Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

SONY PLAYSTATION 3

Ausnutzen von Sicherheitslücken

30. August 2023

Leon Olejar
Timo Christeleit

Studienbereich Design Informatik Medien
Hochschule **RheinMain**

ZIELE

→ Sicherheitsmechanismen der PS3 verstehen

→ Sicherheitslücken finden

→ Exploit entwickeln

→ Spiele modifizieren

AUFBAU SPIELE

GENERELLER AUFBAU

- PS3_DISC.SFB
- PS3_GAME
  - ICON0.PNG
  - PARAM.SFO
  - PIC1.PNG
  - PS3LOGO.DAT
  - USRDIR
    - __dataversion
    - EBOOT.BIN
    - filecachestoragelist.xml
    - icon0.png
    - indexedassets.txt
    - qkl_boot.ast
    - qkl_fe2ig.ast
    - qkl_interface.ast
    - qkl_misc.ast
    - qkl_stream.ast
    - qklstoragelist.xml
    - quickloadbootstoragelist.xml
- PS3_UPDATE

NASCAR 09

- PS3_DISC.SFB
- PS3_GAME
  - ICON0.PNG
  - LICDIR
    - LIC.DAT
  - PARAM.SFO
  - PIC1.PNG
  - PS3LOGO.DAT
  - TROPDIR
    - NPWR00612_00
  - USRDIR
    - BINARIES
      - PS3
      - EBOOT_borderlands.BIN
      - EBOOT.BIN
    - ENGINE
    - WILLOWGAME
- PS3_UPDATE

Borderlands

- PS3_DISC.SFB
- PS3_GAME
  - ICON0.PNG
  - LICDIR
  - PARAM.SFO
  - PIC1.PNG
  - PS3GAME.CHK
  - PS3GAME.INI
  - PS3GAME.VR0
  - PS3LOGO Kopie.DAT
  - PS3LOGO.DAT
  - SND0.AT3
  - TROPDIR
  - USRDIR
    - Common
    - DLC
    - EBOOT.BIN
    - EBOOT.BIN_orig
    - music
    - PS3
- PS3_UPDATE

Minecraft

GENERELLER AUFBAU

- PS3_DISC.SFB
- PS3_GAME
  - ICON0.PNG
  - PARAM.SFO
  - PIC1.PNG
  - PS3LOGO.DAT
  - USRDIR
    - dataversion
    - EBOOT.BIN
    - filecachestoragelist.xml
    - icon0.png
    - indexedassets.txt
    - qkl_boot.ast
    - qkl_fe2ig.ast
    - qkl_interface.ast
    - qkl_misc.ast
    - qkl_stream.ast
    - qklstoragelist.xml
    - quickloadbootstoragelist.xml
- PS3_UPDATE

NASCAR 09

- PS3_DISC.SFB
- PS3_GAME
  - ICON0.PNG
  - LICDIR
    - LIC.DAT
  - PARAM.SFO
  - PIC1.PNG
  - PS3LOGO.DAT
  - TROPDIR
    - NPWR00612_00
  - USRDIR
    - BINARIES
      - PS3
      - EBOOT_borderlands.BIN
      - EBOOT.BIN
    - ENGINE
    - WILLOWGAME
- PS3_UPDATE

Borderlands

- PS3_DISC.SFB
- PS3_GAME
  - ICON0.PNG
  - LICDIR
  - PARAM.SFO
  - PIC1.PNG
  - PS3GAME.CHK
  - PS3GAME.INI
  - PS3GAME.VR0
  - PS3LOGO Kopie.DAT
  - PS3LOGO.DAT
  - SND0.AT3
  - TROPDIR
  - USRDIR
    - Common
    - DLC
    - EBOOT.BIN
    - EBOOT.BIN_orig
    - music
    - PS3
- PS3_UPDATE

Minecraft

WOHER KOMMEN DIE DATEIEN?

Playstation Store:

→ Spiele werden über das Internet auf die Festplatte der PS3 installiert
→ Werden für die konkrete Konsole signiert

Blu-Ray Disc:

→ Spiele werden von der Blue-Ray Disc gestartet
→ Blue-Ray Discs sind verschlüsselt

SELF

EBOOT.BIN ist eine SELF-Datei

SELF

Signed Executable and Linkable Format

1. SELF ist das Dateiformat für ausführbare Dateien auf der PS3
2. SELF-Dateien sind mehrfach verschlüsselt und signiert
3. ELF-Datei entält den Einstiegspunkt für die Ausführung

SELF AUFBAU

Unverschlüsselt:

→ CF-Header: Dateigröße, Version, Kategorie

→ EXT-Header: Strukturdaten, /-Offsets

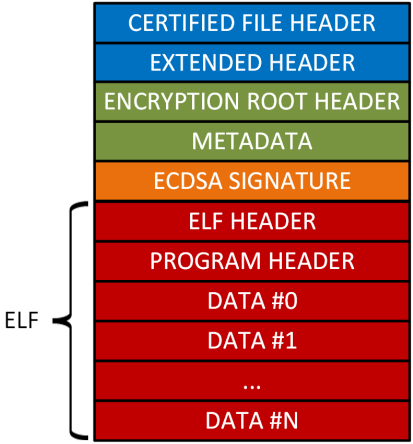
Verschlüsselt mit AES:

→ ENC-Header: Key, IV für Entschlüsselung

→ Metadata: Signatur-Offset, ELF-Hash

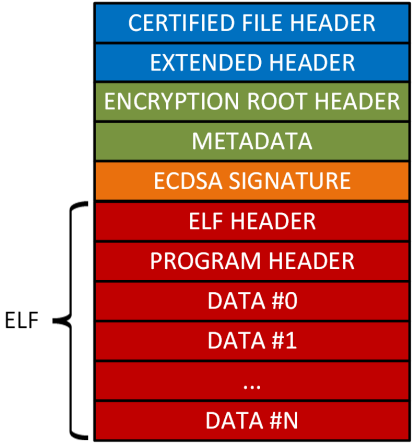
→ ECDSA-Signatur über Header-Struktur

→ ELF-Datei



ELF MODIFIZIEREN

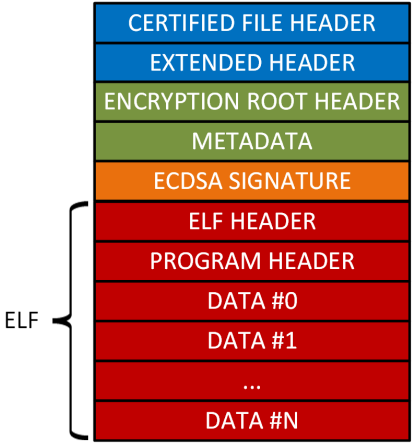
→ EBOOT.BIN entschlüsseln - **Möglich**



ELF MODIFIZIEREN

→ EBOOT.BIN entschlüsseln - **Möglich**

→ ELF entschlüsseln - **Möglich**

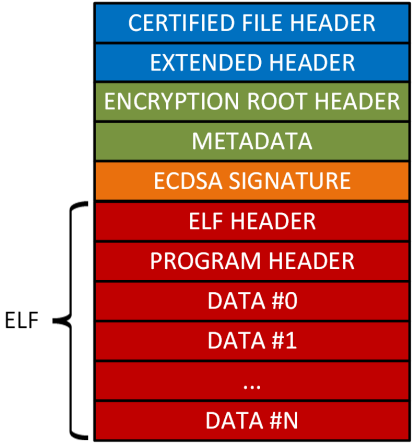


ELF MODIFIZIEREN

→ EBOOT.BIN entschlüsseln - **Möglich**

→ ELF entschlüsseln - **Möglich**

→ ELF Daten modifizieren - **Möglich**



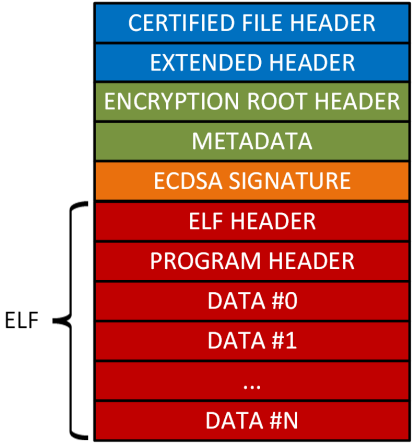
ELF MODIFIZIEREN

→ EBOOT.BIN entschlüsseln - **Möglich**

→ ELF entschlüsseln - **Möglich**

→ ELF Daten modifizieren - **Möglich**

→ Hash über ELF neu berechnen - **Möglich**



ELF MODIFIZIEREN

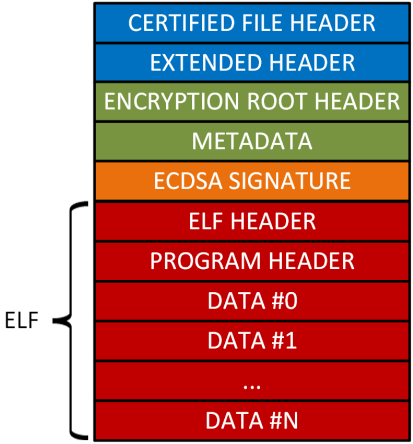
→ EBOOT.BIN entschlüsseln - **Möglich**

→ ELF entschlüsseln - **Möglich**

→ ELF Daten modifizieren - **Möglich**

→ Hash über ELF neu berechnen - **Möglich**

→ ELF mit gleichem Key neu verschlüsseln - **Möglich**



ELF MODIFIZIEREN

→ EBOOT.BIN entschlüsseln - **Möglich**

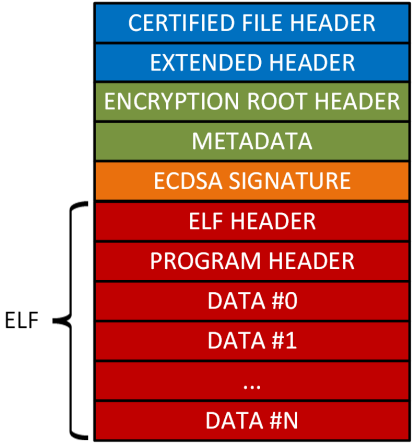
→ ELF entschlüsseln - **Möglich**

→ ELF Daten modifizieren - **Möglich**

→ Hash über ELF neu berechnen - **Möglich**

→ ELF mit gleichem Key neu verschlüsseln - **Möglich**

→ Hash in *Metadata* anpassen - **Möglich**



ELF MODIFIZIEREN

→ EBOOT.BIN entschlüsseln - **Möglich**

→ ELF entschlüsseln - **Möglich**

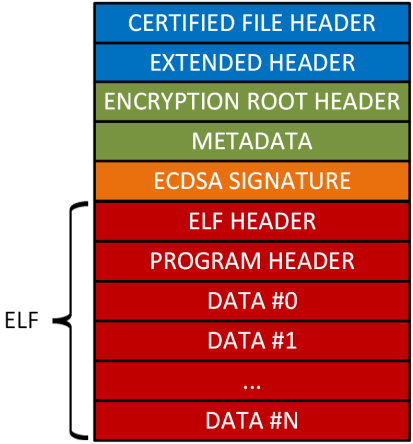
→ ELF Daten modifizieren - **Möglich**

→ Hash über ELF neu berechnen - **Möglich**

→ ELF mit gleichem Key neu verschlüsseln - **Möglich**

→ Hash in *Metadata* anpassen - **Möglich**

→ ECDSA Signatur neu berechnen - **Privater Schlüssel fehlt!**



SICHERHEITSKONZEPT SELF

→ Verschlüsslung der Datei für Obfuscation

→ Hash über ELF-Datei

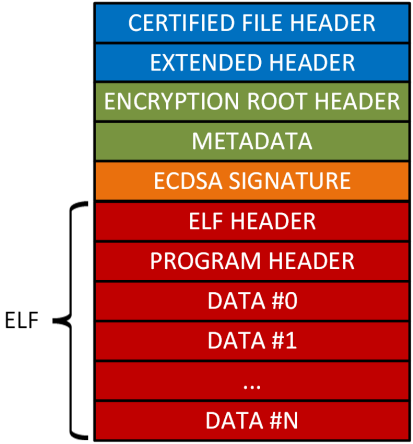
→ ECDSA-Signatur über (kleine) Header-Struktur

→ AES für große Datenmengen

→ ECDSA für kleine Datenmengen

Konzept

Schnelle und *sichere* Code-Integritätsprüfung



SICHERHEITSKONZEPT SELF

→ Verschlüsslung der Datei für Obfuscation

→ Hash über ELF-Datei

→ ECDSA-Signatur über (kleine) Header-Struktur

→ AES für große Datenmengen

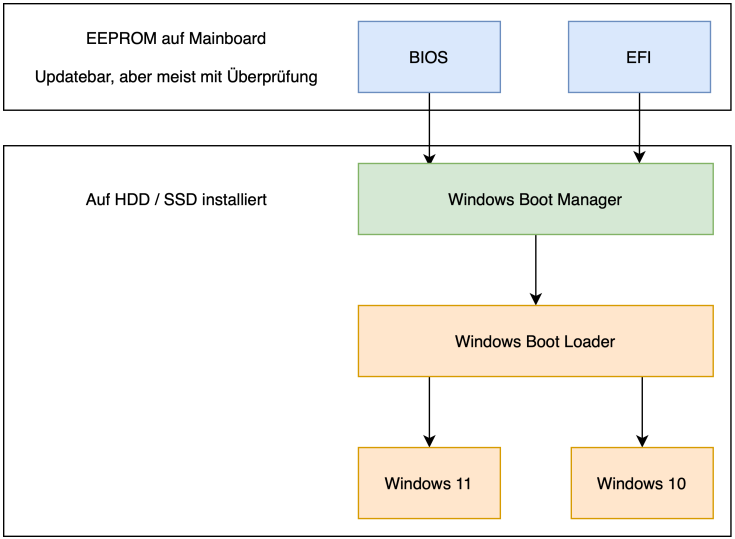
→ ECDSA für kleine Datenmengen

Konzept

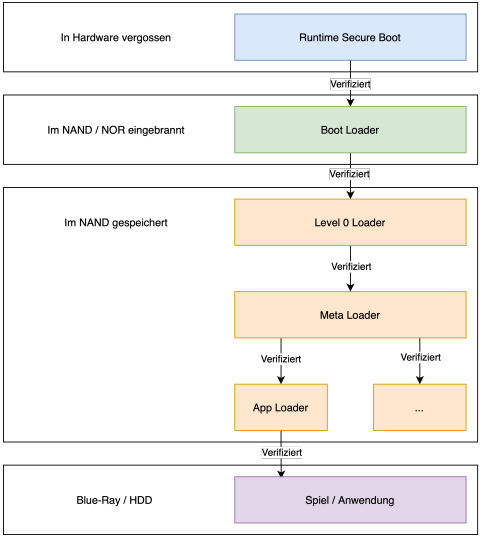
Schnelle und *sichere* Code-Integritätsprüfung

TECHNISCHER AUFBAU

WINDOWS BOOT VORGANG



PS3 BOOT VORGANG



→ Jede Komponente prüft die nächste

→ Basis ist nicht modifizierbar

→ Bootvorgang bei inkonsistenter Komponente abbrechen

PS3 BOOT VORGANG

→ Jede Komponente prüft die nächste

→ Basis ist nicht modifizierbar

→ Bootvorgang bei inkonsistenter Komponente abbrechen

SICHERHEITSMECHANISMEN

WIEDERHOLUNG: ECDSA

ECDSA

Elliptic Curve Digital Signature Algorithm

→ Wähle eine elliptische Kurve $E(p, a, b, q, A)$

→ **Schlüsselgenerierung:**

- → Wähle einen privaten Schlüssel d mit $0 < d < q$
- → Berechne $B = d \cdot A$
- → Setze den öffentlichen Schlüssel $K_{pb} = (p, a, b, q, A, B)$

→ **Signaturerzeugung:**

- → Wähle eine Zufallszahl k_E mit $0 \leqslant k_E \leqslant q$
- → Berechne $R = k_E \cdot A$
- → Setze $r = x_R$
- → Berechne $s = {k_E}^{-1} \cdot (m + d \cdot r) \mod q$

ECDSA IN DER PS3

→ ECDSA-Signaturen befinden sich in jeder SELF-Datei

→ Signatur wird über Header-Struktur gebildet

→ Verwendung von ECDSA160

→ ECDSA wurde **fast** korrekt implementiert

→ Allerdings hat Sony im Prinzip den folgenden Zufallsgenerator verwendet, um k_E zu erzeugen:

IMPLEMENTIERUNGSFEHLER

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

→ **Fehler:** k_E ist nicht zufällig

→ **Folge:** k_E kann aus zwei Signaturen berechnet werden

→ Mit k_E kann der private Schlüssel d berechnet werden

EXPLOIT

VORGEHENSWEISE

→ Zugriff auf EBOOT.BIN von folgenden Spielen verschafft:
  → **NASCAR 09**
  → **Borderlands**

→ Entschlüsselung der Dateien mit Hilfe der Open Source Software **scetool**

→ Extrahierung der Signaturen

→ Berechnung der Hashes über die Header-Struktur

→ Berechnung der „Zufallszahl“ k_E

→ Berechnung des privaten Schlüssels d

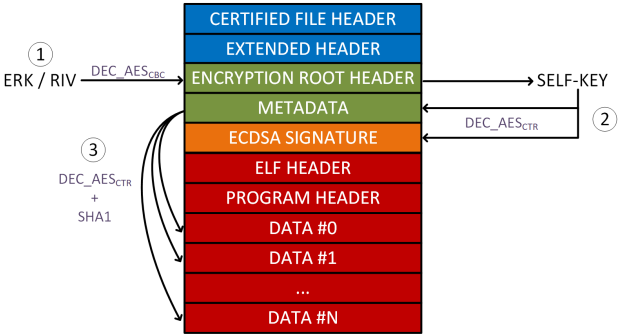
EBOOT.BIN ENTSCHLÜSSELUNG

→ Verwendung von zwei AES-Modi zur Entschlüsselung
- → **CBC** (Cipher Block Chaining Mode)
- → **CTR** (Counter Mode)

1. **Entschlüsseln des Encryption Root Headers:**
   Encryption Round Key (ERK) und Initialisierungs-vektor (RIV) des App Loaders

2. **Entschlüsseln der Metadaten & Signatur:**
   SELF-Key aus Encryption Root Header

3. **Entschlüsseln und Hashen der ELF:**
   Key / IV aus Metadaten und SHA1 Hash über ELF



EXTRAHIEREN DER SIGNATUREN

→ Schreiben eines Parsers für die entschlüsselten EBOOT.BIN Dateien
  → Übersetzt Binärdateien in lesbare Textdateien gemäß SELF-Struktur
  → Ermöglicht Extrahierung der Signaturen

```
                                    Signatur
-------------------------------------------------------------------------------------
r:                          | 00 09 3B DB 96 51 49 3F 5D C1 09 0C C6 24 CF 95 9C 4E 11 CB 53
s:                          | 00 ____________________________________________________ EC
Padding:                    | 00 00 00 00 00 00
```

ECDSA-Signatur: Borderlands

```
                                    Signatur
-------------------------------------------------------------------------------------
r:                          | 00 09 3B DB 96 51 49 3F 5D C1 09 0C C6 24 CF 95 9C 4E 11 CB 53
s:                          | 00 ____________________________________________________ A7
Padding:                    | 00 00 00 00 00 00
```

ECDSA-Signatur: NASCAR 09

EXTRAHIEREN DER SIGNATUREN

→ **Auffällig:** r ist gleich für beide Signaturen
→ **Folge:** k_E ist gleich für beide Signaturen, da $R = k_E \cdot A$ und $r = x_R$

```
                                   Signatur
-------------------------------------------------------------------------------------------
r:                          | 00 09 3B DB 96 51 49 3F 5D C1 09 0C C6 24 CF 95 9C 4E 11 CB 53
s:                          | 00 ____________________________________________________ EC
Padding:                    | 00 00 00 00 00 00
```

ECDSA-Signatur: Borderlands

```
                                   Signatur
-------------------------------------------------------------------------------------------
r:                          | 00 09 3B DB 96 51 49 3F 5D C1 09 0C C6 24 CF 95 9C 4E 11 CB 53
s:                          | 00 ____________________________________________________ A7
Padding:                    | 00 00 00 00 00 00
```

ECDSA-Signatur: NASCAR 09

BERECHNUNG DES HASHES

→ Information in Metadata, welcher Bereich der SELF gehasht wird

→ Diesen Bereich der entschlüsselten SELF-Datei in separate Datei schreiben

→ **Problem:** Big-Endian Format bei PS3, Little-Endian Format bei PC

→ **Folge:** Zu konvertierende Bytes nachvollziehen

→ **Lösung:** scetool kann ELF-Dateien in SELF-Dateien für PS3 umwandeln

- → Bereichskonvertierung von scetool vor Hash-Berechnung nachvollziehen
- → Bereichskonvertierung auf die eigenen SELF-Dateien anwenden

→ Hash-Berechnung mit SHA-1 Algorithmus durchführen

BERECHNUNG DER ZUFALLSZAHL

→ Es sind nun alle Parameter für die Berechnung der Zufallszahl k_E bekannt

BORDERLANDS

s_1 0x[redacted]

m_1 0x[redacted]

r 0x093BDB9651493F5DC1090CC624CF959C4E11CB53

NASCAR 09

s_2 0x[redacted]

m_2 0x[redacted]

r 0x093BDB9651493F5DC1090CC624CF959C4E11CB53

Kurvenparameter (öffentlich)

q 0xB0E7CAFFC8DEEE8A55A3050D809ADFE38FA01DAB

BERECHNUNG DER ZUFALLSZAHL

→ Die Zufallszahl k_E kann aus zwei Signaturen wie folgt berechnet werden:

Differenz der Signaturen:

$$s_1 - s_2 = ({k_E}^{-1} \cdot (m_1 + d \cdot r)) - ({k_E}^{-1} \cdot (m_2 + d \cdot r)) \mod q$$

Umstellen nach k_E :

$$k_E = \frac{m_1 - m_2}{s_1 - s_2} \mod q$$

Problem:

→ Reduktion **modulo** q verlangt eine Division
→ Berechnung aufwendig bei großen Zahlen.
→ Hohe zeitliche Komplexität

LANGZAHL-ARITHMETIK

→ 32-oder 64-Bit Register/Instruktionen für artihmetische Grundrechenarten

→ Left-, Right-Shift, bitweises AND ermöglichen effizente Berechnungen von:
- → Multiplikation mit 2^i
- → Division durch 2^i (ohne Rest)
- → Reduktion modulo 2^i

Problem: Kryptographische Operationen mit Zahlen größer als 2^{64}

Lösung: Zahl $x \in \mathbb{N}_0$ zur Basis $b := 2^w$ darstellen:

$$x = \sum_{i=0}^{n} x_i \cdot b^i$$

→ w ist die Prozessorwortlänge, $x_i \in \{0, \ldots, b-1\}$ und $n \in \mathbb{N}_0$.

MONTGOMERY-DARSTELLUNG

Ziel: Effiziente Multiplikation zweier ganzer Zahlen modulo q

$$k_E = (m_1 - m_2) \cdot (s_1 - s_2)^{-1} \mod q$$

Montgomery-Multiplikation:

→ Abbildung h transformiert Zahlen in Montgomery-Darstellung
→ Weitere Berechnungen im isomorphen Ring, statt im Ring $\mathbb{Z}_q$

Vorteile:

→ Benötigt Addition, Multiplikation, Mod- und Div-Operationen mit Zweierpotenz
→ Letztere sind leicht mit wenig aufwendigen Schiebeoperation realisierbar
→ Inverse Abbildung h^{-1} tranformiert das Ergebnis in $\mathbb{Z}_q$ zurück
→ scetool beinhaltet Programmpaket für Langzahl-Arithmetik

BERECHNUNG DER ZUFALLSZAHL

```
1   void generate_Random_and_Private(u8* s1, u8* s2, u8* m1, u8* m2, u8* R, u8* k_E, u8* d_new){
2       [...]
3       //Transformieren der Zahlen in die Montgomery Darstellung
4       bn_to_mon(m1, ec_q, 21);
5       bn_to_mon(m2, ec_q, 21);
6       bn_to_mon(s1, ec_q, 21);
7       bn_to_mon(s2, ec_q, 21);
8       bn_to_mon(R, ec_q, 21);
9
10      //Subtrahieren der Hashwerte
11      bn_sub(m, m1, m2, ec_q, 21);
12
13      //Subtrahieren der Signaturen
14      bn_sub(s, s1, s2, ec_q, 21);
15
16      //Berechnung der Zufallszahl
17      bn_mon_inv(Sinv, s, ec_q, 21);
18      bn_mon_mul(k_E, Sinv, m, ec_q, 21);
19      [...]
20
26      //Ruecktransformation der Zahlen
27      bn_from_mon(k_E, ec_q, 21);
```

Abbildung h transformiert Zahlen in die Montgomery-Darstellung

$m = (m_1 - m_2)$

$s = (s_1 - s_2)$

$s^{-1} = (s_1 - s_2)^{-1}$

$k_E = m \cdot s^{-1} \bmod q$

Inverse Abbildung h^{-1} transformiert Zahlen in Z_q zurück

BERECHNUNG DER ZUFALLSZAHL

```
1   void generate_Random_and_Private(u8* s1, u8* s2, u8* m1, u8* m2, u8* R, u8* k_E, u8* d_new){
2       [...]
3       //Transformieren der Zahlen in die Montgomery Darstellung
4       bn_to_mon(m1, ec_q, 21);
5       bn_to_mon(m2, ec_q, 21);
6       bn_to_mon(s1, ec_q, 21);
7       bn_to_mon(s2, ec_q, 21);
8       bn_to_mon(R, ec_q, 21);
9
10      //Subtrahieren der Hashwerte
11      bn_sub(m, m1, m2, ec_q, 21);
12
13      //Subtrahieren der Signaturen
14      bn_sub(s, s1, s2, ec_q, 21);
15
16      //Berechnung der Zufallszahl
17      bn_mon_inv(Sinv, s, ec_q, 21);
18      bn_mon_mul(k_E, Sinv, m, ec_q, 21);
19      [...]
20
26      //Ruecktransformation der Zahlen
27      bn_from_mon(k_E, ec_q, 21);
```

Abbildung h transformiert Zahlen in die Montgomery-Darstellung

$m = (m_1 - m_2)$

$s = (s_1 - s_2)$

$s^{-1} = (s_1 - s_2)^{-1}$

$k_E = m \cdot s^{-1} \bmod q$

Inverse Abbildung h^{-1} transformiert Zahlen in Z_q zurück

Einfügen der Parameter ergibt folgende Zufallszahl:

0x

BERECHNUNG DES PRIVATEN SCHLÜSSELS

→ Berechnung des privaten Schlüssels d aus der Zufallszahl k_E und der Formel für die Signaturerzeugung:

Formel für die Signaturerzeugung:

$$s = k_E{}^{-1} \cdot (m + d \cdot r) \mod q$$

Umstellen nach d:

$$d = ((k_E \cdot s) - m) \cdot r^{-1} \mod q$$

```
1   void generate_Random_and_Private(u8* s1, u8* s2, u8* m1, u8* m2, u8* R, u8* k_E, u8* d_new){
2       [...]
20      //Berechnung des privaten Schluessels
21      bn_mon_mul(s1, k_E, s1, ec_q, 21);
22      bn_sub(s1, s1, m1, ec_q, 21);
23      bn_mon_inv(Rinv, R, ec_q, 21);
24      bn_mon_mul(d_new, Rinv, s1, ec_q, 21);
25      [...]
28      bn_from_mon(d_new, ec_q, 21);
29  }
```

$s_1 = k_E \cdot s_1$

$s_1 = s_1 - m_1$

r^{-1}

$d = ((k_E \cdot s_1) - m_1) \cdot r^{-1} \mod q$

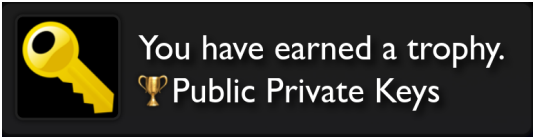
Inverse Abbildung h^{-1} transformiert Zahlen in Z_q zurück

BERECHNUNG DES PRIVATEN SCHLÜSSELS

Einfügen der Parameter ergibt folgenden privaten Schlüssel:

0x [redacted]

BERECHNUNG DES PRIVATEN SCHLÜSSELS



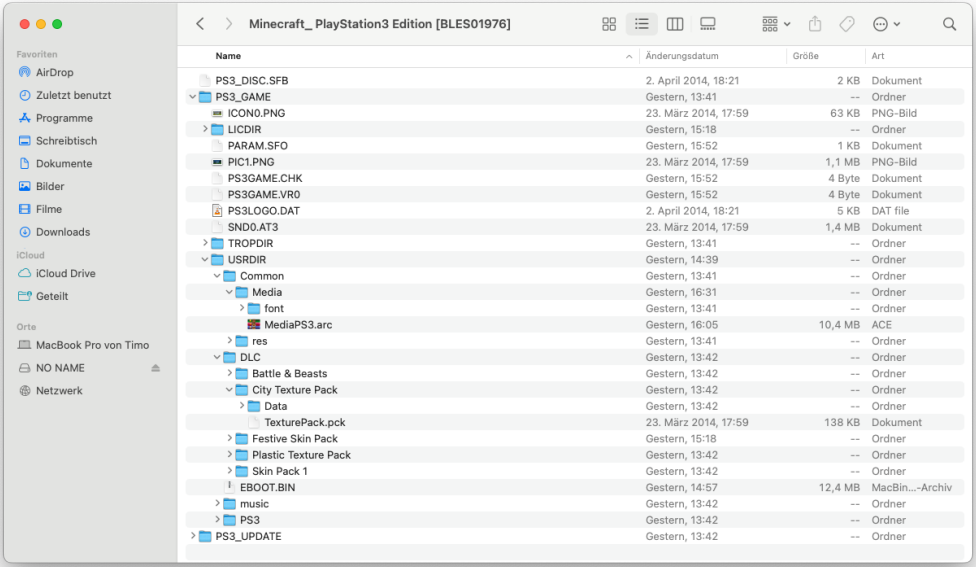
Einfügen der Parameter ergibt folgenden privaten Schlüssel:

0x [redacted]

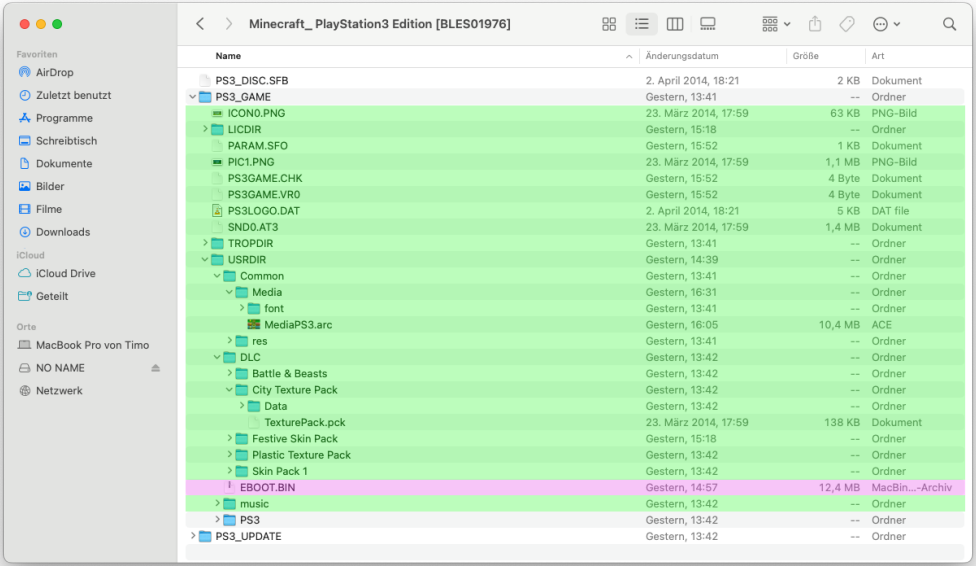
Mit dem privaten Schlüssel können nun eigene Signaturen erstellt werden

GAME MODDING

MINECRAFT DATEIÜBERSICHT



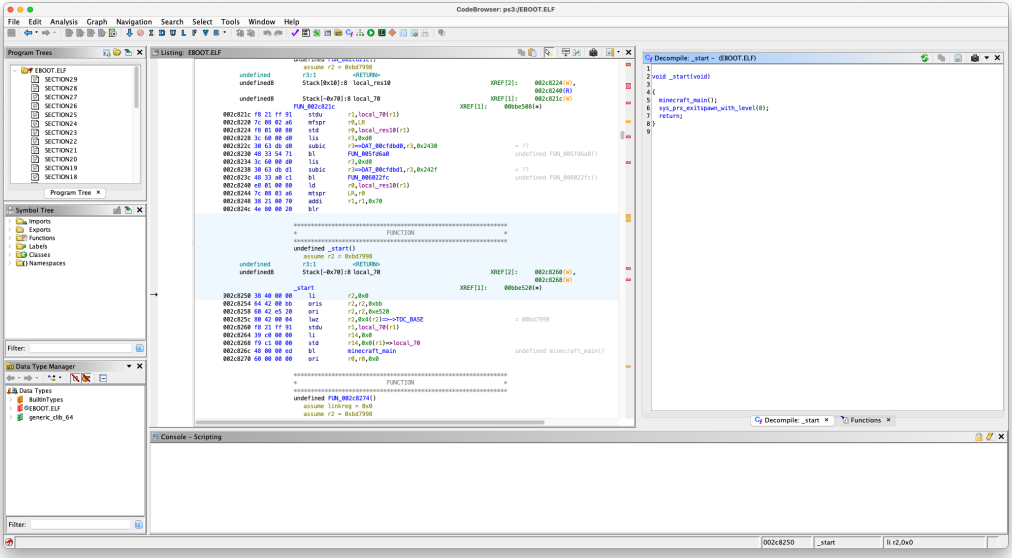
MINECRAFT DATEIÜBERSICHT



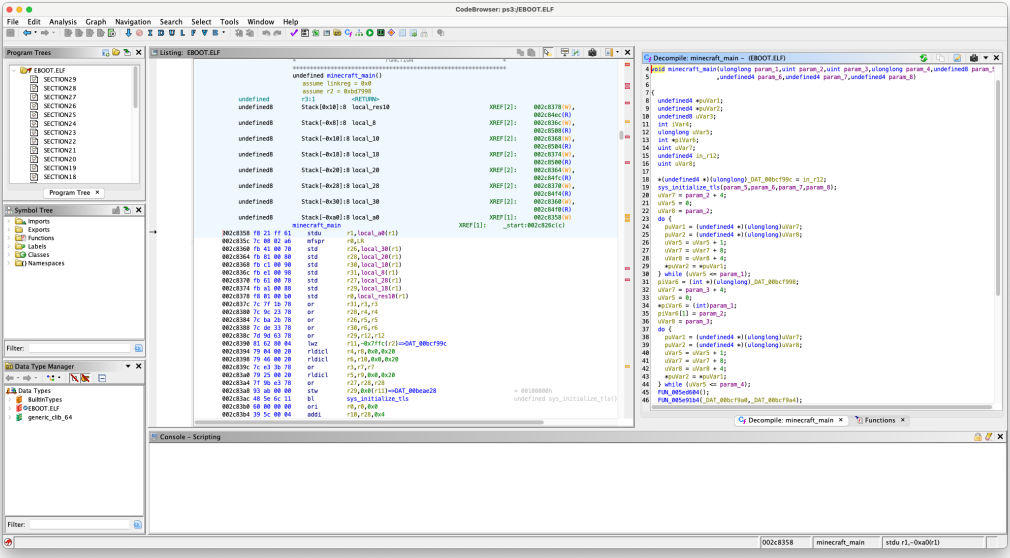
EBOOT.BIN

Minecraft auf der Playstation 3 ist in C++ geschrieben. Heißt wir können den Code disassemblieren und modifizieren.

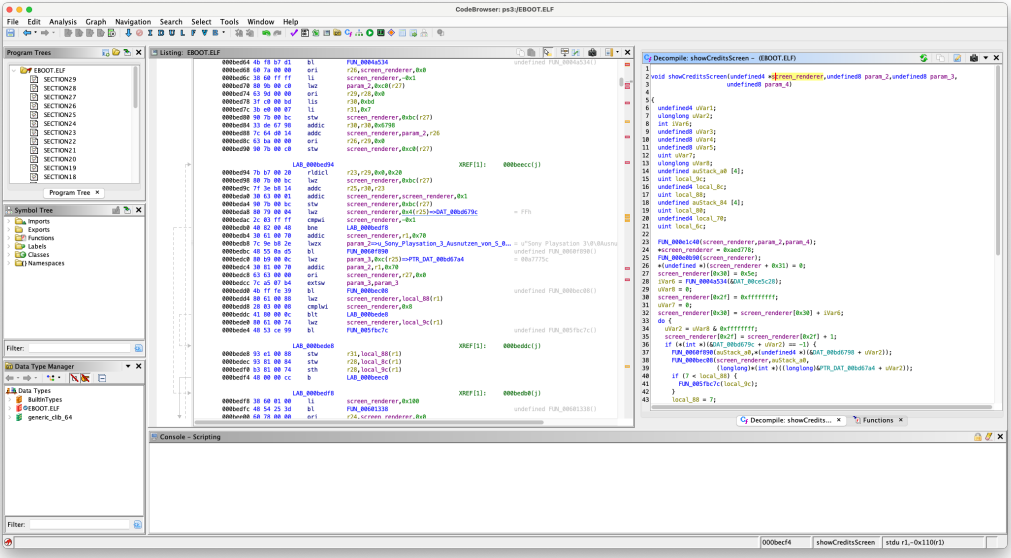
GHIDRA


CodeBrowser: ps3:/EBOOT.ELF
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
EBOOT.ELF
SECTION29
SECTION28
SECTION27
SECTION26
SECTION25
SECTION24
SECTION23
SECTION22
SECTION21
SECTION20
SECTION19
SECTION18
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
0:EBOOT.ELF
generic_clib_64
Filter:
Listing: EBOOT.ELF
Decompile: _start - (EBOOT.ELF)
void _start(void)
{
minecraft_main();
sys_prx_exitspawn_with_level(0);
return;
}
Decompile: _start
Functions
Console - Scripting
002c8250
_start
li r2,0x0

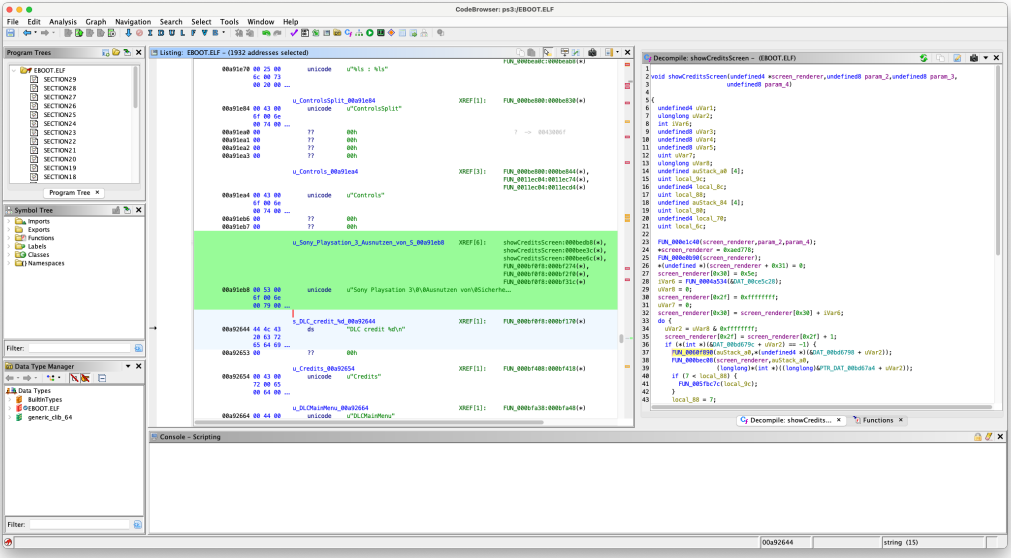
GHIDRA


CodeBrowser: ps3:/EBOOT.ELF
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
EBOOT.ELF
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Data Type Manager
Data Types
BuiltInTypes
0:EBOOT.ELF
generic_clib_64
Listing: EBOOT.ELF
Decompile: minecraft_main - (EBOOT.ELF)
Console - Scripting
Functions
002c8358
minecraft_main
stdu r1,-0xa0(r1)

GHIDRA


CodeBrowser: ps3:/EBOOT.ELF
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
Listing: EBOOT.ELF
Decompile: showCreditsScreen – (EBOOT.ELF)
Symbol Tree
Data Type Manager
Console - Scripting

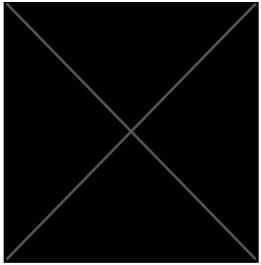
GHIDRA


CodeBrowser: ps3:/EBOOT.ELF
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
EBOOT.ELF
SECTION29
SECTION28
SECTION27
SECTION26
SECTION25
SECTION24
SECTION23
SECTION22
SECTION21
SECTION20
SECTION19
SECTION18
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
0EBOOT.ELF
generic_clib_64
Filter:
Listing: EBOOT.ELF - (1932 addresses selected)
00a91e70 00 25 00 6c 00 73 00 20 00 ... unicode u"%ls : %ls"
u_ControlsSplit_00a91e84 XREF[1]: FUN_000be800:000be830(*)
00a91e84 00 43 00 6f 00 6e 00 74 00 ... unicode u"ControlsSplit"
00a91ea0 00 ?? 00h
00a91ea1 00 ?? 00h
00a91ea2 00 ?? 00h
00a91ea3 00 ?? 00h
u_Controls_00a91ea4 XREF[3]: FUN_000be800:000be844(*), FUN_0011ec04:0011ec74(*), FUN_0011ec04:0011ecd4(*)
00a91ea4 00 43 00 6f 00 6e 00 74 00 ... unicode u"Controls"
00a91eb0 00 ?? 00h
00a91eb1 00 ?? 00h
u_Sony_Playsation_3_Ausnutzen_von_S_00a91eb8 XREF[6]: showCreditsScreen:000bedb8(*), showCreditsScreen:000bee3c(*), showCreditsScreen:000beed6(*), FUN_000bf0f8:000bf274(*), FUN_000bf0f8:000bf28a(*), FUN_000bf0f8:000bf31c(*)
00a91eb8 00 53 00 6f 00 6e 00 79 00 ... unicode u"Sony Playsation 3\0\0Ausnutzen von\0Sicherhe..."
s_DLC_credit_%d_00a92644 XREF[1]: FUN_000bf0f8:000bf170(*)
00a92644 44 4c 43 20 63 72 65 64 69 ... ds "DLC credit %d\n"
00a92653 00 ?? 00h
u_Credits_00a92654 XREF[1]: FUN_000bf408:000bf418(*)
00a92654 00 43 00 72 00 65 00 64 00 ... unicode u"Credits"
u_DLCMainMenu_00a92664 XREF[1]: FUN_000bfa38:000bfa48(*)
00a92664 00 44 00 unicode u"DLCMainMenu"
Decompile: showCreditsScreen - (EBOOT.ELF)
void showCreditsScreen(undefined4 *screen_renderer,undefined8 param_2,undefined8 param_3,
undefined8 param_4)
undefined4 uVar1;
ulonglong uVar2;
int iVar6;
undefined8 uVar3;
undefined8 uVar4;
undefined8 uVar5;
uint uVar7;
ulonglong uVar8;
undefined auStack_a0 [4];
uint local_9c;
undefined4 local_8c;
uint local_88;
undefined auStack_84 [4];
uint local_80;
undefined4 local_70;
uint local_6c;
FUN_000e1c40(screen_renderer,param_2,param_4);
*screen_renderer = 0xaed778;
FUN_000e0b90(screen_renderer);
*(undefined *)(screen_renderer + 0x31) = 0;
screen_renderer[0x38] = 0x5e;
iVar6 = FUN_000a5340(&ART_00ce5c28);
uVar8 = 0;
screen_renderer[0x2f] = 0xffffffff;
uVar7 = 0;
screen_renderer[0x30] = screen_renderer[0x30] + iVar6;
do {
uVar2 = uVar8 & 0xffffffff;
screen_renderer[0x2f] = screen_renderer[0x2f] + 1;
if (*(int *)(&ART_00bd676c + uVar2) == -1) {
FUN_000df890(auStack_a0,*(undefined4 *)(&ART_00bd6798 + uVar2));
FUN_000bec80(screen_renderer,auStack_a0,
(longlong)*(int *)((longlong)&PTR_DAT_00bd67a4 + uVar2));
if (7 < local_88) {
FUN_00519c7c(local_9c);
}
local_88 = 7;
Decompile: showCredits... Functions
Console - Scripting
00a92644
string (15)

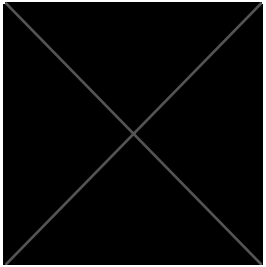
LIVE DEMO

Q&A

PROJEKTDATEIEN



GitLab Repository



Projekt Dokumentation

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

SONY PLAYSTATION 3

Ausnutzen von Sicherheitslücken

30. August 2023

Leon Olejar
Timo Christeleit

Studienbereich Design Informatik Medien
Hochschule **RheinMain**