Hacking (in) Games

Protecting your Games and your Gamers

Michael Schwarz, Daniel Gruss

September 1, 2018

IAIK – Graz University of Technology
Michael Schwarz
PhD student @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at
Daniel Gruss
PostDoc @ Graz University of Technology
@lavados
daniel.gruss@iaik.tugraz.at
Hacking (in) the game industry is as old as the game industry.
• Hacking (in) the game industry is as old as the game industry
• Trainers (“cheat programs”) appeared in 1981
Hacking (in) the game industry is as old as the game industry

- Trainers ("cheat programs") appeared in 1981
  → Exploit insecure implementations of games
Hacking (in) the game industrie is as old as the game industry

- **Trainers** ("cheat programs") appeared in 1981

  → Exploit insecure implementations of games

- Not only cheating
• Hacking (in) the game industry is as old as the game industry
• Trainers (“cheat programs”) appeared in 1981
  → Exploit insecure implementations of games
• Not only cheating → piracy
Hacking (in) the game industry is as old as the game industry

- Trainers (“cheat programs”) appeared in 1981
  → Exploit insecure implementations of games
- Not only cheating → piracy
- Bugs in games allowed subverting gaming console security
Piracy
Piracy

Piracy. /ˈpɪərəsi/ noun.
the unauthorized use or reproduction of another’s work.
“software piracy”
Piracy

Piracy. /ˈpɪərəsi/ noun.
the unauthorized use or reproduction of another’s work.
“software piracy”
synonyms: illegal reproduction, plagiarism, illegal copying, copyright infringement, bootlegging, stealing, theft

Source: Google
Gaming consoles are usually closed systems.
• Gaming consoles are usually closed systems
• No general purpose systems as PCs
Gaming consoles are usually closed systems
- No general purpose systems as PCs
  → Only certified, original games can run
Gaming Consoles

- Gaming consoles are usually closed systems.
- No general purpose systems as PCs.
  - Only certified, original games can run.
- Prevents illegal copies of games.
Goal of many console hackers: run own programs
• Goal of many console hackers: run own programs
→ Homebrew
Goal of many console hackers: run own programs

→ Homebrew

- Often a small step from homebrew to pirated games
Hacking Gaming Consoles

- Goal of many console hackers: run own programs
  → Homebrew
- Often a small step from homebrew to pirated games
- Arbitrary programs can be executed

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
Goal of many console hackers: run own programs
→ Homebrew
Often a small step from homebrew to pirated games
Arbitrary programs can be executed
→ Patched games as well
PIRACY? LET'S CALL IT

"HOMEBREW"
• Running homebrew is a *cat-and-mouse* game
• Running homebrew is a cat-and-mouse game
• Hackers exploit bugs to run code
Running homebrew is a cat-and-mouse game

Hackers exploit bugs to run code

Bugs get fixed in firmware update
Running homebrew is a cat-and-mouse game
Hackers exploit bugs to run code
Bugs get fixed in firmware update
Repeat
• Running homebrew is a cat-and-mouse game
• Hackers exploit bugs to run code
• Bugs get fixed in firmware update
• Repeat
• Side effect: platform and games become more secure
Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in *The Legend of Zelda: Twilight Princess*).
- PS2 (buffer overflow in the BIOS)
- PS3 (heap overflow)
- Xbox (buffer overflow in savegames)
- PSP (buffer overflow in image library and savegame)
Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in \textit{The Legend of Zelda: Twilight Princess})
- PS2 (buffer overflow in the BIOS)
- PS3 (heap overflow)
- Xbox (buffer overflow in savegames)
- PSP (buffer overflow in image library and savegame)
Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in *The Legend of Zelda: Twilight Princess*).
- PS2 (buffer overflow in the BIOS)
- PS3 (heap overflow)
- Xbox (buffer overflow in savegames)
- PSP (buffer overflow in image library and savegame)
Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in *The Legend of Zelda: Twilight Princess*).
- PS2 (buffer overflow in the BIOS)
Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in *The Legend of Zelda: Twilight Princess*).
- PS2 (buffer overflow in the BIOS)
- PS3 (heap overflow)
Bugs for Piracy

Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in *The Legend of Zelda: Twilight Princess*).
- PS2 (buffer overflow in the BIOS)
- PS3 (heap overflow)
- Xbox (buffer overflow in savegames)
Jailbreaks (e.g., getting root) on various devices:

- iOS (multiple exploits)
- Wii (buffer overflow in *The Legend of Zelda: Twilight Princess*).
- PS2 (buffer overflow in the BIOS)
- PS3 (heap overflow)
- Xbox (buffer overflow in savegames)
- PSP (buffer overflow in image library and savegame)
- A **buffer** is a chunk of memory...
• A buffer is a chunk of memory...
  • with boundaries
A buffer is a chunk of memory...
- with boundaries
- defined by a start address and size
• A buffer is a chunk of memory...
  • with boundaries
  • defined by a start address and size
  • storing elements of a certain type
• A buffer is a chunk of memory...
  • with boundaries
  • defined by a start address and size
  • storing elements of a certain type

• Example: Arrays in C/C++

```c
char buffer[12];
strcpy(buffer, "Hello");
```
• Not all buffers check their bounds
- Not all buffers check their bounds
- Out-of-bounds reads/writes access *something*
• Not all buffers check their bounds
• Out-of-bounds reads/writes access *something*
• Most commonly: array index out of bounds
Buffer Overflow

- Not all buffers check their bounds
- Out-of-bounds reads/writes access *something*
- Most commonly: array index out of bounds
- Example: Buffer overflow in C/C++

```c
char buffer[4];
strcpy(buffer, "Hello");
```
1972 First documentation of buffer overflows
1972 First documentation of buffer overflows
1988 Morris Worm (aka “The Internet Worm”)

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
1972  First documentation of buffer overflows
1988  Morris Worm (aka “The Internet Worm”) 
1996  AlephOne’s Phrack article
      “Smashing the Stack for Fun and Profit”
1972 First documentation of buffer overflows
1988 Morris Worm (aka “The Internet Worm”)
1996 AlephOne’s Phrack article
   “Smashing the Stack for Fun and Profit”
1998 DilDog’s tutorial “The Tao of Windows Buffer Overruns”
1972 First documentation of buffer overflows
1988 Morris Worm (aka “The Internet Worm”)
1996 AlephOne’s Phrack article
   “Smashing the Stack for Fun and Profit”
1998 DilDog’s tutorial “The Tao of Windows Buffer Overruns”
2000 Buffer overflows are “Bug of the decade” (beating Y2K bug)
1972  First documentation of buffer overflows
1988  Morris Worm (aka “The Internet Worm”)
1996  AlephOne’s Phrack article
       “Smashing the Stack for Fun and Profit”
1998  DilDog’s tutorial “The Tao of Windows Buffer Overruns”
2000  Buffer overflows are “Bug of the decade” (beating Y2K bug)
2001  Halvar Flake predicted heap overflows to be the next wave
1972 First documentation of buffer overflows
1988 Morris Worm (aka “The Internet Worm”)
1996 AlephOne’s Phrack article
   “Smashing the Stack for Fun and Profit”
1998 DilDog’s tutorial “The Tao of Windows Buffer Overruns”
2000 Buffer overflows are “Bug of the decade” (beating Y2K bug)
2001 Halvar Flake predicted heap overflows to be the next wave
2002 Slapper infected Apache webservers using heap overflows
1972 First documentation of buffer overflows
1988 Morris Worm (aka “The Internet Worm”)
1996 AlephOne’s Phrack article “Smashing the Stack for Fun and Profit”
1998 DilDog’s tutorial “The Tao of Windows Buffer Overruns”
2000 Buffer overflows are “Bug of the decade” (beating Y2K bug)
2001 Halvar Flake predicted heap overflows to be the next wave
2002 Slapper infected Apache webservers using heap overflows
2003 Buffer overflows in Xbox games used to run unlicensed software
Buffer overflows are very common

![Graph showing the increase in CWE-119 (Buffer Errors) from 2002 to 2016 for stack and heap vulnerabilities.]
• Local buffers are on the stack
• Local buffers are on the stack
• What is next to the buffer?
- Local buffers are on the stack
- What is next to the buffer?
  - Other variables
  - Function parameters
  - Saved return addresses
Buffer Overflow (Stack)

- **Local** buffers are on the stack
- What is next to the buffer?
  - Other variables
  - Function parameters
  - Saved return addresses
- Attacker controls the buffer input, overwrites this data
Buffer Overflow (Stack)

- Local buffers are on the stack
- What is next to the buffer?
  - Other variables
  - Function parameters
  - Saved return addresses
- Attacker controls the buffer input, overwrites this data
- Changes control flow or manipulates data
Practical Example: Stack Overflow
```c
#include <stdio.h>
#include <string.h>

void printName(char* buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
```
% gdb --args ./hello Students
(gdb) r
Starting program: /home/hello Students
Hello Students
[Inferior 1 (process 21312) exited normally]
% gdb --args ./hello Students
(gdb) r
Starting program: /home/hello Students
Hello Students
[Inferior 1 (process 21312) exited normally]

% gdb --args ./hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
(gdb) r
Starting program: /home/hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
Practical Example Analysis: Stack Overflow
#include <stdio.h>
#include <string.h>

void printName(char* buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
# include <stdio.h>
# include <string.h>

void printName(char* buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
```c
#include <stdio.h>
#include <string.h>

void printName(char * buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
```
#include <stdio.h>
#include <string.h>

void printName(char* buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
```c
#include <stdio.h>
#include <string.h>

void printName(char* buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
```

```plaintext
EIP 0x41414141
EBP 0x41414141
name
```

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
# include <stdio.h>
#include <string.h>

void printName(char* buffer) {
    char name[16];
    strcpy(name, buffer);
    printf("Hello %s\n", name);
}

int main(int argc, char* argv[]) {
    if(argc > 1) printName(argv[1]);
    return 0;
}
Practical Example Impact: Stack Overflow
• Attacker can jump to arbitrary location in memory
• Attacker can jump to arbitrary location in memory
• Every function that is mapped in the address space can be executed
• Attacker can jump to arbitrary location in memory
• Every function that is mapped in the address space can be executed
• Attacker has effectively **full control** over the program
Pokémon Red

- Released 1998 (USA)/1999 (Europe) for Game Boy

Completely reverse engineered: [https://github.com/pret/pokered](https://github.com/pret/pokered)

Source code analysis reveals many bugs...

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
Pokémon Red

- Released 1998 (USA)/1999 (Europe) for Game Boy
- Among the best games of all time (according to IGN)

**Pokémon Red**

©'95-'96. 98 GAME FREAK inc.
Pokémon Red

- Released 1998 (USA)/1999 (Europe) for Game Boy
- Among the best games of all time (according to IGN)
- Sad that they never produced any sequel (or similar)

Completely reverse engineered: https://github.com/pret/pokered

Source code analysis reveals many bugs...
Pokémon Red

- Released 1998 (USA)/1999 (Europe) for Game Boy
- Among the best games of all time (according to IGN)
- Sad that they never produced any sequel (or similar)
- Completely reverse engineered: https://github.com/pret/pokered
- Source code analysis reveals many bugs...
Pokémon Centers have a so-called **Cable Club**

- Connecting two Game Boys with a Link cable allows to exchange Pokémon during the exchange → cloning Pokémon.
- Related idea: modify Pokémon during exchange ([source](http://www.adanscotney.com/2014/01/spoofing-pokemon-trades-with-stellaris.html))
- Maybe there is something more interesting?
Pokémon Centers have a so-called **Cable Club**

Connecting two Game Boys with a Link cable allows to exchange Pokémon in the Cable Club

*As a kid: Pull cable out during exchange → cloning Pokémon*


*Maybe there is something more interesting?*
Pokémon Centers have a so-called Cable Club.

Connecting two Game Boys with a Link cable allows to exchange Pokémon in the Cable Club.

As a kid: Pull cable out during exchange → cloning Pokémon.


Maybe there is something more interesting?
- Pokémon Centers have a so-called **Cable Club**
- Connecting two Game Boys with a Link cable allows to exchange Pokémons in the Cable Club
- As a kid: Pull cable out during exchange → cloning Pokémon
- Related idea: **modify Pokémons** during exchange

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
Pokémon Centers have a so-called **Cable Club**

Connecting two Game Boys with a Link cable allows to exchange Pokémons in the Cable Club

As a kid: Pull cable out during exchange → cloning Pokémon


Maybe there is something more interesting?
Pokémon List

- Pokémon list is terminated with special character
Pokémon List

- Pokémon list is terminated with special character
- What if we do not terminate the list?

void TradeCenter_PrintPartyListNames (char *start_ids, char *dest) {
    char *current_id = start_ids;
    while (*current_id != 0xFF) {
        char *name_str = GetMonName(*current_id);
        PlaceString(name_str, dest);
        dest += 20;
    }
}

Buffer overflow with Pokémon names into stack
Pokémon List

- Pokémon list is terminated with special character
- What if we do **not terminate** the list?
- Names overflow and game crashes
Pokémon List

- Pokémon list is terminated with special character
- What if we do not terminate the list?
- Names overflow and game crashes
- ```c
void TradeCenter_PrintPartyListNames(char* start_ids, char* dest) {
    char* current_id = start_ids;
    while (*current_id != 0xFF) {
        char* name_str = GetMonName(*current_id);
        PlaceString(name_str, dest);
        dest += 20;
    }
}
```
Pokémon List

- Pokémon list is terminated with special character
- What if we do not terminate the list?
- Names overflow and game crashes
- ```
void TradeCenter_PrintPartyListNames(char* start_ids, char* dest) {
    char* current_id = start_ids;
    while (*current_id != 0xFF) {
        char* name_str = GetMonName(*current_id);
        PlaceString(name_str, dest);
        dest += 20;
    }
}
```
- Buffer overflow with Pokémon names into stack
Live Demo

Running Self-Compiled Code in Pokémon Red
- We can run any code $\leq 192$ bytes
• We can run any code $\leq 192$ bytes
• For larger programs: use Link cable to request more data
- We can run any code \( \leq 192 \) bytes
- For larger programs: use Link cable to request more data
- Also works on real Game Boy hardware (©vaguilar)
- “Only” a simple buffer overflow
Consequence

• “Only” a simple buffer overflow
• Allows running arbitrary code
Consequence

- “Only” a simple buffer overflow
- Allows running arbitrary code

→ “Homebrew” on Game Boy
Consequence

- “Only” a simple buffer overflow
- Allows running arbitrary code
- “Homebrew” on Game Boy
- Could be used to run pirated games
• “Only” a simple buffer overflow
• Allows running arbitrary code
→ “Homebrew” on Game Boy
• Could be used to run pirated games
• (Non-existing) Security completely subverted
Countermeasures

- Binary hardening
Countermeasures

- Binary **hardening**
  - Compiler flags
  - Dedicated tools (e.g., static code analyzer)
Countermeasures

• Binary hardening
  • Compiler flags
  • Dedicated tools (e.g., static code analyzer)
• Use existing libraries
Countermeasures

- Binary **hardening**
  - Compiler flags
  - Dedicated tools (e.g., static code analyzer)
- Use existing **libraries**
  - Well-tested
  - Smaller chance of bugs
Countermeasures

- Binary **hardening**
  - Compiler flags
  - Dedicated tools (e.g., static code analyzer)
- Use existing **libraries**
  - Well-tested
  - Smaller chance of bugs
- **Don’t trust** user input

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
Countermeasures

- Binary **hardening**
  - Compiler flags
  - Dedicated tools (e.g., static code analyzer)

- Use existing **libraries**
  - Well-tested
  - Smaller chance of bugs

- **Don’t trust** user input
  - Sanitize everything
  - Use libraries for user input
Countermeasures

- **Binary hardening**
  - Compiler flags
  - Dedicated tools (e.g., static code analyzer)
- **Use existing libraries**
  - Well-tested
  - Smaller chance of bugs
- **Don’t trust user input**
  - Sanitize everything
  - Use libraries for user input
- **Test (a lot) → automated tests**
Things to keep in mind

- User input is evil
Things to keep in mind

- User input is evil
- Whitelisting is more secure than blacklisting
Things to keep in mind

- User input is evil
- Whitelisting is more secure than blacklisting
- Input can be indirect → network packets, files
I DON'T ALWAYS TRUST
But when I trust, I verify.
Cheating
Cheating

cheat. /tʃiːt/ verb.

- gerund or present participle: cheating.
- act dishonestly or unfairly in order to gain an advantage.

Source: Google
Cheating was always part of video games.
Cheating was always part of video games

Developers implemented cheat codes for testing
• Cheating was always part of video games
• Developers implemented cheat codes for testing
• 1981 cheat programs (aka trainers) appeared
- Cheating was always part of video games
- Developers implemented cheat codes for testing
- 1981 cheat programs (aka trainers) appeared
- Modified game before launching
• Everything is stored in memory
• Everything is stored in memory

→ Health, money, stats...
• Everything is stored in memory
→ Health, money, stats...
• Trainers modify these values
• Everything is stored in memory
  → Health, money, stats...
• Trainers modify these values
• Gives player (unfair) advantages
• There are different formats for storing numbers
- There are different formats for storing numbers
- **Binary** for unsigned integers, only positive numbers
There are different formats for storing numbers:
- **Binary** for unsigned integers, only positive numbers
- **Two’s complement** for signed integers, positive and negative
There are different formats for storing numbers

- **Binary** for unsigned integers, only positive numbers
- **Two’s complement** for signed integers, positive and negative
- **Sign bit + Magnitude** for floating point numbers
• An $n$-bit integer $x$ is represented as

$$x = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0) = \sum_{i=0}^{n-1} 2^i \cdot x_i$$

• The range of representable values is

$$0 \leq x < 2^n$$

• On overflow, the value is reduced modulo $2^n$

$$x = \begin{cases} x & x < 2^n \\ x \mod 2^n & x \geq 2^n \end{cases}$$
• An $n$-bit integer $x$ is represented as

$$x = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0) = -2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i \cdot x_i$$

• The range of representable values is

$$-2^{n-1} \leq x < 2^{n-1}$$
• A single-precision (IEEE 754-2008) float $x$ is represented as

$$x = (x_{31}, x_{30}, \ldots, x_1, x_0)$$

$$= (-1)^{x_{31}} \cdot \left(1 + \sum_{i=1}^{23} x_{23-i}2^{-i}\right) \cdot 2^{([x_{30} : x_{23}] - 127)}$$

• A single-precision float can encode numbers up to

$$\approx 3.4 \times 10^{38}$$

• All integers with $\leq 6$ decimal digits can be encoded

• All values $2^n$ with $-126 \leq n \leq 127$ can be encoded
Example: $x = 3.3125$

- Normalize to 1.
  - $\times 2^{1} = 1.10101_{b} \times 2^{1}$
- Sign bit: 0 as it is positive
- Exponent: $e + 127 = 1 + 127 = 128$
- Fraction: $0.10101_{b} \times 2^{23}$
  - $0.10101_{b} \times 2^{23} = 0.65625 \times 2^{23} = 5505024$
- Result: $01000000010101000000000000000000_{b}$

```c
int i = 0b01000000010101000000000000000000;
float f = *(float *)&i;
printf(" %.4f
", f);
```

// prints 3.3125
• Example: \( x = 3.3125 = 11.0101_b \)
Example: \( x = 3.3125 = 11.0101_b \)

- Normalize to \( 1.bbb \times 2^e \)

```c
int i = 0b01000000010101000000000000000000;
float f = *(float *)&i;
printf("%.4f\n", f);  // prints 3.3125
```
• Example: $x = 3.3125 = 11.0101_b$

• Normalize to $1.bbb \times 2^e = 1.10101_b \times 2^{1}$
Example: \( x = 3.3125 = 11.0101_b \)

- Normalize to \( 1.bbb \times 2^e = 1.10101_b \times 2^1 \)
- Sign bit: 0 as it is positive

```c
int i = 0b01000000010101000000000000000000;
float f = *( float *)&i;
printf (" %.4f\n", f);
// prints 3.3125
```
Example: $x = 3.3125 = 11.0101_b$

- Normalize to $1.bbb \times 2^e = 1.10101_b \times 2^1$
- Sign bit: 0 as it is positive
- Exponent: $e + 127$
Example: $x = 3.3125 = 11.0101_b$

- Normalize to $1.00101 \times 2^1 = 1.10101_b \times 2^1$
- Sign bit: 0 as it is positive
- Exponent: $e + 127 = 1 + 127 = 128$

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
Example: \( x = 3.3125 = 11.0101_b \)

- Normalize to \( 1.bbb \times 2^e = 1.10101_b \times 2^1 \)
- Sign bit: 0 as it is positive
- Exponent: \( e + 127 = 1 + 127 = 128 \)
- Fraction: \( 0.bbb \times 2^{23} \)
- Example: $x = 3.3125 = 11.0101\_b$
- Normalize to $1.bbb \times 2^e = 1.10101\_b \times 2^1$
- Sign bit: 0 as it is positive
- Exponent: $e + 127 = 1 + 127 = 128$
- Fraction: $0.bbb \times 2^{23}$
  - $= 0.10101\_b \times 2^{23} = 0.65625 \times 2^{23} = 5505024$
Example: \( x = 3.3125 = 11.0101_2 \)

- Normalize to \( 1.\underline{bbb} \times 2^e = 1.10101_2 \times 2^1 \)
- Sign bit: 0 as it is positive
- Exponent: \( e + 127 = 1 + 127 = 128 \)
- Fraction: \( 0.\underline{bbb} \times 2^{23} \)
  \[ = 0.10101_2 \times 2^{23} = 0.65625 \times 2^{23} = 5505024 \]
- Result: \( 01000000010101000000000000000000_2 \)
• Example: $x = 3.3125 = 11.0101_2$

• Normalize to $1.bbb \times 2^e = 1.10101_2 \times 2^1$

• Sign bit: 0 as it is positive

• Exponent: $e + 127 = 1 + 127 = 128$

• Fraction: $0.bbb \times 2^{23}$
  \[= 0.10101_2 \times 2^{23} = 0.65625 \times 2^{23} = 5505024\]

• Result: $01000000010101000000000000000000_2$

```c
int i = 0b01000000010101000000000000000000;
float f = *(float *)&i;
printf("%.4f\n", f); // prints 3.3125
```
- Patterns can be searched in memory
• Patterns can be searched in memory
• Dedicated tools (e.g., ArtMoney, Cheat Engine)
• Patterns can be searched in memory
• Dedicated tools (e.g., ArtMoney, Cheat Engine)
• Works for many games
Memory Hacking

- Patterns can be searched in memory
- Dedicated tools (e.g., ArtMoney, Cheat Engine)
- Works for many games
- Easy to use for users
Patterns can be searched in memory
Dedicated tools (e.g., ArtMoney, Cheat Engine)
Works for many games
Easy to use for users
Found values can be changed or frozen
Hacking Candy Crush
WHAT YEAR IS IT?
Do not trust user data!
Do not trust user data!

- Store **important** values (e.g., money) on **server**
Do not trust user data!

- Store important values (e.g., money) on **server**
- Calculate values on **server** based on user action
Do not trust user data!

- Store important values (e.g., money) on server
- Calculate values on server based on user action
- Use checksums for data
Do not trust user data!

- Store **important** values (e.g., money) on **server**
- Calculate values on server based on user action
- Use **checksums** for data
- Encrypt values in memory (simple XOR is already a lot harder to hack)
Game is split into protected and unprotected part
● Game is split into protected and unprotected part
● Unprotected part for logic, rendering, input, ...
• Game is split into **protected** and **unprotected** part
• Unprotected part for logic, rendering, input, ...
  → has to be **fast**, data is **not relevant** for cheating
• Game is split into protected and unprotected part
• Unprotected part for logic, rendering, input, ...
→ has to be fast, data is not relevant for cheating
• Protected part for important values (e.g., health, money, weapons, ...)

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
- Game is split into **protected** and **unprotected part**
- Unprotected part for logic, rendering, input, ...
  → has to be **fast**, data is **not relevant** for cheating
- Protected part for important values (e.g., health, money, weapons, ...)
  → has to be **secure**, data is **relevant** for cheating
Values are encrypted
• Values are encrypted
• Memory is readonly
• Values are encrypted
• Memory is readonly
→ Simply overwriting values is not possible
Anti Memory Hacking - GTA IV

- Values are encrypted
- Memory is read-only
  → Simply overwriting values is not possible
- Values can only be changed through certain functions
Values are encrypted
Memory is readonly
→ Simply overwriting values is not possible
Values can only be changed through certain functions
These functions also require keys/secrets
• Values are not only in memory
• Values are not only in memory
• Savegames store important values
• Values are not only in memory
• **Savegames** store important values
• Similar method as memory hacking
Values are not only in memory
Savegames store important values
Similar method as memory hacking
Modifying savegames is often even easier
SimCity 2000 (1993)
CHAPTER 10: TIPS AND TRICKS

This byte is in Hexadecimal form
This is the ASCII or character equivalent of the Hex code for the byte located at the left

Figure 10-5
The Scenario data structure for Dullsville as viewed from within Norton Utilities

Photos: https://twitter.com/0xabad1dea/status/1021813237032132608

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
SimCity 2000 (1993)

Photos: https://twitter.com/0xabad1dea/status/1021813237032132608

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
SimCity 2000 (1993)

Next, you change the **00** (located before the fourth **02** hex marker in your city file) to a **7F**. This adds $2,139,029,504 to your city’s coffers! The new byte string will look like this, assuming you have $20,000 (hex 4E 20) to begin with:

\[ 7F \quad 02 \quad 4E \quad 20 \]

Change this byte from **00** to **7F**.

The hex-editing procedures outlined below are to be used only on saved cities with the file extension .SC2 that you have previously created on an IBM PC or compatible. *You can’t use this technique on the scenario files. (See the subdirectory of your SC2000 directory.)*

Photos: https://twitter.com/0xabad1dea/status/1021813237032132608

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
SimCity Build It (2014)

Do you think this still works 21 years later?
SimCity Build It (2014)

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
SimCity Build It (2014)

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
SAVEGAME HACKS?

WHAT YEAR IS IT?
Savegame hacking is similar to memory hacking.
Savegame hacking is similar to memory hacking

→ Similar countermeasures
Savegame hacking is similar to memory hacking

→ Similar countermeasures

• Store important values on server
Savegame hacking is similar to memory hacking

→ Similar countermeasures

• Store important values on server

• Add checksums
Savegame hacking is similar to memory hacking

→ Similar countermeasures

• Store important values on server

• Add checksums

• Encrypt savegame
The User Controls the Environment
• User can modify binary...
• User can modify binary...
• ...but also the environment
User can modify binary...
...but also the environment

→ Files, network packets
• User can modify binary...
• ...but also the environment
  → Files, network packets
• And also functions → hooking/preloading
• Microsoft provides Detours framework
  ➔ https://github.com/Microsoft/Detours
Microsoft provides Detours framework

https://github.com/Microsoft/Detours

Replace any Windows function with custom function
Microsoft Detours

- Microsoft provides Detours framework
  - https://github.com/Microsoft/Detours
- Replace any Windows function with custom function
  → Filter functions or function arguments
Microsoft Detours

- Microsoft provides Detours framework
  - https://github.com/Microsoft/Detours
- Replace any Windows function with custom function
  - Filter functions or function arguments
- Example: redirect internet connection to own local server
• LD_PRELOAD is used by the dynamic linker/loader
• LD_PRELOAD is used by the dynamic linker/loader
• Contains one or more ELF shared object files
• LD_PRELOAD is used by the dynamic linker/loader
• Contains one or more ELF shared object files
• Object files are loaded before anything else
● LD_PRELOAD is used by the dynamic linker/loader
● Contains one or more ELF shared object files
● Object files are loaded before anything else
● Overwrites functions in other shared libraries
Fun Example: LD_PRELOAD
```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char* argv[]) {
    char buffer[32];
    strcpy(buffer, "ultra secret password");
    if(getuid() == 0) {
        printf("Password: %s\n", buffer);
    } else {
        printf("Only root can get the password\n");
    }
}
```
% ./secret
Only root can get the password
% ./secret
Only root can get the password

#include <stdio.h>

char *strcpy(char *dest, const char *src) {
    printf("Copy: %s\n", src);
    while((*dest++ = *src++));
}

% gcc -shared -fPIC strcpy.c -o strcpy.so
% ./secret
Only root can get the password

#include <stdio.h>
char *strcpy(char *dest, const char *src) {
    printf("Copy: %s\n", src);
    while((*dest++ = *src++));
}

% gcc -shared -fPIC strcpy.c -o strcpy.so
% LD_PRELOAD=$PWD/strcpy.so ./secret
Copy: ultra secret password
Only root can get the password
Live Demo

Cheating in Tetris with LD_PRELOAD
• Cheating might be okay in single-player games
Cheating

- Cheating might be okay in single-player games
- Problematic for multi-player games
• Cheating might be okay in single-player games
• Problematic for multi-player games
→ Gamers get frustrated
Cheating might be okay in single-player games

Problematic for multi-player games

→ Gamers get frustrated

Problematic if in-game purchases are cheated
Cheating might be okay in single-player games

- Problematic for multi-player games
  - Gamers get frustrated

- Problematic if in-game purchases are cheated
  - Potential monetary loss
Preventing cheating is nearly impossible
• Preventing cheating is nearly impossible
• Some ways to make it harder
• Preventing cheating is nearly impossible
• Some ways to make it harder
  • Store important info on server
Cheating Prevention

- Preventing cheating is nearly impossible
- Some ways to make it harder
  - Store important info on server
  - Add plausibility checks
• Preventing cheating is nearly impossible
• Some ways to make it harder
  • Store important info on server
  • Add plausibility checks
  • Use cryptography
• Preventing cheating is nearly impossible
• Some ways to make it harder
  • Store important info on server
  • Add plausibility checks
  • Use cryptography
• Detecting and banning users sometimes easier
• Some hackers do all this for fun
• Some hackers do all this for fun
• Many also want to profit
What else?

- Some hackers do all this for fun
- Many also want to profit
- Piracy is (usually) profitable
What else?

- Some hackers do all this for fun
- Many also want to profit
- Piracy is (usually) profitable
- Cheating can be profitable
Phase 1  Phase 2  Phase 3
PHASE 1  PHASE 2  PHASE 3

Hack Game
PHASE 1  PHASE 2  PHASE 3

Hack
Game
Phase 1: Hack
Game

Phase 2: ?

Phase 3: Profit
Fraud
Fraud

Fraud. /frɔːd/ noun.

wrongful or criminal deception intended to result in financial or personal gain.

Source: Google
• There is a lot of money in the game industry
• Gamers pay for/in online games
- Gamers pay for/in online games
- User information stored on servers
Online Payment

- Gamers pay for/in online games
- User information stored on servers
- Personal information
Online Payment

- Gamers pay for/in online games
- User information stored on servers
- Personal information
- Even credit card data
PlayStation Network data breach (2011)
• PlayStation Network data breach (2011)
• 77 million accounts exposed
• PlayStation Network data breach (2011)
• 77 million accounts exposed
• 12.3 million credit cards
PlayStation Network data breach (2011)
- 77 million accounts exposed
- 12.3 million credit cards
- No evidence that credit card data was stolen
• PlayStation Network data breach (2011)
• 77 million accounts exposed
• 12.3 million credit cards
• No evidence that credit card data was stolen
• Still some reports of fraud
• Credit card info not the only target
• Credit card info not the only target
• Personal info can be sold on black market
Credit card info not the only target
- Personal info can be sold on black market
- Depending on data quite valuable
### BRAZILIAN UNDERGROUND

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set of business application account credentials</td>
<td>US$155–193</td>
</tr>
<tr>
<td>Set of credit card credentials</td>
<td>US$35–135</td>
</tr>
<tr>
<td>Set of online service account credentials</td>
<td>US$19</td>
</tr>
<tr>
<td>List of mobile phone numbers</td>
<td>US$290–1,236</td>
</tr>
<tr>
<td>List of landline phone numbers</td>
<td>US$317–1,931</td>
</tr>
</tbody>
</table>
Server backend requires security as well
• Server backend requires security as well
• Also receives user input
Server backend requires security as well
Also receives user input
Sometimes not obvious
• Server **backend** requires **security** as well
• Also receives **user input**
• Sometimes **not obvious**
→ **chats**, **highscore name**, **inventory**, ...
• Online tutorials often without security
Online tutorials often without security

→ Name used for database query
Online tutorials often without security

→ Name used for database query

• Enables SQL injections
Online tutorials often without security
→ Name used for database query
• Enables SQL injections
→ Gamer controls part of query, can exfiltrate data
Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something?

In a way -

Did you really name your son Robert'; drop table Students--;?

Oh, yes. Little Bobby Tables, we call him.

Well, we've lost this year's student records. I hope you're happy.

And I hope you've learned to sanitize your database inputs.

https://xkcd.com/327/
• Securing backend is important
- Securing backend is important
- Does not only protect infrastructure...
• Securing backend is important
• Does not only protect infrastructure...
• ...but also the gamers
Securing backend is important
Does not only protect infrastructure...
...but also the gamers
Private data is valuable and needs good protection
Try the Dark Side
Hacking games sounds like fun?
• Hacking games sounds like fun?
• It is fun!
Hacking games sounds like fun?

It is fun!

Side effect: learn about security
Hacking Games

- Hacking games sounds like fun?
- It is fun!
- Side effect: learn about security
- Where to start?
• 3D, first-person, open-world MMORPG
• 3D, first-person, open-world MMORPG
• Intentionally vulnerable game
● 3D, first-person, open-world MMORPG
● Intentionally **vulnerable** game
● Learning “platform” for game developers
Pwn Adventures

- 3D, first-person, open-world MMORPG
- Intentionally vulnerable game
- Learning “platform” for game developers
- Understand how design and programming flaws create vulnerabilities
Challenges

• Multiple quests in the game
Challenges

- Multiple quests in the game
- Can only be solved by cheating/hacking
Challenges

- Multiple quests in the game
- Can only be solved by cheating/hacking
- Requires multiple techniques
Challenges

- Multiple quests in the game
- Can only be solved by cheating/hacking
- Requires multiple techniques

→ Some covered in this talk, e.g., hooking/preloading
• Intercepting network → teleport
Techniques

- Intercepting network → teleport
- Reverse engineering → hidden elements
- Intercepting network $\rightarrow$ teleport
- Reverse engineering $\rightarrow$ hidden elements
- Hooking $\rightarrow$ speed/gravity hack
Techniques

- Intercepting network → teleport
- Reverse engineering → hidden elements
- Hooking → speed/gravity hack
- ...

Michael Schwarz (@misc0110), Daniel Gruss (@lavados) — IAIK – Graz University of Technology
• This was of course not exhaustive
• This was of course not exhaustive
• Many techniques, tricks, and tools
This was of course not exhaustive
Many techniques, tricks, and tools
Good start:

The Ultimate Game Hacking Resource
https://github.com/dsasmblr/game-hacking/
PLAYING A GAME
PROGRAMMING A GAME
HACKING A GAME
PROGRAMMING A SECURE GAME WITH TECHNIQUES LEARNED FROM HACKING A GAME
Conclusion
• Security in game design is often neglected
• Security in game design is often neglected
• Old hacking techniques still applicable
• Security in game design is often neglected
• Old hacking techniques still applicable
• Vulnerabilities can cost money
• Security in game design is often neglected
• Old hacking techniques still applicable
• Vulnerabilities can cost money
• Hard to retrofit security mechanisms
• Never trust the user (input)
Take Aways

- Never trust the user (input)
- Hacking games is often easy
• Never trust the user (input)
• Hacking games is often easy
• Security should be part of the design process
Take Aways

- Never trust the user (input)
- Hacking games is often easy
- Security should be part of the design process
- Small implementation details make hacking already harder
Thank you!
Hacking (in) Games

Protecting your Games and your Gamers

Michael Schwarz, Daniel Gruss
September 1, 2018

IAIK – Graz University of Technology