

Predator's kill switch: undocumented anti-analysis techniques in iOS spyware

By Jamf Threat Labs

Archived: 2026-04-10 03:15:51 UTC

A deep dive into the error code taxonomy and detection mechanisms that prior research didn't cover.

January 14 2026 by

Jamf Threat Labs

By: Shen Yuan and Nir Avraham

Introduction

In December 2025, Google's Threat Intelligence Group (GTIG) published extensive research on Intellexa's Predator spyware, documenting its zero-day exploit chains and the PREYHUNTER stager component. Their research identified that the "watcher" module detects developer mode, jailbreak tools, security applications and network interception configurations.

However, while conducting independent reverse engineering of a Predator sample, Jamf Threat Labs discovered several undocumented mechanisms that reveal how sophisticated this spyware's anti-analysis capabilities truly are. This post presents original findings including:

- A complete error code taxonomy (301-311) that enables operators to diagnose exactly why an implant failed
- Implementation details for each detection method that go beyond high-level descriptions
- An undocumented crash reporter monitoring system for anti-forensics
- SpringBoard hooking to hide recording indicators from victims
- Kernel exploitation class names that reveal the internal architecture

These findings demonstrate that Predator's operators have granular visibility into failed deployments — a capability that has significant implications for researchers attempting to analyze these samples.

The CSWatcherSpawner architecture

The implant contains a C++ class named `CSWatcherSpawner::CSWatcherSpawner` that orchestrates all anti-analysis checks. The class implements a comprehensive set of detection methods with a sophisticated reporting mechanism.

What makes this architecture notable is not just the breadth of checks, but the reporting mechanism that provides operators with precise diagnostic information when deployment fails.

```

-----
text:00000000100005900
text:00000000100005904
text:00000000100005908
text:0000000010000590C
text:00000000100005910
text:00000000100005914
text:00000000100005918
text:0000000010000591C
text:00000000100005920
text:00000000100005924
text:00000000100005928
text:0000000010000592C
text:00000000100005930
text:00000000100005934
text:00000000100005938
text:0000000010000593C
text:00000000100005940
text:00000000100005944
text:00000000100005948
text:0000000010000594C
text:00000000100005950
text:00000000100005954
text:00000000100005958
text:0000000010000595C
text:00000000100005960
text:00000000100005964
text:00000000100005968
text:0000000010000596C
text:00000000100005970
text:00000000100005974
-----
PACIBSP
SUB SP, SP, #0x40
STP X22, X21, [SP, #0x30+var_20]
STP X20, X19, [SP, #0x30+var_10]
STP X29, X30, [SP, #0x30+var_s0]
ADD X29, SP, #0x30
MOV X19, X0
NOP
LDR X16, __imp__open
PACIZA X16
MOV X2, X16
MOV W0, #0
MOV X1, #0
MOV W3, #8
BL __unicopy
BL __ZN5Utils13getCountNamesEv ; Utils::getCountNames(void)
CMP W0, #2
BLT loc_100005978
LDR X20, [X19, #0x110]
MOV X0, X20 ; __s
BL __strlen
ADD X0, X0, #1 ; __count
MOV W1, #1 ; __size
BL __calloc
MOV X21, X0
MOV X1, X20 ; __src
BL __strcpy
ADR X2, a311 ; "311"
NOP
B loc_100005A78

```

Figure 1: check_perform() entry point showing getCountNames() check and error code 311 dispatch for multiple instance detection

The error code taxonomy

The most significant undocumented finding is Predator's error code system. When any anti-analysis check triggers, the malware doesn't simply terminate — it reports a specific error code to its command-and-control infrastructure before cleaning up and exiting.

The `check_perform()` function reveals the complete taxonomy:

Code	Trigger condition	Detection method
311	Multiple instances running	<code>getCountNames() >= 2</code>
310	Debug console attached	<code>is_console_attached()</code>
309	Device in restricted region	<code>Locale == "US" "IL"</code>
301	Developer mode OR jailbreak	<code>is_developer() is_not_phone()</code>
304	Security/analysis tools running	<code>is_unsafe_running()</code>
307	HTTP proxy configured	<code>is_proxy_running()</code>
308	Custom root CA installed	<code>is_rootca_installed()</code>

Table 1: Error codes and their conditions

The missing error codes: 302, 303, 305, 306

A careful examination of the binary's string section reveals an interesting gap in the error code taxonomy. The error codes stored in the `__cstring` section appear sequentially, but with notable gaps:

```

__cstring:00000001000412A4 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::is_console_attached(void)+14r0
__cstring:00000001000412A4 ; CSWatcherSpawner::CSWatcherSpawner::is_console_attached(void)+78r0 ...
__cstring:00000001000412E8 DCB "ist",0
__cstring:00000001000412E8 aGlobal DCB "global",0 ; DATA XREF: __cfstring:cfstr_Global:io
__cstring:00000001000412EF a311 DCB "311",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+6Cr0
__cstring:00000001000412F3 a310 DCB "310",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+A4r0
__cstring:00000001000412F7 a309 DCB "309",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+184r0
__cstring:00000001000412FB a301 DCB "301",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+170r0
__cstring:00000001000412FB a304 DCB "304",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+21Cr0
__cstring:0000000100041303 a307 DCB "307",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+254r0
__cstring:0000000100041307 a308 DCB "308",0 ; DATA XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+290r0
__cstring:000000010004130B aPrivateVarPref_0 DCB "/private/var/preferences/SystemConfiguration/preferences.plist",0
__cstring:000000010004130B ; DATA XREF: __cfstring:cfstr_PrivateVarPref_0:io
__cstring:000000010004134A aHttpproxy DCB "HTTPProxy",0 ; DATA XREF: __cfstring:cfstr_Httpproxy:io
__cstring:0000000100041354 aHttpenable DCB "HTTPEnable",0 ; DATA XREF: __cfstring:cfstr_Httpenable:io
__cstring:000000010004135F aPrivateVarProt DCB "/private/var/protected/trustd/private/TrustStore.sqlite3",0

```

Figure 2: Error code strings in __cstring section — note the sequential addresses but non-sequential codes (301, 304, 307, 308) revealing missing 302, 303, 305, 306

Notably, error codes 302, 303, 305, and 306 are completely absent from this sample. These gaps in the numbering scheme suggest several possibilities:

- **Reserved codes:** Placeholder codes for future functionality or checks not yet implemented
- **Variant-specific:** These codes may be used in other Predator variants or versions targeting different platforms
- **Deprecated checks:** Detection methods that were removed but whose error codes were preserved in the taxonomy
- **Shared taxonomy:** The error code system shared across multiple Intellexa products, with different products implementing different subsets

The non-sequential numbering (jumping from 301 to 304, and from 304 to 307) indicates this error taxonomy has evolved over time or is designed to accommodate checks that may be conditionally compiled based on the target configuration.

Why this matters

This error code system transforms failed deployments from black boxes into diagnostic events. When an operator deploys Predator against a target and receives error code 304, they know the target is running security tools — not that the exploit failed, not that the device is incompatible, but specifically that active analysis is occurring.

This has direct implications for targeted individuals: if security analysis tools like Frida are running, Predator will abort deployment and report error code 304 to operators, who can then troubleshoot why their deployment failed.

Detection implementation details

While Google's research mentioned that Predator detects "custom HTTP proxies" and "custom root CAs," the actual implementation details were not published. Here's what the binary reveals:

Multiple instance detection (error 311)

The first check in `check_perform()` calls `getCountNames()` to detect if multiple Predator instances are running. This prevents researchers from running multiple analysis instances simultaneously:

```

1  int64 __fastcall Utils::getCountNames(Utils *this)
2  {
3  pid_t *v1; // x19
4  __int64 v2; // x20
5  size_t v4; // x23
6  pid_t *v5; // x24
7  int v6; // w22
8  char buffer[4096]; // [xsp+8h] [xbp-1058h] BYREF
9  int v8[2]; // [xsp+1008h] [xbp-58h] BYREF
10 int v9; // [xsp+1010h] [xbp-50h]
11 size_t v10; // [xsp+1018h] [xbp-48h] BYREF
12
13 __chkstk_darwin(this);
14 v10 = 0;
15 *(_QWORD *)v8 = 0xE00000001LL;
16 v9 = 0;
17 if ( sysctl(v8, 3u, 0, &v10, 0, 0) < 0 )
18     return 0xFFFFFFFFLL;
19 v1 = (pid_t *)malloc(v10);
20 if ( sysctl(v8, 3u, v1, &v10, 0, 0) < 0 )
21     return 0xFFFFFFFFLL;
22 if ( v10 >= 0x288 )
23 {
24     v2 = 0;
25     v4 = v10 / 0x288;
26     v5 = v1 + 10;
27     do
28     {
29         v6 = *v5;
30         if ( (getpgid(*v5) & 0x80000000) == 0 && proc_pidpath(v6, buffer, 0x1000u) )
31         {
32             if ( strstr(buffer, "/private/var/tmp/") )
33                 v2 = (unsigned int)(v2 + 1);
34             else
35                 v2 = (unsigned int)v2;
36         }
37         v5 += 162;
38         --v4;
39     }
40     while ( v4 );
41 }
42 else
43 {
44     v2 = 0;
45 }
46 free(v1);
47 return v2;
48 }

```

Figure 3: getCountNames() pseudocode — enumerates all processes via sysctl, uses proc_pidpath to get executable paths, counts processes running from /private/var/tmp/

The function iterates through all running processes using `sysctl`, retrieves each process's executable path via `proc_pidpath()`, and counts how many are running from `/private/var/tmp/` (Predator's staging directory). If the count is two or more, error code 311 is reported.

Developer mode detection: targeting researchers

Building on Google's finding that Predator detects Developer Mode, we reverse engineered how this check works.

The `is_developer()` function specifically targets iOS Developer Mode using the `sysctlbyname` API:

```
__text:0000000100005A10
__text:0000000100005A10 loc_100005A10 ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+DC+j
__text:0000000100005A14 STR WZR, [SP,#0x30+var_24]
__text:0000000100005A14 MOV W8, #4
__text:0000000100005A18 STR X8, [SP,#0x30+var_30]
__text:0000000100005A1C ADR X0, aSecurityMacAmf ; "security.mac.amfi.developer_mode_status"
__text:0000000100005A20 NOP
__text:0000000100005A24 ADD X1, SP, #0x30+var_24 ; void *
__text:0000000100005A28 MOV X2, SP ; size_t *
__text:0000000100005A2C MOV X3, #0 ; void *
__text:0000000100005A30 MOV X4, #0 ; size_t
__text:0000000100005A34 BL _sysctlbyname
__text:0000000100005A38 CBNZ W0, loc_100005A44
__text:0000000100005A3C LDR W8, [SP,#0x30+var_24]
__text:0000000100005A40 CBNZ W8, loc_100005A4C
__text:0000000100005A44
```

Figure 4: Developer Mode detection implementation showing `sysctlbyname("security.mac.amfi.developer_mode_status")` call

This is significant because Developer Mode was introduced in iOS 16 specifically for security researchers and developers. By detecting this, Predator effectively says: "If you've enabled developer features, you're probably not a normal target."

Jailbreak detection: `is_not_phone()`

The `is_not_phone()` function checks for the presence of jailbreak-related files and directories:

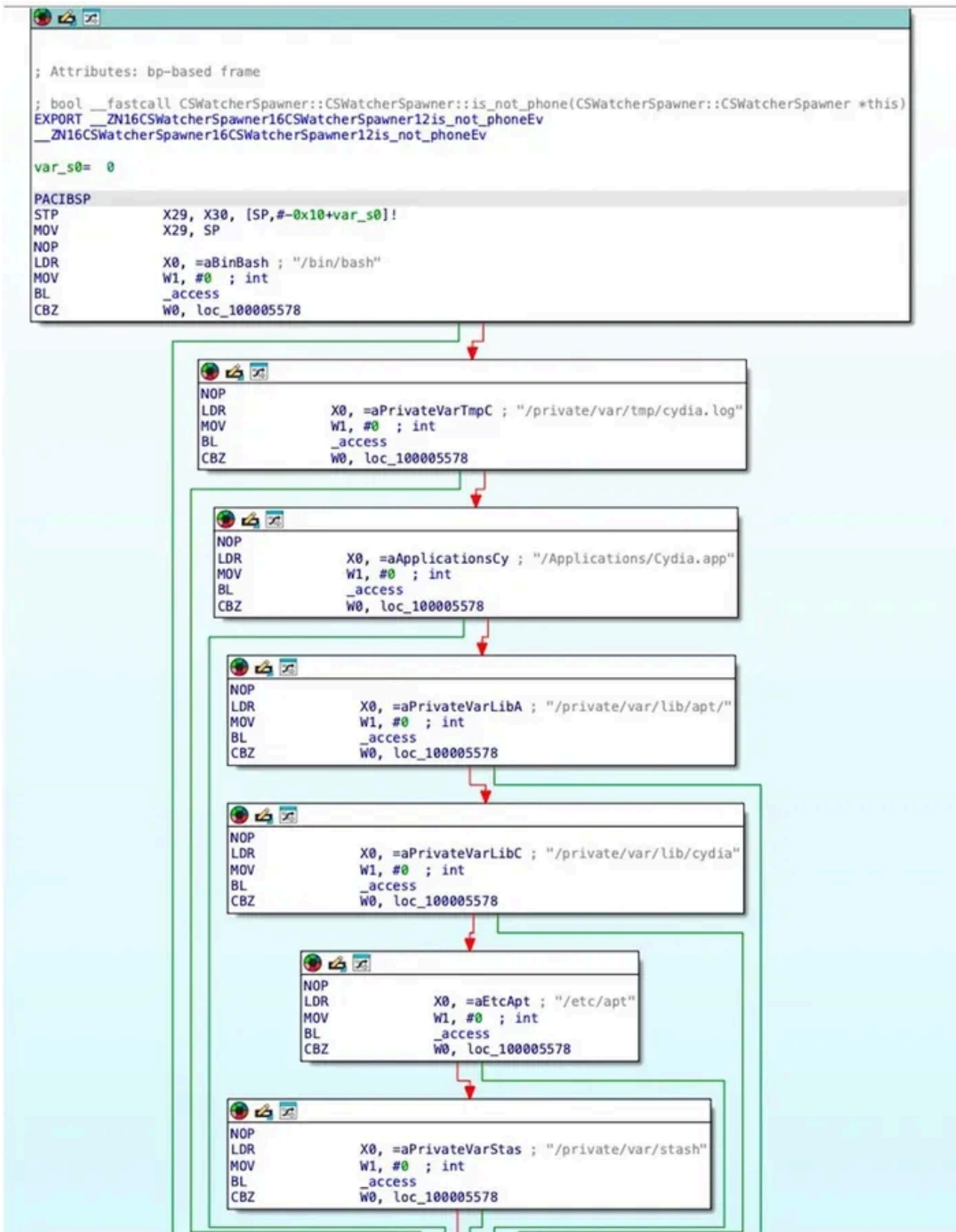


Figure 5: is_not_phone() graph view — cascading file existence checks for jailbreak indicators

The function checks for the following paths using the `access()` syscall:

- `/bin/bash` — Shell binary (not present on stock iOS)

- /private/var/tmp/cydia.log — Cydia package manager log
- /Applications/Cydia.app — Cydia application
- /private/var/lib/apt/ — APT package manager directory
- /private/var/lib/cydia — Cydia data directory
- /etc/apt — APT configuration
- /private/var/stash — Jailbreak stash directory

Additionally, it checks if /bin/ contains more than two entries (stock iOS has minimal binaries in /bin).

Geographic restrictions: avoiding US and Israeli jurisdiction

Google's research identified that Predator checks for US and Israeli locales. Our analysis documented the specific implementation of the geographic check (error code 309), which deserves further attention. Predator refuses to execute on devices with US or Israeli locale settings:

```

__TEXT:__PAGE:__PAGE:
__text:00000001000059B0 ;
__text:00000001000059B0 loc_1000059B0 ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+7C7j
__text:00000001000059B0 NOP
__text:00000001000059B0 LDR X0, =_OBJC_CLASS_$_NSLocale
__text:00000001000059B4 BL _objc_msgSend$autoupdatingCurrentLocale ; +[NSLocale autoupdatingCurrentLocale] ...
__text:00000001000059B8 BL _objc_msgSend$countryCode ; -[NSLocale countryCode] ...
__text:00000001000059BC BL _objc_msgSend$UTF8String ; -[NSString UTF8String] ...
__text:00000001000059C0 ADR X0, aUs ; "US"
__text:00000001000059C4 NOP
__text:00000001000059C8 CMP X0, X8
__text:00000001000059CC ADR X8, aIl ; "IL"
__text:00000001000059D0 NOP
__text:00000001000059D4 CCMP X0, X8, #4, NE
__text:00000001000059D8 B.NE loc_100005A10
__text:00000001000059E0 LDR X20, [X19, #0x110]
__text:00000001000059E4 MOV X0, X20 ; __s
__text:00000001000059E8 BL __strlen
__text:00000001000059EC ADD X0, X0, #1 ; __count
__text:00000001000059F0 MOV W1, #1 ; __size
__text:00000001000059F4 BL __calloc
__text:00000001000059F8 MOV X21, X0
__text:00000001000059FC MOV X1, X20 ; __src
__text:0000000100005A00 BL __strcpy
__text:0000000100005A04 ADR X2, a309 ; "309"
__text:0000000100005A08 NOP
__text:0000000100005A0C B loc_100005A78

```

Figure 6: Geographic restriction check using NSLocale — comparing country code against "US" and "IL", triggering error 309

[Enlarge image](#)

While we cannot confirm, this is likely a deliberate decision to avoid legal exposure in jurisdictions with strong cybercrime laws and active law enforcement cooperation with Intellexa's European base.

Console detection: timing-based analysis

The is_console_attached() function uses a clever timing-based approach:

```

__TEXT:__PAGE:__PAGE:
__text:0000000100005978 ;
__text:0000000100005978 loc_100005978 ; CODE XREF: CSWatcherSpawner16CSWatcherSpawner19is_console_attachedEv ; CSWatcherSpawner::CSWatcherSpawner::is_console_attached(void)
__text:000000010000597C BL __ZN16CSWatcherSpawner16CSWatcherSpawner19is_console_attachedEv
__text:0000000100005980 LDR X0, loc_100005980
__text:0000000100005984 MOV X20, [X19, #0x110]
__text:0000000100005988 BL __strlen
__text:000000010000598C ADD X0, X0, #1 ; __count
__text:0000000100005990 MOV W1, #1 ; __size
__text:0000000100005994 BL __calloc
__text:0000000100005998 MOV X21, X0
__text:000000010000599C MOV X1, X20 ; __src
__text:00000001000059A0 BL __strcpy
__text:00000001000059A4 ADR X2, a310 ; "310"
__text:00000001000059A8 NOP
__text:00000001000059AC B loc_100005A78

```

Figure 7: Debug console detection showing is_console_attached() call and error code 310 dispatch

[Enlarge image](#)

This detection compares the modification time of the diagnosticd filter file against the system boot time. If someone enabled console logging after the device booted (as a researcher would do), the file's mtime will be newer than the boot time.

Security tool detection: the complete list

Google's research mentioned "Bash, tcpdump, frida, sshd, or checkrain" but the actual list in the binary is longer. The `is_unsafe_running()` function reveals the complete process detection list:

```

--text:00000010005C10 ; bool __fastcall CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(CSWatcherSpawner::CSWatcherSpawner *this, const char *)
--text:00000010005C10 EXPORT __ZN16CSWatcherSpawner16CSWatcherSpawner17is_unsafe_runningEv
--text:00000010005C10 ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void):loc_10005AF0+P
--text:00000010005C10 ;
--text:00000010005C10 var_s0 = 0
--text:00000010005C10
--text:00000010005C14 PACI8SP
--text:00000010005C18 STP X29, X30, [SP, #-0x10+var_s0]!
--text:00000010005C18 MOV X29, SP
--text:00000010005C1C NOP
--text:00000010005C20 LDR X0, =Tcpdump ; "tcpdump"
--text:00000010005C24 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005C28 CBZ W0, loc_10005C3C
--text:00000010005C2C MOV W0, #1
--text:00000010005C30 MOV X0, X8
--text:00000010005C34 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005C38 RETAB
--text:00000010005C3C ;
--text:00000010005C3C ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+18+J
--text:00000010005C3C NOP
--text:00000010005C40 LDR X0, =FridaServer ; "frida-server"
--text:00000010005C44 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005C48 CBZ W0, loc_10005C5C
--text:00000010005C4C MOV W0, #1
--text:00000010005C50 MOV X0, X8
--text:00000010005C54 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005C58 RETAB
--text:00000010005C5C ;
--text:00000010005C5C ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+38+J
--text:00000010005C5C NOP
--text:00000010005C60 LDR X0, =Netstat ; "netstat"
--text:00000010005C64 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005C68 CBZ W0, loc_10005C7C
--text:00000010005C6C MOV W0, #1
--text:00000010005C70 MOV X0, X8
--text:00000010005C74 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005C78 RETAB
--text:00000010005C7C ;
--text:00000010005C7C ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+58+J
--text:00000010005C7C NOP
--text:00000010005C80 LDR X0, =Sshd ; "sshd"
--text:00000010005C84 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005C88 CBZ W0, loc_10005C9C
--text:00000010005C8C MOV W0, #1
--text:00000010005C90 MOV X0, X8
--text:00000010005C94 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005C98 RETAB
--text:00000010005C9C ;
--text:00000010005C9C ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+78+J
--text:00000010005C9C NOP
--text:00000010005CA0 LDR X0, =Checkra1nd ; "checkra1nd"
--text:00000010005CA4 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005CA8 CBZ W0, loc_10005CBC
--text:00000010005CAC MOV W0, #1
--text:00000010005CB0 MOV X0, X8
--text:00000010005CB4 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005CB8 RETAB
--text:00000010005CBC ;
--text:00000010005CBC ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+98+J
--text:00000010005CBC NOP
--text:00000010005CC0 LDR X0, =Loader ; "loader"
--text:00000010005CC4 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005CC8 CBZ W0, loc_10005CDC
--text:00000010005CCC MOV W0, #1
--text:00000010005CD0 MOV X0, X8
--text:00000010005CD4 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005CD8 RETAB
--text:00000010005CDC ;
--text:00000010005CDC ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+118+J
--text:00000010005CDC NOP
--text:00000010005CE0 LDR X0, =McAfee ; "McAfee"
--text:00000010005CE4 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005CE8 CBZ W0, loc_10005CFC
--text:00000010005CEC MOV W0, #1
--text:00000010005CF0 MOV X0, X8
--text:00000010005CF4 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005CF8 RETAB
--text:00000010005CFC ;
--text:00000010005CFC ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+138+J
--text:00000010005CFC NOP
--text:00000010005D00 LDR X0, =Avastmobilesec ; "AvastMobileSecurity"
--text:00000010005D04 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005D08 CBZ W0, loc_10005D1C
--text:00000010005D0C MOV W0, #1
--text:00000010005D10 MOV X0, X8
--text:00000010005D14 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005D18 RETAB
--text:00000010005D1C ;
--text:00000010005D1C ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)+158+J
--text:00000010005D1C NOP
--text:00000010005D20 LDR X0, =Nortonmobilese ; "NortonMobileSecurity"
--text:00000010005D24 BL __ZN5Utils6getPidEPKc ; Utils::getPid(char const*)
--text:00000010005D28 CBZ W0, #0
--text:00000010005D2C CSET X0, NE
--text:00000010005D30 MOV X0, X8
--text:00000010005D34 LDP X29, X30, [SP+var_s0],#0x10
--text:00000010005D38 RETAB
--text:00000010005D38 ; End of function CSWatcherSpawner::CSWatcherSpawner::is_unsafe_running(void)

```

Figure 8: is_unsafe_running() assembly — complete process detection list including tcpdump, frida-server, netstat, sshd, checkra1nd, loader, McAfee, AvastMobileSecurity, NortonMobileSecurity

[Enlarge image](#)

The complete list of detected processes also includes:

- `tcpdump` — network packet capture
- `frida-server` — dynamic instrumentation framework
- `netstat` — network statistics
- `sshd` — SSH daemon
- `checkra1nd` — checkra1n jailbreak
- `loader` — generic loader process
- `McAfee` — McAfee mobile security
- `AvastMobileSecurity` — Avast mobile security
- `NortonMobileSecurity` — Norton mobile security

The inclusion of netstat is noteworthy — it suggests Predator is concerned about targets who might be monitoring their own network connections, not just researchers with specialized tools. A privacy-conscious user simply checking their network connections would trigger this detection.

Self-cleanup after error reporting and shutdown

After reporting an error code to C2, Predator performs self-cleanup by removing its staging directory:

```

__TEXT:__TEXT 00000000100005A84 ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+19C+j
__TEXT:00000000100005AB4 loc_100005AB4 ; CODE XREF: CSWatcherSpawner::CSWatcherSpawner::check_perform(void)+19C+j
__TEXT:00000000100005AB4 ADR X0, _WatcherPathSecond ; "/private/var/tmp/"
__TEXT:00000000100005AB8 NOP
__TEXT:00000000100005ABC BL _remove
__TEXT:00000000100005AC0 MOV W0, #0 ; exc_buf
__TEXT:00000000100005AC4 LDP X29, X30, [SP,#0x30+var_s0]
__TEXT:00000000100005AC8 LDP X20, X19, [SP,#0x30+var_10]
__TEXT:00000000100005ACC LDP X22, X21, [SP,#0x30+var_20]
__TEXT:00000000100005AD0 ADD SP, SP, #0x40 ; '@'
__TEXT:00000000100005AD4 RETAB

```

Figure 9: Self-cleanup code calling `_remove` on `/private/var/tmp/` staging directory after error reporting

This cleanup happens after the C2 callback, ensuring operators receive the diagnostic information even if the malware is immediately removed.

Another cleanup mechanism is tied to the device shutdown lifecycle, the implant registers a Darwin notification observer for “com.apple.springboard.deviceWillShutDown”.

```

1 void __fastcall CSWatcherSpawner::CSWatcherSpawner::listenForShutdown(CSWatcherSpawner::CSWatcherSpawner *this)
2 {
3     __CFNotificationCenter *DarwinNotificationCenter; // x0
4     __int64 vars8; // [xsp+18h] [xpb+8h]
5
6     DarwinNotificationCenter = CFNotificationCenterGetDarwinNotifyCenter();
7     if ( ((vars8 ^ (2 * vars8)) & 0x4000000000000000LL) != 0 )
8         __break(0xC471u);
9     CFNotificationCenterAddObserver(
10        DarwinNotificationCenter,
11        this,
12        (CFNotificationCallback)CSWatcherSpawner::CSWatcherSpawner::onShutdown, // callback
13        CFSTR("com.apple.springboard.deviceWillShutDown"),
14        0,
15        CFNotificationSuspensionBehaviorDeliverImmediately);
16 }

```

Figure 10: Code to register listener for shutdown signal

When the shutdown notification is received, it transits to a cleanup routine removes specific on-disk evidences if they exist.

```

1 void __fastcall CSWatcherSpawner::CSWatcherSpawner::stop(CFRunLoopTimerRef *this)
2 {
3     std::error_code *v2; // x1
4     std::error_code *v3; // x1
5     __CFRunLoop *Main; // x0
6     __CFRunLoop *v5; // x0
7     __int64 vars8; // [xsp+18h] [xbp+8h]
8
9     if ( access(BitPath, 0) )
10    {
11        if ( access(HelperPath, 0) )
12            goto LABEL_3;
13    }
14    else
15    {
16        remove((const std::__fs::filesystem::path *)BitPath, v2);
17        if ( access(HelperPath, 0) )
18            goto LABEL_3;
19    }
20    remove((const std::__fs::filesystem::path *)HelperPath, v3);
21 LABEL_3:
22    Main = CFRunLoopGetMain();
23    CFRunLoopRemoveTimer(Main, this[49], kCFRunLoopDefaultMode);
24    v5 = CFRunLoopGetMain();
25    if ( ((vars8 ^ (2 * vars8)) & 0x4000000000000000LL) != 0 )
26        __break(0xC471u);
27    CFRunLoopStop(v5);
28 }

```

Figure 11: Self-cleanup code on shutdown process

Undocumented anti-forensics: crash reporter monitoring

One capability not mentioned in any public research is the `monitoringCrashReporter()` function:

```

1 // Anti-forensics: Monitors /private/var/mobile/Library/Logs/CrashReporter/ using kqueue for file system events.
2 __int64 __fastcall CSWatcherSpawner::CSWatcherSpawner::monitoringCrashReporter(
3     CSWatcherSpawner *this)
4 {
5     int v1; // w0
6     int v2; // w19
7     void *v3; // x22
8     kevent v5; // [xsp+0h] [xbp-70h] BYREF
9     kevent changelist; // [xsp+20h] [xbp-50h] BYREF
10
11     sleep(1u);
12     v1 = kqueue();
13     if ( (v1 & 0x80000000) == 0 )
14     {
15         v2 = v1;
16         changelist.ident = open("/private/var/mobile/Library/Logs/CrashReporter/", 0);
17         *(_QWORD *)&changelist.filter = 0xF0015FFFCLL;
18         changelist.data = 0;
19         changelist.udata = 0;
20         while ( 1 )
21         {
22             while ( kevent(v2, &changelist, 1, &v5, 1, 0) < 0 )
23                 ;
24             v3 = objc_autoreleasePoolPush();
25             -[NSArray enumerateObjectsUsingBlock:]{
26                 -[NSFileManager contentsOfDirectoryAtPath:error:]{
27                     +[NSFileManager defaultManager]{&OBJC_CLASS__NSFileManager, "defaultManager"},
28                     "contentsOfDirectoryAtPath:error:",
29                     CFSTR("/private/var/mobile/Library/Logs/CrashReporter/"),
30                     0,
31                     "enumerateObjectsUsingBlock:",
32                     &stru_100044A68);
33             objc_autoreleasePoolPop(v3);
34         }
35     }
36     return 0xFFFFFFFFLL;
37 }

```

Figure 12: monitoringCrashReporter() — kqueue-based monitoring of /private/var/mobile/Library/Logs/CrashReporter/

This function uses kqueue to monitor the CrashReporter directory for any new files. When a crash occurs that might expose Predator's presence, the handler processes or removes the crash log before it can be synced or examined.

Memory forensics suppression: killing mmaintenanced

The crash handler specifically targets memory forensics evidence:

```

1 // Block callback for crash reporter monitoring. Checks if crash report filename contains "SystemMemory" and kills mmaintenanced (signal 9) to suppress memory forensics evidence.
2 void __cdecl crash_reporter_block_handler(id a1, id a2, unsigned __int64 a3, bool *a4)
3 {
4     const char *v4; // x1
5     pid_t Pid; // w0
6     __int64 _8; // [xsp+8h] [xbp+8h]
7
8     if ( (unsigned int)objc_msgSend(a2, "containsString:", CFSTR("SystemMemory"), a4) )
9     {
10        Pid = Utils::getPid((Utils *)"mmaintenanced", v4);
11        if ( Pid )
12        {
13            if ( ( (_8 ^ (2 * _8)) & 0x400000000000000LL) != 0 )
14                break(0xC471u);
15            kill(Pid, 9);
16        }
17    }
18 }

```

Figure 13: crash_reporter_block_handler — kills mmaintenanced when SystemMemory crash reports are detected

[Enlarge image](#)

When a crash report filename contains "SystemMemory," the handler immediately locates and kills the mmaintenanced process (memory maintenance daemon) using SIGKILL (signal 9). This prevents investigators from capturing memory dumps.

This is significant for forensics: crash logs are a valuable artifact for detecting exploitation attempts, and Predator actively suppresses them.

Recording indicator hiding

The TestHooker() function reveals how Predator hides the iOS recording indicator from victims:

```

1 // UMHooker core function! Sets up hooks on remote processes using Mach exception handling. Uses mach_port_type_trap to verify port validity. Manages up to 16 hook slots per Hooker instance.
2 int __cdecl UMHooker(const char *processName, int hookCount, int *hookSlots)
3 {
4     mach_port_name_t v0; // w1
5     int i;
6     int j;
7     int k;
8     int l;
9     int m;
10    int n;
11    int o;
12    int p;
13    int q;
14    int r;
15    int s;
16    int t;
17    int u;
18    int v;
19    int w;
20    int x;
21    int y;
22    int z;
23    int aa;
24    int ab;
25    int ac;
26    int ad;
27    int ae;
28    int af;
29    int ag;
30    int ah;
31    int ai;
32    int aj;
33    int ak;
34    int al;
35    int am;
36    int an;
37    int ao;
38    int ap;
39    int aq;
40    int ar;
41    int as;
42    int at;
43    int au;
44    int av;
45    int aw;
46    int ax;
47    int ay;
48    int az;
49    int ba;
50    int bb;
51    int bc;
52    int bd;
53    int be;
54    int bf;
55    int bg;
56    int bh;
57    int bi;
58    int bj;
59    int bk;
60    int bl;
61    int bm;
62    int bn;
63    int bo;
64    int bp;
65    int bq;
66    int br;
67    int bs;
68    int bt;
69    int bu;
70    int bv;
71    int bw;
72    int bx;
73    int by;
74    int bz;
75    int ca;
76    int cb;
77    int cc;
78    int cd;
79    int ce;
80    int cf;
81    int cg;
82    int ch;
83    int ci;
84    int cj;
85    int ck;
86    int cl;
87    int cm;
88    int cn;
89    int co;
90    int cp;
91    int cq;
92    int cr;
93    int cs;
94    int ct;
95    int cu;
96    int cv;
97    int cw;
98    int cx;
99    int cy;
100   int cz;
101   int da;
102   int db;
103   int dc;
104   int dd;
105   int de;
106   int df;
107   int dg;
108   int dh;
109   int di;
110   int dj;
111   int dk;
112   int dl;
113   int dm;
114   int dn;
115   int do;
116   int dp;
117   int dq;
118   int dr;
119   int ds;
120   int dt;
121   int du;
122   int dv;
123   int dw;
124   int dx;
125   int dy;
126   int dz;
127   int ea;
128   int eb;
129   int ec;
130   int ed;
131   int ee;
132   int ef;
133   int eg;
134   int eh;
135   int ei;
136   int ej;
137   int ek;
138   int el;
139   int em;
140   int en;
141   int eo;
142   int ep;
143   int eq;
144   int er;
145   int es;
146   int et;
147   int eu;
148   int ev;
149   int ew;
150   int ex;
151   int ey;
152   int ez;
153   int fa;
154   int fb;
155   int fc;
156   int fd;
157   int fe;
158   int ff;
159   int fg;
160   int fh;
161   int fi;
162   int fj;
163   int fk;
164   int fl;
165   int fm;
166   int fn;
167   int fo;
168   int fp;
169   int fq;
170   int fr;
171   int fs;
172   int ft;
173   int fu;
174   int fv;
175   int fw;
176   int fx;
177   int fy;
178   int fz;
179   int ga;
180   int gb;
181   int gc;
182   int gd;
183   int ge;
184   int gf;
185   int gg;
186   int gh;
187   int gi;
188   int gj;
189   int gk;
190   int gl;
191   int gm;
192   int gn;
193   int go;
194   int gp;
195   int gq;
196   int gr;
197   int gs;
198   int gt;
199   int gu;
200   int gv;
201   int gw;
202   int gx;
203   int gy;
204   int gz;
205   int ha;
206   int hb;
207   int hc;
208   int hd;
209   int he;
210   int hf;
211   int hg;
212   int hh;
213   int hi;
214   int hj;
215   int hk;
216   int hl;
217   int hm;
218   int hn;
219   int ho;
220   int hp;
221   int hq;
222   int hr;
223   int hs;
224   int ht;
225   int hu;
226   int hv;
227   int hw;
228   int hx;
229   int hy;
230   int hz;
231   int ia;
232   int ib;
233   int ic;
234   int id;
235   int ie;
236   int if;
237   int ig;
238   int ih;
239   int ii;
240   int ij;
241   int ik;
242   int il;
243   int im;
244   int in;
245   int io;
246   int ip;
247   int iq;
248   int ir;
249   int is;
250   int it;
251   int iu;
252   int iv;
253   int iw;
254   int ix;
255   int iy;
256   int iz;
257   int ja;
258   int jb;
259   int jc;
260   int jd;
261   int je;
262   int jf;
263   int jg;
264   int jh;
265   int ji;
266   int jj;
267   int jk;
268   int jl;
269   int jm;
270   int jn;
271   int jo;
272   int jp;
273   int jq;
274   int jr;
275   int js;
276   int jt;
277   int ju;
278   int jv;
279   int jw;
280   int jx;
281   int jy;
282   int jz;
283   int ka;
284   int kb;
285   int kc;
286   int kd;
287   int ke;
288   int kf;
289   int kg;
290   int kh;
291   int ki;
292   int kj;
293   int kk;
294   int kl;
295   int km;
296   int kn;
297   int ko;
298   int kp;
299   int kq;
300   int kr;
301   int ks;
302   int kt;
303   int ku;
304   int kv;
305   int kw;
306   int kx;
307   int ky;
308   int kz;
309   int la;
310   int lb;
311   int lc;
312   int ld;
313   int le;
314   int lf;
315   int lg;
316   int lh;
317   int li;
318   int lj;
319   int lk;
320   int ll;
321   int lm;
322   int ln;
323   int lo;
324   int lp;
325   int lq;
326   int lr;
327   int ls;
328   int lt;
329   int lu;
330   int lv;
331   int lw;
332   int lx;
333   int ly;
334   int lz;
335   int ma;
336   int mb;
337   int mc;
338   int md;
339   int me;
340   int mf;
341   int mg;
342   int mh;
343   int mi;
344   int mj;
345   int mk;
346   int ml;
347   int mm;
348   int mn;
349   int mo;
350   int mp;
351   int mq;
352   int mr;
353   int ms;
354   int mt;
355   int mu;
356   int mv;
357   int mw;
358   int mx;
359   int my;
360   int mz;
361   int na;
362   int nb;
363   int nc;
364   int nd;
365   int ne;
366   int nf;
367   int ng;
368   int nh;
369   int ni;
370   int nj;
371   int nk;
372   int nl;
373   int nm;
374   int nn;
375   int no;
376   int np;
377   int nq;
378   int nr;
379   int ns;
380   int nt;
381   int nu;
382   int nv;
383   int nw;
384   int nx;
385   int ny;
386   int nz;
387   int oa;
388   int ob;
389   int oc;
390   int od;
391   int oe;
392   int of;
393   int og;
394   int oh;
395   int oi;
396   int oj;
397   int ok;
398   int ol;
399   int om;
400   int on;
401   int oo;
402   int op;
403   int oq;
404   int or;
405   int os;
406   int ot;
407   int ou;
408   int ov;
409   int ow;
410   int ox;
411   int oy;
412   int oz;
413   int pa;
414   int pb;
415   int pc;
416   int pd;
417   int pe;
418   int pf;
419   int pg;
420   int ph;
421   int pi;
422   int pj;
423   int pk;
424   int pl;
425   int pm;
426   int pn;
427   int po;
428   int pp;
429   int pq;
430   int pr;
431   int ps;
432   int pt;
433   int pu;
434   int pv;
435   int pw;
436   int px;
437   int py;
438   int pz;
439   int qa;
440   int qb;
441   int qc;
442   int qd;
443   int qe;
444   int qf;
445   int qg;
446   int qh;
447   int qi;
448   int qj;
449   int qk;
450   int ql;
451   int qm;
452   int qn;
453   int qo;
454   int qp;
455   int qq;
456   int qr;
457   int qs;
458   int qt;
459   int qu;
460   int qv;
461   int qw;
462   int qx;
463   int qy;
464   int qz;
465   int ra;
466   int rb;
467   int rc;
468   int rd;
469   int re;
470   int rf;
471   int rg;
472   int rh;
473   int ri;
474   int rj;
475   int rk;
476   int rl;
477   int rm;
478   int rn;
479   int ro;
480   int rp;
481   int rq;
482   int rr;
483   int rs;
484   int rt;
485   int ru;
486   int rv;
487   int rw;
488   int rx;
489   int ry;
490   int rz;
491   int sa;
492   int sb;
493   int sc;
494   int sd;
495   int se;
496   int sf;
497   int sg;
498   int sh;
499   int si;
500   int sj;
501   int sk;
502   int sl;
503   int sm;
504   int sn;
505   int so;
506   int sp;
507   int sq;
508   int sr;
509   int ss;
510   int st;
511   int su;
512   int sv;
513   int sw;
514   int sx;
515   int sy;
516   int sz;
517   int ta;
518   int tb;
519   int tc;
520   int td;
521   int te;
522   int tf;
523   int tg;
524   int th;
525   int ti;
526   int tj;
527   int tk;
528   int tl;
529   int tm;
530   int tn;
531   int to;
532   int tp;
533   int tq;
534   int tr;
535   int ts;
536   int tt;
537   int tu;
538   int tv;
539   int tw;
540   int tx;
541   int ty;
542   int tz;
543   int ua;
544   int ub;
545   int uc;
546   int ud;
547   int ue;
548   int uf;
549   int ug;
550   int uh;
551   int ui;
552   int uj;
553   int uk;
554   int ul;
555   int um;
556   int un;
557   int uo;
558   int up;
559   int uq;
560   int ur;
561   int us;
562   int ut;
563   int uu;
564   int uv;
565   int uw;
566   int ux;
567   int uy;
568   int uz;
569   int va;
570   int vb;
571   int vc;
572   int vd;
573   int ve;
574   int vf;
575   int vg;
576   int vh;
577   int vi;
578   int vj;
579   int vk;
580   int vl;
581   int vm;
582   int vn;
583   int vo;
584   int vp;
585   int vq;
586   int vr;
587   int vs;
588   int vt;
589   int vu;
590   int vv;
591   int vw;
592   int vx;
593   int vy;
594   int vz;
595   int wa;
596   int wb;
597   int wc;
598   int wd;
599   int we;
600   int wf;
601   int wg;
602   int wh;
603   int wi;
604   int wj;
605   int wk;
606   int wl;
607   int wm;
608   int wn;
609   int wo;
610   int wp;
611   int wq;
612   int wr;
613   int ws;
614   int wt;
615   int wu;
616   int wv;
617   int ww;
618   int wx;
619   int wy;
620   int wz;
621   int xa;
622   int xb;
623   int xc;
624   int xd;
625   int xe;
626   int xf;
627   int xg;
628   int xh;
629   int xi;
630   int xj;
631   int xk;
632   int xl;
633   int xm;
634   int xn;
635   int xo;
636   int xp;
637   int xq;
638   int xr;
639   int xs;
640   int xt;
641   int xu;
642   int xv;
643   int xw;
644   int xx;
645   int xy;
646   int xz;
647   int ya;
648   int yb;
649   int yc;
650   int yd;
651   int ye;
652   int yf;
653   int yg;
654   int yh;
655   int yi;
656   int yj;
657   int yk;
658   int yl;
659   int ym;
660   int yn;
661   int yo;
662   int yp;
663   int yq;
664   int yr;
665   int ys;
666   int yt;
667   int yu;
668   int yv;
669   int yw;
670   int yx;
671   int yy;
672   int yz;
673   int za;
674   int zb;
675   int zc;
676   int zd;
677   int ze;
678   int zf;
679   int zg;
680   int zh;
681   int zi;
682   int zj;
683   int zk;
684   int zl;
685   int zm;
686   int zn;
687   int zo;
688   int zp;
689   int zq;
690   int zr;
691   int zs;
692   int zt;
693   int zu;
694   int zv;
695   int zw;
696   int zx;
697   int zy;
698   int zz;
699   int aa;
700   int ab;
701   int ac;
702   int ad;
703   int ae;
704   int af;
705   int ag;
706   int ah;
707   int ai;
708   int aj;
709   int ak;
710   int al;
711   int am;
712   int an;
713   int ao;
714   int ap;
715   int aq;
716   int ar;
717   int as;
718   int at;
719   int au;
720   int av;
721   int aw;
722   int ax;
723   int ay;
724   int az;
725   int ba;
726   int bb;
727   int bc;
728   int bd;
729   int be;
730   int bf;
731   int bg;
732   int bh;
733   int bi;
734   int bj;
735   int bk;
736   int bl;
737   int bm;
738   int bn;
739   int bo;
740   int bp;
741   int bq;
742   int br;
743   int bs;
744   int bt;
745   int bu;
746   int bv;
747   int bw;
748   int bx;
749   int by;
750   int bz;
751   int ca;
752   int cb;
753   int cc;
754   int cd;
755   int ce;
756   int cf;
757   int cg;
758   int ch;
759   int ci;
760   int cj;
761   int ck;
762   int cl;
763   int cm;
764   int cn;
765   int co;
766   int cp;
767   int cq;
768   int cr;
769   int cs;
770   int ct;
771   int cu;
772   int cv;
773   int cw;
774   int cx;
775   int cy;
776   int cz;
777   int da;
778   int db;
779   int dc;
780   int dd;
781   int de;
782   int df;
783   int dg;
784   int dh;
785   int di;
786   int dj;
787   int dk;
788   int dl;
789   int dm;
790   int dn;
791   int do;
792   int dp;
793   int dq;
794   int dr;
795   int ds;
796   int dt;
797   int du;
798   int dv;
799   int dw;
800   int dx;
801   int dy;
802   int dz;
803   int ea;
804   int eb;
805   int ec;
806   int ed;
807   int ee;
808   int ef;
809   int eg;
810   int eh;
811   int ei;
812   int ej;
813   int ek;
814   int el;
815   int em;
816   int en;
817   int eo;
818   int ep;
819   int eq;
820   int er;
821   int es;
822   int et;
823   int eu;
824   int ev;
825   int ew;
826   int ex;
827   int ey;
828   int ez;
829   int fa;
830   int fb;
831   int fc;
832   int fd;
833   int fe;
834   int ff;
835   int fg;
836   int fh;
837   int fi;
838   int fj;
839   int fk;
840   int fl;
841   int fm;
842   int fn;
843   int fo;
844   int fp;
845   int fq;
846   int fr;
847   int fs;
848   int ft;
849   int fu;
850   int fv;
851   int fw;
852   int fx;
853   int fy;
854   int fz;
855   int ga;
856   int gb;
857   int gc;
858   int gd;
859   int ge;
860   int gf;
861   int gg;
862   int gh;
863   int gi;
864   int gj;
865   int gk;
866   int gl;
867   int gm;
868   int gn;
869   int go;
870   int gp;
871   int gq;
872   int gr;
873   int gs;
874   int gt;
875   int gu;
876   int gv;
877   int gw;
878   int gx;
879   int gy;
880   int gz;
881   int ha;
882   int hb;
883   int hc;
884   int hd;
885   int he;
886   int hf;
887   int hg;
888   int hh;
889   int hi;
890   int hj;
891   int hk;
892   int hl;
893   int hm;
894   int hn;
895   int ho;
896   int hp;
897   int hq;
898   int hr;
899   int hs;
900   int ht;
901   int hu;
902   int hv;
903   int hw;
904   int hx;
905   int hy;
906   int hz;
907   int ia;
908   int ib;
909   int ic;
910   int id;
911   int ie;
912   int if;
913   int ig;
914   int ih;
915   int ii;
916   int ij;
917   int ik;
918   int il;
919   int im;
920   int in;
921   int io;
922   int ip;
923   int iq;
924   int ir;
925   int is;
926   int it;
927   int iu;
928   int iv;
929   int iw;
930   int ix;
931   int iy;
932   int iz;
933   int ja;
934   int jb;
935   int jc;
936   int jd;
937   int je;
938   int jf;
939   int jg;
940   int jh;
941   int ji;
942   int jj;
943   int jk;
944   int jl;
945   int jm;
946   int jn;
947   int jo;
948   int jp;
949   int jq;
950   int jr;
951   int js;
952   int jt;
953   int ju;
954   int jv;
955   int jw;
956   int jx;
957   int jy;
958   int jz;
959   int ka;
960   int kb;
961   int kc;
962   int kd;
963   int ke;
964   int kf;
965   int kg;
966   int kh;
967   int ki;
968   int kj;
969   int kk;
970   int kl;
971   int km;
972   int kn;
973   int ko;
974   int kp;
975   int kq;
976   int kr;
977   int ks;
978   int kt;
979   int ku;
980   int kv;
981   int kw;
982   int kx;
983   int ky;
984   int kz;
985   int la;
986   int lb;
987   int lc;
988   int ld;
989   int le;
990   int lf;
991   int lg;
992   int lh;
993   int li;
994   int lj;
995   int lk;
996   int ll;
997   int lm;
998   int ln;
999   int lo;
1000  int lp;
1001  int lq;
1002  int lr;
1003  int ls;
1004  int lt;
1005  int lu;
1006  int lv;
1007  int lw;
1008  int lx;
1009  int ly;
1010  int lz;
1011  int ma;
1012  int mb;
1013  int mc;
1014  int md;
1015  int me;
1016  int mf;
1017  int mg;
1018  int mh;
1019  int mi;
1020  int mj;
1021  int mk;
1022  int ml;
1023  int mm;
1024  int mn;
1025  int mo;
1026  int mp;
1027  int mq;
1028  int mr;
1029  int ms;
1030  int mt;
1031  int mu;
1032  int mv;
1033  int mw;
1034  int mx;
1035  int my;
1036  int mz;
1037  int na;
1038  int nb;
1039  int nc;
1040  int nd;
1041  int ne;
1042  int nf;
1043  int ng;
1044  int nh;
1045  int ni;
1046  int nj;
1047  int nk;
1048  int nl;
1049  int nm;
1050  int nn;
1051  int no;
1052  int np;
1053  int nq;
1054  int nr;
1055  int ns;
1056  int nt;
1057  int nu;
1058  int nv;
1059  int nw;
1060  int nx;
1061  int ny;
1062  int nz;
1063  int oa;
1064  int ob;
1065  int oc;
1066  int od;
1067  int oe;
1068  int of;
1069  int og;
1070  int oh;
1071  int oi;
1072  int oj;
1073  int ok;
1074  int ol;
1075  int om;
1076  int on;
1077  int oo;
1078  int op;
1079  int oq;
1080  int or;
1081  int os;
1082  int ot;
1083  int ou;
1084  int ov;
1085  int ow;
1086  int ox;
1087  int oy;
1088  int oz;
1089  int pa;
1090  int pb;
1091  int pc;
1092  int pd;
1093  int pe;
1094  int pf;
1095  int pg;
1096  int ph;
1097  int pi;
1098  int pj;
1099  int pk;
1100  int pl;
1101  int pm;
1102  int pn;
1103  int po;
1104  int pp;
1105  int pq;
1106  int pr;
1107  int ps;
1108  int pt;
1109  int pu;
1110  int pv;
1111  int pw;
1112  int px;
1113  int py;
1114  int pz;
1115  int qa;
1116  int qb;
1117  int qc;
1118  int qd;
1119  int qe;
1120  int qf;
1121  int qg;
1122  int qh;
1123  int qi;
1124  int qj;
1125  int qk;
1126  int ql;
1127  int qm;
1128  int qn;
1129  int qo;
1130  int qp;
1131  int qq;
1132  int qr;
1133  int qs;
1134  int qt;
1135  int qu;
1136  int qv;
1137  int qw;
1138  int qx;
1139  int qy;
1140  int qz;
1141  int ra;
1142  int rb;
1143  int rc;
1144  int rd;
1145  int re;
1146  int rf;
1147  int rg;
1148  int rh;
1149  int ri;
1150  int rj;
1151  int rk;
1152  int rl;
1153  int rm;
1154  int rn;
1155  int ro;
1156  int rp;
1157  int rq;
1158  int rr;
1159  int rs;
1160  int rt;
1161  int ru;
1162  int rv;
1163  int rw;
1164  int rx;
1165  int ry;
1166  int rz;
1167  int sa;
1168  int sb;
1169  int sc;
1170  int sd;
1171  int se;
1172  int sf;
1173  int sg;
1174  int sh;
1175  int si;
1176  int sj;
1177  int sk;
1178  int sl;
1179  int sm;
1180  int sn;
1181  int so;
1182  int sp;
1183  int sq;
1184  int sr;
1185  int ss;
1186  int st;
1187  int su;
1188  int sv;
1189  int sw;
1190  int sx;
1191  int sy;
1192  int sz;
1193  int ta;
1194  int tb;
1195  int tc;
1196  int td;
1197  int te;
1198  int tf;
1199  int tg;
1200  int th;
1201  int ti;
1202  int tj;
1203  int tk;
1204  int tl;
1205  int tm;
1206  int tn;
1207  int to;
1208  int tp;
1209  int tq;
1210  int tr;
1211  int ts;
1212  int tt;
1213  int tu;
1214  int tv;
1215  int tw;
1216  int tx;
1217  int ty;
1218  int tz;
1219  int ua;
1220  int ub;
1221  int uc;
1222  int ud;
1223  int ue;
1224  int uf;
1225  int ug;
1226  int uh;
1227  int ui;
1228  int uj;
1229  int uk;
1230  int ul;
1231  int um;
1232  int un;
1233  int uo;
1234  int up;
1235  int uq;
1236  int ur;
1237  int us;
1238  int ut;
1239  int uu;
1240  int uv;
1241  int uw;
1242  int ux;
1243  int uy;
1244  int uz;
1245  int va;
1246  int vb;
1247  int vc;
1248  int vd;
1249  int ve;
1250  int vf;
1251  int vg;
1252  int vh;
1253  int vi;
1254  int vj;
1255  int vk;
1256  int vl;
1257  int vm;
1258  int vn;
1259  int vo;
1260  int vp;
1261  int vq;
1262  int vr;
1263  int vs;
1264  int vt;
1265  int vu;
1266  int vv;
1267  int vw;
1268  int vx;
1269  int vy;
1270  int vz;
1271  int wa;
1272  int wb;
1273  int wc;
1274  int wd;
1275  int we;
1276  int wf;
1277  int wg;
1278  int wh;
1279  int wi;
1280  int wj;
1281  int wk;
1282  int wl;
1283  int wm;
1284  int wn;
1285  int wo;
1286  int wp;
1287  int wq;
1288  int wr;
1289  int ws;
1290  int wt;
1291  int wu;
1292  int wv;
1293  int ww;
1294  int wx;
1295  int wy;
1296  int wz;
1297  int xa;
1298  int xb;
1299  int xc;
1300  int xd;
1301  int xe;
1302  int xf;
1303  int xg;
1304  int xh;
1305  int xi;
1306  int xj;
1307  int xk;
1308  int xl;
1309  int xm;
1310  int xn;
1311  int xo;
1312  int xp;
1313  int xq;
1314  int xr;
1315  int xs;
1316  int xt;
1317  int xu;
1318  int xv;
1319  int xw;
1320  int xx;
1321  int xy;
1322  int xz;
1323  int ya;
1324  int yb;
1325  int yc;
1326  int yd;
1327  int ye;
1328  int yf;
1329  int yg;
1330  int yh;
1331  int yi;
1332  int yj;
1333  int yk;
1334  int yl;
1335  int ym;
1336  int yn;
1337  int yo;
1338  int yp;
1339  int yq;
1340  int yr;
1341  int ys;
1342  int yt;
1343  int yu;
1344  int yv;
1345  int yw;
1346  int yx;
1347  int yy;
1348  int yz;
1349  int za;
1350  int zb;
1351  int zc;
1352  int zd;
1353  int ze;
1354  int zf;
1355  int zg;
1356  int zh;
1357  int zi;
1358  int zj;
1359  int zk;
1360  int zl;
1361  int zm;
1362  int zn;
1363  int zo;
1364  int zp;
1365  int zq;
1366  int zr;
1367  int zs;
1368  int zt;
1369  int zu;
1370  int zv;
1371  int zw;
1372  int zx;
137
```

[Enlarge image](#)

This code locates the SpringBoard process, uses kernel exploitation primitives to inject into SpringBoard, and hooks SBRecordingIndicatorManager methods to suppress the recording indicator. When Predator activates the microphone or camera, victims won't see the orange/green indicator dot.

Kernel exploitation class names

The code reveals several internal class names that indicate the exploitation architecture:

Table 2: Class names and their purposes

The `NSTaskROP::WithoutDeveloperMode` template is particularly interesting — it suggests Intellexa has developed ROP techniques that work even when developer mode is disabled, which is the default state for most users.

Stubbed functionality: `is_corellium()`

An interesting artifact is the `is_corellium()` function at address `0x100005bb8` :

Corellium is a cloud-based iOS device virtualization platform used by security researchers. The presence of this stubbed function suggests that Intellexa is aware of Corellium as an analysis platform; detection may have been implemented and later disabled, or detection is planned for future versions.

This function is not called from `check_perform()` in this sample, but its presence indicates awareness of the research community's tools.

Indicators of compromise

File access patterns

- `/private/var/preferences/SystemConfiguration/preferences.plist` (proxy check)
- `/private/var/preferences/Logging/com.apple.diagnostics.filter.plist` (console check)
- `/private/var/protected/trustd/private/TrustStore.sqlite3` (root CA check)
- `/private/var/mobile/Library/Logs/CrashReporter/` (crash monitoring)

Jailbreak detection paths

Process detection targets

SQL queries (TrustStore)

Conclusion

This analysis reveals that Predator's anti-analysis capabilities are more sophisticated than previously documented. The error code taxonomy demonstrates that Intellexa operators have granular visibility into why deployments fail, enabling them to adapt their approaches for specific targets.

For researchers, these findings highlight the importance of:

- **Air-gapped analysis environments:** The C2 callback means any network-connected analysis will alert operators.
- **Crash log preservation:** Enable crash log collection before analysis begins.
- **Process name awareness:** Even running netstat triggers detection.
- **Boot-time considerations:** Console logging configured before boot may evade timing-based detection.

For the broader security community, this analysis demonstrates that commercial spyware vendors invest significant engineering effort into detecting researchers — not just evading security products. The presence of the `is_corellium()` stub shows they're watching our tools as closely as we're watching theirs.

Appendix: function reference

Function	Address	Description
<code>check_perform()</code>	0x100005900	Main orchestration, error code dispatch
<code>is_developer()</code>	0x1000055a8	sysctlbyname developer_mode_status
<code>is_not_phone()</code>	0x1000054b0	Jailbreak path detection
<code>is_unsafe_running()</code>	0x100005c10	Process name detection (incl. netstat)
<code>is_proxy_running()</code>	0x100005d60	SystemConfiguration.plist parsing
<code>is_rootca_installed()</code>	0x100005e3c	TrustStore.sqlite3 queries
<code>is_console_attached()</code>	0x10000579c	diagnosticd mtime comparison
<code>is_restricted_country()</code>	0x100005758	NSLocale country code check
<code>getCountNames()</code>	0x10000c85c	/private/var/tmp/ process counting
<code>is_corellium()</code>	0x100005bb8	Stubbed Corellium detection
<code>ReportAbort()</code>	0x100005620	C2 error reporting
<code>monitoringCrashReporter()</code>	0x1000067d4	kqueue crash log monitoring
<code>TestHooker()</code>	0x1000068fc	SpringBoard indicator hooking

Table 3: List of functions and associated address and descriptions

Dive into more Jamf Threat Labs research on our blog.

Tags:

Source: <https://www.jamf.com/blog/predator-spyware-anti-analysis-techniques-ios-error-codes-detection/>