für die Domain ist und stellt in Zukunft alle Anfragen für die Domain gegen den Angreifer-Server.

Die Gegenmaßnahme hierfür nutzt aus, dass DNS Case-Insensitive ist. Wenn jetzt der Caching Server beim autoritativen Server mit Anfragen mit zufällig gewählter Groß- und Kleinschreibung stellt, kann die Antwort auf Plausibilität geprüft werden, indem geprüft wird, ob die Antwort die selbe Groß- und Kleinschreibung hat. So wird ein Bit Entropie pro Buchstaben gewonnen, hilft also nur bei langen DNS-Namen.

Der nächste Angriff nutzt die Fragmentierung von IPv4 aus. Ziel des Angreifers ist, Anfragen zu stellen, die möglichst große Antworten provozieren. Dadurch sollen fragmentierte Pakete provoziert werden. Das erste Fragment soll möglichst uninteressante Daten enthalten, während das zweite Fragment die zu fälschende Antwort enthält. Der Angreifer kann nun das zweite Fragment senden, bevor der autoritative DNS-Server überhaupt das zweite Fragment gesendet hat, um falsche Daten an den Caching Server zu senden. Dafür müssen zuvor ICMP-Pakete für PMTU Discovery gefälscht werden, um die Grenze für die Fragmentierung zu bestimmen. So kann kontrolliert werden, welche Daten im ersten und welche im zweiten Fragment stehen. Da jedoch die Fragmente wieder eine zufällige ID haben, muss der Angreifer sehr viele Anfragen stellen, um hoffentlich mit seiner Antwort die richtige ID zu raten.

Eine Gegenmaßnahme hierfür ist, große Pakete und PMTU-Discovery zu vermeiden. So kann im DNS-Server festgelegt werden, dass große Anfragen nur über TCP-Verbindungen beantwortet werden. Mit ausgeschalteter PMTU Discovery wird verhindert, dass Fragmentierung von außen provoziert werden kann.

5.5.2 Split-Horizon DNS

Um Cache Poisoning von externen Geräten zu verhindern, ist der Gedanke von *Split-Horizon DNS*, die Funktionen des Nameservers unterschiedlich bereitzustellen, je nachdem, von wo Anfragen kommen. So werden im eigenen Netz Anfragen rekursiv aufgelöst, während nach außen hin nur autoritative Anfragen beantwortet werden. Dazu kann der rekursive DNS-Server per Firewall von Anfragen von außen abgeschirmt werden. Der externe Server ist nur autoritativ konfiguriert und akzeptiert immer Anfragen von außen, aber niemals Updates von außen. Zonentransfers sind weiterhin von außen nach innen erlaubt. Für weitere Absicherung kann auch der primäre autoritative DNS-Server versteckt werden, während nur die Backup-Server tatsächlich externe Anfragen beantworten.

5.6 Robustness Towards Data Corruption: Data Integrity

Bisher ist sich zwar darum gekümmert worden, dass Anfragen korrekt beantwortet werden, aber es wurde noch nicht sichergestellt, dass der antwortende Server überhaupt

berechtigt ist, bestimmte Antworten zu geben.

Secret Key Transaction Authentication for DNS (TSIG) nutzt Signaturen, um Daten bei Zonentransfers vom Master zum Slave zu sichern. Zwischen den Entities werden symmetrische Schlüssel verteilt, um mit MD5 Hashes oder HMAC mit SHA-1 oder SHA-2 Signaturen zu erzeugen. Die Signatur wird dabei in einen TSIG-RR eingebettet, der dann vom Slave-Server geprüft wird.

In großen Domänen ist die Authentisierung schwierig, da symmetrische Schlüssel zwischen Servern ausgetauscht werden müssen. Aus Sicherheitsgründen müssten dabei zum einen paarweise zwischen Servern Schlüssel ausgehandelt werden und andererseits sollten diese regelmäßig gewechselt werden, was bei manuellem Schlüsselaustausch quasi nie passiert. Eine weitere Idee ist hier noch der Einsatz von Kerberos oder Sicherstellung, dass Zonentransfers nur innerhalb geschützter Tunnel (z. B. IPsec) durchgeführt werden können.

Ein Hauptproblem an dieser Stelle ist jedoch weiterhin, dass DNS-Anfragen nicht abgesichert sind, sondern nur Zonentransfers.

5.7 DNSSEC

DNSSEC soll das Problem lösen, Ende-zu-Ende Data Origin Authentication und Integrität global sicherzustellen. Dafür soll eine PKI aufgebaut werden, mit der DNS-Antworten der autoritativen Server signiert werden. DNSSEC bietet keine Verbesserung der Verfügbarkeit (eher im Gegenteil, da jetzt mehr Arbeit pro DNS-Anfrage nötig ist), gar keine Vertraulichkeit oder kontrollierten Zugriff. Außerdem ist es weiter möglich, dass ein kompromittierter Server falsche Daten korrekt signiert.

DNSSEC sendet zusätzlich zur eigentlichen Antwort signierte RRSets (Gruppen von RRs). Die Public Keys zur Validierung werden per DNS verteilt. Kindzonen werden über Eltern entsprechend der Zonenhierarchie authentisiert, wodurch eine Chain of Trust etabliert wird. So muss nur der Signaturschlüssel (*Key Signing Key*, KSK) der Root-Zone manuell verteilt werden, um eine vollständige Vertrauenshierarchie (für DNSSEC-fähige Zonen) zu etablieren.

Da es in DNSSEC keine Key Revocation gibt, sollten Signaturen immer nur kurze Gültigkeit haben.

5.7.1 Means of Securing RRSets

DNSSEC führt mehrere neue RR-Typen ein, um RRSets zu authentisieren und Integrität sicherzustellen. Dafür wird Public Key Kryptographie mit Trust Chains genutzt.

RRSig RR für Signaturen übertragener RRs

DNSKEY RR für Public Keys

DS RR für Trust Chains (Trust Anchor signiert den Schlüssel der Kindzonen)

NSEC RR für die nächste sichere Zone in kanonischer Ordnung (Zertifikat, dass es einen Nameserver in der Hierarchie nicht gibt, da der nächste sichere Server angegeben wird, führt zu weiteren Privatsphäreproblemen)

NSEC3 RR für die nächste sichere Zone in kanonischer Ordnung (gehasht)

5.7.2 Authority Delegation and Trust Chaining

Zu Beginn wird ein Trust Anchor benötigt, der festlegt, welcher Public Key für die Root-Zone zuständig ist. Dafür wird ein *Key Signing Key* (KSK) genutzt, mit dem sich die Zone einen *Zone Signing Key* (ZSK) signiert, mit dem sie die Public Keys von Kindzonen signiert. In der Elternzone werden DS-Records für die Kindzonen angelegt, die angeben, welche Schlüssel-ID für welche Kindzone zuständig ist (mit Signatur durch den ZSK der Elternzone in einem RRSIG-RR). In der Kindzone gibt es wieder den von der Elternzone signierten KSK sowie den ZSK, deren Public Keys über DNSKEY-RRs veröffentlicht werden. So lässt sich die Hierarchie beliebig verschachteln und einzig durch manuelle Verteilung des öffentlichen Teils des Root-KSKs eine Chain of Trust etablieren. Kindzonen signieren ihre RRs, indem für jeden Record zusätzlich die Signatur in einem RRSIG-Record hinterlegt wird. Da sich diese Signaturen über die Lebensdauer des ZSKs nicht ändern, bietet es sich an, die Signaturen nicht auf dem DNS-Server zu erzeugen, sondern extern zu erzeugen und in der Zonendatei des DNS-Servers wie andere RRs zu hinterlegen.

5.7.3 DNSSEC Deployment

Die KSKs der Root-Zone sind auf zwei redundanten HSMs gespeichert. Jede HSM kann durch 3 von 7 internationalen Experten (*Crypto Officers*) aktiviert werden. Recovery ist durch 5 von 7 internationalen Experten (*Recovery Key Share Holder*) möglich. Mit diesen HSMs werden die ZSKs für die Root-Zone signiert. Dies steht aktuell unter der Kontrolle von Verisign, einer US-amerikanischen Firma.

In Unterzonen können KSK und ZSK aufgeteilt werden, um verschiedene Sicherheitslevel zu erzielen. Der KSK generiert langlebige Signaturen und wird offline gespeichert (z. B. auf einem HSM). Der ZSK hingegen erzeugt kurzlebige Signaturen auf einem leichter verfügbaren System. Wenn der ZSK jetzt jedoch kompromittiert wird, kann dieser nicht schnell geändert werden, da KSK-Signaturen langlebig sind.

5.7.4 DNSSEC Resource Records

DNSSEC führt einige neue RRs ein:

RRSIG Enthält Signaturen für verschiedene Einträge. Neben der Signatur sind außerdem Informationen über den Algorithmus, TTL, Gültigkeit und auch ID des verwendeten Schlüssels enthalten.

DNSKEY Enthält Public Keys, die bei DNSSEC eingesetzt werden sowie die dazugehörigen Algorithmen und Typen.

DS Enthält den Hashwert des DNSKEYs eines Name Servers einer Sub-Zone, um zusammen mit einem NS-Record den Nameserver der Sub-Zone zu authentifizieren und damit die Trust Chain zu bilden.

NSEC Gibt Informationen über die nächste Subdomain in kanonischer Ordnung, die DNSSEC-fähig ist. Damit lässt sich die Information, dass ein nachgelagerter DNS-Server nicht DNSSEC-fähig ist, authentifizieren. Da jedoch die Informatino über die nächste DNSSEC-fähige Domain explizit im Record enthalten ist, besteht hier die Möglichkeit, durch Auslesen der NSEC-Records herauszufinden, welche Sub-Zonen existieren. Dies kann je nach Ansicht als Verletzung der Privatsphäre angesehen werden (da DNS ja eigentlich für die Auflösung *bekannter* Namen und nicht für das Finden unbekannter Namen gedacht ist und das Wissen über Zonen bspw. mit whois auch das Auslesen weiterer Informationen ermöglicht). Dieser Record-Typ wird durch NSEC3 ersetzt.

NSEC3 Statt den Rechnernamen des nächsten DNSSEC-fähigen Nameservers bereitzustellen (wie es bei NSEC der Fall ist), wird hier nur der Salted Hashwert des nächsten DNSSEC-fähigen Servers angegeben. So kann man aus NSEC3-Records nicht mehr auslesen, welcher DNS-Server gemeint ist, aber (unter Kenntnis des Namens) dennoch prüfen, ob der Server DNSSEC-fähig sein sollte oder nicht. Der Salt wird hierfür im NSEC3-Record neben dem Hashalgorithmus und anderen Parametern für das Hashing auch bereitgestellt.

Da DNS-Namen meist möglichst kurz und leicht merkbar gewählt werden, hilft Hashing nur bedingt, da ein Angreifer weiterhin bspw. mit Wörterbüchern ausprobieren könnte, welcher DNS-Name auf den Hash passt. Die einzige Möglichkeit, dies etwas zu mitigieren, ist die dynamische Erzeugung von Signaturen on demand, die sich jedes Mal ändert. Hier besteht jedoch wieder ein Risio für DoS und erfordert entsprechende Gegenmaßnahmen wie Rate Limiting.

5.7.5 DNSSEC Issues

Vorteil von DNSSEC ist es, sich nicht autorisierte DNS-Records abzusichern.

Nachteil ist klar die gestiegene Komplexität. Es ist leicht möglich, hier Fehler zu machen. Außerdem werden durch die vielen Records für DNSSEC auch die allgemeine Last auf dem DNS-Server und damit auch das DoS-Potential größer. Zonen müssen außerdem vollständig signiert sein, um überhaupt einen Sicherheitsgewinn zu erhalten. Die Anfragen und Antworten sind weiterhin nicht vertraulich, da nicht verschlüsselt. Mit NSEC und NSEC3 erhält man zwar Authenticated Denial, aber auch das Risiko, dass Angreifer alle Subzonen herausfinden können.

Ein Kernproblem beim Vertrauen ist außerdem, wer die Anchor Keys für die Domain . hält. Diese werden aktuell von einer US-amerikanischen Firma gehalten.

5.7.6 Alternatives to DNSSEC

Verschiedene Alternativen zu DNSSEC ausprobiert. Dabei wurden verschiedene Ziele erreicht. Insgesamt ist es möglich, nur zwei der drei Ziele (Zookos triangle) zu erreichen: Sicherheit, verteiltes System, menschlich lesbare Namen.

DNSCurve versucht, online kryptographische Funktionen zu benutzen, um DNS-Anfragen verschlüsselt zu stellen und zu beantworten. Dafür werden Anfragen mit symmetrischen Schlüsseln verschlüsselt. Für die ursprüngliche Anfrage wird weiter ein Public Key Verfahren eingesetzt. Diese werden nicht über DNS-Records verteilt, sondern im Namen des DNS-Servers enkodiert. So ist der Name des DNS-Servers jetzt der Public Key des Servers (plus einer magischen Zeichenfolge, um den Server als DNSCurve-fähig zu markieren). So wird keine Hierarchie benötigt, die sicherstellt, dass der DNS-Server auch zuständig ist. Durch Einbettung in Hyperlinks, etc. kann DNSCurve zur Verifikation von DNSCurve-Servern genutzt werden, ohne der Root-Zone vertrauen zu müssen. Dafür sind die menschen-merkbaren Namen verloren gegangen. Ein Problem weiter, dass die kryptographischen Funktionen hart festgelegt sind.

Peer Name Resolution Protocol (PNRP) versucht, ein verteiltes Protokoll zur Namensauflösung anzubieten. Die Computer sind in einem Overlay-Netz strukturiert. Lesbare Namen werden in einer bestimmten Subdomäne bereitgestellt und können auf unsicherem Weg aufgelöst werden. Für andere Namen können SHA1-Hashes der Public Keys im Namen kodiert werden, um DNS-Anfragen sicher aufzulösen. Da PNRP ein P2P-Netz bildet, weist es hohe Ausfallsicherheit und Skalierbarkeit auf. Verschiedene Teilnetze machen es sehr performant und robust innerhalb eines Netzwerks. Durch eine Vertrauenshierarchie können sichere Namen weiter überprüft werden. Allerdings sind die Namen wieder nicht gut menschen-merkbar. Im unsicheren Namespace gibt es außerdem keine Autorität, die sicherstellt, wer welche Namen registrieren darf. Damit kann jeder Namen von jedem jederzeit beansprucht werden.

GNU Name System (GNS) ist auch ein verteilter Ansatz basierend auf GNUnet und bietet wieder sichere Namen aus Hash und Key an. Damit sind hier die Namen auch wieder nicht merkbar. Dafür lassen sich Aliase bilden.

6 Internet Firewalls

6.1 Introduction

Firewalls sind inspiriert von in Bauarbeiten benutzen Brandmauern, die Schaden im Brandfall eingrenzen sollen. Die Firewalls im Internet entsprechen eher Mauern und Zugbrücken von Burganlagen, die nur für bestimmte Sachen an kontrollierten Punkten durchlässig sind. Damit soll es möglich gemacht werden, dass Angreifer nur an bestimmten, besonders gut kontrollierten Eintrittspunkten in das Netz einbrechen kann. Sämtlicher Traffic, der das Netz wieder verlassen soll, muss auch an diesen kontrollierten Punkten vorbei. Außerdem kann der Einsatz von Firewall das das Erreichen anderer Verteidungsmechanismen erschweren. Firewalls ermöglichen damit Sicherheit auf Subnetzebene.

Firewalls stehen im Fokus von Sicherheitsentscheidungen. Hier können bestimmte Sicherheits-Policies durchgesetzt werden, bspw. um den Zugriff auf bestimmte Netzbereiche einzuschränken. Außerdem kann Internetaktivität an der Firewall effektiv geloggt werden, da sämtlicher Traffic an einer Firewall vorbeikommt, bevor er ins Internet geht. Da sämtlicher Traffic durch die Firewall geht, wird die Angriffsfläche eines Teils des Netzwerks verkleinert, bspw. indem (potentiell unsicher konfigurierte) Client-Geräte nicht mehr aus dem Internet erreichbar sind.

Firewalls können jedoch nicht gegen böswillige Geräte im eigenen Netz schützen, da sie nur Traffic filtern können, der durch sie geleitet wird. So sind Geräte immer noch angreifbar, wenn bspw. ein bösartiger WLAN-AP ohne Authentifizierung an einen internen Port des Netzes angeschlossen wird. Gegen neuartige Angriffe kann eine Firewall nicht schützen. Gegen Viren hilft eine Firewall auch nicht. Außerdem müssen Firewalls von einem Administrator korrekt konfiguiert werden.

6.2 Fundamental Approaches Regarding Firewall Policy

Eine grundlegende Strategie ist, standardmäßig alles zu verbieten. Dabei wird alles, was nicht explizit erlaubt ist, verboten. Dafür muss bekannt sein, welche Dienste die Nutzer des geschützten Netzbereiches benötigen. Für jeden dieser Dienste muss geprüft werden, welche Implikationen er auf die Sicherheit hat und wie er sicher bereitgestellt werden kann.

Die gegenteilige Strategie ist, standardmäßig alles zu erlauben, was nicht explizit verboten ist. Dafür muss entschieden werden, welche Dienste als potentiell gefährlich einzustufen sind. Beispiele dafür sind SMB oder X-Windows. Diese Strategie erfordert, dass man immer einen Überblick hat, welche Dienste im Netzwerk verfügbar sind und

wie gefährlich diese sind. Wird etwas übersehen, kann ein potentiell gefährlicher Dienst im Netz angeboten werden. Daher ist diese Strategie für die meisten Netzwerke nicht sinnvoll.

6.3 Protocol Fields Important for Firewalls

Internetdienste werden üblicherweise über Client-Server-Verbindungen realisiert. Anwendungsprotokolle laufen auf solchen Verbindungen. Die PDUs der Anwendungen werden meist in TCP-Verbindungen oder UDP-Paketen eingebettet. Diese wiederum sind in IP-Pakete eingepackt, die mit Ethernet übertragen werden. Verbindungen lassen sich daher meist mittels Quell- und Ziel-IP-Adresse sowie Quell- und Ziel-Port und Layer-4-Protokolltyp identifizieren. Nicht alle diese Informationen stehen zur Verfügung, wenn Verschlüsselung bspw. mittels IPsec eingesetzt wird. Daher ist es eine Policy-Frage, ob Ende-zu-Ende-Verschlüsselung in sicheren Netzen überhaupt eingesetzt werden soll. Eine übliche Architektur terminiert hingegen verschlüsselte Verbindungen bei der Firewall, um hier die Policy des Traffics durchzusetzen. Andere Strategien könnten erlauben, nur bestimmte (vertrauenswürdige) VPN-Verbindungen zu erlauben.

IP-Pakete haben zusätliche Flags und Optionen, die teilweise Routing-Entscheidungen beeinflussen (bspw. Source Routing). Diese werden teilweise abgelehnt bzw. gefiltert, da sie heutzutage kaum Anwendung finden und eher Probleme verursachen.

Bei TCP-Verbindungen lässt sich anhand der Control-Flags erkennen, wann und in welche Richtung Verbindungen aufgebaut werden. Mit RST können Verbindungen abgebaut werden, ohne weitere sinnvolle Fehlermeldungen zu erzeugen, bspw. wenn der Traffic einer Verbindung irgendwann als gefährlich eingestuft wird.

Die Erkennung und Analyse von Anwendungsprotokollen ist schwierig, da die Protokolle stark anwendungsabhängig sind.

6.4 Firewall Terminology and Building Blocks

6.1. Eine **Firewall** ist eine (Menge von) Komponente(n), die die Zugriff auf ein geschütztes Netz von anderen Netzbereichen oder dem Internet einschränkt.

6.2. Ein **Paketfilter** betrachtet einzelne Pakete und entscheidet darüber, ob das Paket akzeptiert oder abgelehnt bzw. andere Regeln ausgewertet werden sollen.

6.3. Der **Bastion Host** ist ein besonders sicher konfigurierter Host, auf den von „außen“ zugegriffen werden kann, um den gesicherten Zugriff auf interne Netzbereiche zu ermöglichen.

6.4. Ein **dual-homed host** ist ein Computer mit wenigstens zwei Netzwerkinterfaces.

6.5. Ein **Proxy** verbindet sich stellvertretend für interne Clients mit externen Servern, um (zulässige) Anfragen von Clients zu echten Servern zu stellen und die Antworten weiterzuleiten.
Versteht und interpretiert der Proxy Befehle eines Anwendungsprotokolls, wird er auch als **Application Level Proxy** bereichnet. Leitet er PDUs nur weiter, ist er ein **Circuit Level Proxy**.

6.6. Network Address Translation ist ein Mechanismus, bei dem Daten in Paketen abgeändert werden, um Netzwerkadressen zu modifizieren. Damit werden interne IP-Adressen verschleiert und es ist möglich, mehrere Hosts über eine IP-Adresse mit dem Internet zu verbinden.

6.7. Das **Perimeter Network** ist ein Subnetz zwischen dem externen und internen Netz, um eine zusätzliche Sicherheitsebene einzuführen. Diese ist auch bekannt als **DMZ** (De-militarized Zone).

6.5 Firewall Architectures

Die einfachste Architektur setzt einen Paketfilter auf einem Router ein, der den Traffic zum Internet leitet. Dies kann sowohl mit einer einfachen Workstation (z. B. Linux PC) mit zwei NICs oder auch mit einem dedizierten Hardware-Router realisiert werden, sofern dieser Paketfilter-Techniken zur Verfügung stellt.

Reicht ein einfacher Paketfilter nicht aus, kann ein dual-homed Bastion Host eingesetzt werden, der Proxys für interne/externe Clients bereitstellt. Auf diesem können auch wieder Filterregeln eingesetzt werden. Nachteil hierbei ist wiederum, dass sämtlicher zulässiger Traffic durch den Bastion Host geleitet werden muss, was wieder ein Bottleneck darstellen kann.

Eine weitere Idee versucht, den Leistungsengpass zu umgehen. Die *Screened Host* Architektur ordnet zwar den Bastion Host logisch der Firewall zu, aber erlaubt es, den Bastion Host irgendwo im Netz zu platzieren. Erlauber IP-Traffic wird über den Screened Host geleitet. Direkter Verkehr von internen Hosts zum Internet wird hingegen abgelehnt. Der Screend Host stellt wieder Proxys bereit. Da jetzt jedoch der Bastion Host hinter der Firewall ist, kann ein kompromittierter Bastion Host viel Schaden anrichten.

Die nächste Architektur benutzt zwei Firewalls, um ein Perimeter-Netz einzurichten. So wird einerseits Traffic zwischen Internet und Perimeter-Netz sowie Traffic zwischen

Perimeter-Netz und internem Netz gefiltert, damit ein kompromittierter Bastion Host nur begrenzt viel Schaden anrichten kann. Im Perimeter-Netz steht wieder ein Proxy zur Verfügung.

Dieser Ansatz kann wieder um einen dual-homed Bastion Host erweitert werden, der zwischen den beiden Firewalls eingesetzt wird. Hier steht wieder das Bottleneck Bastion Host als Problem im Raum. Diese Struktur lässt sich jedoch teilweise horizontal skalieren und wird daher heute auch häufig eingesetzt.

6.6 Packet Filtering

Was aber lässt sich mit Paketfiltern überhaupt realisieren? In der Theorie ist es zwar möglich, alles zu kontrollieren. In der Praxis werden Operationen, die tiefergehendes Wissen über Protokolle höherer Schichten erfordern, nur von Proxys umgesetzt. Paketfilter setzen normalerweise nur einfache Operationen um, die sich schnell und hardwarebeschleunigt prüfen lassen. Einfache Paketfilter benutzen dafür Ziel- und Quelladressen und -ports, Transportprotokolltyp und ggf. noch Flags des Transportprotokolls (z. B. TCP Flags ACK, SYN, RST). Manche Paketfilter können auch für gängige Protokolle wie DNS oder HTTP tiefere Untersuchungen durchführen, bspw. um eingehende UDP-Pakete immer zu erlauben, wenn sie wie DNS-Pakete aussehen.

Manche Paketfilter können auch Verbindungen tracken, was dann als **dynamischer Paketfilter** bezeichnet wird. Das ist i. d. R. die einzige Möglichkeit, UDP-Antwortpakete durch die Firewall zu lassen, ohne UDP generell zu erlauben. Dynamische Paketfilter erfordern, dass nicht nur eingehender, sondern auch ausgehender Traffic angesehen werden. Dabei muss gespeichert werden, welche ausgehenden Verbindungen in der letzten Zeit gesehen wurden.

Paketfilter können verschiedene Aktionen für jedes Paket ausführen. Üblicherweise werden Pakete immer durchgelassen oder verworfen. Zusätzlich kann entschieden werden, Pakete zu loggen oder Fehlermeldungen (z. B. TCP RST) an den Absender des verworfenen Pakets zu senden.

Für die Spezifikation von Regeln werden üblicherweise zwei Richtungen betrachtet:

- Pakete, die vom internen Netz Richtung Internet gehen, sind *inbound*.
- Pakete, die vom Internet zum internen Netz gehen, sind *outbound*.

Für Quell- und Zieladresse können üblicherweise Wildcards spezifiziert werden, bspw. indem statt IP-Adressen ganze Adressbereiche gewählt werden. Für Quell- und Zielports können ähnliche Regeln spezifiziert werden, bspw. dass nur Ports oberhalb eines bestimmten Werts auf die Regel zutreffen. Bei Regeln, die für Antwortpakete von bestehenden TCP-Verbindungen vorgesehen sind, sollte immer sichergestellt werden, dass das ACK-Bit gesetzt sein muss. Ansonsten können die Regeln es ermöglichen, dass Angreifer Verbindungen von außen zu Diensten aufbauen können, die nicht öffentlich erreichbar sein sollen, wenn Angreifer die Quellports passend ändern, sodass eine solche Regel zutreffen würde.

Der Einfachheit halber werden Regeln in Paketfiltern normalerweise immer von oben nach unten abgearbeitet. Die erste Regel, die zutrifft, wird dabei angewendet. Das hat auch Sicherheitsimplikationen, da bei einer schwer nachvollziehbaren Regelauswertung Administratoren möglicherweise nicht mehr verstehen, wie die Regeln funktionieren und welche Sicherheitspolitik diese durchsetzen.

6.7 Bastion Hosts

Der Bastion Host ist dem Internetverkehr stärker ausgesetzt als andere interne Hosts. Diese Art von Hosts bietet üblicherweise Proxy-Dienste, Paketfilter oder auch von außen erreichbare Dienste an. Als Faustregel für solche Systeme gilt immer das KISS-Prinzip, also dass das System so wenig wie möglich kann, um die nötigen Funktionalitäten umzusetzen. Weitere Sicherungsmaßnahmen liegen darin, dass diesem Host nicht mehr als notwendig vertraut wird. Idealerweise sollte der Bastion Host nicht einmal direkten Zugang zum internen Netzwerk haben, um Sniffing u. Ä. zu verhindern.

Um laufende Angriffe erkennen zu können, sollten ungewöhnliche Events geloggt werden. Diese Logs müssen ordentlich abgesichert werden, damit diese nicht gelöscht werden können. Dafür bietet sich eine Art von Remote Logging über Schnittstellen an, die das Löschen vom loggenden System selbst unmöglich machen (bspw. Logging auf eine serielle Konsole, deren Gegenseite nicht im Internet hängt).

Weiter sollte ein Ziel sein, dass der Bastion Host möglichst unattraktiv für Angreifer ist, indem der Host möglichst wenige Funktionen hat. So kann man versuchen, dass der Bastion Host nicht in der Lage ist, Dateien herunterzulasen oder Verbindungen ins interne Netz aufzubauen, damit ein infiziertes System möglichst wenige Möglichkeiten hat. Ansonsten gelten für Bastion Hosts ähnliche Sicherheitsmaßnahmen wie für andere Serversysteme, bspw. regelmäßige Updates, Sperrung unnötiger Benutzeraccounts, lange und verschiedene Passwörter (oder besser nur Public Key Authentifikation), regelmäßige Backupts, Monitoring, etc.

6.7.1 Proxy Services

Eine wichtige Art von Bastion Hosts sind Proxy-Server, die Anfragen von internen Hosts stellvertretend gegenüber externen Diensten stellen. Beispiel dafür wären FTP, HTTP, SSH, DNS und SMTP. Diese Server sind meist als Application Level Proxy ausgelegt, d. h. sie verstehen die Anwendungsprotokolle und erlauben es auch, tiefergehende Inspektion der Daten (bspw. für Virenscans, Caching) durchzuführen. Circuit Level Proxys sind weniger wichtig, da sie nur PDUs unverändert weitergeben können. Diese finden immer weniger Verbreitung und werden meist nur benutzt, wenn kein passender Application Level Proxy existiert.

Für die Verbindung zum Proxy gibt es verschiedene Ansätze. So kann sich ein Nutzer manuell in das Proxy-System einloggen, um die Anwendung zu benutzen. Eine andere Variante ist, den Proxy-Server als solchen in der Client-Software zu konfigurieren. Dies ist bspw. bei HTTP-Proxys für Webbrowser möglich. Komfortabler ist es, dass das Betriebssystem automatisch den Proxy für alle Anwendungen benutzt und möglicherweise

sogar per Autodiscovery (z. B. WPAD) die Proxy-Information vom Netzwerk erhält. Zuletzt kann der Router Proxy-aware gemacht werden, also den Traffic routerseitig zum Proxy umleiten. Dies wird auch als **Transparent Proxy** bezeichnet.

6.7.2 Aspects of modern Firewall Systems

Eine weitere Variante von Firewalls ist das **One-Way Gateway**, was Traffic nur in eine Richtung zulässt. Das ermöglicht es, dass ein Netzbereich gar nicht von außen über das Netzwerk angegriffen werden kann, da ein Angreifer keine Möglichkeit hat, mit den internen System zu kommunizieren. Realisiert werden kann dies bspw. in dem die Glasfaser nur für eine Richtung gesteckt wird. Eingesetzt wird dies in manchen Hochsicherheitsanwendungen, bspw. in Kernkraftwerken, und erfordert spezielles Protokolldesign, da die Gegenseite nicht signalisieren kann, um die Nachrichten erfolgreich empfangen werden konnten.

Ein weiterer Ansatz ist das **Remote-Controlled Browsers System**, bei dem Anwendungen nicht auf Endgeräten, sondern auf einem Terminal-Server laufen. Diese sich stark überwacht und besser abgesichert als Endgeräte. Die Endgeräte selbst hingegen haben keinen direkten Zugriff zum Internet, sondern nur zum Terminal-Server über eine Videoverbindung.

Moderne Firewalls setzen teilweise auch auf **Deep Packet Inspection**, also die genauere Analyse des Protokollverhaltens bis in die Anwendungsebene hinein. Dazu kommen auch *SSL Inspection* und *SSH Inspection*. Teilweise wird dies durch Verschlüsselung erschwert und erfordert Workarounds wie automatisch MITM-Angriffe der Firewall, die dann Zertifikate einer lokal vertrauenswürdigen CA akzeptieren. Diese Workarounds können teilweise durch Certificate Pinning unbrauchbar gemacht werden, da so der Client nur noch das einmal erhaltene Zertifikat akzeptiert, wodurch die interne CA für die entsprechende Domain nicht mehr akzeptiert wird.

Network Access Control (NAC) und **Unified Threat Management** integriert verschiedene Werkzeuge wie Antvirus, Firewalls, etc. miteinander.

7 Intrusion Detection Systems

7.1 Introduction

> **7.1.** Eine **Intrusion**/Eindringen ist eine Aktion bzw. Folge von Aktionen, die Vertraulichkeit, Integrität oder Verfügbarkeit eines Dienstes oder Systems kompromittieren soll.

Für die Verteidigung vor Eindringlingen gibt es mehrere Kategorien:

- Präventive Verteidigung strebt an, dass ein Angreifer gar nicht erst versucht, in das System einzudringen.
- Erkennung zielt darauf ab, einen Eindringling zu erkennen.
- Wurde ein Eindringling erkannt, zielt die Response darauf ab, den Eindringling zu entfernen, zu identifizieren, etc.

Die meisten Angriffe werden erst innerhalb von Monaten erkannt. Ein signifikanter Teil wird auch erst nach Jahren erkannt. Nur etwa ein Viertel wird innerhalb von Sekunden bis Tagen erkannt! Im Laufe der Jahre werden die erfolgreichen Angriffe auch ausgeklügelter. Durch die Verbereitung von automatisierten Angriffstools wird es zudem immer einfacher, komplexe Angriffe sehr leicht auszuführen. Zudem werden immer mehr (staatliche) Akteure aktiv, die über viele Jahre einen Angriff vorbereiten und durchführen (bspw. der xz-Utils Angriff [3]).

Das generelle Ziel ist also, die Computersysteme und Kommunikationsinfrastruktur zu überwachen und Angriffe und falsche Benutzung zu erkennen. Das hat einfach den Grund, dass ein vollumfänglicher Schutz nicht möglich ist. Starke Sicherheitsmaßnahmen sind zu teuer oder haben zu geringe Flexibilität, z. B.weil nicht jede Funktionalität in ASICs gebaut werden kann. Im einfachsten Fall können schon aus Compliance-Gründen bestimmte Systeme (z. B. im Gesundheitswesen) nicht „mal eben“ aktualisiert werden. Zudem kann es sein, dass zu viele präventive Sicherheitsmaßnahmen Nutzer stören, was diese wiederum dafür motiviert, die Sicherheitsmaßnahmen zu umgehen (was wiederum neue Sicherheitsprobleme einführt).

Intrusion Detection hat nicht nur den Vorteil, dass Angriffe erkannt werden können. Auch der Missbrauch der Computersysteme durch die eigenen (legitimen) Nutzer kann so erkannt werden.

7.2 Intrusion Detection Systems

Ein IDS ergibt nur sinn, wenn es auch Konsequenzen hat. So muss nach der Erkennung eines Angriffs darauf reagiert und der Schaden behoben werden. Anschließend müssen neue Sicherungsmaßnahmen umgesetzt werden, damit der Angriff nicht erneut glückt.

Ereignisse werden an einem System (z. B. Terminal, Router, etc.) überwacht und an ein zentrales IDS/SIEM geloggt. Dort läuft eine Erkennung, die dann möglicherweise eine Reaktion auslöst (bspw. Warnung ausgeben, Alarmierung) oder eine automatische Reaktion an das überwachende System weitergibt.

7.3 Tasks of an Intrusion Detection System

IDS bestehen aus drei Kernaufgaben:

Audit zeichnet alle sicherheitsrelevanten Events auf einem überwachten System auf. Hier werden aufgezeichnete Daten bereits vorverarbeitet und verwaltet.

Erkennung analysiert automatisch die Audit-Daten. Hier gibt es verschiedene Ansätze, bspw. Signaturanalyse, Erkennung von abnormalen Verhalten (basierend auf Wissen) oder Erkennung von Anomalien (basierend auf einem gelernten „normalen“ Level). Bei automatisch lernenden Verfahren muss insbesondere darauf geachtet werden, dass kein Angriff läuft, während das System sich in der Lernphase befindet, da sonst das Angriffsverhalten als Normalverhalten gelernt wird. Hier können sowohl falsch positive als auch falsch negative Ergebnisse auftreten.

Response reagiert auf erkannte Angriffe. Diese werden mindestens gemeldet, können aber möglicherweise auch automatisierte Sicherungsmaßnahmen einleiten. Dies wiederum kann aber auch gezielt von Angreifern ausgenutzt werden, wenn bspw. das Herunterfahren von Systemen durch das IDS genau das Ziel des Angriffs war.

7.4 Requirements of Intrusion Detection Systems

Neben einer guten Erkennungsgenauigkeit sollte ein IDS gut in das bestehende System/Netzwerk integriert werden können. Es sollte einfach konfigurier- und wartbar sein und autonom, ausfalltolerant und mit wenigen Ressourcen laufen Außerdem sollte gerade das IDS gut abgesichert sein, damit Angreifer nicht das IDS angreifen können, um den nachfolgenden (eigentlichen) Angriff unerkannt zu lassen.

7.5 Classification of IDS

Eine Einteilung kann einerseits auf den Umfang durchgeführt werdne. Host-basierte IDS analysieren System-Events, bspw. von einem Logging-Daemon übermittelt. Netzwerkbasierte IDS analysieren die im Netz ausgetauschten Informationen (IP-Pakete). Hybride Ansätze hingegen kombinieren beide Arten von Informationen.

Andererseits ist auch eine Einteilung basierend auf Zeit möglich. Läuft das IDS im Regelbetrieb, betreibt es Online-Analyse. Demgegenüber steht die Post-Mortem-Analyse, bspw. durch Forensiker durchgeführt, die möglicherweise deutlich langsamer und gründlicher ist und daher online nicht ausgeführt werden kann.

7.6 Host Intrusion Detection Systems

HIDS arbeiten mit Informationen, die auf dem System verfügbar sind. Dazu gehören bspw. Betriebssystem- und Applikations-Logs, Veränderungen am Dateisystem, unzulässige Dateizugriffe (oder auch Erkennung ungewöhnlich vieler Dateizugriffe), das Login-Verhalten (bspw. fehlgeschlagene Loginversuche) oder auch Ressourcenverbrauch. Bei der Analyse vieler verdächtiger Verhaltensweisen (bspw. viele Dateizugriffe, Spitzen im Netzwerkverkehr) kann es auch zu falsch positiven Meldungen (z. B. bei Systemupdates) kommen. Ein HIDS muss dies bei der Analyse berücksichtigen.

HIDS haben den großen Vorteil, dass sie auch Angriffe innerhalb vom Netz erkennen können. Allerdings muss das HIDS auch auf jedem System laufen. Das bedeutet auch, dass eine große Anzahl System verwaltet werden muss. Außerdem sind HIDS nicht notwendigerweise für alle Plattformen verfügbar. Ein Problem dabei ist, dass ein erfolgreich eingedrungener Angreifer das HIDS auch deaktivieren könnte. Weitere Probleme sind, dass viele (möglicherweise nutzlose) Informationen erzeugt werden und dadurch auch eine Online-Analyse nur schwer möglich machen.

7.7 Network Intrusion Detection Systems

NIDS analysieren Monitoring-Informationen aus dem Netzwerk. Dabei wird meist im Network Layer begonnen, da insbesondere bei Angriffen aus dem Internet der Data Link Layer vom eigenen Router erzeugt wird und damit selten nützliche Informationen bietet. Bestehende Systeme sind damit in der Lage, signaturbasiert Angriffe zu erkennen, Abweichungen vom definierten Protokollverhalten zu sehen oder auch statistische Anomalien zu finden.

Durch NIDS können verschiedene Angriffe wie DoS mit Buffer Overflow Angriffen erkannt werden. Gerade dies ist nützlich, da ein Buffer Overflow gerne mal einen Systemabsturz erzeugen kann, was möglicherweise gesammelte Daten eines HIDS unbrauchbar machen kann. Weiterhin können natürlich auch typische über das Netzwerk durchgeführte Angriffe wie ungültige Pakete auf Anwendungsebene, DDoS, Spoofing oder auch Port Scans erkannt werden. Übliches Einsatzfeld von NIDS ist bei Netzwerkhubs, da so ganze Segmente des Netzwerks überwacht werden können.

Probleme mit NIDS sind vor allem, dass sie keine Offline-Angriffe (z. B. Kopie auf USB-Stick) erkennen können und durch Verschlüsselung die Analyse höherer Protokollschichten stark erschwert wird.

NIDs können an verschiedenen Stellen vom Netzwerk installiert werden. So erlaubt eine Installation im LAN, Angriffe innerhalb des Netzes zu erkennen. Außerdem ist so die Last des IDS nicht so hoch. Platziert man das IDS hingegen noch vor die erste Firewall, hat das

IDS mit viel Load zu rechnen. Durch die vielen Pakete können auch viele falsche Alarme erzeugt werden. Allerdings kann man hier wirklich alle Angriffsversuche von außen sehen, da keine von der Firewall herausgefiltert wurden. Eine weitere Platzierungsmöglichkeit ist in der DMZ, wo bereits vorgefilterter Traffic analysiert werden kann, der aber noch von außen mit Servern reden kann. So können zwar nur Angriffe gegen die DMZ erkannt werden, aber das IDS kann neben Angriffen von außen auch kompromittierte LAN-Geräte erkennen.

IDS werden häufig auch an mehreren Stellen installiert und können ihre Informationen an eine zentrale Stelle (z. B. *SIEM*) gesendet werden. Zum Austausch von Nachrichten wird dabei das *Intrusion Detection Message Exchange Format* (IDMEF, RFC 4765) benutzt. Hier stehen u. A. Heartbeats und Alert-Nachrichten zur Verfügung. Die Events enthalten Informationen über das analysierende IDS, die Klassifikation des Events, die Quelle, dass Ziel sowie eine Einschätzung über die Schwere des Alarms.

7.8 Signature-based Detection

Ziel der *Signaturerkennung* ist die Erkennung von Angriffsmustern, die bestimmte Angriffe erzeugen. Die Entwicklung solcher *Angriffssignaturen* ist aufwändig, da bekannte Angriffe analysiert und daraus Signaturen entwickelt werden müssen. Dies ist von einzelnen Organisationen für den Eigengebrauch kaum zu stemmen. Stattdessen gibt es Dienstleister oder Open-Source-Bibliotheken, die solche Signaturdatenbanken bereitstellen.

Um die Performance solcher Erkennungen hoch zu halten, werden Daten in verschiedenen Abstufungen verschiedener Granularität verarbeitet. So können langsame Analysen wie die Reassemblierung von Fragmenten und Segmenten oder die Suche mit regulären Ausdrücken erst durchgeführt werden, nachdem anderen Kritieren wie ein erfolgreicher Verbindungsaufbau zutreffen. Die Analyse läuft also normalerweise in mehreren Schritten, die sich Schrittweise im Protokollstack nach oben arbeiten und jeweils Regeln und Abbruchkritieren prüfen. In jedem Schritt können Aktionen definiert werden, um Pakete zu verwerfen, zu loggen oder einen Alarm auszulösen.

Signaturbasierte Erkennung ist durch vordefinierte Datenbanken leicht einzurichten und erzielt in manchen Umgebungen akzeptable falsch-positiv-Raten. Allerdings müssen auch alle potentiellen Angriffe im Vorfeld bekannt und vor allem auch analysiert und in der Datenbank eingetragen sein. Daher muss die Signaturdatenbank auch durchgehend aktualisiert werden. Da die Angriffe nicht weniger werden, wachsen die Datenbanken ständig und sind daher auch schwer wartbar. Außerdem können auch nicht alle Angriffe über solche Signaturen erkannt werden, sondern benötigen spezielle Plugins, die zusätzliche Features in der Regelsprache bereitstellen, bspw. um Port Scans erkennen zu können. Durch das Preprocessing von IP und TCP werden außerdem viele Ressourcen verbraucht. Zudem hilft das IDS auch nur, wenn der Angreifer es nicht schafft, seine Signature so zu ändern, dass das IDS den Angriff nicht erkennt. Da der Angreifer theoretisch das IDS auch im eigenen Labor betreiben kann, kann er offline testen, ob er das IDS umgehen kann. Gerade bei Open-Source-Signaturdatenbanken kann leicht gesehen

werden, welche Signaturen erkannt werden können.

Ein IDS erlaubt letztlich eine sehr präzise Erkennung von Angriffen. Allerdings muss das IDS auch dafür sehr präzises Wissen über mögliche Angriffe haben.

7.9 Detection of Abnormal Behavior

Die Grundidee ist hier, Verhalten zu erkennen, welches signifikant von normalem Verhalten abweicht. Dazu wird ausgenutzt, dass Nutzer und Systeme ein gewisses „normales“ Verhalten (Aktivitätsmuster, benutzte Protokolle und Protokollzustände, kontaktierte Server, Verkehrsaufkommen, etc.) haben.

Angenommen wird jetzt, dass ein Administrator in der Lage ist, „normales“ Verhalten zu beschreiben. Dafür ist eine Spezifikation bspw. durch eine Regelsprache notwendig. Für generische Protokolle kann möglicherweise auch durch Hersteller eine Beschreibung vordefiniert werden. Das IDS kann dann einfach das Netzwerkverhalten analysieren und die Regeln prüfen. Sämtliche Abweichungen werden als Anomalien gemeldet.

Neben üblichen IDS, die Flow-Informationen, Protokollanalyse etc. passiv durchführen, können auch sog. *Honey Pots* aufgestellt werden. Diese Systeme habe keinen Zweck für normale Nutzer (werden also von Nutzern auch normalerweise nicht kontaktiert), sondern dienen dazu, Angreifer „anzulocken“, d. h. ihn zu provozieren, das System anzugreifen. Jeder Zugriff auf solche Honey Pots kann dann als illegitim gewertet werden, da klar ist, dass normale Nutzer das System überhaupt nicht benutzen. Um den Angreifer etwas länger zu beschäftigen und möglicherweise eine genauere Analyse zu ermöglichen, können diese Systeme dem Angreifer auch vortäuschen, dass sie sich tatsächlich angreifen lassen (bspw. indem sie Dateizugriffe scheinbar gewähren). So soll der Angreifer möglichst lange auf dem System unterwegs sein.

Hauptvorteil von Anomalieerkennung ist, dass auch unbekannte Angriffe erkannt werden können. Da man keine bestimmten Muster prüft, ist es auch schwer für Angreifer, sich gegen die Erkennung vorzubereiten. Wenn das IDS gut aufgesetzt ist, können damit auch akzeptable falsch-positiv-Raten erzielt werden. Die erkannten Ereignisse sind dabei auch recht leicht interpretierbar.

Nachteile solcher IDS sind hoher administrativer Aufwand. Außerdem ist es gerade bei sehr heterogenen Netzen (z. B. öffentliches WLAN) schwierig, normales Netzwerkverhalten zu definieren.

7.10 Automatic Anomaly Detection

7.10.1 Übersicht

Statt manuell normales Verhalten zu definieren, ist der Ansatz hier, normales Verhalten automatisch zu lernen. Dabei wird angenommen, dass sich normales Nutzerverhalten statistisch beschreiben lässt. In einer Lernphase, in der angenommen wird, dass kein Angriff im Netzwerk stattfindet, wird das System angelernt, Netzwerkmuster zu erkennen, die „normal“ sind. Dies erlaubt die Erkennung von deutlich mehr Features als ein

Administrator sie manuell spezifizieren könnte.

In der Analysephase werden aufgezeichnete Events mit dem gelernten Referenzprofil verglichen und statistische Ausreißer gemeldet. Die Analysen können dabei auf verschiedenen Ebenen durchgeführt werden, bspw. die Analyse von Bandbreiten verschiedener Protokolle.

Durch signifikante Ereignisse wie bspw. Änderungen von Konfigurationen des Netzes oder auch Ereignisse, die das Verhalten der Nutzer kurzzeitig ändern (z. B. wichtiges Update, großes Fußballspiel), können auch durch normales Verhalten Anomalien im Netzwerk auftreten, die vom IDS erkannt werden würden. Dennoch können so auch Anomalien wie DoS-Angriffe oder Port Scans erkannt werden, weil dabei bspw. kurzzeitig die Anzahl Flows pro Sekunde stark ansteigt. So können Anomalien schon mit einfachen Mitteln erkannt werden.

7.10.2 Systemmodell

Am Netzwerk angeschlossen sind viele Probes, die Informationen an ein zentrales System senden. Dies bildet das Sensor-Subsystem. Events werden hier klassifiziert. In der Trainingsphase, in der angenommen wird, dass es nur „normales“ Netzwerkverhalten gibt, werden alle Events (auch negativ klassifizierte Events) an das Modeling Subsystem weitergeleitet, in dem die Events für das Ableiten eines Modells genutzt werden. Das tranierte Modell wird im Analyse-Subsystem eingesetzt, welches vom Sensor-Subsystem positive Events empfängt und genauer analysiert, ob es sich um eine Anomalie handelt.

7.10.3 Klassifikationskriterien

7.10.4 Anomalietypen

Zur Erkennung von *Punktanomalien* werden Messpunkte in einen n-dimensionalen Raum abgebildet. Beim Normalverhalten clustern sich Punkte. Anomalien sind dann Punkte, die außerhalb von Clustern liegen. Für eine gute Erkennung sollten möglicht wenige Dimensionen gewählt werden, da sonst die Cluster ein viel größeres Volumen annehmen.

Kontextanomalien sind Datenpunkte, die für sich gesehen nicht verdächtig sind, in dem Kontext, in dem sie auftreten, untypisch sind. Beispiele dafür sind große Datenübertragungen bei eingebetteten Geräten oder wenig Traffic zu Peak-Zeiten.

Kollektive Anomalien werden durch Abweichungen von Zustandsautomaten erkannt. Datenpunkte werden als nicht verdächtig angesehen, solange sie in einer bestimmten Reihenfolge auftreten. Abweichung davon werden als Anomalien erkannt. Beispiele dafür sind der Download von Dateien, ohne zuvor einen erfolgreichen Login durchgeführt zu haben.

7.10.5 Erkennungstypen

Statistiches Profiling nutzt einfache statistische Methoden (Histogramme, Abschätzung von Parametern basieren duaf Verteilungen, Regressionen, etc.), um signifikaten Abweichungen zu erkennen. Gerade die aktuelle Forschung untersucht neuronale Netze, um

normales Verhalten zu erlernen und Angriffe in Daten zu erkennen. Ein großes Problem mit neuronalen Netzen ist, dass durch die schiere Größe der Netzwerke nicht mehr genau gesagt werden kann, nach welchen Kriterien das neuronale Netz bestimmte Einschätzungen trifft.

Das Ergebnis der Lernvorgänge können einerseits direkt verschiedene Modelle sein, die die Erkennung durchführen. Andererseits ist es aber auch möglich, dass die Modelle zur Berechnung von Regelbäumen benutzt werden, die die dann für die eigentliche Erkennung genutzt werden.

Um auch zeitlich abhängiges Normalverhalten zu erkennen, wird spektrale Analyse bspw. durch Fouriertransformation und anschließende Analyse des Frequenzspektrums durchgeführt. So kann auch erkannt werden, dass bspw. große Dateitransfers durch regelmäßige Backups nicht als Angriff erkannt werden und trotzdem große Datenvolumen außerhalb der Backupzeit erkannt werden.

7.10.6 Eigenschaften

Hauptvorteil automatischer Anomalieerkennung ist, dass unbekannte Angriff erkannt werden können. Dadurch ist es auch vergleichsweise einfach einzurichten ist.

Die Nachteile liegen in verschiedenen Aspekten. Privatsphäre der Nutzer wird insofern eingeschränkt als dass das Verhalten der Nutzer für die Anomalieerkennung analysiert wird. Außerdem müssen immer wieder „normale“ Verhaltensweisen im Netz neu trainiert werden. Trotzdem erzeugt ein solches System eine hohe Zahl an falsch positiven Ergebnissen. Positive Ergebnisse (auch echt positive) lassen sich zudem nur schwer analysieren, um bspw. den Ursprung des Angriffs zu finden. Außerdem funktioniert das IDS nur, wenn der Angriff nicht wie normales Nutzungsverhalten aussieht.

7.11 Testing and Benchmarking of IDS

IDS wurden in verschiedenen Umgebungen getestet. So werden bspw. große Mengen von Daten in das IDS gefüttert oder auch Traffic basierend auf definierten Service-Modellen generiert, um Netzwerktraffic einer kleinen Organisation zu simulieren.

Für Tests von IDS ist die Open Source Philosophie prädominant. Es werden individuelle Testumgebungen aufgesetzt und bestehende Exploits und Angriffe gesucht. Der Traffic-Generator wird mit einer Mischung aus „normalem“ (harmlosen) und simuliertem Angriffs-Traffic gefüttert. Anschließend älsst sich die Erkennungsrate (falsch positiv / falsch negativ) ermittelt.

Dabei stellt sich natürlich weiter die Frage, ob die Testumgebung sich auch wirklich so verhält wie eine echte Umgebung. Das Einspielen von Angriffs-Traffic in ein Produktivsystem kann nämlich auch Schaden in dem entsprechenden System verursachen.

Stichwortverzeichnis

accountability, 6
Address Space Layout Randomization, 16
Angriff, 6
Angriffssignatur, 58
Anomalie
    kollektiv, 60
    Kontext-, 60
    Punkt-, 60
Anycast, 24, 42
Application Level Proxy, 51
ASLR, 16
Attack Tree
    DDoS, 27
Audit, 56
authorization violation, 7

Backdoor, 18
Backscatter, 22
Bastion Host, 50
Bedrohung, 6
Bedrohungsbaum, 8
BGPSEC, 39
Blackhole, 32
Bloom Filter, 28
Blue Pill, 19
Buffer Overflow, 13

canary, 16
CDN, 24
Circuit Level Proxy, 51
Command And Control, 23
Controlled Delivery, 32
Cross-Site-Scripting, 18
Crypto Officer, 46

DDoS, 23
Deep Packet Inspection, 54
Distributed Denial of Service, 23
DMZ, 51
DNSCurve, 48
DNSKEY, 46
DNSSEC, 45
DS, 47
dual-homed host, 51

eavesdropping, 7
Erkennung, 9, 56

Fälschung, 7
Firewall, 49, 50
Format-String, 17

GNS, 48
GNUnet, 48
Greyhole, 32

HIDS, 57
Honey Pot, 59
honeypot, 9

IANA, 35
ICMP Traceback, 28
inbound, 52
Integrität, 6
Interdomain Route Validation, 38
Internet Assigned Numbers Authority, 35
Intrusion, 55
Intrusion Detection Message Exchange Format, 58
IP Traceback, 29
IRV, 38

Key Signing Key, 45, 46
kontrollierter Zugriff, 6
KSK, 45, 46

Lamport Signatures, 37
Loop, 32

Malware, 18
masquerade, 7
Memory Integrity Checking, 16
Merkle Hash Tree, 36
Minimum Route Advertisement Interval, 37

Network Access Control, 54
Network Address Translation, 51
NIDS, 57
NOP-slide, 15
NSEC, 47
NSEC3, 47

One-Way Gateway, 54
outbound, 52
Overclaiming, 33

Paketfilter, 50
    dynamisch, 52
Path Authentication, 35
PDU, 7
Penetrate-and-Patch, 21
Perimeter Network, 51
Periphery Nodes, 39
PNRP, 48
Prävention, 9
Proxy, 51
    Application Level, 51
    Circuit Level, 51
    transparent, 54

Quine, 19

Race Condition, 17
Ratenkontrolle, 24
Reaktion, 10
Recovery Key Share Holder, 46
Reflection, 22
Remote-Controlled Browsers System, 54
repudiation, 7
Resource Exhaustion, 33
Response, 56
Return-Oriented Programming, 16
RIR, 35
Rootkit, 18
ROP, 16
RPKI, 39
RRSIG, 46

S-BGP, 35
Sabotage, 7
Screened Host, 51
Secure Origin Authentication, 35
security service, 10
setgid, 13
setuid, 13
Sicherheitsanforderung, 21
Sicherheitsziel, 6
SIEM, 58
Signaturerkennung, 58
Sinkhole, 33
Smurf, 22
Split-Horizon, 44
SQL Injection, 18
SSH Inspection, 54
SSL Inspection, 54
Stack Frame, 14
Stack Smashing, 15
Stackguard, 16

threat, 6
threat tree, 8
trojanisches Pferd, 18
TSIG, 45

Underclaiming, 33
Unified Threat Management, 54

Verfügbarkeit, 6
Vertraulichkeit, 6
Virus, 19

Web of Trust, 38
Wormhole, 33

XSS, 18

Zone Signing Key, 46
ZSK, 46

Literatur

[1] *AddressSanitizer — Clang 19.0.0git documentation*. URL: https://clang.llvm.org/docs/AddressSanitizer.html (besucht am 25. 04. 2024).

[2] *Anycast*. In: *Wikipedia*. Page Version ID: 234090017. 28. Mai 2023. URL: https://de.wikipedia.org/w/index.php?title=Anycast&oldid=234090017 (besucht am 23. 05. 2024).

[3] Lasse Collin. *XZ Utils backdoor*. 2024. URL: https://tukaani.org/xz-backdoor/ (besucht am 02. 05. 2024).

[4] *Formatted Output (The GNU C Library)*. URL: https://www.gnu.org/software/libc/manual/html_node/Formatted-Output.html (besucht am 25. 04. 2024).

[5] Andy Heffernan. *Protection of BGP Sessions via the TCP MD5 Signature Option*. RFC 2385. Aug. 1998. DOI: 10.17487/RFC2385. URL: https://www.rfc-editor.org/info/rfc2385.

[6] *NVD - CVE-2008-0166*. 2008. URL: https://nvd.nist.gov/vuln/detail/CVE-2008-0166 (besucht am 02. 05. 2024).

[7] *NVD - CVE-2024-3094*. 2024. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-3094 (besucht am 02. 05. 2024).

[8] Günter Schäfer. "Schutz von Kommunikationsinfrastrukturen". 2024.

[9] *Valgrind Home*. URL: https://valgrind.org/ (besucht am 25. 04. 2024).

[10] *What is a content delivery network (CDN)? | How do CDNs work?* URL: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/ (besucht am 23. 05. 2024).