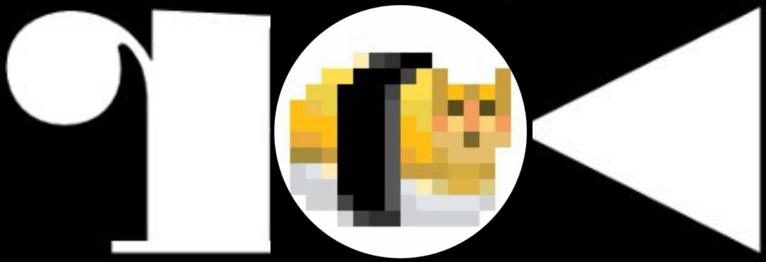
Decompiling with AI

by pancake



Whoami

Sergi Àlvarez aka **pancake**

- Mobile Security Research Engineer at NowSecure
- Author and leader of the Radare project
- Free Software enthusiast and developer

Introduction

Since the beginning of the boom of large language models I've been playing with them in order to find useful applications for it.

R2AI initially supported llama for local models only, but now supports a large set of commercial and free backends, from ollama to claude.

But **R2AI** being written in Python make things “hard”

What's AI for and what's not for?

Hallucinations, we all love them. But probably not at work time.

We can't trust **LLM** outputs without reviewing, but they can help us a lot in understanding complex things and get better strategies to solve our challenges.

Nowadays using local models is a good option for privacy reasons.

- Use it wisely, we all know how harmful this technology can be for the planet. It's a tool, not a replacement for researchers.

Installation

Decomposing

- **R2AI**
  - CLI tool with rest server in Python that uses r2pipe, manages models, uses llama or many remote implementations OpenAI, Claude, ..
- **R2AI-plugin**
  - Same R2AI module running *inside* r2, adds the r2ai cmd
- **R2AI-server**
  - Shellscript that spawns r2ai -w, or openapi/llama servers
- **Decai**
  - R2JS script that depends on curl to talk to r2ai-server or

R2PM

As usual, everything we need in radare2 it's available via r2pm.

```
$ r2pm -s r2ai        # search for r2ai packages

$ r2pm -ci decai      # install the decai script
```

Focus on Decai

How decai was born

R2JS is native to r2, so having a way to use language models without having to install anything was the easiest way to use.

Problems?

- We can't use llama directly and function calling needs to be reimplemented, but we can get all this in few KBs of js instead of 1GB of python code.

Why not C?

Yes, this can be written in C, actually even LlamaCpp exposes a C API, and there's no problem in using libcurl or even RSocket to talk to OpenAPI rest instances.

- There's the r2ai-native plugin for that

This may eventually be the way to go, but right now it was funnier and faster to experiment with Py/JS and get things done easier.

Features

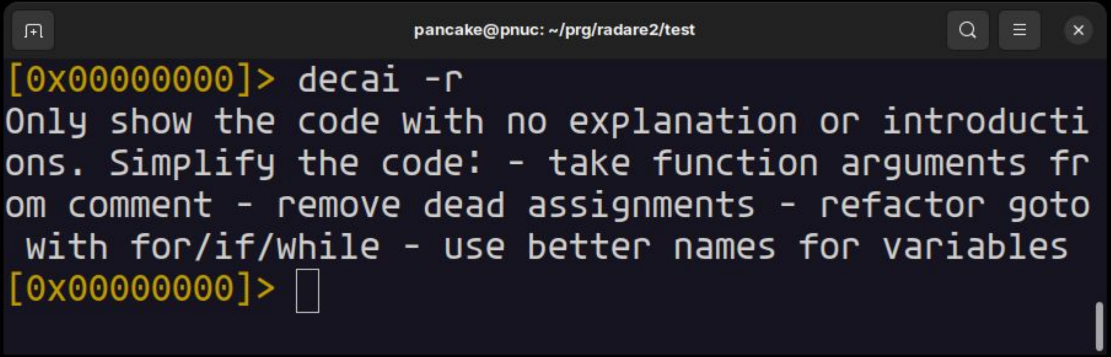
What Can Decai Do?

- Use openapi, huggingface, openai and claude endpoints
- Decompile into many languages
- Explain what a function is doing, in many languages
- Find a better function signature for it
- Guess the function name and perform type propagation
- Identify local variables, its names and types
- Find vulnerabilities
- Combine multiple functions for context decompilation
- Infer ESIL findings into strings or function arguments

Focus On Decompilation

Decompiling with Decai

- Use decai -e cmds to select the commands to feed the model
    - By default uses the pseudo decompiler from r2
- The prompt will do the rest for us



```
[0x00000000]> decai -r
Only show the code with no explanation or introducti
ons. Simplify the code: - take function arguments fr
om comment - remove dead assignments - refactor goto
 with for/if/while - use better names for variables
[0x00000000]>
```

Options

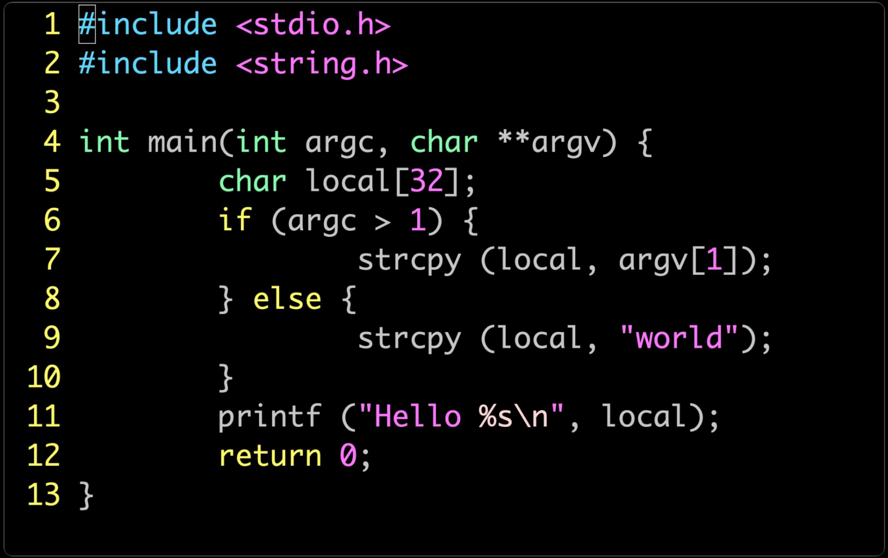
There are few things we can tune here:

- **Cache**: uses “ano” to save decompile output to save the planet
- **Lang**: can be anything, from C to Python, Swift or even Bash
- **Cmds**: comma separated list of commands to run
  - decai -e cmds=pdg,pdd,pdc
- **HLang**: select the human langauge to use for explanations and comments (default is english)

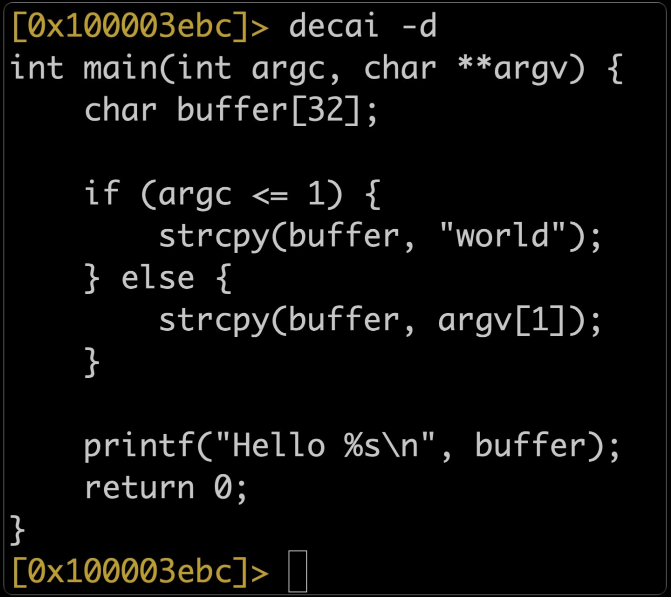
Demo Time

Decompiling a Vulnerable Program

Source and Decai based decompilation

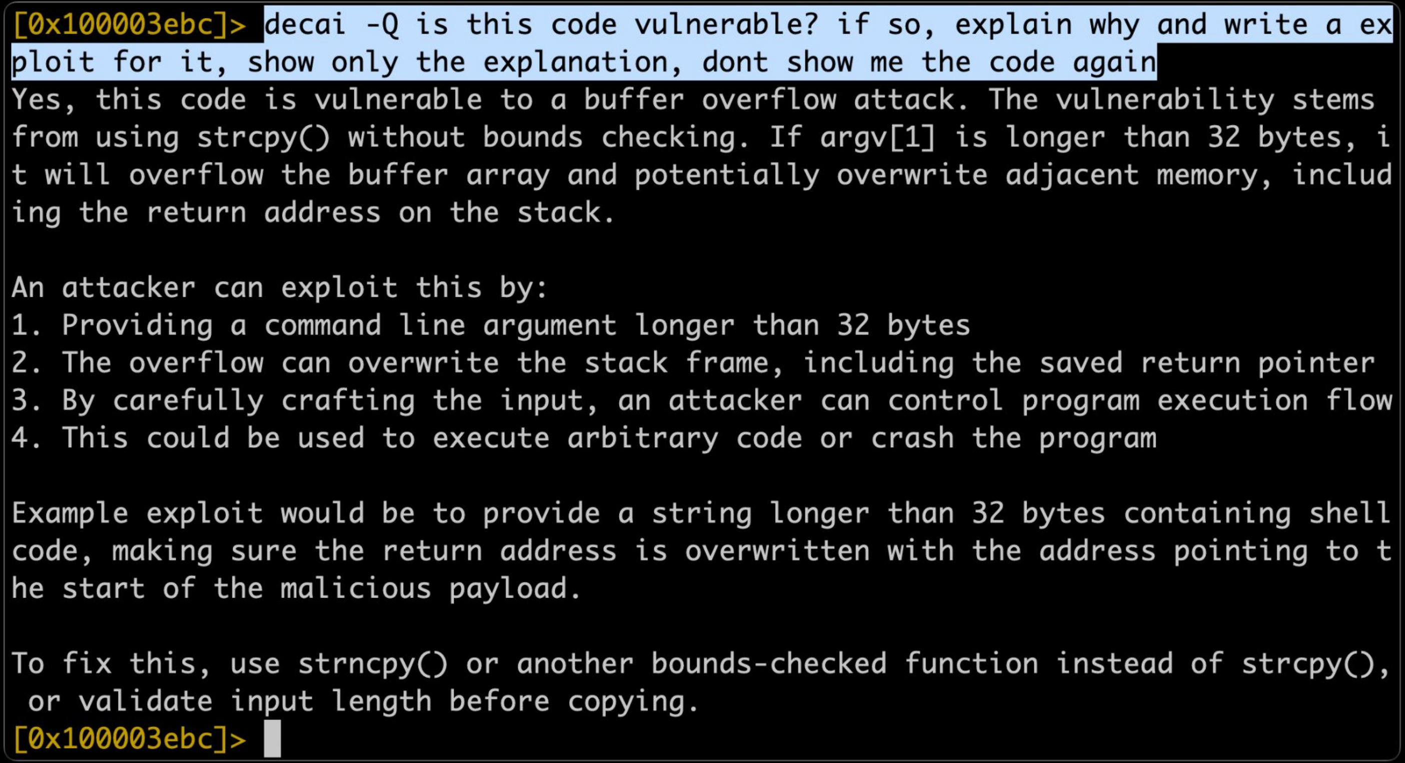


```c
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
        char local[32];
        if (argc > 1) {
                strcpy (local, argv[1]);
        } else {
                strcpy (local, "world");
        }
        printf ("Hello %s\n", local);
        return 0;
}
```



```
[0x100003ebc]> decai -d
int main(int argc, char **argv) {
    char buffer[32];

    if (argc <= 1) {
        strcpy(buffer, "world");
    } else {
        strcpy(buffer, argv[1]);
    }

    printf("Hello %s\n", buffer);
    return 0;
}
[0x100003ebc]> 
```

Finding the bug and writing the exploit



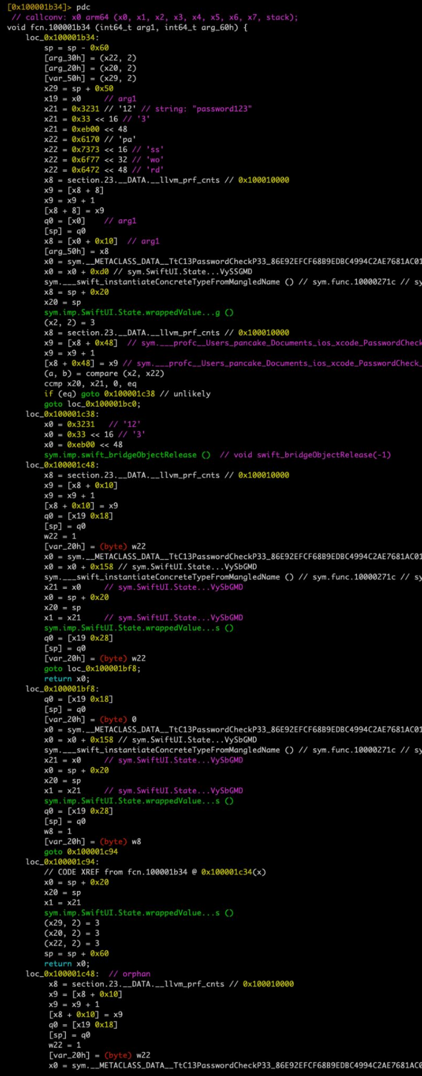
```
[0x100003ebc]> decai -Q is this code vulnerable? if so, explain why and write a ex
ploit for it, show only the explanation, dont show me the code again
Yes, this code is vulnerable to a buffer overflow attack. The vulnerability stems
from using strcpy() without bounds checking. If argv[1] is longer than 32 bytes, i
t will overflow the buffer array and potentially overwrite adjacent memory, includ
ing the return address on the stack.

An attacker can exploit this by:
1. Providing a command line argument longer than 32 bytes
2. The overflow can overwrite the stack frame, including the saved return pointer
3. By carefully crafting the input, an attacker can control program execution flow
4. This could be used to execute arbitrary code or crash the program

Example exploit would be to provide a string longer than 32 bytes containing shell
code, making sure the return address is overwritten with the address pointing to t
he start of the malicious payload.

To fix this, use strncpy() or another bounds-checked function instead of strcpy(),
 or validate input length before copying.
[0x100003ebc]> 
```

DEMO: Decompiling Swift

Decompiling Swift Crackme

Using the internal pseudodecompiler.. (pdc)

```swift
func checkPassword(_ s: String) -> Bool {
    return "password123" == s;
}
```

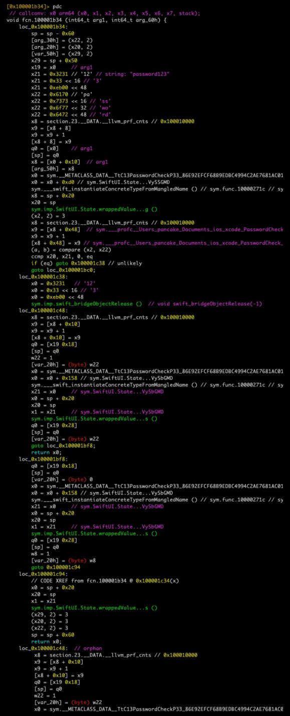


```
[0x100001b34]> pdc
 // callconv: x0 arm64 (x0, x1, x2, x3, x4, x5, x6, x7, stack);
void fcn.100001b34 (int64_t arg1, int64_t arg_60h) {
    loc_0x100001b34:
        sp = sp - 0x60
        [arg_30h] = (x22, 2)
        [arg_20h] = (x20, 2)
        [var_50h] = (x29, 2)
        x29 = sp + 0x50
        x19 = x0      // arg1
        x21 = 0x3231 // '12' // string: "password123"
        x21 = 0x33 << 16 // '3'
        x21 = 0xeb00 << 48
        x22 = 0x6170 // 'pa'
        x22 = 0x7373 << 16 // 'ss'
        x22 = 0x6f77 << 32 // 'wo'
        x22 = 0x6472 << 48 // 'rd'
        x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
        x9 = [x8 + 8]
        x9 = x9 + 1
        [x8 + 8] = x9
        q0 = [x0]     // arg1
        [sp] = q0
        x8 = [x0 + 0x10]  // arg1
        [arg_50h] = x8
        x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC01
        x0 = x0 + 0xd0 // sym.SwiftUI.State...VySSGMD
        sym.___swift_instantiateConcreteTypeFromMangledName () // sym.func.10000271c // sy
        x8 = sp + 0x20
        x20 = sp
        sym.imp.SwiftUI.State.wrappedValue...g ()
        (x2, 2) = 3
        x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
        x9 = [x8 + 0x48]  // sym.___profc__Users_pancake_Documents_ios_xcode_PasswordCheck
        x9 = x9 + 1
        [x8 + 0x48] = x9 // sym.___profc__Users_pancake_Documents_ios_xcode_PasswordCheck_
        (a, b) = compare (x2, x22)
        ccmp x20, x21, 0, eq
        if (eq) goto 0x100001c38 // unlikely
        goto loc_0x100001bc0;
    loc_0x100001c38:
        x0 = 0x3231   // '12'
        x0 = 0x33 << 16 // '3'
        x0 = 0xeb00 << 48
        sym.imp.swift_bridgeObjectRelease ()  // void swift_bridgeObjectRelease(-1)
    loc_0x100001c48:
        x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
        x9 = [x8 + 0x10]
        x9 = x9 + 1
        [x8 + 0x10] = x9
        q0 = [x19 0x18]
        [sp] = q0
        w22 = 1
        [var_20h] = (byte) w22
        x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC01
        x0 = x0 + 0x158 // sym.SwiftUI.State...VySbGMD
        sym.___swift_instantiateConcreteTypeFromMangledName () // sym.func.10000271c // sy
        x21 = x0      // sym.SwiftUI.State...VySbGMD
        x0 = sp + 0x20
        x20 = sp
        x1 = x21      // sym.SwiftUI.State...VySbGMD
        sym.imp.SwiftUI.State.wrappedValue...s ()
        q0 = [x19 0x28]
        [sp] = q0
        [var_20h] = (byte) w22
        goto loc_0x100001bf8;
        return x0;
    loc_0x100001bf8:
        q0 = [x19 0x18]
        [sp] = q0
        [var_20h] = (byte) 0
        x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC01
        x0 = x0 + 0x158 // sym.SwiftUI.State...VySbGMD
        sym.___swift_instantiateConcreteTypeFromMangledName () // sym.func.10000271c // sy
        x21 = x0      // sym.SwiftUI.State...VySbGMD
        x0 = sp + 0x20
        x20 = sp
        x1 = x21      // sym.SwiftUI.State...VySbGMD
        sym.imp.SwiftUI.State.wrappedValue...s ()
        q0 = [x19 0x28]
        [sp] = q0
        w8 = 1
        [var_20h] = (byte) w8
        goto 0x100001c94
    loc_0x100001c94:
        // CODE XREF from fcn.100001b34 @ 0x100001c34(x)
        x0 = sp + 0x20
        x20 = sp
        x1 = x21
        sym.imp.SwiftUI.State.wrappedValue...s ()
        (x29, 2) = 3
        (x20, 2) = 3
        (x22, 2) = 3
        sp = sp + 0x60
        return x0;
    loc_0x100001c48:  // orphan
         x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
         x9 = [x8 + 0x10]
         x9 = x9 + 1
         [x8 + 0x10] = x9
         q0 = [x19 0x18]
         [sp] = q0
         w22 = 1
         [var_20h] = (byte) w22
         x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC0
```

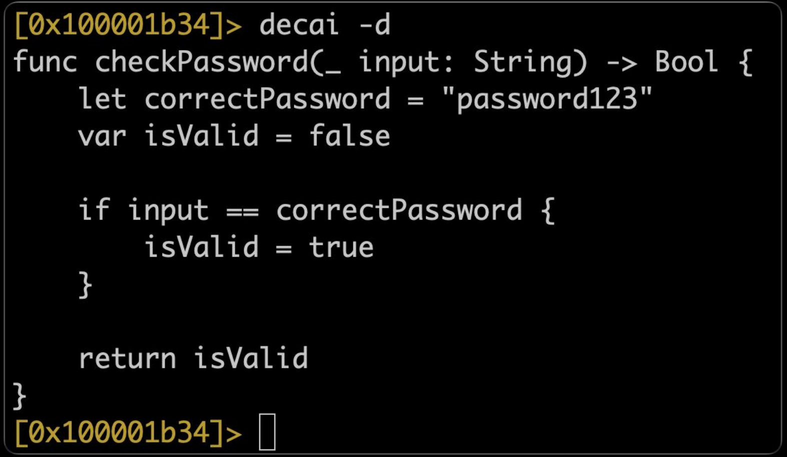
Decompiling Swift Crackme

Feeding that boring pdc output into decai

```swift
func checkPassword(_ s: String) -> Bool {
    return "password123" == s;
}
```



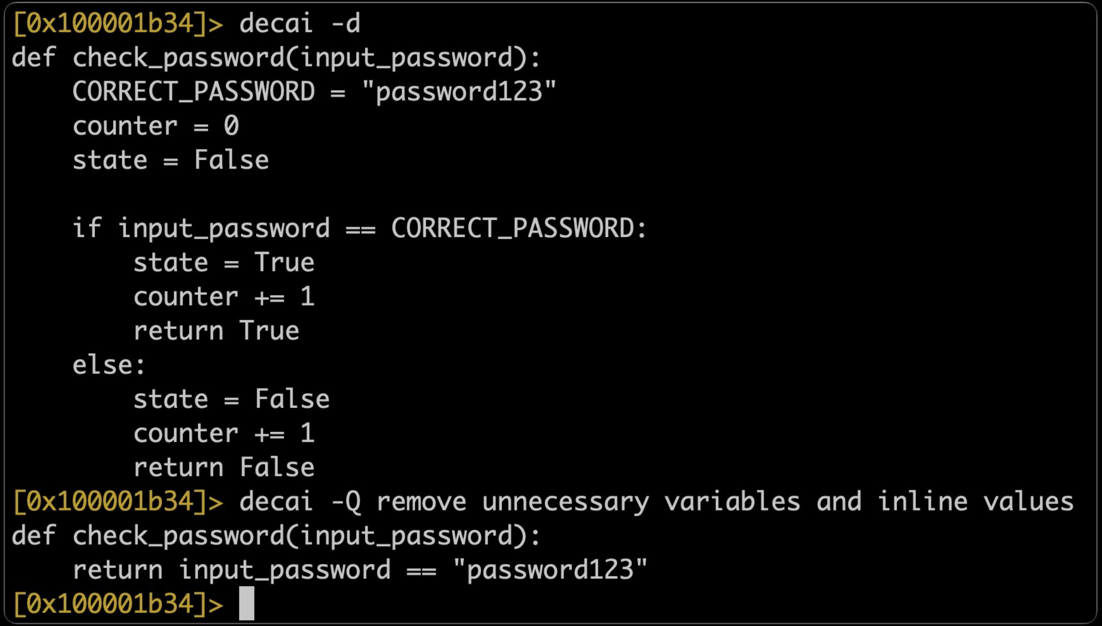
```
[0x100001b34]> pdc
 // callconv: x0 arm64 (x0, x1, x2, x3, x4, x5, x6, x7, stack);
void fcn.100001b34 (int64_t arg1, int64_t arg_60h) {
    loc_0x100001b34:
        sp = sp - 0x60
        [arg_30h] = (x22, 2)
        [arg_20h] = (x20, 2)
        [var_50h] = (x29, 2)
        x29 = sp + 0x50
        x19 = x0     // arg1
        x21 = 0x3231 // '12' // string: "password123"
        x21 = 0x33 << 16 // '3'
        x21 = 0xeb00 << 48
        x22 = 0x6170 // 'pa'
        x22 = 0x7373 << 16 // 'ss'
        x22 = 0x6f77 << 32 // 'wo'
        x22 = 0x6472 << 48 // 'rd'
        x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
        x9 = [x8 + 8]
        x9 = x9 + 1
        [x8 + 8] = x9
        q0 = [x0]    // arg1
        [sp] = q0
        x8 = [x0 + 0x10]  // arg1
        [arg_50h] = x8
        x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC01
        x0 = x0 + 0xd0 // sym.SwiftUI.State...VySSGMD
        sym.___swift_instantiateConcreteTypeFromMangledName () // sym.func.10000271c // sy
        x8 = sp + 0x20
        x20 = sp
        sym.imp.SwiftUI.State.wrappedValue...g ()
        (x2, 2) = 3
        x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
        x9 = [x8 + 0x48]  // sym.___profc__Users_pancake_Documents_ios_xcode_PasswordCheck
        x9 = x9 + 1
        [x8 + 0x48] = x9 // sym.___profc__Users_pancake_Documents_ios_xcode_PasswordCheck_
        (a, b) = compare (x2, x22)
        ccmp x20, x21, 0, eq
        if (eq) goto 0x100001c38 // unlikely
        goto loc_0x100001bc0;
    loc_0x100001c38:
        x0 = 0x3231   // '12'
        x0 = 0x33 << 16 // '3'
        x0 = 0xeb00 << 48
        sym.imp.swift_bridgeObjectRelease ()  // void swift_bridgeObjectRelease(-1)
    loc_0x100001c48:
        x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
        x9 = [x8 + 0x10]
        x9 = x9 + 1
        [x8 + 0x10] = x9
        q0 = [x19 0x18]
        [sp] = q0
        w22 = 1
        [var_20h] = (byte) w22
        x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC01
        x0 = x0 + 0x158 // sym.SwiftUI.State...VySbGMD
        sym.___swift_instantiateConcreteTypeFromMangledName () // sym.func.10000271c // sy
        x21 = x0     // sym.SwiftUI.State...VySbGMD
        x0 = sp + 0x20
        x20 = sp
        x1 = x21     // sym.SwiftUI.State...VySbGMD
        sym.imp.SwiftUI.State.wrappedValue...s ()
        q0 = [x19 0x28]
        [sp] = q0
        [var_20h] = (byte) w22
        goto loc_0x100001bf8;
        return x0;
    loc_0x100001bf8:
        q0 = [x19 0x18]
        [sp] = q0
        [var_20h] = (byte) 0
        x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC01
        x0 = x0 + 0x158 // sym.SwiftUI.State...VySbGMD
        sym.___swift_instantiateConcreteTypeFromMangledName () // sym.func.10000271c // sy
        x21 = x0     // sym.SwiftUI.State...VySbGMD
        x0 = sp + 0x20
        x20 = sp
        x1 = x21     // sym.SwiftUI.State...VySbGMD
        sym.imp.SwiftUI.State.wrappedValue...s ()
        q0 = [x19 0x28]
        [sp] = q0
        w8 = 1
        [var_20h] = (byte) w8
        goto 0x100001c94
    loc_0x100001c94:
        // CODE XREF from fcn.100001b34 @ 0x100001c34(x)
        x0 = sp + 0x20
        x20 = sp
        x1 = x21
        sym.imp.SwiftUI.State.wrappedValue...s ()
        (x29, 2) = 3
        (x20, 2) = 3
        (x22, 2) = 3
        sp = sp + 0x60
        return x0;
    loc_0x100001c48:  // orphan
         x8 = section.23.__DATA.__llvm_prf_cnts // 0x100010000
         x9 = [x8 + 0x10]
         x9 = x9 + 1
         [x8 + 0x10] = x9
         q0 = [x19 0x18]
         [sp] = q0
         w22 = 1
         [var_20h] = (byte) w22
         x0 = sym.__METACLASS_DATA__TtC13PasswordCheckP33_86E92EFCF68B9EDBC4994C2AE7681AC0
```



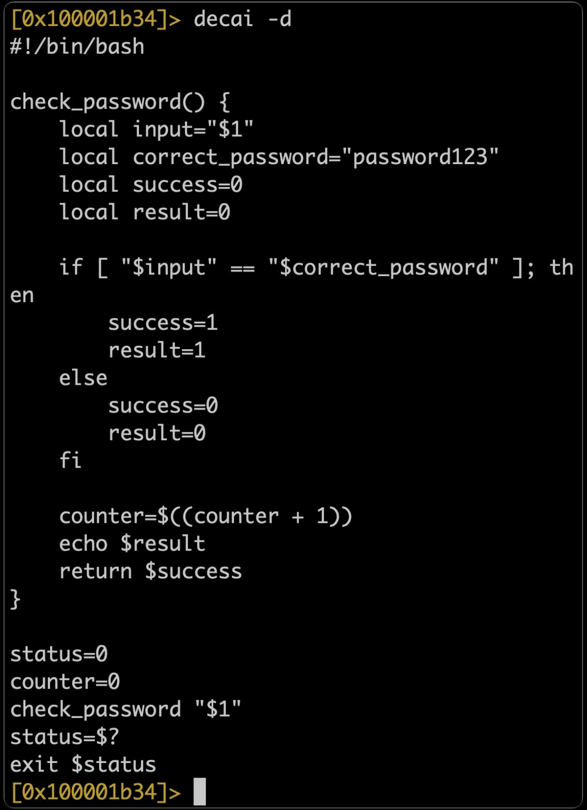
```
[0x100001b34]> decai -d
func checkPassword(_ input: String) -> Bool {
    let correctPassword = "password123"
    var isValid = false

    if input == correctPassword {
        isValid = true
    }

    return isValid
}
[0x100001b34]> 
```

Not just swift, we can choose other languages

Prefer Python? Or maybe Bash?



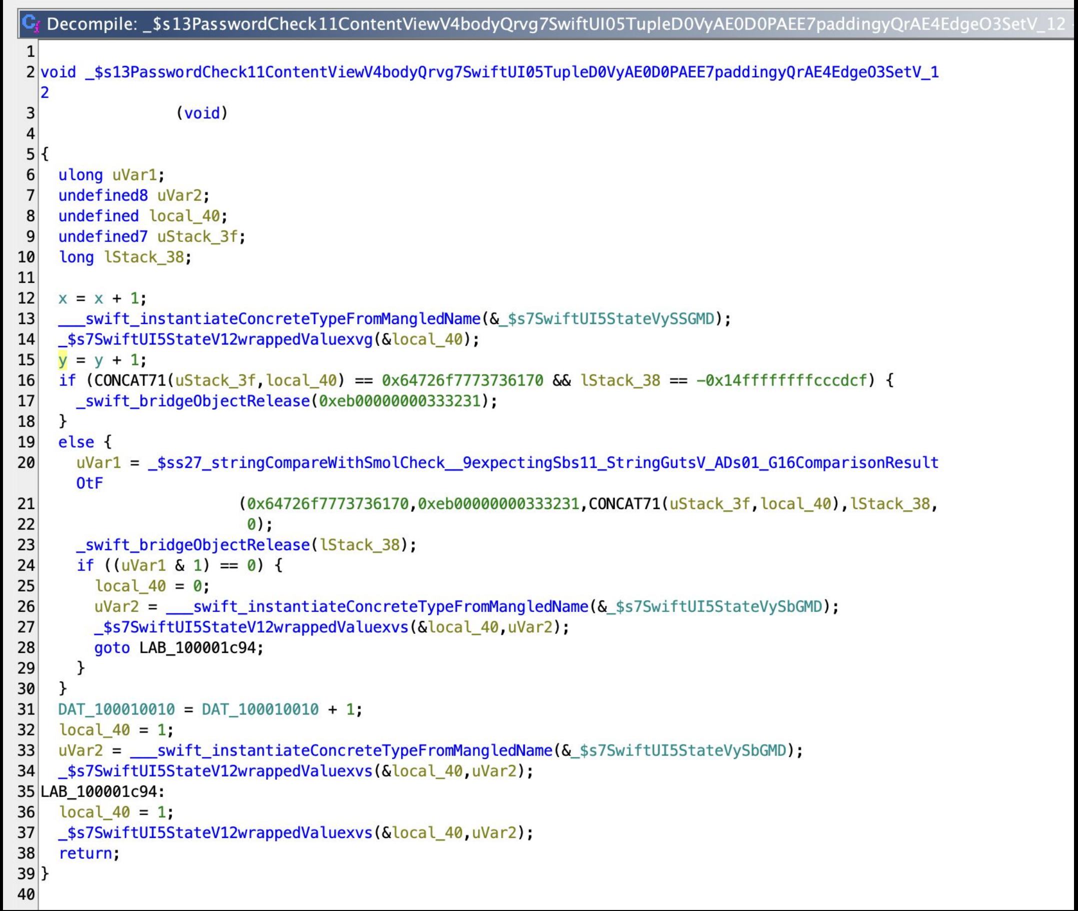
```
[0x100001b34]> decai -d
def check_password(input_password):
    CORRECT_PASSWORD = "password123"
    counter = 0
    state = False

    if input_password == CORRECT_PASSWORD:
        state = True
        counter += 1
        return True
    else:
        state = False
        counter += 1
        return False
[0x100001b34]> decai -Q remove unnecessary variables and inline values
def check_password(input_password):
    return input_password == "password123"
[0x100001b34]> 
```



```
[0x100001b34]> decai -d
#!/bin/bash

check_password() {
    local input="$1"
    local correct_password="password123"
    local success=0
    local result=0

    if [ "$input" == "$correct_password" ]; th
en
        success=1
        result=1
    else
        success=0
        result=0
    fi

    counter=$((counter + 1))
    echo $result
    return $success
}

status=0
counter=0
check_password "$1"
status=$?
exit $status
[0x100001b34]> 
```

What about ghidra?

IDA is even worse


Decompile: _$s13PasswordCheck11ContentViewV4bodyQrvg7SwiftUI05TupleD0VyAE0D0PAEE7paddingyQrAE4EdgeO3SetV_12

```
void _$s13PasswordCheck11ContentViewV4bodyQrvg7SwiftUI05TupleD0VyAE0D0PAEE7paddingyQrAE4EdgeO3SetV_1
2
               (void)

{
  ulong uVar1;
  undefined8 uVar2;
  undefined local_40;
  undefined7 uStack_3f;
  long lStack_38;

  x = x + 1;
  ___swift_instantiateConcreteTypeFromMangledName(&_$s7SwiftUI5StateVySSGMD);
  _$s7SwiftUI5StateV12wrappedValuexvg(&local_40);
  y = y + 1;
  if (CONCAT71(uStack_3f,local_40) == 0x64726f7773736170 && lStack_38 == -0x14ffffffffcccdcf) {
    _swift_bridgeObjectRelease(0xeb00000000333231);
  }
  else {
    uVar1 = _$ss27_stringCompareWithSmolCheck__9expectingSbs11_StringGutsV_ADs01_G16ComparisonResult
    OtF
                      (0x64726f7773736170,0xeb00000000333231,CONCAT71(uStack_3f,local_40),lStack_38,
                       0);
    _swift_bridgeObjectRelease(lStack_38);
    if ((uVar1 & 1) == 0) {
      local_40 = 0;
      uVar2 = ___swift_instantiateConcreteTypeFromMangledName(&_$s7SwiftUI5StateVySbGMD);
      _$s7SwiftUI5StateV12wrappedValuexvs(&local_40,uVar2);
      goto LAB_100001c94;
    }
  }
  DAT_100010010 = DAT_100010010 + 1;
  local_40 = 1;
  uVar2 = ___swift_instantiateConcreteTypeFromMangledName(&_$s7SwiftUI5StateVySbGMD);
  _$s7SwiftUI5StateV12wrappedValuexvs(&local_40,uVar2);
LAB_100001c94:
  local_40 = 1;
  _$s7SwiftUI5StateV12wrappedValuexvs(&local_40,uVar2);
  return;
}
```

Future

Future

Decai was proven to be useful to decompile code from many architectures, even stm8, riscv or msil.

- Improve PDC output to make things easier for the inference
- Play with XML outputs to extract decompilation information
- Customizable and auto-improved prompts
- Improved data sets optimized for decompilation
- Rewrite in C and remove curl dependency, just use RSocket

Questions?