CORELLIUM

Solutions Brief

Mobile Vulnerability Research

The Corellium Arm virtualization platform provides never-before-possible security vulnerability research for iOS and Android phones with deep forensics and introspection tools.

Its high accuracy, Arm-native model enables real platform vulnerability discovery and exploit validation.

This is not an emulator or simulator — it's Arm on Arm.

This is basically magic.

- MARK DOWD, AZIMUTH SECURITY

Virtual iOS and Android Devices On-Demand

The Corellium hypervisor for Arm (CHARM) runs on native Arm processors, in the cloud or on server appliances. A single platform supports high-fidelity security tooling for both iOS and Android phones and tablets. Simply spin-up a near limitless combination of device and OS, from older versions to the very latest, patched or unpatched, jailbroken or not.

Cutting-Edge Vulnerability and Exploit Research

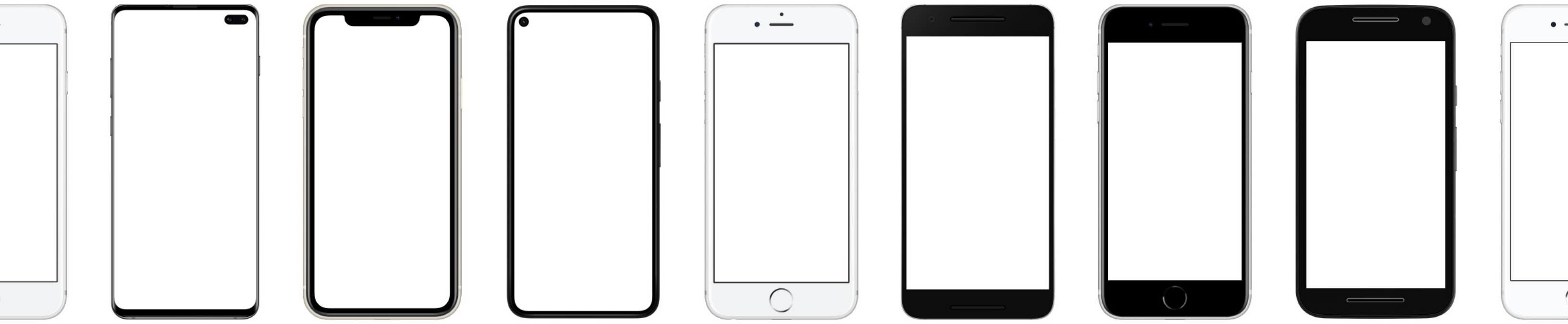
Corellium provides a powerful and polished user interface with built-in security tools for root access, process tracing, file system manipulation, Frida scripting, TLS-stripping network monitoring, kernel debugging, SEP/iBoot debugging, and much more. Combined with a comprehensive API and USBFlux technology, integrating with leading development, security, and forensics tools such as Xcode, Android Studio, IDA Pro, and GDB/LLDB is seamless.

Industries

- ✓ Government
- ✓ Independent Researchers
- ✓ Education

Roles

- ✓ Vulnerability Researchers
- ✓ Independent Verification & Validation (IV&V)
- ✓ Mobile Forensics



Why Choose Corellium for Mobile App Pen Testing

Spin-Up Needed Devices

Easily access near limitless combinations of device models and iOS and Android, from past models and releases to the very latest.

Jailbreak and Root Access

Corellium can root any device configuration, including the latest version of iOS, even when no public jailbreak is available.

Low-Level Debugging

Leverage built-in kernel, iBoot, and SEP debugger tools for unprecedented introspection that can't be performed on physical devices.

Hypervisor Hooks

Dynamically patch the kernel with a performant C-like language to modify runtime behavior, trigger breakpoints from user-space, log function arguments, and more.

Matrix Testing

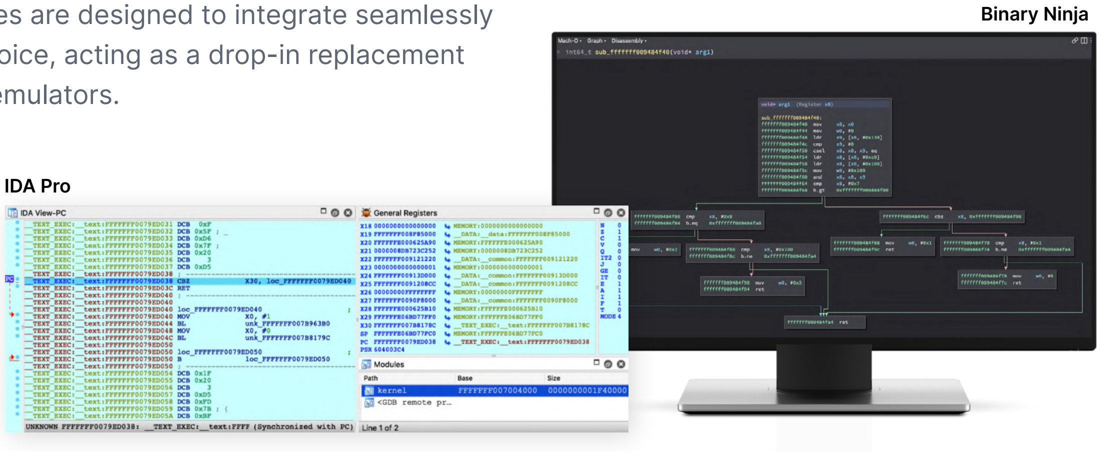
Script and automate vulnerability testing on multiple OS versions and device models simultaneously.

Comprehensive Integrations

Leverage the API and USBFlux to integrate with common tools, such as Xcode, Android Studio, IDA Pro, and GDB/LLDB.

Use with your favorite tools

Corellium's virtual devices are designed to integrate seamlessly with existing tools of choice, acting as a drop-in replacement for physical devices or emulators.



```
__TEXT_EXEC:__text:FFFFFFF0079ED031 DCB  0xF
__TEXT_EXEC:__text:FFFFFFF0079ED032 DCB 0x5F ; _
__TEXT_EXEC:__text:FFFFFFF0079ED033 DCB 0xD6
__TEXT_EXEC:__text:FFFFFFF0079ED034 DCB 0x7F ;
__TEXT_EXEC:__text:FFFFFFF0079ED035 DCB 0x20
__TEXT_EXEC:__text:FFFFFFF0079ED036 DCB    3
__TEXT_EXEC:__text:FFFFFFF0079ED037 DCB 0xD5
__TEXT_EXEC:__text:FFFFFFF0079ED038 ; ---------------------------------------
__TEXT_EXEC:__text:FFFFFFF0079ED038 CBZ              X30, loc_FFFFFFF0079ED040
__TEXT_EXEC:__text:FFFFFFF0079ED03C RET
__TEXT_EXEC:__text:FFFFFFF0079ED040 ; ---------------------------------------
__TEXT_EXEC:__text:FFFFFFF0079ED040
__TEXT_EXEC:__text:FFFFFFF0079ED040 loc_FFFFFFF0079ED040                     ;
__TEXT_EXEC:__text:FFFFFFF0079ED040 MOV              X0, #1
__TEXT_EXEC:__text:FFFFFFF0079ED044 BL               unk_FFFFFFF007B963B0
__TEXT_EXEC:__text:FFFFFFF0079ED048 MOV              X0, #0
__TEXT_EXEC:__text:FFFFFFF0079ED04C BL               unk_FFFFFFF007B8179C
__TEXT_EXEC:__text:FFFFFFF0079ED050
__TEXT_EXEC:__text:FFFFFFF0079ED050 loc_FFFFFFF0079ED050                     ;
__TEXT_EXEC:__text:FFFFFFF0079ED050 B                loc_FFFFFFF0079ED050
__TEXT_EXEC:__text:FFFFFFF0079ED050 ; ---------------------------------------
__TEXT_EXEC:__text:FFFFFFF0079ED054 DCB 0x1F
__TEXT_EXEC:__text:FFFFFFF0079ED055 DCB 0x20
__TEXT_EXEC:__text:FFFFFFF0079ED056 DCB    3
__TEXT_EXEC:__text:FFFFFFF0079ED057 DCB 0xD5
__TEXT_EXEC:__text:FFFFFFF0079ED058 DCB 0xFD
__TEXT_EXEC:__text:FFFFFFF0079ED059 DCB 0x7B ; {
__TEXT_EXEC:__text:FFFFFFF0079ED05A DCB 0xBF
UNKNOWN FFFFFFF0079ED038: __TEXT_EXEC:__text:FFFF (Synchronized with PC)
```

```
X18 0000000000000000  MEMORY:0000000000000000                      N     0
X19 FFFFFFF008F85000  __DATA:__data:FFFFFFF008F85000               Z     1
X20 FFFFFFE000625A90  MEMORY:FFFFFFE000625A90                      C     1
X21 0000008DB723C252  MEMORY:0000008DB723C252                      V     0
X22 FFFFFFF009121220  __DATA:__common:FFFFFFF009121220             Q     0
X23 0000000000000001  MEMORY:0000000000000001                      IT2   0
X24 FFFFFFF00913D000  __DATA:__common:FFFFFFF00913D000             J     0
X25 FFFFFFF0091208CC  __DATA:__common:FFFFFFF0091208CC             GE    0
X26 00000000FFFFFFFF  MEMORY:00000000FFFFFFFF                      IT    0
X27 FFFFFFF0090F8000  __DATA:__common:FFFFFFF0090F8000             E     1
X28 FFFFFFE000625B10  MEMORY:FFFFFFE000625B10                      A     1
X29 FFFFFFE06BD77FF0  MEMORY:FFFFFFE06BD77FF0                      I     1
X30 FFFFFFF007B8178C  __TEXT_EXEC:__text:FFFFFFF007B8178C          F     1
SP  FFFFFFE06BD77FC0  MEMORY:FFFFFFE06BD77FC0                      T     0
PC  FFFFFFF0079ED038  __TEXT_EXEC:__text:FFFFFFF0079ED038          MODE  4
PSR 604003C4
```

Path	Base	Size
kernel	FFFFFFF007004000	0000000001F40000
<GDB remote pr…		

Line 1 of 2

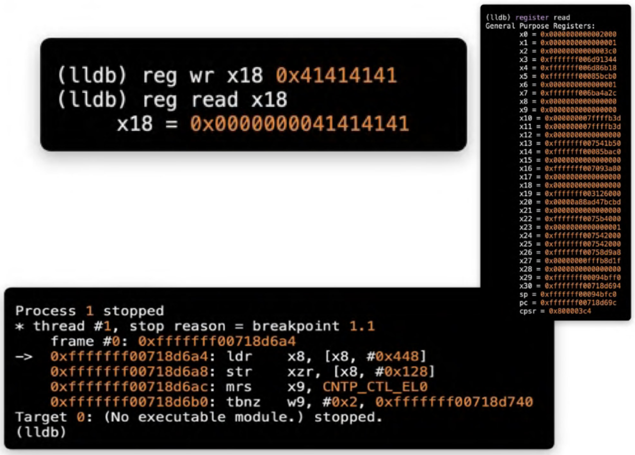
```
int64_t sub_fffffff009484f40(void* arg1)

void* arg1  {Register x0}

sub_fffffff009484f40:
fffffff009484f40  mov     x8, x0
fffffff009484f44  mov     w0, #0
fffffff009484f48  ldr     x9, [x8, #0x138]
fffffff009484f4c  cmp     x9, #0
fffffff009484f50  csel    x8, x8, x9, eq
fffffff009484f54  ldr     x8, [x8, #0xc0]
fffffff009484f58  ldr     x8, [x8, #0x108]
fffffff009484f5c  mov     w9, #0x109
fffffff009484f60  and     x8, x8, x9
fffffff009484f64  cmp     x8, #0x7
fffffff009484f68  b.gt    0xfffffff009484f88

fffffff009484f80  cmp     x8, #0x8
fffffff009484f84  b.eq    0xfffffff009484fa0

mov     w0, #0x2

fffffff009484f88  cmp     x8, #0x100
fffffff009484f8c  b.ne    0xfffffff009484fa4

fffffff009484f90  mov     w0, #0x3
fffffff009484f94  ret

fffffff009484f6c  cbz     x8, 0xfffffff009484f98

fffffff009484f98  mov     w0, #0x1
fffffff009484f9c  ret

fffffff009484f70  cmp     x8, #0x1
fffffff009484f74  b.ne    0xfffffff009484fa4

fffffff009484f78  mov     w0, #0
fffffff009484f7c  ret

fffffff009484fa4  ret
```

Integrate with your favorites.
Use side-by-side with your
existing tools.

Technical Capabilities

Vulnerability Research

- ✓ Easily spin-up any combo of device, firmware, and mobile app
- ✓ Instantly jailbreak and gain root access, including the latest OS versions
- ✓ Supports kernel, iBoot, and SEP debugging
- ✓ Network traffic interception and tracing
- ✓ Built-in matrix testing and retrograding tests for older OS versions
- ✓ Research and test new and known exploits
- ✓ Practice weaponizing n-day iOS and Android exploits
- ✓ Experiment with new mitigations with live introspection and debugging (kernel and user-mode)

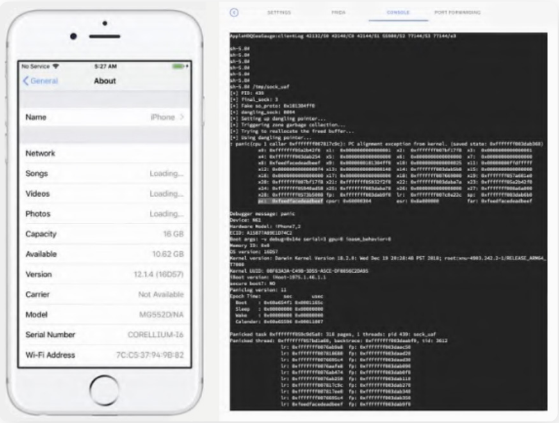


```
(lldb) reg wr x18 0x41414141
(lldb) reg read x18
     x18 = 0x0000000041414141
```

```
(lldb) register read
General Purpose Registers:
        x0 = 0x0000000000002000
        x1 = 0x0000000000000001
        x2 = 0x00000000000003c0
        x3 = 0xfffffff006d91344
        x4 = 0xfffffff006d86b18
        x5 = 0xfffffff00085bcb0
        x6 = 0x0000000000000001
        x7 = 0xfffffff006ba4a2c
        x8 = 0x0000000000000000
        x9 = 0x0000000000000000
       x10 = 0x000000007ffffb3d
       x11 = 0x000000007ffffb3d
       x12 = 0x0000000000000000
       x13 = 0xfffffff007541b50
       x14 = 0xfffffff00085bac0
       x15 = 0x0000000000000000
       x16 = 0xfffffff007093a80
       x17 = 0x0000000000000000
       x18 = 0x0000000000000000
       x19 = 0xfffffff003126000
       x20 = 0x00000a88ad47bcbd
       x21 = 0x0000000000000000
       x22 = 0xfffffff0075b4000
       x23 = 0x0000000000000001
       x24 = 0xfffffff007542000
       x25 = 0xfffffff007542000
       x26 = 0xfffffff00758d9a8
       x27 = 0x00000000fffb8d1f
       x28 = 0x0000000000000000
       x29 = 0xfffffff00094bff0
       x30 = 0xfffffff00718d694
        sp = 0xfffffff00094bfc0
        pc = 0xfffffff00718d69c
      cpsr = 0x800003c4
```

```
Process 1 stopped
* thread #1, stop reason = breakpoint 1.1
    frame #0: 0xfffffff00718d6a4
->  0xfffffff00718d6a4: ldr    x8, [x8, #0x448]
    0xfffffff00718d6a8: str    xzr, [x8, #0x128]
    0xfffffff00718d6ac: mrs    x9, CNTP_CTL_EL0
    0xfffffff00718d6b0: tbnz   w9, #0x2, 0xfffffff00718d740
Target 0: (No executable module.) stopped.
(lldb)
```

Use standard debugging tools such as LLDB.



No Service 5:27 AM

General About

Name	iPhone
Network	
Songs	Loading...
Videos	Loading...
Photos	Loading...
Capacity	16 GB
Available	10.62 GB
Version	12.1.4 (16D57)
Carrier	Not Available
Model	MG552D/NA
Serial Number	CORELLIUM-I6
Wi-Fi Address	7C:C5:37:94:9B:82

```
AppleHDQGasGauge::clientLog [illegible]

sh-5.0#
sh-5.0#
sh-5.0#
sh-5.0#
sh-5.0#
sh-5.0#
sh-5.0# /tmp/sock_uaf
[+] PID: 439
[+] Final_sock: 3
[+] Fake so_proto: [illegible]
[+] dangling_sock: [illegible]
[+] Setting up dangling pointer...
[+] Triggering zone garbage collection...
[+] Trying to reallocate the freed buffer...
[+] Using dangling pointer...
: panic(cpu 1 caller [illegible]): PC alignment exception from kernel. (saved state: [illegible])
[illegible]
      pc: 0xfeedfacedeadbeef cpsr: [illegible]     esr: [illegible]     far: 0xfeedfacedeadbeef

Debugger message: panic
Device: N61
Hardware Model: iPhone7,2
ECID: [illegible]
Boot args: -v debug=0x14e serial=3 gpu=0 [illegible]
Memory ID: 0x0
OS version: 16D57
Kernel version: Darwin Kernel Version 18.2.0: Wed Dec 19 20:28:40 PST 2018; root:xnu-4903.242.2~1/RELEASE_ARM64_
T7000
Kernel UUID: [illegible]
iBoot version: iBoot-[illegible]
secure boot?: NO
Paniclog version: 11
Epoch Time:        sec       usec
  Boot    : [illegible]
  Sleep   : 0x00000000 0x00000000
  Wake    : 0x00000000 0x00000000
  Calendar: [illegible]

Panicked task [illegible]: 318 pages, 1 threads: pid 439: sock_uaf
Panicked thread: [illegible], backtrace: [illegible], tid: [illegible]
              [illegible]
              lr: 0xfeedfacedeadbeef  fp: [illegible]
```

Trigger real 0-day and n-day vulnerabilities.

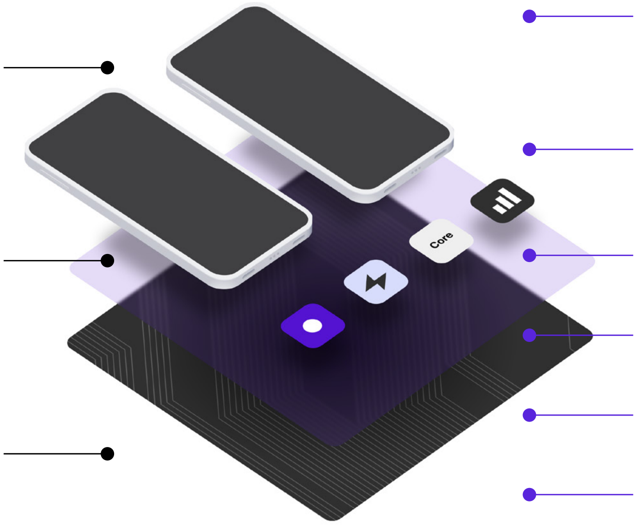
Training & Education

- ✓ Teach mobile security and testing best practices
- ✓ Vulnerability and exploit research training
- ✓ Inject artificial vulnerabilities for training
- ✓ Perform capture-the-flag (CTF) exercises

Compliance & Auditing

- ✓ Regulatory standards development, testing, and auditing
- ✓ Data privacy testing and auditing

Corellium Virtual Hardware Platform



VIRTUAL DEVICES

Arm-powered phones and IoT devices with endless OS and model combinations.

HYPERVISOR

Corellium Hypervisor for Arm (**CHARM**) is a type 1 hypervisor and the only one of its kind.

ARM SERVER

Virtual models run on Arm, just like real devices, combining native fidelity with on-demand availability.

ROOT ACCESS

Root or jailbreak devices instantly, no need to add code or apply security vulnerabilities.

CONTROL

Configure device buttons, sensors, location, environment, battery, device IDs, ports, cameras and mics.

X-RAY VISION

Powerful OS, app, file, system call, and console access and control.

INTROSPECTION

Advanced OS, kernel and boot control and tooling.

NETWORK ANALYSIS

HTTP/S traffic inspection, tracing, and logging.

REPLICATION

Snapshot, clone, and restore device states.

TEAMING

Easy project workspace management and team collaboration.

TOOLING

Simplified connection of IDE, debugging, network and security tools and comprehensive APIs.

AMPERE

Corellium Appliances

Corellium appliances for onsite and air-gapped solutions run on the latest Ampere Altra Arm servers.

aws

Corellium Cloud

Corellium is hosted on AWS using Amazon Graviton Arm servers. Customer private AWS Graviton clouds also supported.

CORELLIUM

Free trials at **Corellium.com**