v1.0.0

VPAND.COM

Virtual Processor Infrastructure for Code Reverse Engineer and VMProtect.

Loading very hard for Qt.wasm, wait a second please...

Powered by LLVM/Clang.

vpand.com

UraniumVM SDK User Manual

UraniumVM

UraniumVM is a lightweight VM infrastructure for **arm/arm64/x86/x86_64** on **Android/iOS/Windows/macOS/Linux**, which can execute any native instructions in your 100% control. Based on it, you can develop your own tracer, vmp, vm, etc.(**ClangVMP**, **PhoneVMP** and **UVMDbg** in A64Dbg are based on UraniumVM). UraniumVM is **an instruction level virtual machine in function unit** for a **process environment**, which can be understood by the following concept table:

Concept	Explanation
Process environment	a running process in the real system, e.g.: apps, services, and daemons in iOS and Android systems.
Instruction level	the x86, x86_64, armv7, arm64, arm64e instruction set.
Function	the binary C function in machine instruction mode.
Virtual machine	a virtual machine based on **UraniumVCPU** that can execute instructions one by one.

Sum up all the concepts, UraniumVM is a virtual machine running in a real physical process environment with assembly functions as the minimum virtualization unit and a single instruction as the minimum execution unit.

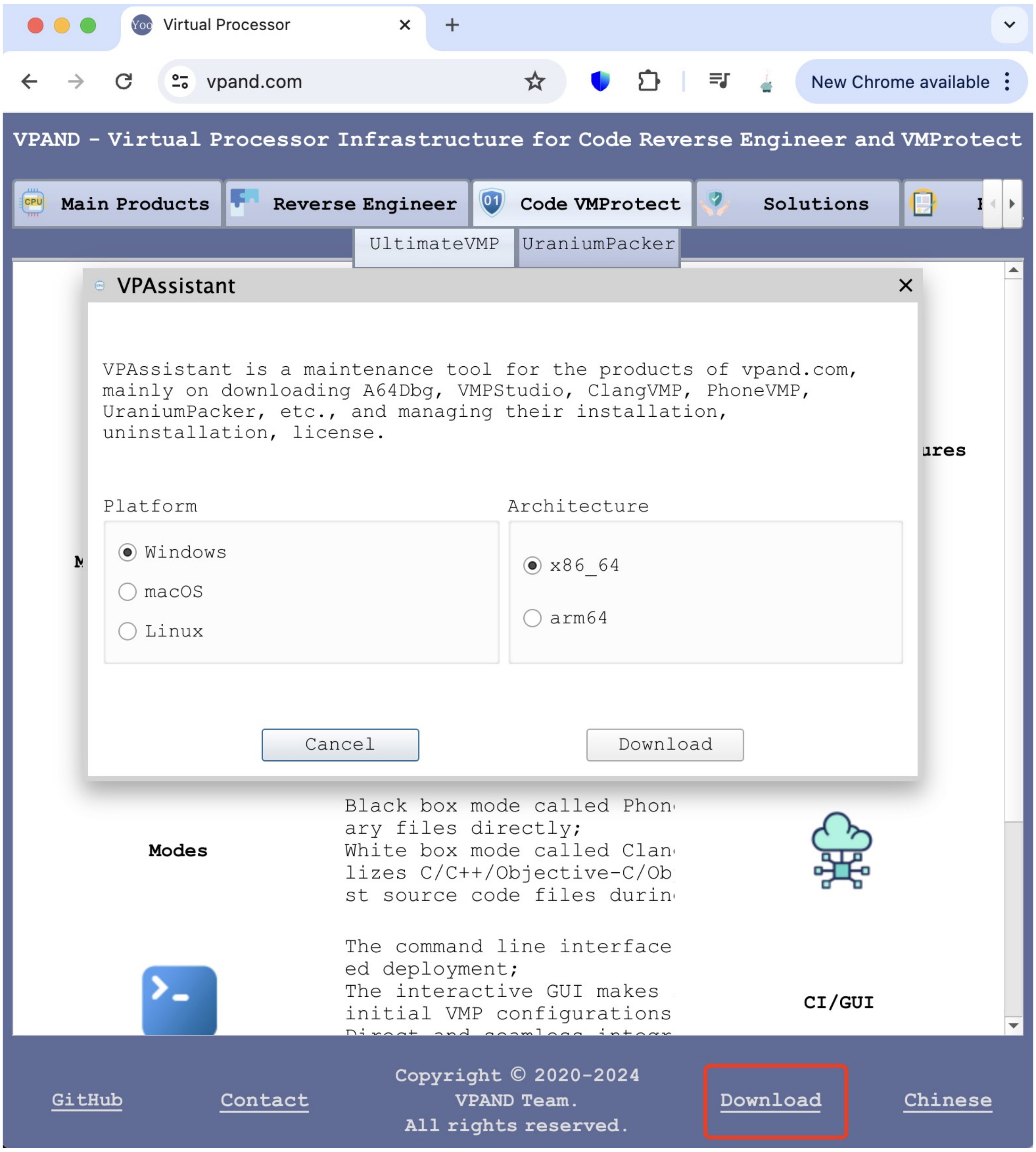
UraniumVM compares to Unicorn Engine:

	UraniumVM	Unicorn
Environment	real os process	simulation environment
Virtual CPU	UraniumVCPU	Qemu
Cross Architecture	No	Yes
Performance	high	low
License	vpand.com	**GPL**

Download

All of our products are hosted in vpand.com, you can download the assistant tool called **VPAssistant** to fetch your interested product like UraniumVM.

Please note that you should download **the exact platform and architecture** which matches your current running OS, all the real product download through VPAssistant highly depends on the architecture that it's running on. For the native performance, you'd better not download x86_64 version on arm64 macOS even though it can directly run on it with Rosetta 2.



VPAssistant

VPAssistant is a maintenance tool for the products of vpand.com, mainly on downloading A64Dbg, VMPStudio, ClangVMP, PhoneVMP, UraniumPacker, etc., and managing their installation, uninstallation, license.

Platform: [✓ Windows] macOS Linux

Architecture: [✓ x86_64] arm64

Cancel Download

Black box mode called Phon
ary files directly;
White box mode called Clan
lizes C/C++/Objective-C/Ob
st source code files durin

The command line interface
ed deployment;
The interactive GUI makes
initial VMP configurations

Copyright © 2020-2024 VPAND Team. All rights reserved.

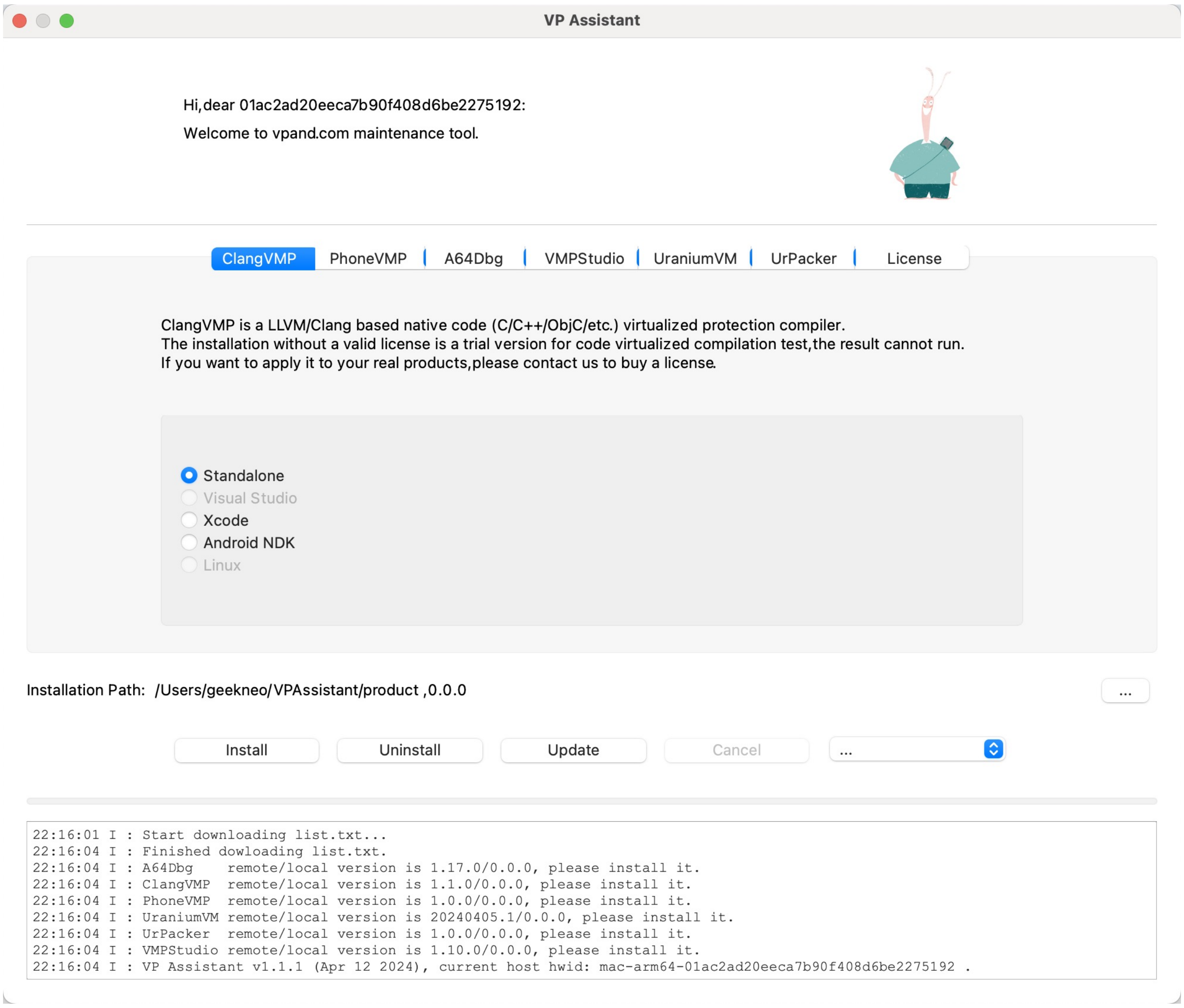
Install

As the VPAssistant on diffrerent platform(like Windows, Linux and macOS) is **absolutely the same**, unless some feature is just available on that specified platform, otherwise all the generic feature screenshots are from macOS. Here we go.

After download, unzip and launch VPAssistant, we're gonna be in the ClangVMP tab widget, then click the **UraniumVM** tab to activate the assistant for UraniumVM. The default installation path is(You can **change it with the "..." button** on the right):

```
macOS/Linux : ~/VPAssistant/product

Windows     : SysDrive:\Users\user-name\VPAssistant\product
```



Hi,dear 01ac2ad20eeca7b90f408d6be2275192:

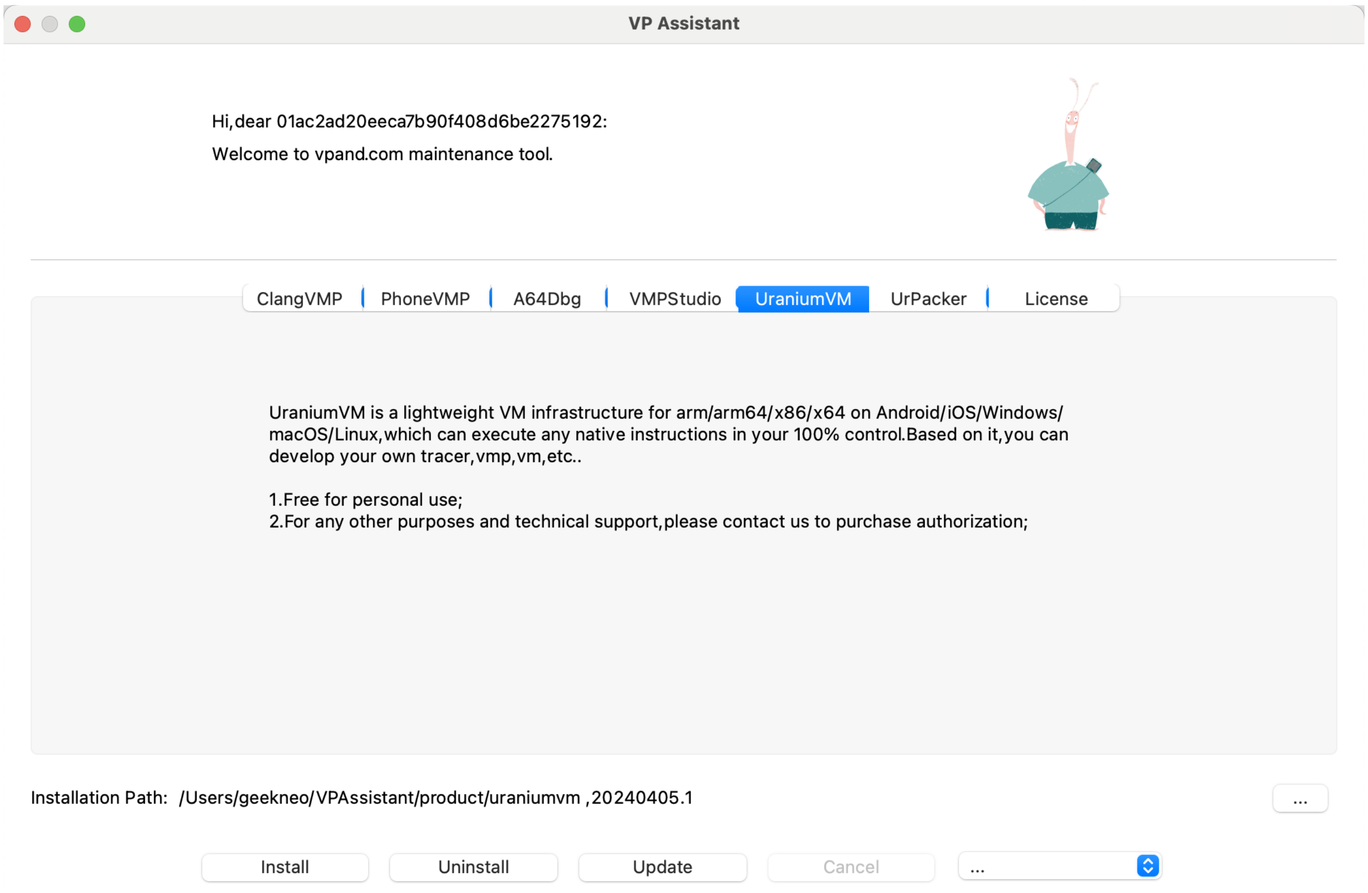
Welcome to vpand.com maintenance tool.

ClangVMP is a LLVM/Clang based native code (C/C++/ObjC/etc.) virtualized protection compiler.
The installation without a valid license is a trial version for code virtualized compilation test,the result cannot run.
If you want to apply it to your real products,please contact us to buy a license.

[✓ Standalone]
Visual Studio
Xcode
Android NDK
Linux

Installation Path: /Users/geekneo/VPAssistant/product ,0.0.0

```
22:16:01 I : Start downloading list.txt...
22:16:04 I : Finished dowloading list.txt.
22:16:04 I : A64Dbg    remote/local version is 1.17.0/0.0.0, please install it.
22:16:04 I : ClangVMP  remote/local version is 1.1.0/0.0.0, please install it.
22:16:04 I : PhoneVMP  remote/local version is 1.0.0/0.0.0, please install it.
22:16:04 I : UraniumVM remote/local version is 20240405.1/0.0.0, please install it.
22:16:04 I : UrPacker  remote/local version is 1.0.0/0.0.0, please install it.
22:16:04 I : VMPStudio remote/local version is 1.10.0/0.0.0, please install it.
22:16:04 I : VP Assistant v1.1.1 (Apr 12 2024), current host hwid: mac-arm64-01ac2ad20eeca7b90f408d6be2275192 .
```



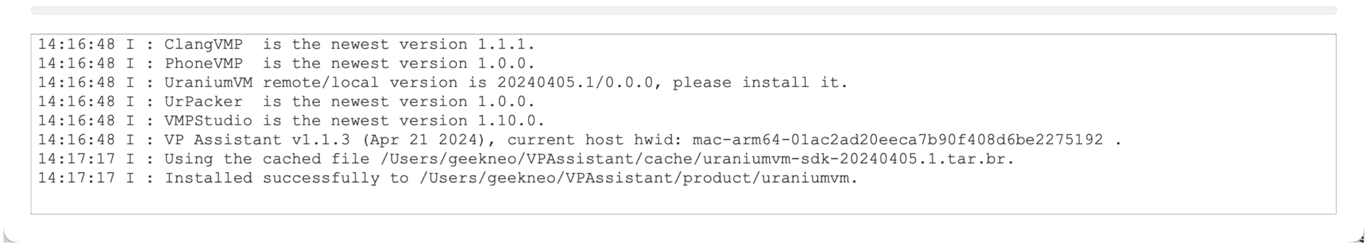
Hi,dear 01ac2ad20eeca7b90f408d6be2275192:

Welcome to vpand.com maintenance tool.

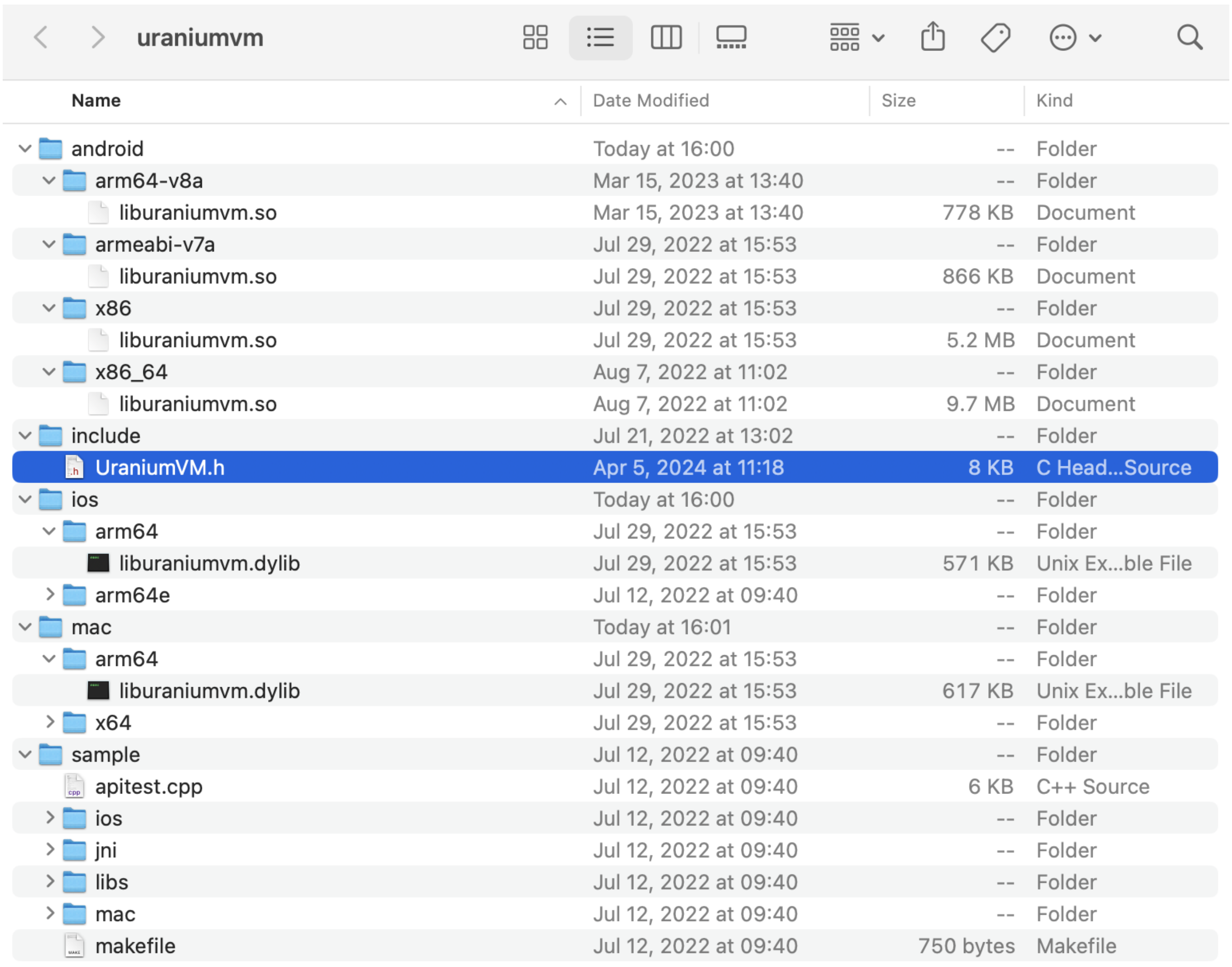
UraniumVM is a lightweight VM infrastructure for arm/arm64/x86/x64 on Android/iOS/Windows/macOS/Linux,which can execute any native instructions in your 100% control.Based on it,you can develop your own tracer,vmp,vm,etc..

1.Free for personal use;
2.For any other purposes and technical support,please contact us to purchase authorization;

Installation Path: /Users/geekneo/VPAssistant/product/uraniumvm ,20240405.1



```
14:16:48 I : ClangVMP  is the newest version 1.1.1.
14:16:48 I : PhoneVMP  is the newest version 1.0.0.
14:16:48 I : UraniumVM remote/local version is 20240405.1/0.0.0, please install it.
14:16:48 I : UrPacker  is the newest version 1.0.0.
14:16:48 I : VMPStudio is the newest version 1.10.0.
14:16:48 I : VP Assistant v1.1.3 (Apr 21 2024), current host hwid: mac-arm64-01ac2ad20eeca7b90f408d6be2275192 .
14:17:17 I : Using the cached file /Users/geekneo/VPAssistant/cache/uraniumvm-sdk-20240405.1.tar.br.
14:17:17 I : Installed successfully to /Users/geekneo/VPAssistant/product/uraniumvm.
```



uraniumvm

Name	Date Modified	Size	Kind
android	Today at 16:00	--	Folder
arm64-v8a	Mar 15, 2023 at 13:40	--	Folder
liburaniumvm.so	Mar 15, 2023 at 13:40	778 KB	Document
armeabi-v7a	Jul 29, 2022 at 15:53	--	Folder
liburaniumvm.so	Jul 29, 2022 at 15:53	866 KB	Document
x86	Jul 29, 2022 at 15:53	--	Folder
liburaniumvm.so	Jul 29, 2022 at 15:53	5.2 MB	Document
x86_64	Aug 7, 2022 at 11:02	--	Folder
liburaniumvm.so	Aug 7, 2022 at 11:02	9.7 MB	Document
include	Jul 21, 2022 at 13:02	--	Folder
UraniumVM.h	Apr 5, 2024 at 11:18	8 KB	C Head...Source
ios	Today at 16:00	--	Folder
arm64	Jul 29, 2022 at 15:53	--	Folder
liburaniumvm.dylib	Jul 29, 2022 at 15:53	571 KB	Unix Ex...ble File
arm64e	Jul 12, 2022 at 09:40	--	Folder
mac	Today at 16:01	--	Folder
arm64	Jul 29, 2022 at 15:53	--	Folder
liburaniumvm.dylib	Jul 29, 2022 at 15:53	617 KB	Unix Ex...ble File
x64	Jul 29, 2022 at 15:53	--	Folder
sample	Jul 12, 2022 at 09:40	--	Folder
apitest.cpp	Jul 12, 2022 at 09:40	6 KB	C++ Source
ios	Jul 12, 2022 at 09:40	--	Folder
jni	Jul 12, 2022 at 09:40	--	Folder
libs	Jul 12, 2022 at 09:40	--	Folder
mac	Jul 12, 2022 at 09:40	--	Folder
makefile	Jul 12, 2022 at 09:40	750 bytes	Makefile

API

uvm_run_interp

```
// interpreter context
typedef struct uvm_context_t {
  void *usrctx;
  uvm_regs_t uvmctx;
  uvm_interp_callback_t callback;
} uvm_context_t;

// run function 'fn' on UraniumVCPU with 'ctx'
// return value is r[0].sx/rax
__URANIUM_VCPU_API__ long uvm_run_interp(const void *fn, const
uvm_context_t *ctx);
```

All you need to do is passing a function and its accompanying register context into it, then you can execute the function in the UraniumVM environment in the real process under your callback function control.

uvm_make_callee

```
// this api is used to make target's function pointer under your control
//
// make a wrapper for function 'fn' with 'usrctx','callback'
// return value is a new function pointer which will run under our VCPU
// you can replace this pointer to target's function pointer
// like C++-Vtable/Script-Native-Bridge
// if return null, you should check errno
__URANIUM_VCPU_API__ const void *uvm_make_callee(const void *fn,
                                                 void *usrctx,
                                                 uvm_interp_callback_t
callback);
```

All you need to do is passing a function into it and it will return a new function with the same functionality but executed in UraniumVM under your callback function control. You can replace any C++-Vtable with this function pointer, or use it as an argument to InlineHook/GotHook.

UVM Parameters

uvm_regs_t

The API uvm_run_interp needs a initial register context which is defined by uvm_context_t.uvm_regs_t:

```
// uvm execution context
#if __a32__
typedef arm_regs_t uvm_regs_t;
#elif __a64__
typedef arm64_regs_t uvm_regs_t;
#else
typedef intel_regs_t uvm_regs_t;
#endif
```

arm_regs_t

```
// arm common register like r0 r1 ... lr sp pc
typedef union arm_cmmreg_t {
  unsigned int w;
  unsigned int x;  // make the same name as arm64
  int sw;
  int sx;
  const void *p;
  const char *s;
} arm_cmmreg_t;

// arm neon register like s0 d0 ...
typedef union arm_neonreg_t {
  unsigned int i[2];
  unsigned long long l;
  int si[2];
  long long sl;
  float f[2];
  double d;
} arm_neonreg_t;

// special register wrapper
#define ARM_FP(c) (c).r[12]
#define ARM_SP(c) (c).r[13]
#define ARM_LR(c) (c).r[14]
#define ARM_PC(c) (c).pc

// arm execution context
typedef struct arm_regs_t {
  arm_cmmreg_t r[16];  // 0-12, 13-sp, 14-lr, 15-reserved
  arm_neonreg_t v[32];
  arm_cmmreg_t pc;
} arm_regs_t;
```

arm64_regs_t

```
// arm64 common register like x0 x1 ... lr sp pc
typedef union arm64_cmmreg_t {
  unsigned int w;
  unsigned long long x;
  int sw;
  long long sx;
  const void *p;
  const char *s;
  unsigned int ws[2];
  int sws[2];
} arm64_cmmreg_t;

// arm64 neon register like s0 d0 q0 ...
typedef union arm64_neonreg_t {
  unsigned int i[4];
  unsigned long long l[2];
  int si[4];
  long long sl[2];
} arm64_neonreg_t;

// special register wrapper
#define ARM64_FP(c) (c).r[29]
#define ARM64_LR(c) (c).r[30]
#define ARM64_SP(c) (c).r[31]
```

```
#define ARM64_PC(c) (c).pc

// arm64 execution context
typedef struct arm64_regs_t {
  arm64_cmmreg_t r[32];  // 0-28,29-fp,30-lr,31-sp
  arm64_neonreg_t v[32];
  arm64_cmmreg_t pc;
} arm64_regs_t;
```

intel_regs_t

```
// x86/x64 common register like eax/rax ...
typedef union intel_cmmreg_t {
  void *p;
  char *s;
  unsigned long long q;
  long long sq;
  unsigned long long x;
  long long sx;
  unsigned int l;
  int sl;
  unsigned int w[2];
  int sw[2];
  unsigned short h[4];
  short sh[4];
  unsigned char b[8];
  char sb[8];
} intel_cmmreg_t;

// x86/x64 sse register like xmm0 ...
typedef union intel_ssereg_t {
  intel_cmmreg_t i[2];
  double d[2];
  float f[4];
} intel_ssereg_t;

// x86/x64 float register like mm0/st(0) ...
typedef union intel_floatreg_t {
  intel_cmmreg_t i;
  char fp80[10];
  double d;
} intel_floatreg_t;

// x86/64 state register
typedef struct intel_state_t {
  unsigned long long rflags;
  unsigned short fctrl;
  unsigned char fstats[26];
} intel_state_t;

// x86/x64 execution context
```

```
typedef struct intel_regs_t {
  intel_cmmreg_t rax;
  intel_cmmreg_t rbx;
  intel_cmmreg_t rcx;
  intel_cmmreg_t rdx;
  intel_cmmreg_t rbp;
  intel_cmmreg_t rsi;
  intel_cmmreg_t rdi;
  intel_cmmreg_t r8;
  intel_cmmreg_t r9;
  intel_cmmreg_t r10;
  intel_cmmreg_t r11;
  intel_cmmreg_t r12;
  intel_cmmreg_t r13;
  intel_cmmreg_t r14;
  intel_cmmreg_t r15;
  intel_ssereg_t xmm[16];
  intel_state_t state;
  intel_floatreg_t stmmx[8];
  intel_cmmreg_t rsp;
  intel_cmmreg_t pc;
} intel_regs_t;
```

uvm_interp_callback_t

```
// callback prototype
typedef uvm_callback_return_t (*uvm_interp_callback_t)(
    uvm_callback_args_t *args);
```

This callback function is called by the UraniumVM runtime when hitting special events, so that the user can decide how to handle advanced operations such as rewriting memory, function redirection, modifying system call parameters, and so on.

uvm_callback_args_t

Callback events are defined as follows:

```
// opcode type for callback args
typedef enum uvm_optype_t {
  vcop_read,    // memory read
  vcop_write,   // memory write
  vcop_call,    // function call
  vcop_return,  // function return
#if __ARM__
  vcop_svc,  // arm syscall
#else
  vcop_syscall,  // intel syscall
#endif
```

```
  vcop_ifetch,  // interpreter fetch instruction
} uvm_optype_t;
```

Callback parameters are defined as follows:

```
// callback args
typedef struct uvm_callback_args_t {
  // your own context passed for uvm_run_interp/uvm_make_callee
  const void *usrctx;
  // uvm execution context
  uvm_regs_t *uvmctx;
  // current opcode
  uvm_optype_t op;
  union {
    // for vcop_read/vcop_write/vcop_ifetch
    struct {
      const void *src;
      void *dst;
      int byte;
    } rw;
    // for vcop_call
    struct {
      const void *callee;
    } call;
    // for vcop_return
    struct {
      const void *hitaddr;  // which address hit return
    } ret;
    // for vcop_svc
    struct {
      // arm
      // parameters are in armctx->r[0...6]
      // syscall number from armctx->r[7]
      //
      // arm64
      // parameters are in arm64ctx->x
      // syscall number
      //
      // x86/x64
      // ...
      int sysno;
    } svc;
  } info;
} uvm_callback_args_t;
```

It records the specific parameters of the callback event in detail. For example, when the **vcop_read** event occurs, **src, dst, and byte** in the **rw** field indicate the **memory source address, write address**, and **byte size** in turn. When the **vcop_svc** event occurs, the **sysno** in the **svc** field gives the call number of the current system call, and then the specific parameters of the system call can be obtained through the uvmctx on the virtual machine. Here's a demonstration callback function:

```
// log runtime information
static uvm_callback_return_t interp_callback_log(uvm_callback_args_t
*args) {
  FILE *logfp = (FILE *)args->usrctx;
  switch (args->op) {
    case vcop_read:
    case vcop_write: {
      break;
    }
    case vcop_call: {
      fprintf(logfp, "vcop call : func %p.\n", args->info.call.callee);
      break;
    }
    case vcop_return: {
      fprintf(logfp, "vcop return : hit address %p.\n", args-
>info.ret.hitaddr);
      break;
    }
#if __ARM__
    case vcop_svc:
#else
    case vcop_syscall:
#endif
    {
      fprintf(logfp, "vcop syscall : syscall number %d.\n",
              args->info.svc.sysno);
      break;
    }
    case vcop_ifetch: {
      break;
    }
    default: {
      fprintf(logfp, "unknown vcop %d.\n", args->op);
      break;
    }
  }
  return cbret_directcall;
  // return cbret_recursive;
}
```

uvm_callback_return_t

The return value of the callback function is used to indicate how the UraniumVM virtual machine interpreter should react, and is defined as follows:

```
// callback return type
typedef enum uvm_callback_return_t {
  cbret_continue,    // let interp continue
  cbret_processed,   // already processed by callback implementation
  cbret_recursive,   // interp this function recursively
  cbret_directcall,  // call this function directly
} uvm_callback_return_t;
```

- **cbret_continue**: tells the interpreter to continue interpreting and executing the current instruction;
- **cbret_processed**: tells the interpreter that the callback function has already processed current instruction, just ignores it;
- **cbret_recursive**: tells the interpreter to recursively execute the function currently called;
- **cbret_directcall**: tells the interpreter to switch to the host environment to execute the function currently called;

API Call Sample

```
// demo interpretee
int print_message(const char *reason, const char **argv);

// run interpretee directly
int vrun_print_message(const char *reason, const char **argv, FILE
*cblogfp,
                       uvm_interp_callback_t cb) {
  uvm_context_t ctx;
  memset(&ctx, 0, sizeof(ctx));
  ctx.usrctx = cblogfp;
  ctx.callback = cb;
#if __ARM__
  ctx.uvmctx.r[0].p = reason;
  ctx.uvmctx.r[1].p = (void *)argv;
#else
#if __x64__
  ctx.uvmctx.rdi.p = (void *)reason;
  ctx.uvmctx.rsi.p = (void *)argv;
#else
  // x86 uses stack to pass parameter
  void *args[3];
  args[0] = (void *)vrun_print_message; // simulate call return address
  args[1] = (void *)reason;
  args[2] = (void *)argv;
  ctx.uvmctx.rsp.p = (void *)&args[0];
#endif
#endif
  return (int)(long)uvm_run_interp((void *)print_message, &ctx);
}

// run interpretee with a wrapper
int wrapper_print_message(const char *reason, const char **argv, FILE
*cblogfp,
                          uvm_interp_callback_t cb) {
  const void *fnptr = uvm_make_callee((void *)print_message, cblogfp, cb);
  return ((int (*)(const char *, const char **))fnptr)(reason, argv);
}
```

License

- **Personal use** is free.
- **Any other use**, **technical support** or **custom requirements** should contact us to purchase authorization, the price is negotiable.

Contact us

Email

If you have any questions or problems on our products or services, feel free to contact us via email at anytime:

- **neoliu2011@gmail.com**

We-Media

Till now, we-media is our main operation method, you can also contact us via the following platforms:

- Facebook
- YouTube
- Reddit
- X
- Instagram