CONTROL FLOW INTEGRITY ON IBM I

REcon 2024

silent signal

WHOAMI

- Bálint Varga-Perke (@buherator)
- Silent Signal co-founder
  - IBM i focused research (2022.)
- Senior Security Expert

silent signal

PRIOR WORK

- **Architecture**
  - Leif Svalgaard
  - Hugo Landau
- Hacking
  - Shalom Carmel - Hacking iSeries
  - Matthew Carpenter blog posts
  - Bart Kulach - Hacking the Legacy
  - Not many others?

silent signal

IBM I

IBM I

- Midrange: more than a microcomputer, less than a mainframe
- Vertically integrated platform (think iPhone)
- POWER CPU (think PowerPC)
  - 2021: POWER 10
- Object-Oriented OS (think ???)
  - Unix subsystem (PASE) is out of scope now

silent signal

OOP OS

Usual OS

- Files, processes, databases, ...
- File system and Memory
- Serialization for persistence

IBM i

- Everything is an object
- Single-Level Store (Disk+Memory)
- Transparent caching/persistence, object encapsulation

silent signal

SINGLE-LEVEL STORE

- Disk+Memory -> single 64-bit address space
  - [40 bit segment ID] || [24 bit offset]
  - Object encapsulation hides caching/persistence
- System-wide address space!
  - Separated program segments on SL50

silent signal

OOP OS - *PGM EXAMPLE

- No loading phase
  - Except for stack/heap
  - Stored thing === Executed thing
  - Edited data === Stored data
- No such thing as .read() on a Program Object
  - It's a *program* not a *file*
  - Not a continuous byte stream!

silent signal

MACHINE INTERFACE (MI)

- Backwards compatibility is a major selling point
- Object-oriented bytecode
- Midrange SW runs "without modification" across different *architectures*

silent signal

SECURITY

silent signal

THE MYTH, THE LEGEND

"The Only Operating System That has Never been infected by Virus, Trojan, Worm, or Malware." - LinkedIn

"The IBM i itself can't be contaminated by viruses created for computers, but its files are considered excellent 'carriers'" - Fortra

"The architecture of the IBM i system makes it highly unlikely that a virus could be written to attack it" - IBM

External interfaces (e.g. IFS) and configuration are known ITW targets!

silent signal

PZ TAKES A LOOK

- CVE-2023-30990: Pre-Auth RCE (pcrappyfuzzer ftw)
- Several LPE's
- Core components (vs. external interfaces)
- Logical bugs, analogous to *Wintel* systems
- See our TROOPERS'24 talk and blog

silent signal

MEMORY CORRUPTION?

silent signal

SPATIAL SAFETY

```c
int main(){
    char buf[4];
    int num;

    scanf("%x %s", &num, buf);

    // Out-of-bounds read in both directions
    for (int i=-2; i < 8; i++){
        printf(" %02x ", buf[i]);
    }

    printf("\n%x\n", num);

    return 0;
}
```

silent signal

SPATIAL SAFETY

```
> 1337 ABCDE
  40  40  c1  c2  c3  c4  c5  00  13  37
  c5001337
  Press ENTER to end terminal session.
```

silent signal

EXPLOITATION?

- **OOB access -> privilege escalation**
  - SLS -> Single address space
  - Even "artificial" corruptions work!
  - Below Security Level 50
- Memory corruptions do occur ItW
  - Search for MCH3601, MCH0601

silent signal

REVERSE

silent signal

MI VS NATIVE

- Every language is first compiled to an intermediate representation
  - OPM -> U-Code -> MI
  - ILE -> W-Code -> (N)MI
  - Think JVM/.NET bytecode
- MI code is embedded in *PGM's
  - The *translator* generates native code from MI
  - MI is independent from HW, so you can "recompile" to new architectures

silent signal

STATIC ANALYSIS

- Program objects are not bitstreams!
- Programs still need to be transferred to other systems etc.
- Save Files (*SAVF): ~Universal object serialization format
- *PGM -> *FILE -> SCP/FTP

silent signal

SAVE FILES (*SAVF)

- Undocumented format
- Free, closed-source reader: jSAVF

silent signal

SERIALIZED *PGM

- SAVF's can contain multiple objects
- PGM: Undocumented format
- Let's write a parser...

silent signal

DYNAMIC ANALYSIS

- System Service Tools (SST)
  - Low-level debugging features for admins
- R/W memory dump
- No breakpoints, tracing, etc.
- Partially masked register state
- "scanf() debugging"

silent signal

BASE STRUCTURE

```
Hex editor

Address    00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F

00001FF0:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00002000:  31 10 00 00 81 00 80 04  1E 57 0E 6D A1 00 12 00
00002010:  1E 57 0E 6D A1 00 13 90  1E 57 0E 6D A1 00 15 20
00002020:  1E 57 0E 6D A1 00 30 50  1E 57 0E 6D A1 01 34 F0
00002030:  1E 57 0E 6D A1 00 12 30  1E 57 0E 6D A1 00 00 00

                                                Display For

Find . . . . . .
. . . +. . . . 1. . . . +. . . . 2. . . . +. . . . 3. . . . +. . . .
        DISPLAY/ALTER/DUMP
MI   PROGRAM                                    SUBTYPE:
PROGRAM HEADER
     PROGRAM ATTRS          3110000081008004
     SEGMENT TBL PTR        1E570E6DA1  001390
     SIG TBL PTR            1E570E6DA1  003050
     ACT GRP INF PTR        1E570E6DA1  001230
     ACT STR                1E570E6DA1  001230
```

KAITAI <3 GHIDRA

- Kaitai Struct allows declarative, multi-language parser creation
  - Debug in Python/JS/... REPL
  - Hexeditor integration (Hobbit, VSCode)
  - Reusable Java code for Ghidra
- Adding XCOFF support to Ghidra with Kaitai Struct

silent signal

```java
public class GhidraPGM400Loader extends AbstractProgramWrapperLoader {
  @Override
  protected void load(ByteProvider provider /*...*/ ){
    ByteBufferKaitaiStream stream =
      new ByteBufferKaitaiStream(provider.readBytes(0, provider.length()))
    // Instantiate strcture with Kaitai
    Savf kaitai = new Savf(stream);
    // Object-oriented access to structure
    Savf.ProgramHeaderBase programBaseHeader =
      kaitai.mainSegment().pgmHeader().programHeader();
    // ...
  }
  // ...
}
```

silent signal

```java
public class GhidraPGM400Loader extends AbstractProgramWrapperLoader {
  @Override
  protected void load(ByteProvider provider /*...*/ ){
    ByteBufferKaitaiStream stream =
      new ByteBufferKaitaiStream(provider.readBytes(0, provider.length()))
    // Instantiate strcture with Kaitai
    Savf kaitai = new Savf(stream);
    // Object-oriented access to structure
    Savf.ProgramHeaderBase programBaseHeader =
      kaitai.mainSegment().pgmHeader().programHeader();
    // ...
  }
  // ...
}
```

silent signal

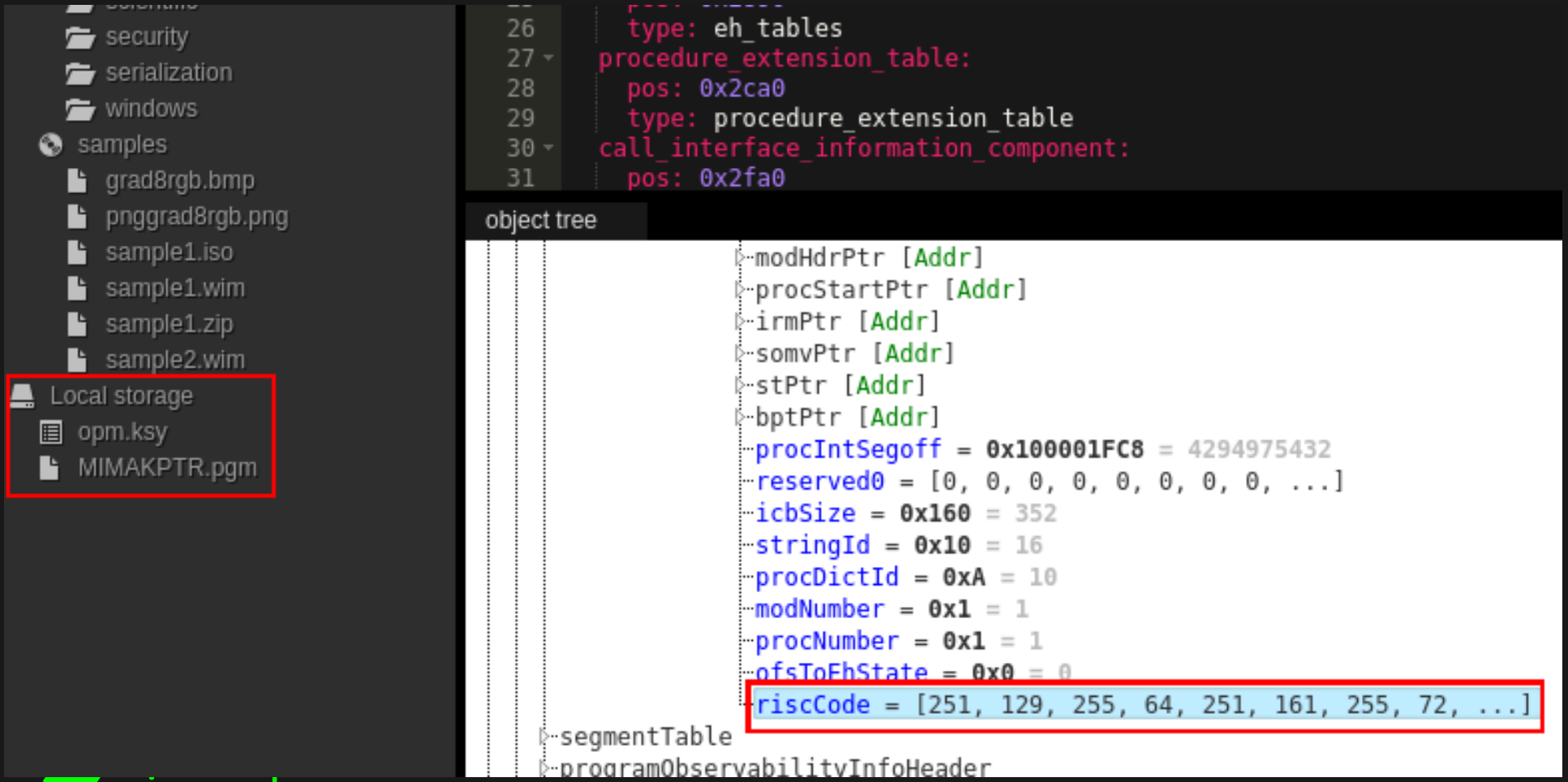
```java
public class GhidraPGM400Loader extends AbstractProgramWrapperLoader {
  @Override
  protected void load(ByteProvider provider /*...*/ ){
    ByteBufferKaitaiStream stream =
      new ByteBufferKaitaiStream(provider.readBytes(0, provider.length()))
    // Instantiate strcture with Kaitai
    Savf kaitai = new Savf(stream);
    // Object-oriented access to structure
    Savf.ProgramHeaderBase programBaseHeader =
      kaitai.mainSegment().pgmHeader().programHeader();
    // ...
  }
  // ...
}
```

silent signal

```java
public class GhidraPGM400Loader extends AbstractProgramWrapperLoader {
  @Override
  protected void load(ByteProvider provider /*...*/ ){
    ByteBufferKaitaiStream stream =
      new ByteBufferKaitaiStream(provider.readBytes(0, provider.length()))
    // Instantiate strcture with Kaitai
    Savf kaitai = new Savf(stream);
    // Object-oriented access to structure
    Savf.ProgramHeaderBase programBaseHeader =
      kaitai.mainSegment().pgmHeader().programHeader();
    // ...
  }
  // ...
}
```

silent signal

KAITAI VS. PGM



signal

```
                                undefined FUN_1e570e6da1003080()
          undefined                r3:1            <RETURN>
                                FUN_1e570e6da1003080
1e570e6da1003080 78 a0 00 21     rldicl.     r0,r5,0x0,0x20
1e570e6da1003084 38 a0 00 00     li          r5,0x0
1e570e6da1003088 4d c2 00 20     beqlr
1e570e6da100308c 7c 09 03 a6     mtspr       CTR,r0
1e570e6da1003090 7c 04 2a 14     add         r0,r4,r5
1e570e6da1003094 7c              ??          7Ch
1e570e6da1003095 20              ??          20h
1e570e6da1003096 20              ??          20h
```

Private ISA extensions: POWER-AS

silent signal

Ghidra
PPC module
can't decode

```
0000305e 00              ??      00h
0000305f 22              ??      22h    "
00003060 38 c0 00 80     li      r6,0x80
00003064 78 c6 c1 c6     rldicr  r6,r6,0x38,0x7
00003068 7c              ??      7Ch    |                    Test patch from 7c 01 03 e6
00003069 01              ??      01h
0000306a 03              ??      03h
0000306b e6              ??      E6h
0000306c 60 c4 00 00     ori     r4,r6,0x0
00003070 61 85 00 00     ori     r5,r12,0x0
00003074 f8              ??      F8h
00003075 8a              ??      8Ah
00003076 00              ??      00h
00003077 32              ??      32h    2
00003078 61 43 00 00     ori     r3,r10,0x0
0000307c 48 00 01 55     bl      scanf                       undefined scanf
```

Search Memory

Search Value: 61850000

Hex Sequence: 61 85 00 00

```
                    Display Formatted Data
                                    Page/Line. . .   52 / 41
                                    Columns. . . :   1 - 78
Find . . . . . . . . . . . . .  2A9EE
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
2A9EE6911E 002074     0000A4     F88A0032      STQ 4,0X30(10)
2A9EE6911E 002078     0000A8     61430000      ORI 3,10,0
2A9EE6911E 00207C     0000AC     48000155      BL 0X154
```

IBM can :)

silent signal

POWER-AS FOR GHIDRA

We have a Processor Module!

Program Trees

- S2DBG.pgm
  - Segment 0
  - Segment 1
  - Segment 2
  - ExceptionSpace
  - StackSpace

Memory mapping

Program Tree

Some symbols

Symbol Tree

- Imports
- Exports
- Functions
  - FUN_3692b02d25001e00
  - FUN_3692b02d25002200
  - increment
  - main
- Labels

Listing: S2DBG.pgm

Complete Disassembly

```
**********************************************
undefined increment()
   r3:1            <RETURN>
increment
3692b02d25002190 7c 08 02 a6     mfspr      r0,LR
3692b02d25002194 f8 01 00 28     std        r0,0x28(r1)
3692b02d25002198 f8 21 ff 21     stdu       r1,-0xe0(r1)
3692b02d2500219c 3c 00 04 03     lis        r0,0x403
3692b02d250021a0 f8 01 00 08     std        r0,0x8(r1)
3692b02d250021a4 41 dd 80 03     bgtla      cr7,SUB_ffffffffffff8000
3692b02d250021a8 f8 83 00 30     std        r4,0x30(r3)
3692b02d250021ac e9 83 00 30     ld         r12,0x30(r3)
3692b02d250021b0 e5 43 00 21     ltptr      r10,0x2(r3),0x2
3692b02d250021b4 7d 2a 62 14     add        r9,r10,r12
3692b02d250021b8 7f a9 50 88     td         0x1d,r9,r10
3692b02d250021bc 38 e0 00 80     li         r7,0x80
3692b02d250021c0 78 e7 c1 c6     rldicr     r7,r7,0x38,0x7
3692b02d250021c4 7c 01 03 e6     settag
3692b02d250021c8 60 e4 00 00     ori        r4,r7,0x0
3692b02d250021cc 61 25 00 00     ori        r5,r9,0x0
3692b02d250021d0 f8 83 00 12     stq        r4,0x10(r3)
3692b02d250021d4 48 00 00 04     b          LAB_3692b02d250021d8

LAB_3692b02d250021d8
3692b02d250021d8 41 dd 81 c3     bgtla      cr7,SUB_ffffffffffff81c0
3692b02d250021dc 38 21 00 e0     addi       r1,r1,0xe0
3692b02d250021e0 e8 01 00 28     ld         r0,0x28(r1)
3692b02d250021e4 7c 08 03 a6     mtspr      LR,r0
```

POWER-AS FOR GHIDRA

```
 2 void increment(longlong param_1,undefined8 param_2)
 3
 4 {
 5   longlong in_r1;
 6   longlong lVar1;
 7   byte in_cr7;
 8   undefined8 in_LR;
 9
10   *(undefined8 *)(in_r1 + 0x28) = in_LR;
11   *(longlong *)(&UNK_ffffffffffffff20 + in_r1) = in_r1;
12   *(undefined8 *)(&UNK_ffffffffffffff28 + in_r1) = 0x4030000;
13   if ((bool)(in_cr7 >> 2 & 1)) {
14     param_1 = func_0xffffffffffff8000();
15   }
16   *(undefined8 *)(param_1 + 0x30) = param_2;
17   lVar1 = *(longlong *)(param_1 + 0x28) + *(longlong *)(param_1 + 0x30);
18   trapDoubleWord(0x1d,lVar1,*(longlong *)(param_1 + 0x28));
19   settag();
20   *(undefined8 *)(param_1 + 0x10) = 0x8000000000000000;
21   *(longlong *)(param_1 + 0x18) = lVar1;
22   if ((bool)(in_cr7 >> 2 & 1)) {
23     func_0xffffffffffff81c0();
24   }
25   return;
26 }
```

POWER-AS FOR GHIDRA

Some findings

- Register usage
- Calling convention
  - Registers + thick pointers on stack
- Function arguments and locals use different stack pointers
  - <u>How to handle that in Ghidra?!</u>
  - Local stack grows up...
- Return addresses are on a *different stack*

silent signal

ATTACK!

```
38 e0 00 80     li          r7,0x80
78 e7 c1 c6     rldicr      r7,r7,0x38,0x7
7c 01 03 e6     settag
60 e4 00 00     ori         r4,r7,0x0
61 25 00 00     ori         r5,r9,0x0
f8 83 00 12     stq         r4,0x10(r3)
```

silent signal

MEMORY TAGGING

- 1-bit of extra information can be encoded for every aligned 128-bit QWORD
  - HW assisted (ECC?), can't be leaked/forged
- SETTAG + store instruction
- Loads check tags
  - LQ sets bits in XER, then TXER traps
  - LTPTR loads NULL if untagged

silent signal

MEMORY TAGGING

```
7c 01 03 e6     settag
f8 88 00 22     stq         r4,0x20(r8)
```

```
e1 1f ff d1     lq          r8,-0x30(r31)
7c 00 05 48     txer        0x0,0x0,0xa
```

silent signal

MEMORY TAGGING

"Was this pointer created in an approved way?"

silent signal

MEMORY TAGGING

Consequence:

Corrupted pointers can't be dereferenced

silent signal

Idea #0

Write shellcode and jump to it

silent signal

TYPED POINTERS

```
printf("[%llx %llx + %d]", buf, offset);
[8000000000000000 ab4dc0ffee002000 + 1337]
```

First 64-bits of a 128-bit ptr contains the type

silent signal

TYPED POINTERS

Load Quadword **DQ-form**

lq RTp,DQ(RA)

56	RTp	RA	DQ	///
0	6	11	16	28 31

"The instruction is also extended with a third operand (in bits 28:31 inclusive), an immediate, which appears to mask the loaded value in some way." - The PowerPC AS Tagged Memory Extensions

silent signal

FUNCTION POINTER CALL

```
lq          r8,-50(r31),0x01  ; Pointer from stack in R8||R9
                              ; Last four reserved bits are: 0b0001
txer        0x0,0x0,0xa       ; Type check
ori         r10,r9,0x0        ; Function address -> R10
ori         r12,r8,0x0        ; Funcptr type -> R12
ori         r3,r7,0x0         ; Setting callee parameter stack
ld          r11,0x0(r10)      ; Dereference function address to R11
; ...
ld          r4,0x8(r11)       ; Another deref from R11 to R4
mtspr       CTR,r4            ; Set program counter from R4
ld          r2,0x0(r11)
bctrl                         ; Branch to CTR
```

silent signal

FUNCTION POINTER CALL

```
lq          r8,-50(r31),0x01  ; Pointer from stack in R8||R9
                              ; Last four reserved bits are: 0b0001
txer        0x0,0x0,0xa       ; Type check
ori         r10,r9,0x0        ; Function address -> R10
ori         r12,r8,0x0        ; Funcptr type -> R12
ori         r3,r7,0x0         ; Setting callee parameter stack
ld          r11,0x0(r10)      ; Dereference function address to R11
; ...
ld          r4,0x8(r11)       ; Another deref from R11 to R4
mtspr       CTR,r4            ; Set program counter from R4
ld          r2,0x0(r11)
bctrl                         ; Branch to CTR
```

silent signal

FUNCTION POINTER CALL

```
lq          r8,-50(r31),0x01  ; Pointer from stack in R8||R9
                              ; Last four reserved bits are: 0b0001
txer        0x0,0x0,0xa       ; Type check
ori         r10,r9,0x0        ; Function address -> R10
ori         r12,r8,0x0        ; Funcptr type -> R12
ori         r3,r7,0x0         ; Setting callee parameter stack
ld          r11,0x0(r10)      ; Dereference function address to R11
; ...
ld          r4,0x8(r11)       ; Another deref from R11 to R4
mtspr       CTR,r4            ; Set program counter from R4
ld          r2,0x0(r11)
bctrl                         ; Branch to CTR
```

silent signal

TYPED POINTERS

LQ-TXER mask pairs (lq_stats.py)

```
LQ mask: 0x1 with TXER mask 0xA - 1 instances
LQ mask: 0xF with TXER mask 0x3 - 3 instances
```

```
LQ mask: 0x2 with TXER mask 0x3 - 9 instances
LQ mask: 0xF with TXER mask 0x3 - 21 instances
```

silent signal

Type Byte	LQ Mask	TXER Mask
0x80	0xF	0x3
0xA1	0x1	0xA
0xA2	0x2	0x3

silent signal

TYPED POINTERS

Consequence:

Can't transfer control to data via pointers

silent signal

Idea #1

Just use a large enough overflow!

silent signal

SEGMENT BOUNDARY CHECKING

```
putchar(str[i]);
```

```
ltptr      r8,0x2(r29),0x2  ; Load address from typed pointer to R8
add        r7,r8,r12        ; Add offset (R12) to address, result in R7
td         0x1d,r7,r8       ; Trap conditionally
lbz        r6,0x0(r7)       ; Load byte from R7
```

silent signal

SEGMENT BOUNDARY CHECKING

```
putchar(str[i]);
```

```
ltptr      r8,0x2(r29),0x2  ; Load address from typed pointer to R8
add        r7,r8,r12        ; Add offset (R12) to address, result in R7
td         0x1d,r7,r8       ; Trap conditionally
lbz        r6,0x0(r7)       ; Load byte from R7
```

silent signal

SEGMENT BOUNDARY CHECKING

The 0x1d (0b11101) mask doesn't make sense!

Trap Doubleword — *X-form*

td TO,RA,RB

31	TO	RA	RB	68	/
0	6	11	16	21	31

```
a ← (RA)
b ← (RB)
if (a < b) & TO0 then TRAP
if (a > b) & TO1 then TRAP
if (a = b) & TO2 then TRAP
if (a <u b) & TO3 then TRAP
if (a >u b) & TO4  then TRAP
```

SEGMENT BOUNDARY CHECKING

```
td_masks.py> Running...
TD Mask: 0b11100 - 44 instances found
TD Mask: 0b11101 - 4 instances found
td_masks.py> Finished!
```

Checking if top 40 bits of src and dst registers match.

silent signal

SEGMENT BOUNDARY CHECKING

DEMO

silent signal

SEGMENT BOUNDARY CHECKING

Consequence:

Can't move pointers outside their original segments

silent signal

Idea #2

Return address overwrite?

silent signal

RETURN ADDRESS OVERWRITE

- Tag/type not checked on function return!
- Return addresses are stored in a separate segment

```
**************************************************************
*                          FUNCTION                          *
**************************************************************
void __stdcall main_(void)
   assume linkreg = 0x0
   assume r1 = 0x1337133713100000
   assume r2 = 0x82001487b0019c0
   assume r31 = 0x1337133713100000
   <VOID>          <RETURN>
   r10:8           strTblPtr_                          XREF[1]:     82001487
   r31:8           puVar5                              XREF[1]:     82001487
   HASH:3fa0a19... lVar1
main_                                          XREF[1]:     82001487b001ec8(c
ff 38     std         r27,-0xc8(r1)
                          $Ue000:8 = INT_ADD 0xffffffffffffff38:8, r1
                          STORE ram($Ue000:8), r27
NKNOWN>
KNOWN>
ff 40     std         r28,-0xc0(r1)
                          $Ue000:8 = INT_ADD 0xffffffffffffff40:8, r1
                          STORE ram($Ue000:8), r28
ff 48     std         r29,-0xb8(r1)
                          $Ue000:8 = INT_ADD 0xffffffffffffff48:8, r1
```

```
 8  undefined *puVar1;
 9  longlong strTblPtr_;
10  undefined8 unaff_r27;
11  undefined8 *puVar2;
12  undefined8 unaff_r28;
13  undefined8 unaff_r29;
14  undefined8 unaff_r30;
15  char *puVar5;
16  byte in_cr7;
17  undefined8 in_LR;
18  char *lVar1;
19
20  strTblPtr_ = 0x82001487b0019c0;
21  uRam1337133713 0fff58 = 0x1337133713100000;
22                      /* Stack frame? */
23  puVar1 = (undefined *)0x13371337130ffec0;
24  uRam13371337130ffec0 = 0x1337133713100000;
25  uRam13371337130ffee0 = 0x82001487b0019c0;
26  uRam13371337130ffec8 = 0x44130000;
27  trapXER();
28  uRam13371337130fff38 = unaff_r27;
29  uRam13371337130fff40 = unaff_r28;
30  uRam13371337130fff48 = unaff_r29;
```

signal

RETURN ADDRESS OVERWRITE

Consequence:

User code can't corrupt return addresses

silent signal

Idea #3

Out-of-context call?

silent signal

OUT-OF-CONTEXT FPTR CALL

```
1  void (*adminCmd)(char*);
2  Command commands[2];
3
4  commands[1].name="lower"; commands[1].exec=lower;
5  commands[0].name="upper"; commands[0].exec=upper;
6  adminCmd = my_system;
7
8  adminCmd("SNDPGMMSG MSG(CRACKMEX) MSGTYPE(*INFO)");
9  // ...
10 while(cmd != 99){
11     // ...
12     Command *tmpCmd = &(commands[cmd]);
13     printf("%llx %llx %llx %llx\n", *tmpCmd);
14     printf("Invoking %s(%s)\n", tmpCmd->name, param);
15     tmpCmd->exec(param);
16 }
```

silent signal

OUT-OF-CONTEXT FPTR CALL

```
1  void (*adminCmd)(char*);
2  Command commands[2];
3
4  commands[1].name="lower"; commands[1].exec=lower;
5  commands[0].name="upper"; commands[0].exec=upper;
6  adminCmd = my_system;
7
8  adminCmd("SNDPGMMSG MSG(CRACKMEX) MSGTYPE(*INFO)");
9  // ...
10 while(cmd != 99){
11     // ...
12     Command *tmpCmd = &(commands[cmd]);
13     printf("%llx %llx %llx %llx\n", *tmpCmd);
14     printf("Invoking %s(%s)\n", tmpCmd->name, param);
15     tmpCmd->exec(param);
16 }
```

silent signal

OUT-OF-CONTEXT FPTR CALL

```
1  void (*adminCmd)(char*);
2  Command commands[2];
3
4  commands[1].name="lower"; commands[1].exec=lower;
5  commands[0].name="upper"; commands[0].exec=upper;
6  adminCmd = my_system;
7
8  adminCmd("SNDPGMMSG MSG(CRACKMEX) MSGTYPE(*INFO)");
9  // ...
10 while(cmd != 99){
11     // ...
12     Command *tmpCmd = &(commands[cmd]);
13     printf("%llx %llx %llx %llx\n", *tmpCmd);
14     printf("Invoking %s(%s)\n", tmpCmd->name, param);
15     tmpCmd->exec(param);
16 }
```

silent signal

OUT-OF-CONTEXT FPTR CALL

```
1  void (*adminCmd)(char*);
2  Command commands[2];
3
4  commands[1].name="lower"; commands[1].exec=lower;
5  commands[0].name="upper"; commands[0].exec=upper;
6  adminCmd = my_system;
7
8  adminCmd("SNDPGMMSG MSG(CRACKMEX) MSGTYPE(*INFO)");
9  // ...
10 while(cmd != 99){
11     // ...
12     Command *tmpCmd = &(commands[cmd]);
13     printf("%llx %llx %llx %llx\n", *tmpCmd);
14     printf("Invoking %s(%s)\n", tmpCmd->name, param);
15     tmpCmd->exec(param);
16 }
```

silent signal

CRACKMEX

DEMO

silent signal

FAILS / FUTURE WORK

- Misaligned loads / Type confusion
  - How do I get useful pointers?
- TOCTOU
  - No tags on regs!
- Did't look
  - Translator bugs
  - Microarchitecture

silent signal

TEMPORAL SAFETY

```c
for (int i=0; i < BUFS_COUNT; i++){
  bufs[i]=(char*)malloc(0xffe000);
  memset(bufs[i], 0x41, 0xffe000);
}
for (int i=0;i<BUFS_COUNT;i++){
  free(bufs[i]);
}

for (int j=0; j < 10; j++){
  for (int i=0; i<BUFS_COUNT; i++){
      for (int k=0; k<0xffe000; k++){
          if (bufs[i][k] != 0x41){
              printf("%02x ", bufs[i][k]);
          }
      }
  }
}
```

silent signal

TEMPORAL SAFETY

```c
for (int i=0; i < BUFS_COUNT; i++){
  bufs[i]=(char*)malloc(0xffe000);
  memset(bufs[i], 0x41, 0xffe000);
}
for (int i=0;i<BUFS_COUNT;i++){
  free(bufs[i]);
}

for (int j=0; j < 10; j++){
  for (int i=0; i<BUFS_COUNT; i++){
      for (int k=0; k<0xffe000; k++){
          if (bufs[i][k] != 0x41){
              printf("%02x ", bufs[i][k]);
          }
      }
  }
}
```

silent signal

TEMPORAL SAFETY

```c
for (int i=0; i < BUFS_COUNT; i++){
  bufs[i]=(char*)malloc(0xffe000);
  memset(bufs[i], 0x41, 0xffe000);
}
for (int i=0;i<BUFS_COUNT;i++){
  free(bufs[i]);
}

for (int j=0; j < 10; j++){
  for (int i=0; i<BUFS_COUNT; i++){
      for (int k=0; k<0xffe000; k++){
          if (bufs[i][k] != 0x41){
              printf("%02x ", bufs[i][k]);
          }
      }
  }
}
```

silent signal

TEMPORAL SAFETY

```
00 00 00 00 00 ff e0 10 d3 fa d1 28 71 00 20 00 00 00 00 00 00 ff e0 10 ca a4 7f 9f 52 00 20 00 00
00 ff e0 10 d9 c0 49 43 e8 00 20 00 00 00 00 00 00 ff e0 10 c1 4e 2e 16 33 00 20 00 00 00 00 00 00
d0 69 f7 ba c9 00 20 00 00 00 00 00 00 ff e0 10 f7 f7 dc 8d 14 00 20 00 00 00 00 00 00 ff e0 10 df
aa 00 20 00 00 00 00 00 00 ff e0 10 f4 67 03 1f 6c 00 20 00 00 00 00 00 00 ff e0 10 db f4 e7 f1 b7
00 00 00 00 00 ff e0 10 d2 9e 96 68 98 00 20 00 00 00 00 00 00 ff e0 10 e1 ba 60 0d 2e 00 20 00 00
00 ff e0 10 ff f1 f3 56 5a 00 20 00 00 00 00 00 00 ff e0 10 f6 9b a1 cd 3b 00 20 00 00 00 00 00 00
d4 d3 35 16 67 00 20 00 00 00 00 00 00 ff e0 10 e3 ee fe ba fd 00 20 00 00 00 00 00 00 ff e0 10 f3
29 00 20 00 00 00 00 00 00 ff e0 10 d1 42 5b a8 bf 00 20 00 00 00 00 00 00 ff e0 10 e0 5e 25 4d 55
00 00 00 00 00 ff e0 10 e6 23 9d 68 cc 00 20 00 00 00 00 00 00 ff e0 10 f5 3f 67 0d 62 00 20 00 00
00 ff e0 10 eb e9 15 84 43 00 20 00 00 00 00 00 00 ff e0 10 e2 92 c3 fb 24 00 20 00 00 00 00 00 00
d9 3c 72 72 05 00 20 00 00 00 00 00 00 ff e0 10 f7 74 05 bb 31 00 20 00 00 00 00 00 00 ff e0 10 c6
a8 00 20 00 00 00 00 00 00 ff e0 10 cc 55 47 7b 3e 00 20 00 00 00 00 00 00 ff e0 10 f3 e3 2c 4d 89
00 00 00 00 00 ff e0 10 d2 1a bf 96 b5 00 20 00 00 00 00 00 00 ff e0 10 c8 c4 6e 0d 96 00 20 00 00
00 ff e0 10 e6 fc 01 56 c2 00 20 00 00 00 00 00 00 ff e0 10 f6 17 ca fb 58 00 20 00 00 00 00 00 00
d4 4f 5e 44 84 00 20 00 00 00 00 00 00 ff e0 10 e3 6b 27 e9 1a 00 20 00 00 00 00 00 00 ff e0 10 da
91 00 20 00 00 00 00 00 00 ff e0 10 f8 4c 69 a9 27 00 20 00 00 00 00 00 00 ff e0 10 c7 68 33 4d bd
00 00 00 00 00 ff e0 10 e5 9f c6 96 e9 00 20 00 00 00 00 00 00 ff e0 10 f4 bb 90 3b 7f 00 20 00 00
00 ff e0 10 fa 81 08 56 f6 00 20 00 00 00 00 00 00 ff e0 10 c9 9c d1 fb 8c 00 20 00 00 00 00 00 00
e7 d4 65 44 b8 00 20 00 00 00 00 00 00 ff e0 10 f6 f0 2e e9 4e 00 20 00 00 00 00 00 00 ff e0 10 c6
7a 00 20 00 00 00 00 00 00 ff e0 10 cb d1 70 a9 5b 00 20 00 00 00 00 00 00 ff e0 10 da ed 3a 4d f1
00 00 00 00 00 ff e0 10 ca a4 7f 9f 52 00 20 00 00 00 00 00 00 ff e0 10 f2 32 64 71 9d 00 20 00 00
```

silent signal

IDEAS?

silent signal

ENDPGM

signal

SUMMARY

- Memory corruption **is** possible on IBM i
- CFI makes exploitation **hard**
  - Logical bugs are more appealing
  - Even Data-oriented exploitation is harder on segmented memory
- Accepting objects from untrusted systems is Achilles' Heel of the platform
  - Now we have tools to attack it

silent signal

RABBIT HOLE

- IBM i cloud instances - pub400.com
- Custom IBM POWER hardware - raptorcs.com

silent signal

THANK YOU!

Writeup **Ghidra ext.** **Kaitai defs**

greetz: depth, hl, s2, alligators, hipsters

silent signal