
UNCHARTED TERRITORY
macOS Operations and Development
BEACON
OUTFLANK
clear advice with a hacker mindset

WHOAMI

Kyle Avery – @kyleavery_

- Offensive Developer @ Outflank
- Red team background / .NET and C development
- Lone US Outflank member

OUTFLANK

- Outflank Security Tooling (OST)
- Red Teaming Services

BEACON

AGENDA

macOS Internals

- Relevant security controls
- EDR telemetry sources

Red Team Operations

- Typical macOS environment
- Initial access techniques

Malware Development

- Current reflective loaders
- PIC Mach-O loader

MACOS INTERNALS

BEACON

OUTFLANK

clear advice with a hacker mindset

CODE SIGNING

- Most executables and libraries are signed
  - Certificates available through Apple Developer Program ($99/year)
  - Can be revoked by Apple

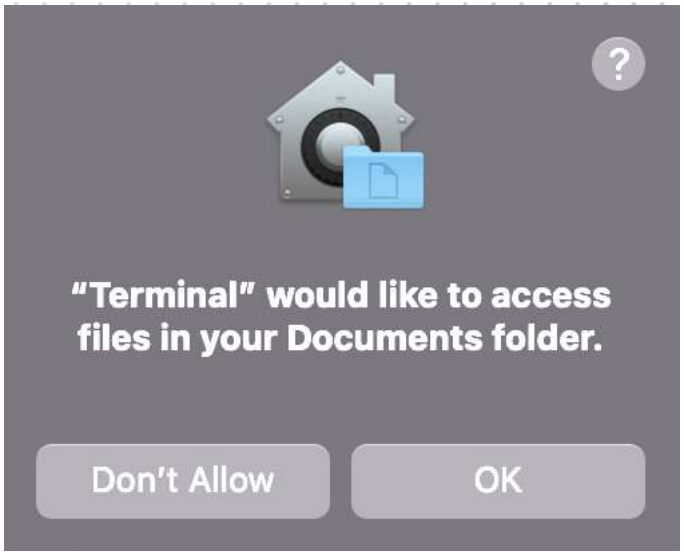
- Hardened runtime
  - Enforces signed dylibs (sideloading/hijacking)
  - Prevents remote task ports (process handle)

- Notarization
  - Submit binary to Apple for “review”
  - Must be compiled with hardened runtime

TRANSPARENCY, CONSENT, & CONTROL

- Restricts access, even for root
  - Desktop, Documents, Downloads
  - Camera, Microphone

- Some resources require **entitlements**
  - Part of signature
  - May disable hardened runtime features:
    - Unsigned executable memory
    - Library validation
    - DYLD environment variables



APP SANDBOX AND SIP

- **App Sandbox**
  - Opt-in restrictions

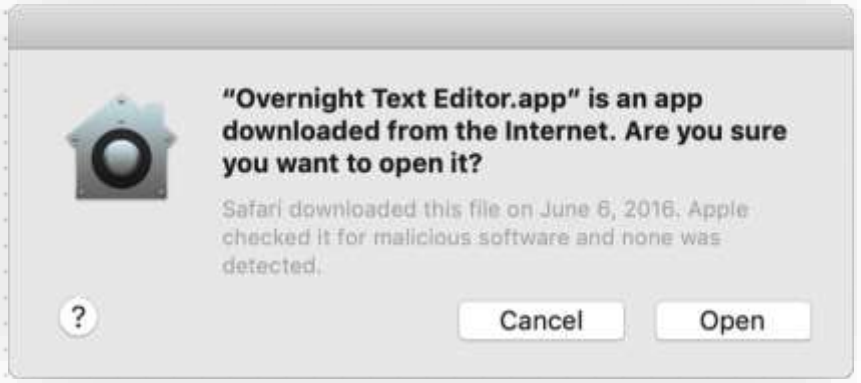
```
codesign -d --entitlements - /Applications/Microsoft\ Excel.app

[Key] com.apple.security.app-sandbox
[Value]
  [Bool] true
[Key] com.apple.security.temporary-exception.sbpl
[Value]
[Array]
[String] (allow file-read* file-write*
         (require-all (vnode-type REGULAR-FILE) (regex #"(^|/)~\$[^/]+$")) )
```

- **System Integrity Protection**
  - Enforces code signatures, TCC, and sandbox

XPROTECT AND GATEKEEPER

- **XProtect** – Built-in antivirus
  - Silent updates
  - Very specific rules, not really an issue

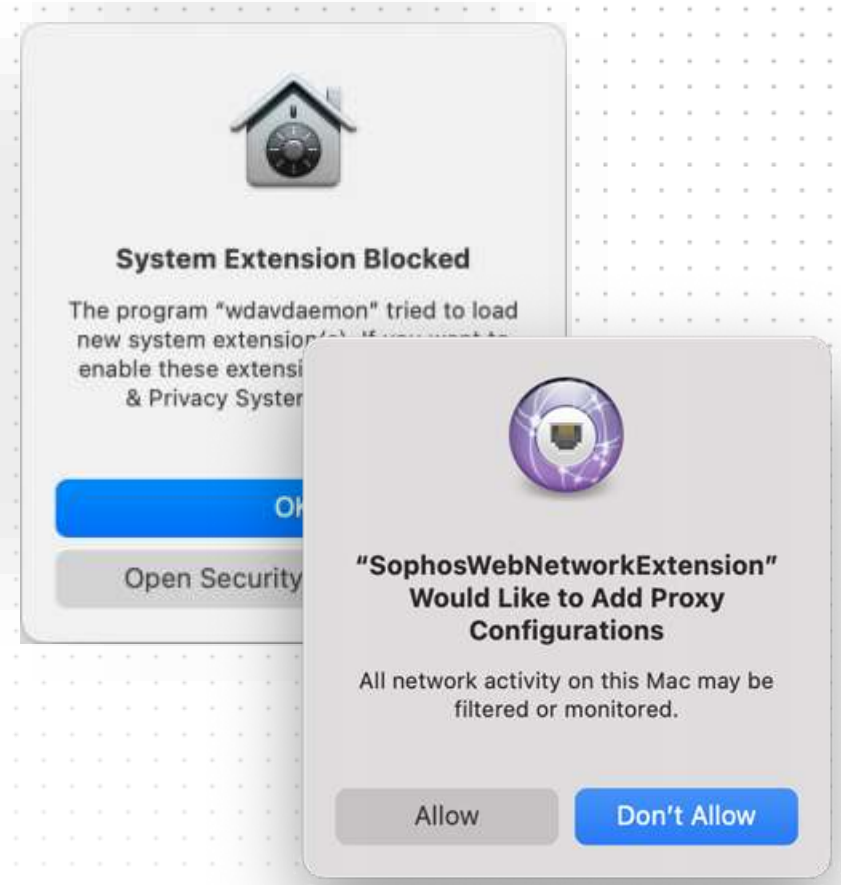
- **Gatekeeper**
  - Validates signature+notarization
  - Similar to Windows SmartScreen



EDR TELEMETRY

- **Endpoint Security API**
  - Authentication attempts
  - Process creation and termination
  - File access, modification, and creation

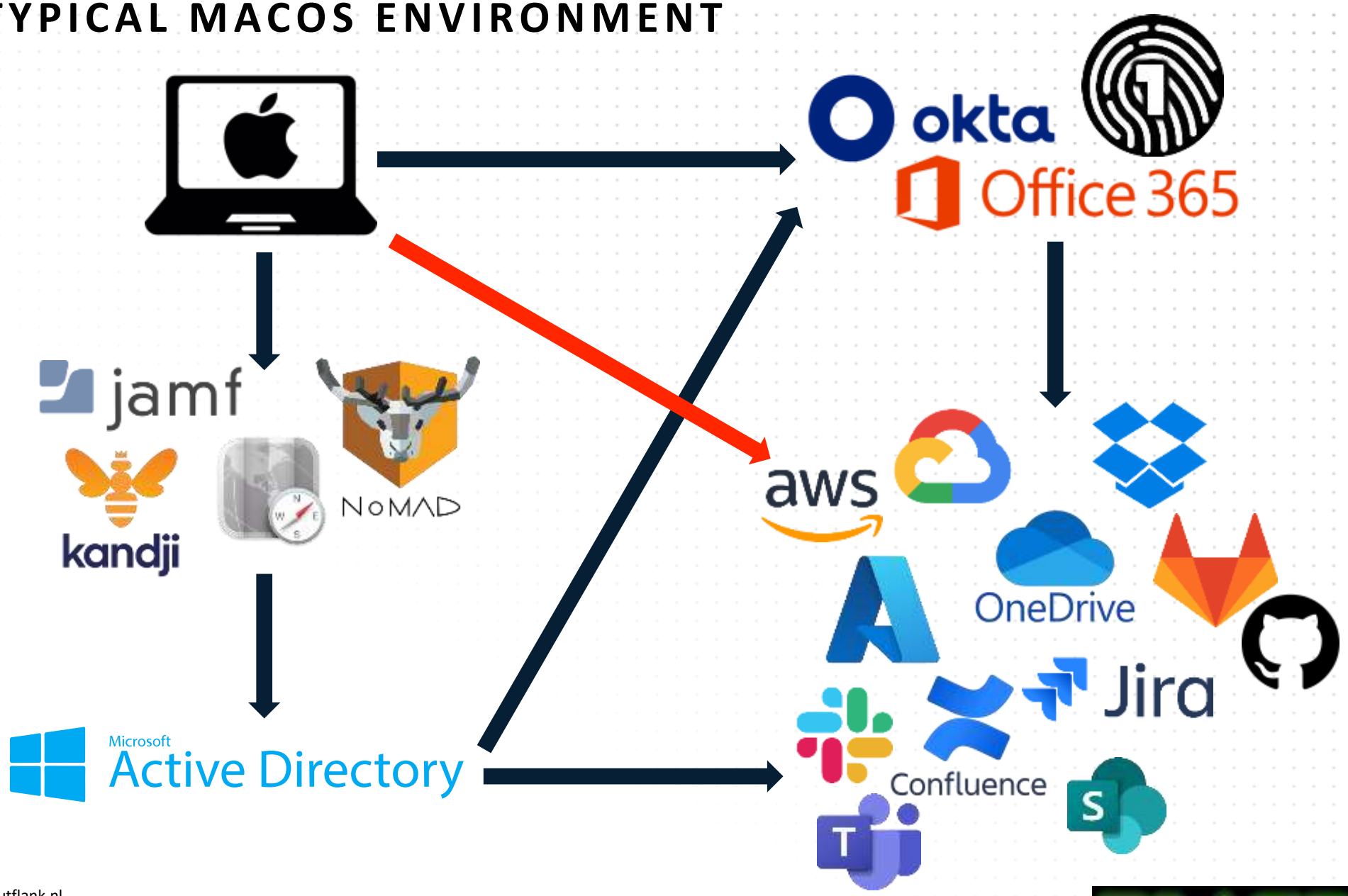
- **Network Extensions** – Full packet capture



Source: https://www.outflank.nl/blog/2024/06/03/edr-internals-macos-linux/


RED TEAM OPS
BEACON
OUTFLANK
clear advice with a hacker mindset

TYPICAL MACOS ENVIRONMENT



MACOS C2

- Mythic
  - Multiple agents with different formats
    - Hermes, Poseidon – Mach-O
    - Apfell – JXA
- Other agents:
  - Sliver – Mach-O
  - Empire – Python

OFFICE MACROS

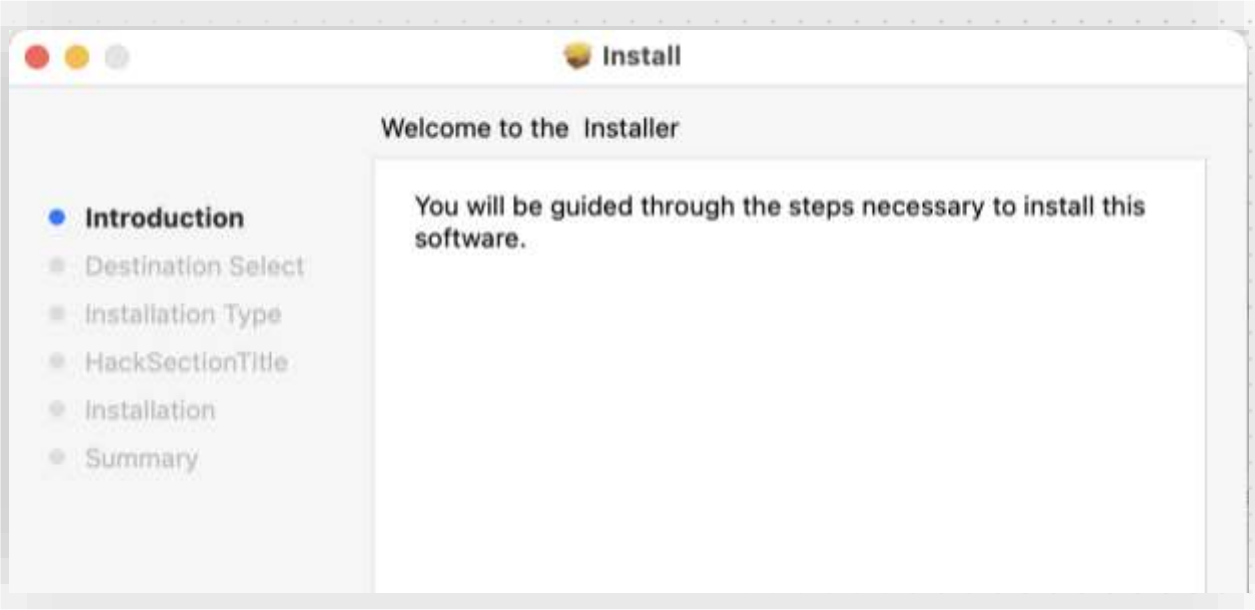
- Good reasons to use macros:
  - Signature not required
  - No AMSI
  - Full API access

```
Private Declare PtrSafe Function mach_task_self Lib "libSystem.B.dylib"
    () As LongPtr

Private Declare PtrSafe Function memcpy Lib "libc.dylib"
    (ByVal dst As LongPtr, ByVal src As LongPtr, ByVal sz As LongPtr) As LongPtr
```

- Restricted by sandbox
  - Cannot access local files
  - Possible to escape, currently requires reboot
  - Full network access

INSTALLER PACKAGES

- Three execution methods:
  - Pre/Post install Bash scripts
  - Installer plugins (Mach-O binary)
  - Distribution XML (Installer JS)

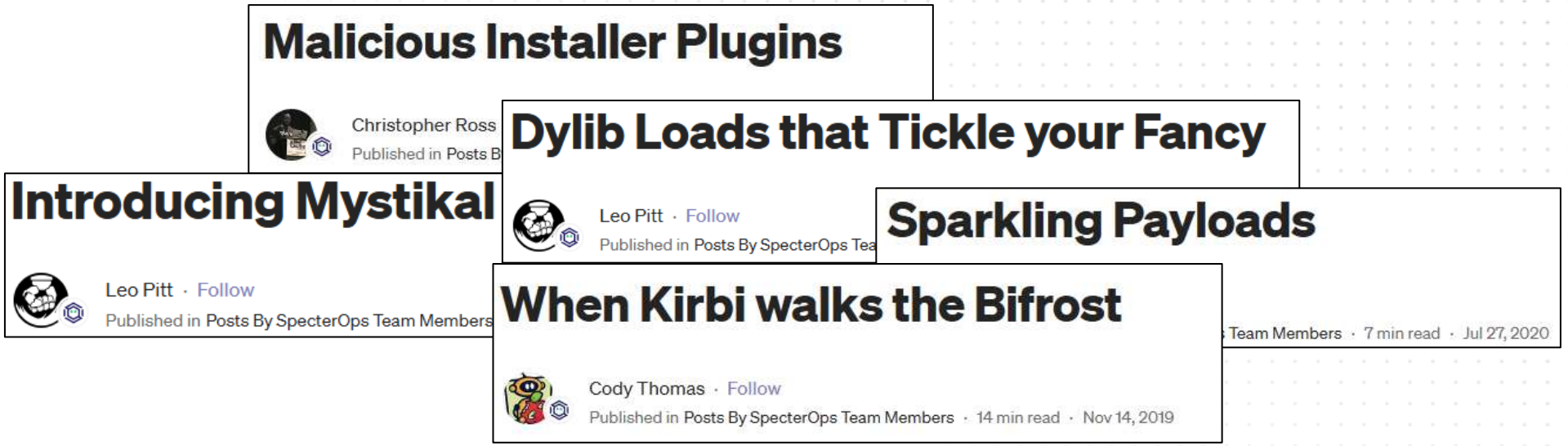


DMG WRAPPER

- Notarization “blocks” unsigned executables and installer packages
- Use DMG for social engineering / delivery

MORE MACOS RED TEAM

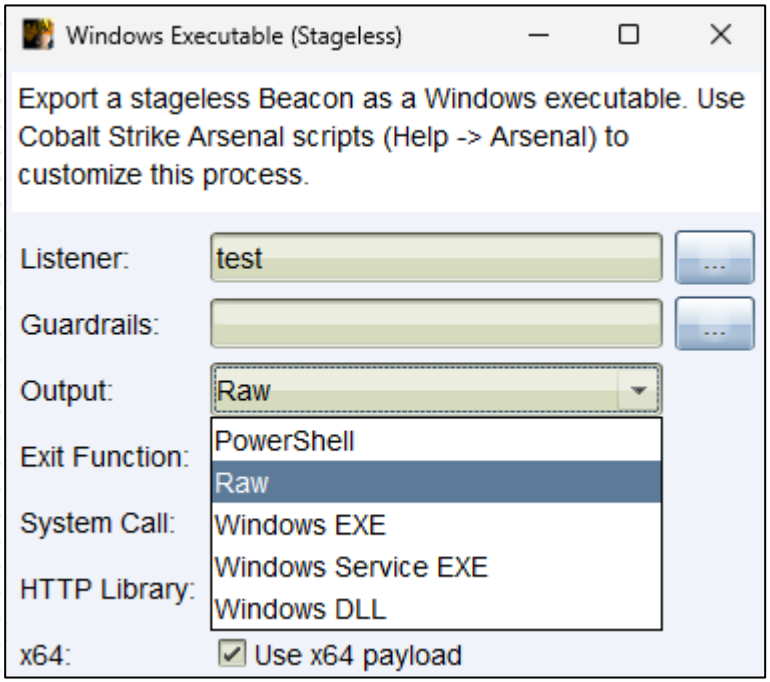
- SpecterOps Blog
  - https://posts.specterops.io/tagged/macos
- Objective by the Sea Conference
  - https://www.youtube.com/@objectiveseefoundation



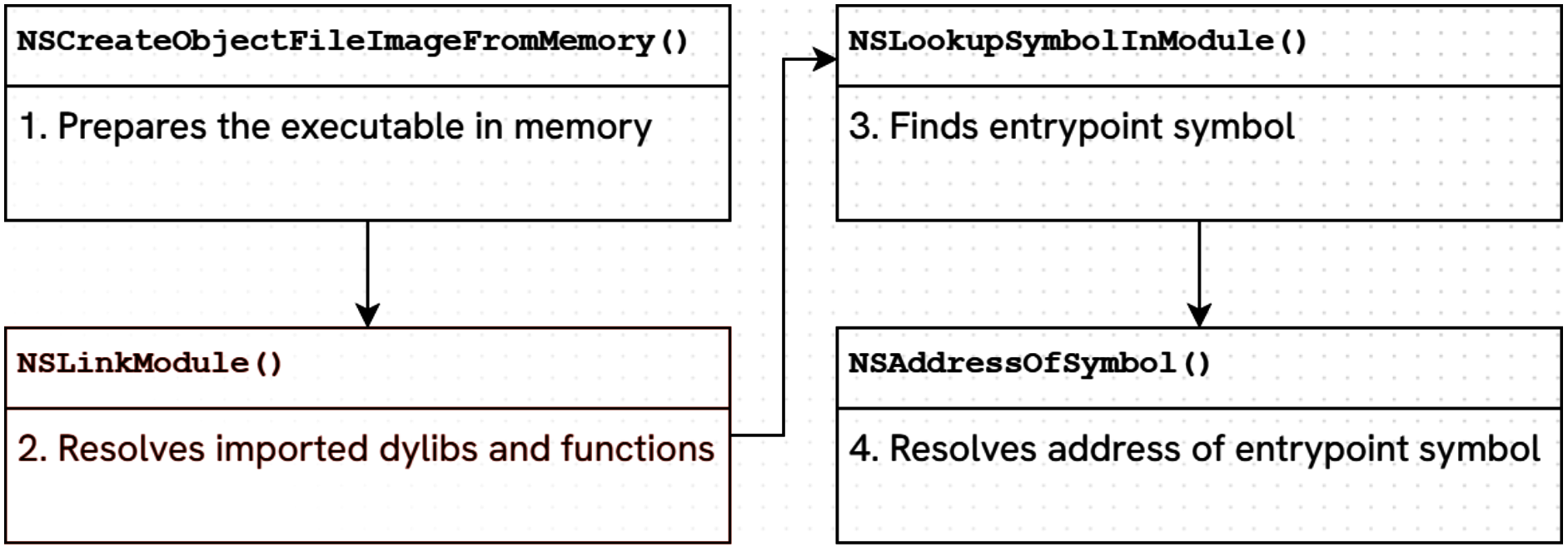

MALWARE DEVELOPMENT
BEACON
OUTFLANK
clear advice with a hacker mindset

SOMETHING MORE FLEXIBLE?

- Learning from Windows
  - Reflective DLL + PIC loader
  - Executed with shellcode loader
- macOS Restrictions
  - Unsigned executable memory
  - Hardened runtime exceptions:
    - Excel, PowerPoint
    - PyCharm, Android Studio
    - GoTo Meeting, Discord



COULD IT BE THAT EASY?



- Newer versions of dyld write the module to disk and load with **`dlopen()`**
  - Static prefix **`NSCreateObjectFileImageFromMemory-`**

Source: The Mac Hacker's Handbook (2009)

CUSTOM MACH-O LOADER

- XPN already solved this! Mostly…
  - Written in Objective-C
  - Resolves imports with dlopen/dlsym

- Converting to PIC:
  1. Rewrite without libc or objc libraries
  2. Control code placement and order
  3. Manually resolve imported functions

Source: https://blog.xpnsec.com/building-a-mach-o-memory-loader-part-1/

ORDER FILE – LINUX & WINDOWS

- Linux GCC/Clang and MinGW
  - Decorate functions with section name

```
/* gcc -Wl,-T,link.ld

__attribute__((section(".text$A"))) void Example();
*/

SECTIONS {
    .text : {
        *( .text$A )
        *( .text$B )
        *( .rdata* )
        *( .text$C )
    }
}
```

ORDER FILE – MACOS

- macOS Clang doesn't support linker scripts
  - Must list every function in "order file"

```
/* clang -Wl,-order_file,"link.order"

void Example();
*/

_Example
```

DYNAMIC IMPORTS

- Windows makes this easy:
  1. Find the PEB at **`FS:[0x30]`** / **`GS:[0x60]`**
  2. Go to **`InLoadOrderModuleList`**
  3. Walk the list of modules to find NTDLL

- macOS is similar:
  1. Find dyld in memory
  2. Resolve **`dyld_get_all_image_infos`**
  3. Walk the list of modules to find libdyld

FIND DYLD WITH CHMOD

- Memory scanning may access invalid memory
  - Use chmod to check each region!

```
ERRORS
   The chmod() system call will fail and the file mode will be unchanged if:

   [EFAULT]            Path points outside the process's allocated address
                       space.
```

- Don’t we need to import chmod?

```
int
chmod(const char *path, mode_t mode)
{
        int res = __chmod(path, mode);
```

Source: https://blogs.blackberry.com/en/2017/02/running-executables-on-macos-from-memory

IT'S SO OVER

- This won't work – why?

```
int sys_chmod(const char* path, mode_t mode) {
    int ret = 0;

    register long rax asm("rax") = SYS_chmod;
    register long rdi asm("rdi") = (long)path;
    register long rsi asm("rsi") = (long)mode;

    __asm__ volatile (
        "syscall"
        : "=r" (ret)
        : "0" (rax), "D" (rdi), "S" (rsi)
        : "memory", "cc"
    );

    return ret;
}
```

MACOS SYSTEM CALL CLASSES

- Multiple syscall “classes”
  - Must set higher-order bits of eax to class number
  - Apple open-source headers define the classes

```
#define SYSCALL_CLASS_NONE      0       /* Invalid */
#define SYSCALL_CLASS_MACH      1       /* Mach */
#define SYSCALL_CLASS_UNIX      2       /* Unix/BSD */
#define SYSCALL_CLASS_MDEP      3       /* Machine-dependent */
#define SYSCALL_CLASS_DIAG      4       /* Diagnostics */
#define SYSCALL_CLASS_IPC       5       /* Mach IPC */
```

Source: https://dustin.schultz.io/mac-os-x-64-bit-assembly-system-calls.html

WE'RE SO BACK

- Use Apple's macro to set the syscall class:

```
int sys_chmod(const char* path, mode_t mode) {
    int ret = 0;

    register long rax asm("rax") = SYSCALL_CONSTRUCT_UNIX(SYS_chmod);
    register long rdi asm("rdi") = (long)path;
    register long rsi asm("rsi") = (long)mode;

    __asm__ volatile (
        "syscall"
        : "=r" (ret)
        : "0" (rax), "D" (rdi), "S" (rsi)
        : "memory", "cc"
    );

    return ret;
}
```

THANK YOU

- Try it yourself:
  - macOS Reflective Loader
    https://github.com/outflanknl/macho-loader

- If you have questions about OST or macOS, let's chat!

OUTFLANK

clear advice with a hacker mindset