Kantonsschule Im Lee

Hacking und Cybersicherheit

Die Schattenseiten der digitalen Welt

Corsin Streit (4a)[1] unter Aufsicht von Thomas Graf[2]

[1]Student, Kantonsschule Im Lee, Winterthur

[2]Fachschaft Informatik, Kantonsschule Im Lee, Winterthur

Winterthur, 23. Dezember 2024

Abstrakt

Das Ziel dieser Arbeit besteht darin, durch Literaturrecherche und selbstständiger Nachforschung einen für die Allgemeinheit verständlichen Überblick über das Thema Hacking und Cybersicherheit zu schaffen. Dazu werden technische Grundlagen und Hacking-Methoden erläutert und anhand von konkreten Beispielen anschaulich dargestellt. Ferner wird auf das Thema Dark Web eingegangen.

Stichworte: Hacking, Cybersicherheit, Hacking-Methoden, Dark Web

Inhaltsverzeichnis

1 Einführung

1.1 Zielsetzung

Hacking ist ein riesiges Themenfeld, dessen tiefgründiges Verständnis Kenntnisse in vielen Bereichen der Computerwissenschaften voraussetzt. Jede Person, die ein Smartphone, Laptop, Tablet, etc. benutzt, ist den Gefahren des Hackings täglich ausgesetzt, doch nur ein kleiner Bruchteil der Benutzer weiss, was bei einer Hacking-Attacke eigentlich passiert und wie die Konsequenzen aussehen können. Ziel dieser Arbeit ist es, einen Überblick über das Thema Hacking und Cybersicherheit in einer Form zu schaffen, die für die Allgemeinheit verständlich ist. Dabei wird bewusst vor Allem auf leicht verständliche Themenbereiche oder solche, die uns im Alltag am Meisten betreffen eingegangen und die Detailgenauigkeit je nach Thema reduziert. Analysen von modernen Hacking-Attacken sollen dazu beitragen, ein Verständnis der Implementation der Hacking-Methoden zu erlangen und das mögliche Ausmass der Konsequenzen aufzeigen. Zuletzt wird auf das Thema Dark Web eingegangen, da dies als Plattform zur Anonymisierung bei vielen Hacking-Attacken eine Rolle spielt.

1.2 Motivationserklärung

Ich bin grundsätzlich an Computern und dem Programmieren interessiert. Meistens finde ich in meinem durch Sporttrainings ausgefüllten Alltag aber keine Zeit, mich mit dieser Leidenschaft auseinanderzusetzen. Die Maturitätsarbeit schien mir als passende Gelegenheit, mir Wissen in einem computerbezogenen Themengebiet anzueignen. Das Thema Hacking und Cybersicherheit interessiert mich besonders, da ich gerne die Limiten von in diesem Fall Computersystemen teste und es ein Themengebiet ist, das kreatives Denken und logische Schlussfolgerungen voraussetzt und fördert.

1.3 Methodik

Es gibt viele verschiedene Methoden, sich mit dem Thema Hacking auseinanderzusetzen. Einsteigern werden oft Videokurse oder Bücher empfohlen, die Themenbereiche, auf die sie einen Fokus setzten, sind aber oft sehr unterschiedlich. Ich habe mich entschieden, meine Arbeit hauptsächlich auf Literatur zu basieren und das Werk “Hacking - The Art of Exploitation” von Jon Erickson als Hauptinformationsquelle zu gebrauchen. Unterstützend arbeite ich mit wissenschaftlichen Arbeiten und weiteren, weniger umfangreichen Büchern. Es fällt mir leichter, Informationen anhand von Geschriebenem zu erarbeiten. Ausserdem ist es bei Büchern tendenziell einfacher einzuschätzen, welcher Inhalt erwartet werden kann. Das Buch “Hacking - The Art of Exploitation” ist relativ alt (2008), dies passt aber gut zu meiner Zielsetzung, da die Komplexität des Hackings seither exponentiell zugenommen hat und ich eine tiefgründige, verständliche Analyse einer einfachen Hacking-Attacke gegenüber einer oberflächlichen Analyse einer komplexen Attacke bevorzuge. Die Themen weisen aber dennoch einen Bezug zur aktuellen Zeit auf.

Quellenangabe

Der Hauptteil des Wissens in Kapitel 3 und 4 wurde mit Hilfe des Buches “Hacking - The Art of Exploitation” [3] erarbeitet. Zur besseren Übersichtlichkeit wurden deshalb die Quellenangaben dieses Buches auf eine am Anfang der Kapitel beschränkt. Explizite Zitate werden immer gekennzeichnet.

Disclaimer

Sprache Es finden sich englische Begriffe in dieser Arbeit. Da Computerwissenschaften normalerweise in englischer Sprache praktiziert werden, existieren für einige Begriffe keine passende deutsche Übersetzungen. In diesem Falle wird auf die englische Form zurückgegriffen.

Gender Teilweise wird nur die männliche Form verwendet. Dies hat zwei Gründe: Zum einen ist die überwiegende Mehrheit der Personen, die in diesem Themenfeld arbeiten und gearbeitet haben, männlich, zum anderen ist die Einbeziehung beider Geschlechter layouttechnisch schwierig ansprechend umzusetzen. In den meisten Fällen sind jedoch beide Geschlechter gemeint, es sei denn, es wird explizit darauf hingewiesen.

2 Ursprung und Bedeutung

Der Stereotyp ist schlecht wegzudenken. Schwarzer Kapuzenpulli, abgedunkelter Raum und ein Bildschirm mit kryptografischen Zeichen. Auf dem Computer laufen bösartige Programme, die Schwachstellen von anderen ausnutzen und dabei Unheil anrichten. Die Handlung ist kriminell und bestrafbar. Auch wenn diese stereotypische Beschreibung wohl auf eine sehr kleine Gruppe von Personen zutreffen mag, sieht die Realität bedeutend anders aus. Jon Erickson formulierte folgenden Satz: "Beim Hacken geht es eher darum, das Gesetz zu befolgen als es zu brechen. Das Wesen des Hackens besteht darin, unbeabsichtigte oder übersehene Anwendungen für die Gesetze und Eigenschaften einer gegebenen Situation zu finden und sie dann auf neue und erfinderische Weise anzuwenden, um ein Problem zu lösen - was auch immer es sein mag." `(Erickson, 2008, S. 1, übersetzt mit DeepL)` [3]

2.1 Bedeutung und Urpsrung des Begriffes

Im Grunde genommen ist ein "Hack" nichts mehr als eine "von Innovation, Stil und technischem Können durchdrungene" Lösung zu einem Problem, das nicht einmal einen Bezug zu Computern aufweisen muss. Den Ursprung nahm der Begriff im Modelleisenbahnclub des Massachusetts Institute of Technology (MIT). Aus geschenkten Elektronikbauteilen bauten die Clubmitglieder die Steuerung ihrer Modellzüge. Durch verschiedene Optimierung, Hacks, versuchten sie, diese so elegant wie möglich zu gestalten. Ihr Ziel war nicht das simple Funktionieren. Es war nicht einmal wichtig, dass die Lösung einen grossen Nutzen zeigte. Was zählte, war die technische Eleganz, die hinter der Lösung steckte.
Mit dem Erscheinen erster Computer und der Gründung eines Computerclubs im MIT wurde die Bedeutung auf die technische Welt ausgeweitet. Als Steven Levy 1984 die Leiterprinzipen des Hackings zu einer "Hackerethik" zusammenfasste, formulierte er folgenden Satz: "Du kannst mit Computern Kunst und Schönheit schaffen." [14]

Das Negative, das viele Personen heute mit dem Hacking assoziieren, wurde erst nach und nach dem Bedeutungskonzept hinzugefügt. Nach weiteren Grundsätzen der Hackerethik sollten sich Hacker für freie Informationszugänglichkeit, Dezentralisierung und unlimitierten Zugriff auf alles, was einen etwas über die Welt lehren kann, einsetzen. Mit den gesetzlichen Richtlinien nahmen sie es nicht genau. Die Neugier, Wissensbegierde und allgemein das technische Interesse führte zur Entdeckung und Erkundung neuer "Spielwiesen", was in den 1960er-Jahre zum ersten "modernen" Hack führten. Findige Hacker fanden heraus, dass sich durch das Abspielen einer ganz bestimmten Frequenz das Telefonnetz so manipulieren liess, dass Anrufe nicht verrechnet wurden. Sogar Tech-Legenden wie die späteren Apple-Gründer Steve Jobs und Steve Wozniak beteiligten sich an dem sogenannten Blue-Boxing und verdienten Geld durch das Verkaufen von Frequenzgeneratoren, den Blue-Boxes.
Zeitungsartikel und Filme, insbesondere der 1983 veröffentlichte Titel War Games, trugen in grossem Masse zur heutigen Auffassung des Hackers bei. Ein Hacker ist "jemand, der ohne Erlaubnis in die Computersysteme anderer Leute eindringt, um Informationen herauszufinden oder etwas Illegales zu tun." `(Cambridge University Press, Oktober 2024, übersetzt mit DeepL)` [14, 4]

2.2 Motivation, Ethik und Begriffserklärung

Die Motivationen und Ethiken, die Hacker verfolgen, fallen sehr unterschiedlich aus. Grob kann man sie in zwei Kategorien einteilen: die "Black Hats" (auch "Cracker" genannt) und die "White Hats" (auch "Penetration Tester" genannt).
Black Hat Hacker verfolgen kriminelle Ziele, die (umwelt-)politisch, monetär oder egoistisch motiviert sein können. Oftmals verbreiten sie Malware (Schadsoftware). Dabei ignorieren sie legale und ethische Grundsätze.
Ihre Gegenspieler sind die White Hat Hacker. Diese benutzen dieselben Methoden, arbeiten aber im Rahmen der Gesetze und halten sich an ethische Grundsätze. Ihre Aufgabe ist es, Privatpersonen, Firmen oder ganz allgemein gefährdete Computer vor den Black Hats zu schützen. Um dies zu bewerkstelligen, hacken sie sich nach vorheriger Absprache mit den Besitzer in Computersysteme ein, um mögliche Sicherheitslücken zu finden und zu beheben.
Das Resultat dieses Wettkampfes ist gewinnbringend. "Das Endergebnis dieser Interaktion ist positiv, denn es führt zu intelligenteren Menschen, verbesserter Sicherheit, stabilerer Software, erfinderischen Problemlösungstechniken und sogar zu einer neuen Wirtschaft." `(Erickson, 2008, S. 4, übersetzt mit DeepL)` [3]
Zwischen den beiden Extremen liegen die Grey Hat Hacker, die zwar oftmals illegal vorgehen, dabei aber keine negativen Absichten verfolgen. Beispielsweise hacken sie sich unberechtigt in ein Firmensystem ein, kommunizieren aber danach die gefunden Schwachstellen. [1, 2]

Ein weiterer oft verwendeter Begriff ist "Script Kiddies". Dies sind ungeschulte (meist junge) Person ohne tiefgründige Hacking-Kenntnisse, die "von anderen entwickelten Skripts oder Programme für hauptsächlich böswillige Zwecke verwenden" `(Wikipedia, Oktober 2024, übersetzt mit DeepL)`. Nichtsdestotrotz können sie grosse Schäden anrichten. [21]

3 Grundlagen der Informatik

Zum Verständnis des Kapitel Methoden und Techniken sind Grundkenntnisse der Informatik nötig. Diese werden in diesem Kapitel erarbeitet. Wie am Anfang bereits erwähnt diente, sofern keine andere Quelle angegeben wurde, das Buch “Hacking - The Art of Exploitation” [3] als Hauptquelle dieses Kapitels.

3.1 Hardwaretechnische Grundlagen

Ein Computer besteht aus verschiedenen Bestandteilen, die im Zusammenspiel Verarbeitung von Informationen ermöglichen. Folgende Komponenten sind zur Ausführung eines Programms wichtig.

3.1.1 CPU

Die Central Processing Unit (CPU) beschreibt den wichtigsten Prozessor eines Computers. Mithilfe einer komplexen Platinenstruktur führt er, den Anweisungen eines Programms folgend, Berechnungen durch.

Register Wichtig für die Ausführung eines Programms sind Register. Register sind “Speicherbereiche für Daten, auf die der Prozessor besonders schnell zugreifen kann”. [20] Sie speichern zum Beispiel den Fortschritt der Ausführung eines Programms. Einige der wichtigsten Register sind folgende:

- **Extended Instruction Pointer (EIP)**: Befehlszeiger für die jetzige Instruktions-Adresse [7]
- **Extended Stack Pointer (ESP)**: Stapelzeiger für die vorherige Adresse [7]
- **Extended Base Pointer (EBP)**: Basiszeiger bei Funktionsaufruf, dient zur Referenzierung lokaler Variablen im Stack [7]
- **Saved Frame Pointer (SFP)**: Gesicherter Basiszeiger für das vorherige Stack Frame, dient zur Wiederherstellung des ursprünglichen Stack-Zustands [7]

Die Bedeutung dieser Funktionen wird im Kapitel Funktionsweise eines Programms genauer erklärt.

3.1.2 RAM (Arbeitsspeicher)

Das Random Access Memory (RAM) ist ein elektronischer Computerspeicher, dessen Daten sehr schnell und in beliebiger Reihenfolge gelesen und geändert werden können. Er wird deshalb für Daten aktuell laufender Programme verwendet. Oft wird ein Vergleich zum menschlichen Kurzzeitgedächtnis hergestellt.

Bit und Binärzahlen Computer sind nur imstande, Nullen (0) und Einsen (1) zu speichern, sie arbeiten also nur mit zwei Zuständen. Ein binary digit (Bit) beschreibt einen einzelnen

solchen Zustand. Werden mehrere Bits zur Datenspeicherung verwendet, steigt die Anzahl möglicher Kombinationen exponentiell auf 2^n (bei $n = Anzahl\ Bits$) Bits. Zahlen werden folgendermassen gespeichert: Die Stelle in der Binärzahl von hinten (also rechts) beginnend dient als Exponent (startend bei null) zur Basis 2. Bei einer 1 wird die Zahl zur Gesamtzahl addiert, bei 0 nicht. Die (dezimale) Zahl 21 zum Beispiel entspricht der Binärzahl 10101, da sie sich durch (von rechts nach links) $1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 = 21$ zusammensetzt. Zur Konversion können Online-Rechner wie zum Beispiel `https://www.rapidtables.com/convert/number/` verwendet werden.

Big and Little Endian Byte Order Ein Byte umfasst acht Bits. Zur Ordnung von Bytes existieren zwei Strukturen: Big und Little Endian. Big Endian speichert das wichtigste, also signifikanteste Byte zuerst, während Little Endian dieses zuletzt speichert. Das signifikanteste Byte ist dasjenige, das den grössten Einfluss auf die Gesamtzahl hat, daher normalerweise das am weitesten links stehende.

Speicheradressen Jedes Bit im RAM ist einzeln adressierbar. Heutzutage arbeiten die meisten Computer mit 32bit oder 64bit Systemen. Dies entspricht der Adresslänge der Speicheradressen und limitiert somit die Anzahl adressierbarer Bits. Bei 32bit ist die Ansteuerung von $2^{32} = 4\,294\,967\,296$ Bits möglich, bei 64bit $2^{64} = 18\,446\,744\,073\,709\,551\,616$. In der Praxis wird dies nur selten ausgereizt. Einfachheitshalber erfolgt die Angabe der Speicheradresse normalerweise im hexadezimalen Format (Basis 16). Dazu werden die Zeichen 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E und F verwendet. Ein Hexadezimalzeichen entspricht vier Bits (da $2^4 = 16$), eine 32bit Adresse setzt sich also aus acht Hexadezimalstellen zusammen. Hexadezimalzahlen werden üblicherweise mit dem Präfix “0x” angekündet. Eine 32bit Speicheradresse könnte folgendermassen aussehen: 0xb7ec819b. [19]

3.1.3 HDD/SSD

Speichermedien, wie hauptsächlich Hard Disk Drive (HDD) und Solid-State Drive (SSD), dienen der permanenten Speicherung von Daten, also auch über den Neustart eines Systems hinaus. Ihr Lese- und Schreibzugriff ist langsamer als bei RAM, die Speicherkapazität dafür meist höher. Oft wird ein Vergleich zum menschlichen Langzeitgedächtnis hergestellt.

3.2 Funktionsweise eines Programms

Hacker schreiben Code, nutzen ihn aber auch aus. Ein Verständnis grundlegender Programmierkonzepte ist deshalb notwendig.

3.2.1 Einführung

“Ein Programm ist nichts anderes als eine Reihe von Anweisungen, die in einer bestimmten Sprache geschrieben sind.” `(Erickson, 2008, S. 20, übersetzt mit DeepL)` [3] Es ist vergleichbar mit einem Küchenrezept. Genauso wie das Küchenrezept Menschen sagt, was sie tun sollen, sagt ein Programm dem Computer, was er zu tun hat. Im Gegensatz zu Menschen

hat ein Computer aber keinen eigenen Willen und muss sich an die gegebenen Instruktionen halten. In dieser Arbeit liegt der Fokus exemplarisch auf der Programmiersprache C, da damit verbundene Hacking-Methoden beschrieben werden, die grundlegenden Konzepte können aber auf viele Programmiersprachen übertragen werden.

3.2.2 Grundlagen

Ein Programm besteht aus verschiedenen Elementen. Die wichtigsten Elemente werden im Folgenden beschrieben.

Kontrollstrukturen Kontrollstrukturen steuern den Ablauf eines Programms.

If-then-else If-then-else-Strukturen erlauben die Ausführung eines Programmteiles nur unter bestimmten Bedingungen. Falls die Bedingung nicht erfüllt ist, wird der Programmteil übersprungen oder, wenn angegeben, ein alternativer Programmteil ausegführt.

While-Schleifen While-Schleifen wiederholen einen bestimmten Programmteil solange, bis eine Bedingung erfüllt ist.

For-Schleifen For-Schleifen werden normalerweise gebraucht, um einen bestimmten Programmteil eine definierte Anzahl mal zu durchlaufen.

Variablen und Konstanen Variablen und Konstanten speichern Daten. Variablen sind veränderbar, während Konstanten über die Laufzeit eines Programmes unverändert bleiben. Im Folgenden wird einfachheitshalber von Variablen gesprochen, die beschriebenen Eigenschaften sind aber auch auf Konstanten übertragbar.

Datentypen Bei Variablen wird zwischen verschiedenen Datentypen unterschieden. Einige Programmiersprachen sind bei der Handhabung flexibel und erlauben Veränderungen des Datentyps einer Variable, während andere Variablen bestimmte Datentypen zuordnen. Die Liste bietet eine Auflistung der wichtigsten Datentypen:

- **String**: Text (“Hello”, “World”)
- **Int**: Ganzzahl (2, 37)
- **Float**: Gleitkommazahl (3.1415, 2.7182)
- **Boolean**: Wahrheitswert (True, False)
- **Array**: Geordnete Liste mit festem Datentyp ([1, 3, 2, 8])
- **List**: Geordnete Liste mit gemischten Datentypen ([2, “Hello”, True])
- **Dict**: Liste mit Schlüssel-Wert-Paaren ([“Mehl”: 500, “Zucker”: 200, “Zitrone”: False])

Gültigkeitsbereich und Lebensdauer Variablen sind je nach Initialisierung (Erstellung) nur für bestimmte Teile eines Programms zugänglich. Wird eine Variable beispielsweise in einer Funktion initialisiert, existiert sie nur während der Funktionsausführung. Variablen besitzen folgende Gültigkeitsbereiche:

- **Globale Variablen**: Zugänglich vom gesamten Programm
- **Lokale Variablen**: Zugänglichkeit auf Funktion oder Programmteil limitiert
- **Statische Variablen**: Zugänglichkeit auf Funktion limitiert, behalten jedoch Wert bei erneutem Funktionsaufruf

Die verschiedenen Gültigkeitsbereiche führen dazu, dass gleichnamige Variablen mehrere Speicheradressen haben können. In einigen Programmiersprachen müssen Variablen vor der ersten Verwendung initialisiert, also angekündigt, werden.

Arithmetische Operatoren Arithmetische Operatoren werden verwendet, um mathematische Operationen durchzuführen. In C stehen die unten aufgelisteten Operatoren zur Verfügung:

- **Addition (+)**: 2 + 3 = 5
- **Subtraktion (-)**: 5 - 3 = 2
- **Multiplikation (*)**: 5 * 3 = 15
- **Division (/)**: 15 / 3 = 5
- **Modulus (%)**: 15 % 4 = 3 ((positiver) Rest der Division)
- **Inkrement (++)**: a++ = a + 1
- **Dekrement (–)**: a– = a - 1
- **Exponentiation (pow())**: pow(a, 3) = a^3

Für einige Operatoren können Gleichungen, deren Resultat-Variable auch in der Berechnung vorkommt, abgekürzt werden. Statt $a = a + 5$ lässt sich beispielsweise a += 5 schreiben.

Vergleichsoperatoren Vergleichsoperatoren dienen dem Vergleich von Variablen oder Werten. Sie geben jeweils True oder False aus. Folgende Operatoren können verwendet werden:

- **Gleichheit (==)**: 5 == 5 → True
- **Ungleichheit (!=)**: 5 != 3 → True
- **Größer als (>)** : 5 > 3 → True
- **Kleiner als (<)**: 5 < 3 → False
- **Größer oder gleich (>=)**: 5 >= 5 → True
- **Kleiner oder gleich (<=)**: 3 <= 5 → True

Funktionen Funktionen dienen zur Wiederholung bestimmter Programmteile. Sie erlauben einen In- und Output. Der arithmetische Operator pow() ist beispielsweise eine Funktion. Sie erlaubt den Input von `base` und `exponent` und gibt die Potenz der beiden Zahlen zurück. Eine Potenz-Funktion für Ganzzahlen könnte so aussehen:

```
1 int pow(int base, int exponent) {
2   int result = 1; // Startwert für die Multiplikation
3   for (int i = 0; i < exponent; i++) {
4     result *= base; // Multiplikation
5   }
6   return result;
7 }
```

Dateizugriff Dateien, die permanent auf einem Speichermedium gespeichert sind, können gelesen, bearbeitet und ausgeführt werden. Ein Benutzer kann dabei aber nur diejenigen Operationen verwenden, deren Rechte er besitzt. Die Identifikation des Benutzers erfolgt über eine ID (beispielsweise 999).

Pseudo-Zufallszahlen Bei bestimmten Anwendungen, wie zum Beispiel der Simulation eines Wettermodells oder einem Computerspiel, werden Zufallszahlen benötigt. Computer sind nicht in der Lage, zufällige Zahlen zu generieren. Hauptsächlich wird deshalb auf zwei Alternativen zurückgegriffen: Bei der ersten Möglichkeit werden mathematische Operationen verwendet, die zufällig aussehen. Eine bekannte Methode ist beispielsweise das Quadrieren zehnstelliger Dezimalzahlen. Vom Resultat werden die mittleren zehn Ziffern als neue Zufallszahl angesehen. Eine zweite Möglichkeit ist die Verwendung unvorhersehbarer Prozesse. Dabei wird aus unvorhersehbaren Prozessen wie dem atmosphärischen Rauschen Zahlen generiert. [8]

3.2.3 Abstraktions-Ebenen

Computer können nur mit aus Einsen und Nullen bestehender Maschinensprache umgehen. Da die Programmierung in Maschinensprache sehr aufwendig und unintuitiv ist, wurden verschiedene Abstraktions-Ebenen eingeführt.

Höhere Programmiersprache (High Level Language) Höhere Programmiersprache liegt nahe bei der englischen Sprache und ist (zumindest in Grundzügen) für die meisten Leute intuitiv verständlich. Normalerweise wird in höherer Programmiersprache programmiert. Das Beispiel der Potenz-Funktion oben ist in höherer Programmiersprache angegeben.

Maschinensprache (Machine Language) Maschinensprache besteht nur aus Einsen und Nullen und ist auch für die meisten professionellen Programmierer unverständlich. Der Code $x = 3 + 5$ könnte so aussehen: b8 05 00 00 00 83 c0 03 . (Angabe im hexadezimalen Format)

Assemblersprache (Assembly Language) Assemblersprache kann als die Verbalisierung der Maschinensprache angesehen werden und wird teilweise zur Programmierung verwendet. Dies aber nur bei sehr systemnahen Anwendungen. Derselbe Code $x = 3 + 5$ könnte in Assemblersprache so aussehen: mov eax, 5 add eax, 3 . Hier werden auch die in Kapitel 3.1.1 erwähnten Register ersichtlich.

Zur Übersetzung zwischen den Abstraktionsebenen dienen der Compiler und Assembler. Der Compiler transformiert höhere Programmiersprache in Maschinensprache, der Assembler Assemblersprache in Maschinensprache.

3.2.4 Ablauf zur Ausführung

Die Ausführung eines Programms folgt folgendem Grundprinzip:

1. Der Code wird kompiliert (oder assembliert) und als Maschinencode im RAM gespeichert.
2. Die Instruktion, zu der das EIP-Register zeigt, wird gelesen.
3. Die Bitlänge der Instruktion wird zum EIP addiert, der EIP zeigt jetzt also zur nächsten Instruktion.
4. Die Instruktion wird ausgeführt.
5. Neustart bei Schritt 2.

3.2.5 Speichersegmentierung in C

Die Datenanordnung im RAM folgt einer bestimmten Struktur. Bei C wird sie in fünf Segmente aufgeteilt.

- **Text Segment**: Speichert Code (in Maschinensprache), ist unveränderlich
- **Initialized Data**: Speichert initialisierte globale und statische Variablen
- **Uninitialized Data**: Speichert uninitialisierte Variablen
- **Heap Segment**: Ist kontrollierbar durch den Programmierer, wächst von niedrigen zu hohen Adressen [13]
- **Stack Segment**: Speichert den Kontext und lokale Variablen bei Funktionsaufrufen, funktioniert nach LIFO-Prinzip (last-in-first-out): wächst von hohen zu niedrigen Adressen und wird umgekehrt abgearbeitet [13]

Unter dem Stack werden Umgebungsvariablen und Befehlszeilenargumente gespeichert. Umgebungsvariablen enthalten Information zur Umgebung des Systemes [11] und Befehlszeilenargumente sind vom Benutzer beim Programmstart eingegebene Daten.

Stack Frame Ein Stack Frame ist “ein Abschnitt des Stacks, der einem bestimmten Funktionsaufruf gewidmet ist.” `(NordVPN, Dezember 2024, übersetzt mit DeepL)`[12] Das Stack

Frame der aufgerufenen Funktion wird oberhalb des Stack Frame der aktuellen Funktion erstellt und enthält alle wichtigen Informationen. Dazu gehören unter anderem Variablen, der EBP und die Rückkehradresse. Die Rückkehradresse wird benötigt, um an der richtigen Stelle zur ursprünglichen Funktion zurückzukehren.

3.3 Funktionsweise eines Netzwerks

Viele Hacking-Attacken nutzen Schwachstellen in Netzwerken aus. Für Hacker sind Netzwerk-Attacken sehr attraktiv, da sie keinen physischen Zugriff auf Infrastruktur erfordern.

3.3.1 Einführung

Ein Netzwerk ist ein "großes System, das aus vielen ähnlichen Teilen besteht, die miteinander verbunden sind, um eine Bewegung oder Kommunikation zwischen den Teilen oder zwischen den Teilen und einem Kontrollzentrum zu ermöglichen." `(Cambridge University Press, Dezember 2024, übersetzt mit DeepL)` Anwendungen wie das World Wide Web (WWW) oder WhatsApp-Nachrichten basieren auf einer Netzwerkstruktur. Ein Netzwerk ist aber nicht dem Internet gleichzusetzen und kann auch lokal existieren. naja

3.3.2 OSI-Modell

Das Open Systems Interconnection (OSI)-Modell standardisiert den Netzwerkverkehr und bietet damit die Grundlage der Datenübertragung. Das Modell wurde in in den 1980er-Jahren von der International Organization for Standardization (ISO) entwickelt und ist heute mehrheitlich von dem Transmission Control Protocol (TCP)/Internet Protocol (IP)-Modell abgelöst, wird wegen besserer Übersichtlichkeit aber oft zu Unterrichtszwecken verwendet. [18]

Einführung Das OSI-Modell ist in sieben Ebenen mit jeweils spezifischen Funktionen zum Datentransport organisiert. Ebene sieben ist dabei am nächsten bei der Anwendung, während Ebene eins am nächsten bei der physischen Hardware liegt. Bevor Daten transportiert werden, werden sie bei Ebene sieben beginnend in alle Ebenen verpackt. Beim Erhalt werden sie wieder entpackt. Zwischengeräte wie Router, die für die Weiterleitung der Datenpakete verantwortlich sind, packen die Daten nur soweit, wie für die Weiterleitung nötig, aus. Meist entspricht dies Ebene drei. Datenpakete bestehen aus einem Header und Daten. Der Header enthält wichtige Informationen zum Datentransport, so zum Beispiel die Datenlänge oder die Senderadresse.

Netzwerkprotokolle Netzwerkprotokolle standardisieren die Struktur eines Datenpaketes. Sie sind für die korrekte Formatierung der Daten verantwortlich und bestimmen den Inhalt des Headers. Damit ermöglichen sie verschiedenen Geräten eine einheitliche Kommunikationsbasis. Die Auflistung der sieben Ebenen weiter unten bietet einige Beispiele.

Die sieben Ebenen Im Anschluss findet sich eine Auflistung der sieben Ebenen. Wichtige Eigenschaften oder Protokolle einer Ebene werden untergeordnet angegeben.

- **1. Bitübertragung**: Verantwortlich für die physische Verbindung zwischen zwei Punkten, überträgt einzelne Bits (zum Beispiel Ethernet-Kabel, Bluetooth)
- **2. Sicherung**: Zuständig für zuverlässige Übertragung der Daten inklusive Fehlerkorrektur und Flusskontrolle.
  - Media Access Control (MAC)-Adressen: Hardware-Adressen, werden zur Kommunikation auf Level 2 benötigt. Normalerweise (im Gegensatz zur IP-Adresse) gerätespezifisch.
  - Address Resolution Protocol (ARP): Wandelt IP-Adressen (Level 3) in MAC-Adressen (Level 2) um. Dazu sendet es Anfrage mit einer IP-Adresse an die Broadcasting-Adresse eines Netzwerkes (Adresse, die alle Geräte in einem Netzwerk umfasst). Das Gerät mit entsprechender IP-Adresse antwortet mit seiner MAC-Adresse.
- **3. Vermittlung**: Entscheidet über den Pfad der Daten, kümmert sich um die Fragmentierung (Aufteilung von Datenpaketen bei Überschreitung der maximalen Datenmenge).
  - IP-Adresse: Wird jedem internetfähigen Gerät zugewiesen wird und zur Kommunikation auf Level 3 benötigt.
  - Internet Control Message Protocol (ICMP): Meldet Fehler und führt Netzdiagnosen durch. Das `ping`-Kommando testet beispielsweise die Verbindung zwischen zwei Geräten. [15]
- **4. Transport**: Sorgt für Ende-zu-Ende-Kommunikation zwischen Anwendungen. Hauptsächlich werden zwei Protokolle verwendet.
  - User Datagramm Protocol (UDP): Einseitiges, verbindungsloses, unzuverlässiges, aber ressourcenschonendes Protokoll, dass für zeitkritische Datenübertragung geeignet wie Videostreaming oder Online-Gaming geeignet ist.
  - TCP: Zweiseitiges, verbindungsbasiertes, zuverlässiges, dafür ressourcenintensives Protokoll, dass für Anwendungen mit hoher Datenintegrität wie Dateiübertragungen, E-Mails oder Webinhalte geeignet ist.
- **5. Kommunikation**: Verwaltet Verbindungen durch Erstellung, Aufrechterhaltung und Auflösung.
- **6. Darstellung**: Formatiert und ver- oder entschlüsselt Daten für die Endanwendung.
- **7. Anwendung**: Schnittstelle einer Anwendungen zum Netzwerk.
  - Hypertext Transfer Protocol (HTTP) und Hypertext Transfer Protocol Secure (HTTPS): Kommunikationsprotokolle für das WWW. HTTPS ist im Gegensatz zu HTTP verschlüsselt. [17, 5] Die wichtigsten Anfrage-Methoden sind folgende:
    - GET: Fordert spezifische Ressource an.
    - HEAD: Fordert nur Header ohne Kontext an.
    - POST: Sendet Daten an Server.

- Domain Name System (DNS): Protokoll zur Umwandlung von Hostnamen wie `www.google.com` in IP-Adressen.
- Post Office Protocol (POP3): Veraltetes Protokoll zum Senden und Empfangen von E-Mails. [16]

3.3.3 Sockets

Sockets dienen als Endpunkte einer Verbindung und werden vom Betriebssystem bereitgestellt. [22] Sie operieren hauptsächlich auf Level 4. Sockets werden anhand von drei Eigenschaften unterschieden

Domain Die Domain legt fest, wie Adressen angegeben werden. Mögliche Optionen sind sind zum Beispiel IP (Version 4) (`AF_INET`) oder Bluetooth (`AF_BLUETOOTH`).

Typ Drei Typen von Sockets existieren: Stream Sockets, Datagram Sockets und raw sockets. Normalerweise werden Stream Sockets mit dem TCP benutzt, währenddessen Datagram Sockets das UDP verwenden. Dies führt auch zu den jeweiligen Eigenschaften: Stream Sockets sind zuverlässig aber langsam, Datagram Sockets schnell aber unzuverlässig. Raw Sockets erlauben die Miteinbeziehung und Analyse von tiefer liegenden Netzwerkebenen.

Protokoll Das Protokoll spezifiziert, welches Protokoll auf Ebene vier verwendet werden sollte. Im Normalfall muss dies nicht explizit angegeben werden, da das System standardmässig das zum Typ des Sockets passende Protokoll auswählt.

4 Methoden und Techniken

Hacking-Attacken sind sehr unterschiedlich und fokussieren sich auf verschiedene Bereiche eines Computersystems. Einige erfordern physischen Zugriff auf ein System während andere über ein Netzwerk erfolgen. Dieses Kapitel zeigt exemplarisch einige gut verständliche Hacking-Methoden und Techniken auf. Die Kapitel folgen jeweils derselben Struktur: Zuerst wird die allgemeine Funktionsweise einer Attacke beschrieben und danach auf die Abwehrmechanismen eingegangen. In einigen Kapiteln findet sich auch ein konkretes Beispiel mit Code. Zuletzt folgt ein Kapitel mit Erklärungen zu bekannten Begriffen, die bis dahin noch nicht erklärt wurden.
Wie am Anfang bereits erwähnt, diente, sofern keine andere Quelle angegeben wurde, das Buch “Hacking - The Art of Exploitation” [3] als Hauptquelle und Inspiration dieses Kapitels. Einige Beispiele wurden direkt aus dem Buch übernommen. Diese sind so gekennzeichnet.

4.1 Buffer-overflow

Ein Buffer beschreibt einen reservierten Speicherbereich, um Daten temporär zu speichern. Ein Buffer-overflow tritt auf, wenn mehr Daten in diesen Bereich geschrieben werden, als die Kapazität erlaubt. Dies kann dazu führen, dass zur Ausführung des Programms wichtige Daten überschrieben werden und das Programm anders abläuft, als vom Programmierer ursprünglich geplant. Dies ist besonders kritisch, wenn das Programm mit Root-, also Administrationsrechten ausgeführt wird, da der Angreifer dann die Kontrolle über das gesamte System erlangen kann. Programmiersprachen wie C, bei denen sich RAM sehr direkt kontrollieren lässt, sind besonders anfällig für Attacken dieser Art. Moderne Programmiersprachen besitzen oft eine automatische Speicherverwaltung, die die direkte Kontrolle des RAM erschweren oder sogar verunmöglichen.

4.1.1 Allgemeine Funktionsweise

Voraussetzung dieser Attacke ist ein Programm, das Daten (zum Beispiel Text) als Input annimmt und diese in das RAM kopiert, ohne vorher zu überprüfen, ob ausreichend Platz reserviert wurde. Der Angreifer kann den Input dann bewusst zu lange wählen, so dass er die Kapazität des buffers überschreitet. Ziel ist oft die Überschreibung der Rückkehradresse einer Funktion. Diese liegt unterhalb des Buffers im Stack Frame. Da Daten im Stack in Richtung tiefer Adressen gespeichert werden, wird die Rückkehradresse ab einer bestimmten Inputlänge überschrieben. Bei geschickter Zeichenfolge wird sie durch eine neue Speicheradresse ersetzt, die auf einen eigens im RAM platzierten Code zeigt. Dieser oft bösartige Code wird als Shellcode bezeichnet. Er muss in Assembler- oder Maschinensprache angegeben werden. Zur Platzierung des Shellcodes im RAM können verschiedene Methoden verwendet werden. In diesem Fall wird er als Input eingegeben, die Platzierung in Umgebungsvariablen ist beispielsweise aber auch möglich, erfordert aber deren Zugriff.
Der zur Ausführung dieser Attacke nötige Input enthält also eine neue Rückkehradresse und den Shellcode. Die Rückkehradresse zeigt dabei zur Speicheradresse des Shellcodes und lässt diesen bei Funktionsende ausführen. Zwei Probleme treten auf:

1. Meist ist es schwer vorherzusagen, wo genau die neue Rückkehradresse im Input platziert werden muss, damit sie die originale Rückkehradresse trifft. Die Speichersegmentierung kann zwischen Programmausführungen variieren und lässt eine exakte Vorhersage nur schwer zu. Dieses Problem kann einfach durch die wiederholte Angabe der neuen Rückkehradresse gelöst werden. Wenn man statt nur eine anzugeben einen ganzen Bereich mit Rückkehradressen füllt, erhöht sich die Wahrscheinlichkeit, die originale Rückkehradresse zu treffen, massiv.
2. Das zweite Problem bezieht sich auf die Vorhersage der Speicheradresse des Shellcodes. Da eine exakte, genau auf den Start des Shellcodes zeigende Rückkehradresse angegeben werden muss, kann das Problem nicht durch wiederholte Angabe gelöst werden. Hier greift man auf einen no-operation (NOP) sled zurück.

   NOP sled Ein NOP sled beschreibt eine Reihe von NOP-Instruktionen. Diese Instruktionen tun nichts, ausser zur nächsten Instruktion zu springen, bis schliesslich der finale Code (in diesem Fall Shellcode) erreicht wird.

   Die Platzierung eines langen NOP sled vor dem Shellcode macht eine genaue Vorhersage der Speicheradresse des Shellcodes überflüssig. Es reicht, wenn die Rückkehradresse den NOP sled trifft.

Der finale Input hat dementsprechend folgendes Format (RET für Rückkehradresse):

| NOP | NOP | NOP | SHELLCODE | RET | RET | RET |

Er führt zur Ausführung des Shellcodes.

4.1.2 Abwehrmechanismen

Bei modernen Computersystem ist diese Attacke wegen verschiedener Sicherheitsmechanismen relativ schwierig durchzuführen.

- **Bound-Checking**: Die simpelste Abwehrmethode ist das Bound-Checking. Sie verhindert das Schreiben von Daten bei Überschreitung der Kapazität eines Buffers.
- **Stack Canaries**: Stack Canaries beschreiben eine Methode, bei der bewusst Prüfdaten im RAM platziert werden, die nicht zur Veränderung gedacht sind. Eine Veränderung dieser Daten hat eine Alarmierung des Systems zur Folge. [6]
- **Adress Space Layout Randomization (ASLR)**: ASLR ordnet ausführbarem Code zufällige Adressen zu. Dies erschwert auch die ungefähre Vorhersage von Datenplatzierung im RAM. [10]
- **Non-Executable Memory**: Gewisse Speicherbereiche im RAM werden als nicht ausführbar markiert. Wird der Shellcode in einem dieser Bereiche platziert, ist er nutzlos.

4.1.3 Konkretes Beispiel

Dieses Beispiel inklusive Code wurde direkt aus dem Buch "Hacking - The Art of Exploitation" [3] übernommen. Es kann selbst nachgestellt werden. Zur Handhabung der Beispiele und Links zum Code siehe Kapitel Beispiele aus "Hacking - The Art of Exploitation" [3]. Zum besseren Verständnis wurde als zusätzlich zum Buch ein Stack Overflow-Eintrag verwendet. [9]

Voraussetzung für die Durchführung dieses Beispiels ist ein 32bit Linux System, bei welchem die oben angegebenen Abwehrmechanismen nicht vorhanden oder deaktiviert sind. Für das Beispiel werden folgende Programme verwendet:

- **notetaker.c**: Ein Notizprogramm, das Notizen in einer Datei (var/notes) speichert. Da das Programm von mehreren Benutzern benutzt werden können soll, gehört es root.
- **notesearch.c**: Ein Programm, das die Notizen einer bestimmten Person wiedergibt und einen Suchinput zulässt. Es gehört ebenfalls root. Das Programm hat eine Sicherheitslücke (im Programmausschnitt unten in Linie 5). Es kopiert den Suchstring, der als erstes Befehlszeilenargument eingegeben werden kann, ohne Überprüfung der Länge in den auf 100 Bytes limitierten Buffer. Dies geschieht durch die Funktion `strcpy()`.

```
1 int main(int argc, char *argv[]) {
2   int userid, printing=1, fd;
3   char searchstring[100];
4   if(argc > 1)
5     strcpy(searchstring, argv[1]);
6   else
7     searchstring[0] = 0;
```

- **exploit_notesearch.c**: Dieses Programm wird zur Erstellung des präparierten Suchstrings verwendet. Die Funktionsweise wird im Folgenden genauer erläutert. Das Programm ist im Anhang abgedruckt.

Erstellung des Suchstrings Zur ungefähren Abschätzung der Speicheradresse der Speicheradresse des Suchstring-Buffers wird eine Variable des **exploit_notesearch.c**-Programmes zur Hilfe genommen, dessen Adresse sich während der Ausführung abfragen lässt. Es kann davon ausgegangen werden, dass **notesearch.c**-Programm ähnliche Speicheradressen verwendet, da der Prozess ungefähr an derselben Stelle gestartet wird. Experimentieren ist aber nötig, weshalb ein `offset`-Wert definiert wird, der eine Verschiebung der Speicheradressen ermöglicht. Anhand dieser Daten wird ein Suchstring nach obigem Format erstellt. Der Shellcode öffnet in diesem Fall eine Root-Kommandozeile.

Ausführung von notesearch.c Das Programm wird mit dem präparierten Suchstring gestartet. Dabei wird ein Skript verwendetet, welches systematisch verschiedene `offset`-Werte ausprobiert. Falsche `offset`-Werte bringen das **notesearch.c**-Programm in den meisten Fällen zum Absturz. Sobald der richtige `offset`-Wert gefunden wurde, wird die Attacke erfolg-

reich durchgeführt und eine Kommandozeile geöffnet. Durch das Kommando `whoami` lässt sich überprüfen, dass die Kommandozeile Root-Berechtigungen besitzt.

4.2 Format String Exploitation

In den meisten Programmiersprachen gibt es Möglichkeiten, Text in die Kommandozeile zu drucken. Für variablen Input werden in C Format Specifiers gebraucht. Format Specifiers geben an, welcher Datentyp erwartet werden soll. Beispielsweise kann der Format Specifier `%d` für Ganzzahlen auf diese Weise eingesetzt werden:

```
1 int main() {
2   int zahl = 42;
3   printf("Ganzzahl: %d", zahl);
4 }
```

Die Ausgabe des Programms lautet:

```
  Ganzzahl: 42
```

Eine fehlerhafte Implementierung von Format Specifiers kann ausgenutzt werden, um unerwünschte Speicherzugriffe oder -manipulationen durchzuführen.

4.2.1 Allgemeine Funktionsweise

Voraussetzung dieser Attacke ist ein Programm, welches eine Angabefunktion wie `prinf()` verwendet, um vom Benutzer gegebenen Input zu drucken. Dabei wird der Input aber nicht explizit als String, sondern direkt gedruckt (siehe Code unten).

```
1 printf("%s", text); // korrekte Art
2 printf(text);  // falsche Art
```

Die falsche Implementierung erlaubt dem Angreifer die Benutzung von eigenen Format Specifiers. Zum Verständnis der Attacke muss die Speichersegmentierung bei einem `printf()`-Funktionsaufruf bekannt sein.

Speichersegmentierung bei printf() Bei Funktionsaufruf einer `printf()`-Funktion wird eine Rückkehradresse zur Ursprungsfunktion, die Speicheradresse im Text Segment des zu druckenden Textes und die Argumente der `printf()`-Funktion in dieser Reihenfolge auf den Stack gelegt. Die Argumente stehen jeweils direkt hintereinander. Je nach Datentyp werden die Werte direkt, also nicht als Referenzierung durch Speicheradressen, auf den Stack gelegt. Beim Format Specifier `%x`, der aus vier Bytes bestehende hexadezimale Zahlen in den Text einbettet, ist dies der Fall.

Die Falsche Implementierung der `printf()`-Funktion kann auf zwei Arten ausgenutzt werden:

1. **Speicherlesen**: Format Specifier wie beispielsweise `%x` erwarten Argumente. Werden keine Argumente angegeben, lesen sie die Daten an der Stelle aus, an der die Argumente erwarten werden. Durch Wiederholung von `%x` lassen sich so ab der Stelle, bei der die `printf()`-Funktion Argumente im RAM erwartet, Daten in die Kommandozeile drucken. Dies erlaubt das vom Programmierer möglicherweise unerwünschte Auslesen von Daten.

2. **Speichermanipulation**: Bedeutend schlimmer ist die unerwünschte Veränderung von Daten im RAM. Dazu kann der Format Specifier `%n` genutzt werden. Er schreibt die bisherige Anzahl gedruckter Zeichen als Binärzahl an eine als Argument angegebene Speicheradresse. Das Beispiel unten zeigt die Funktionalität von *%n*. Der Referenzoperator `&` wird benötigt, um die Variable `count` zu referenzieren. Dies ist erforderlich, da `%n` eine Speicheradresse erwartet.

```
1 int main() {
2   int count;
3   printf("Hallo, Welt!%n", &count);
4   printf("\nAnzahl der ausgegebenen Zeichen: %d", count);
5 }
```

   Die Ausgabe des Programms lautet:

```
Hallo, Welt!
Anzahl der ausgegebenen Zeichen: 12
```

   Diese Attacke ist speziell kritisch, wenn der gedruckte Text weiter unten im Stack gespeichert wird und so auf sich selbst referenzieren kann, denn dadurch können gezielt Speicheradressen manipuliert und durch geschickte Wahl des Input mit eigenen Daten gefüllt werden. Das Beisiel in Kapitel 4.2.3 demonstriert das Schreiben einer eigenen Speicheradresse an einen bestimmten Punkt im RAM. Mithilfe eines Shellcodes kann dies dieselben Konsequenzen wie eine Buffer-overflow-Attacke von sich tragen.

4.2.2 Abwehrmechanismen

Erneut ist die Attacke bei modernen Computersystemen wegen verschiedenen Abwehrsystemen nur schwer durchzuführen.

- **Drucken als String**: Wird der Input mit dem Format Specifier *%s* als String gedruckt, werden in ihm enthaltene Format Specifier ignoriert.
- **Stack Canaries, ASLR und Non-Executable Memory**: Die Sicherheitsmechanismen, die bereits in Kapitel 4.1.2 beschrieben wurden, schützen auch in diesem Fall vor ungewollten Änderungen im RAM oder der Ausführung des Shellcodes.

4.2.3 Konkretes Beispiel

Dieses Beispiel inklusive Code wurde direkt aus dem Buch "Hacking - The Art of Exploitation" [3] übernommen. Es kann selbst nachgestellt werden. Zur Handhabung der Beispiele und Links zum Code siehe Kapitel Beispiele aus "Hacking - The Art of Exploitation" [3].

Voraussetzung für die Durchführung dieses Beispiels ist ein 32bit Linux System, bei welchem die oben angegebenen Abwehrmechanismen nicht vorhanden oder deaktiviert sind. Für das Beispiel wird das Programm **fmt_vuln.c** verwendet.

fmt_vuln.c Das Programm **fmt_vuln.c** druckt einen vom Benutzer als Kommandozeilenargument gegebenen Input in die Kommandozeile. Die Sicherheitslücke besteht darin, dass das Programm den Text nicht explizit als String, sondern direkt druckt. Die zwei wichtigsten Zeilen sind unten abgedruckt.

```
1 strcpy(text, argv[1]);
2 prinft(text);
```

Eine weitere wichtige Eigenschaft des Programms ist, dass der `text`-Buffer, der den vom Benutzer gegebenen Input speichert, unterhalb (bei höheren Speicheradressen) des `printf()`-Stack Frame im Stack gespeichert ist. Die Angabefunktion muss auf dem höchsten Stack Frame sein, die Speicherung des `text`-Buffers muss also unterhalb geschehen. Dies macht eine Selbstreferenzierung möglich.

Zu Demonstrationszwecken wird der Wert einer Variable `test_val` mit einer Speicheradresse überschrieben. Analog zu Kapitel 4.1.3 kann die in diesem Kapitel beschriebene Technik aber zur Überschreibung einer Rückkehradresse und Ausführung von Shellcode verwendet werden. Eine gekürzte Version des Programm **fmt_vuln.c** ist im Anhang abgedruckt.

Zur Ausführung der Attacke sind folgende Schritte notwendig:

1. Durch Ausprobieren mit wiederholter Angabe des Format Specifiers `%x` lässt sich herausfinden, das wievielte Argument dem Beginn des `text`-Buffer entspricht. In diesem Fall ist es das vierte.
2. Ein geschickter Input wird eingegeben. Dieser hat folgendes Format:

| S1 | JUNK | S2 | JUNK | S3 | JUNK | S4 | %x | %x | %125x | %n | %201x | %n | %15x | %n | %175x | %n |

- **Speicheradressen (S)**: S1, S2, S3 und S4 stehen für die vier einzelnen Bytes der `test_val`-Variable. S1 zeigt zum ersten Byte, S4 zum letzten. **JUNK**: Das Wort JUNK wird als Platzhalter verwendet, von dem die `%x` Format Specifier lesen können. Die Zeichenfolge ist nicht von Bedeutung und kann ersetzt werden, das Wort muss jedoch eine Länge von vier Bytes aufweisen.

- **%x**: Die `%x` Format Specifier lesen hexadezimale Wörter der Länge vier Bytes und betten diese in den Text ein. Eine Zahl vor dem `x` gibt die Mindestlänge der zu lesenden Daten an. Mithilfe der Mindestlänge kann die Anzahl der bisher geschriebenen Zeichen auf die gewünschte Länge erhöht werden.
- **%n**: Die `%n` Format Specifier schreiben die Anzahl bisher geschriebenen Zeichen als Integer der Länge vier Bytes an die als Argumente angegebenen Speicheradressen. Bei korrekter Anpassung des Inputs verwenden sie S1, S2, S3 und S4.

Die ersten drei `%x` Format Specifier werden gebraucht um die Distanz zum `text`-Buffer zu überwinden. Der dritte Format Specifier erhöht die Anzahl gedruckter Zeichen um 125. Diese Anzahl wird dann durch den `%n` Format Specifier an die Speicheradresse S1 geschrieben. Da das am wenigsten signifikante Byte wegen der Little Endian Architektur zuerst gespeichert wird, wird das Byte an Stelle S1 mit der Anzahl gespeicherter Zeichen überschrieben. Der nächste Format Specifier erhöht die Anzahl gedruckter Zeichen erneut um die gewünschte Anzahl, in diesem Fall um 201. Sollte das zweite Byte der Speicheradresse tiefer sein als das erste ergibt sich ein Problem, da nur Erhöhung, nicht aber Vertiefung der bisher gedruckten Zeichen möglich ist. Dies lässt sich durch das Wrapping Around-Prinzip lösen.

Wrapping Around Ein Byte entspricht einem Zeichen bestehend aus acht Bit. Die Binärzahl `11111111` entspricht der Dezimalzahl 255. Bei Erhöhung um eins bekommt die Binärzahl ein neuntes Zeichen und sieht wie folgt aus: `100000000`. In diesem Fall sind nur die letzten acht Bits von Bedeutung. Diese bleiben bei einer Erhöhung der Zahl um die Dezimalzahl 256 gleich. Dadurch lässt sich das Problem der bereits zu hohen Anzahl gedruckter Zeichen eifach lösen, indem statt der ursprünglichen Zahl die nächsthöhere Zahl in Schritten von 256 verwendet wird.

3. Bei Ausführung des Programms wird dieses Verfahren insgesamt viermal angewendet und dadurch eine ganze Speicheradresse an die Adresse der `test_val` geschrieben. Eine Veränderung der nächsten drei Bytes hinter der `test_val`-Variable lässt sich nicht verhindern, dies ist aber nicht weiter störend.

4.3 Network Sniffing

4.3.1 Allgemeine Funktionsweise

4.3.2 Abwehrmechanismen

4.3.3 Konkretes Beispiel

4.4 Man-in-the-Middle

4.4.1 Allgemeine Funktionsweise

4.4.2 Abwehrmechanismen

4.5 Denial of Service

4.5.1 Allgemeine Funktionsweise

4.5.2 Abwehrmechanismen

4.6 Erklärung weiterer Begriffe

- Phishing
- Passwort-Attacken
- Ransomware
- Viren
- Würmer
- Trojaner
- Malware

5 Analyse bekannter Hackerangriffen

5.1 Russische Einflussnahme auf die Präsidentschaftswahlen (2016)

5.2 Angriff auf das ukrainische Stromnetz (2015)

5.3 WannaCry Ransomware (2017)

6 Das Dark Web

6.1 Grundlagen

6.2 Zugangsmechanismen und Tools

6.3 Sicherheitsrisiken

7 Diskussion

was ist rausgekommen?

8 Ausblick

was hätte man noch machen können?

A Appendix

A Beispiele aus “Hacking - The Art of Exploitation” [3]

Literatur

[1] Jean Abeoussi. *White Hat and Black Hat - The thin line of ethics.edited*. Okt. 2019.

[2] *Black Hat-, White Hat- & Grey Hat-Hacker*. https://www.kaspersky.de/resource-center/definitions/hacker-hat-types.

[3] Jon Erickson. *Hacking: The Art of Exploitation*. 2nd. San Francisco, CA: No Starch Press, 2008. ISBN: 978-1-59327-144-2.

[4] *HACKER | English meaning - Cambridge Dictionary*. https://dictionary.cambridge.org/dictionary/english/hacker. Okt. 2024.

[5] *HTTP versus HTTPS - Unterschied zwischen Übertragungsprotokollen - AWS*. [Online; aufgerufen am 26.12.2024]. URL: https://aws.amazon.com/de/compare/the-difference-between-https-and-http/.

[6] Michiel Lemmens. *Stack Canaries – Gingerly Sidestepping the Cage | SANS Institute*. [Online; aufgerufen am 26.12.2024]. Feb. 2021. URL: https://www.sans.org/blog/stack-canaries-gingerly-sidestepping-the-cage/.

[7] Sharon Lin. *Useful Registers in Assembly. As someone who occasionally… | by Sharon Lin | Medium*. [Online; aufgerufen am 25.12.2024]. Jan. 2019. URL: https://medium.com/@sharonlin/useful-registers-in-assembly-d9a9da22cdd9.

[8] Raúl Rojas. *Mathematik: Wie kommt der Zufall in den PC? - WELT*. [Online; accessed 2024-12-26]. Apr. 2008. URL: https://www.welt.de/wissenschaft/article1924410/Wie-kommt-der-Zufall-in-den-PC.html.

[9] *security - buffer overflow exploit example from "Hacking: The Art of Exploitation Stack Overflow*. [Online; aufgerufen am 27.12.2024]. Mai 2013. URL: https://stackoverflow.com/questions/16781308/buffer-overflow-exploit-example-from-hacking-the-art-of-exploitation.

[10] Sharon Shea. *What is address space layout randomization (ASLR)? | Definition from TechTarget*. [Online; aufgerufen am 26.12.2024]. Juni 2014. URL: https://www.techtarget.com/searchsecurity/definition/address-space-layout-randomization-ASLR.

[11] *SO WIRD'S GEMACHT: Verwalten von Umgebungsvariablen in Windows XP - Microsoft-Support*. [Online; aufgerufen am 26.12.2024]. URL: https://support.microsoft.com/de-de/topic/so-wird-s-gemacht-verwalten-von-umgebungsvariablen-in-windows-xp-5bf6725b-655e-151c-0b55-9a8c9c7f747d.

[12] *Stack frame definition – Glossary | NordVPN*. [Online; aufgerufen am 26.12.2024]. Okt. 2023. URL: https://nordvpn.com/de/cybersecurity/glossary/stack-frame/.

[13] *Stack vs Heap Memory - Simple Explanation - YouTube*. [Online; aufgerufen am 26.12.2024]. Nov. 2022. URL: https://www.youtube.com/watch?v=5OJRqkYbK-4&t=28s.

[14] Christian Stöcker. *Kleine Geschichte der Hackerkultur | Cybersicherheit | bpb.de*. https://www.bpb.de/shop/zeitschriften/apuz/cybersicherheit-2023/521304/kleine-geschichte-der-hackerkultur. Mai 2025.

[15] *What is ICMP (Internet Control Message Protocol)? | Fortinet.* [Online; aufgerufen am 26.12.2024]. URL: https://www.fortinet.com/resources/cyberglossary/internet-control-message-protocol-icmp.

[16] *What is the difference between POP and IMAP? - Microsoft Support.* [Online; aufgerufen am 26.12.2024]. URL: https://support.microsoft.com/en-us/office/what-is-the-difference-between-pop-and-imap-85c0e47f-931d-4035-b409-af3318b194a8.

[17] Contributors to Wikimedia projects. *HTTP - Wikipedia.* [Online; aufgerufen am 26.12.2024]. URL: https://en.wikipedia.org/wiki/HTTP.

[18] Contributors to Wikimedia projects. *OSI model - Wikipedia.* [Online; aufgerufen am 26.12.2024]. URL: https://en.wikipedia.org/wiki/OSI_model.

[19] Autoren der Wikimedia-Projekte. *Bit – Wikipedia.* [Online; aufgerufen am 25.12.2024]. URL: https://de.wikipedia.org/wiki/Bit.

[20] Autoren der Wikimedia-Projekte. *Register (Prozessor) – Wikipedia.* [Online; aufgerufen am 25.12.2024]. URL: https://de.wikipedia.org/wiki/Register_(Prozessor).

[21] Autoren der Wikimedia-Projekte. *Script kiddie - Wikipedia.* [Online; aufgerufen am 25.12.2024]. URL: https://en.wikipedia.org/wiki/Script_kiddie.

[22] Autoren der Wikimedia-Projekte. *Socket – Wikipedia.* [Online; aufgerufen am 26.12.2024]. URL: https://de.wikipedia.org/wiki/Socket.

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

ARP Address Resolution Protocol. 15

ASLR Adress Space Layout Randomization. 18, 21

Bit binary digit. 8, 9, 15, 23

CPU Central Processing Unit. 8

DNS Domain Name System. 16

EBP Extended Base Pointer. 8, 14

EIP Extended Instruction Pointer. 8, 13

ESP Extended Stack Pointer. 8

HDD Hard Disk Drive. 9

HTTP Hypertext Transfer Protocol. 15

HTTPS Hypertext Transfer Protocol Secure. 15

ICMP Internet Control Message Protocol. 15

IP Internet Protocol. 14–16

ISO International Organization for Standardization. 14

MAC Media Access Control. 15

MIT Massachusetts Institute of Technology. 6

NOP no-operation. 18

OSI Open Systems Interconnection. 14

POP3 Post Office Protocol. 16

RAM Random Access Memory. 8, 9, 17, 18, 21

SFP Saved Frame Pointer. 8

SSD Solid-State Drive. 9

TCP Transmission Control Protocol. 14–16

UDP User Datagramm Protocol. 15, 16

WWW World Wide Web. 14, 15