

วันที่ 22 ตุลาคม 2569

PHP2Shell พบ 2 ช่องโหว่ร้ายแรงใน PHP เสี่ยงถูกแฮกและรันโค้ดจากระยะไกลโดยไม่ต้องยืนยันตัวตน



วันที่ 21 ตุลาคม 2026 ทีม PHP ออกแพตช์ฉุกเฉินนอกกำหนด เพื่อปิดช่องโหว่ร้ายแรง 2 รายการที่อยู่ในตัว runtime ของ PHP เอง ไม่ใช่แอปพลิเคชันที่รันอยู่บน PHP ทำให้แม้แต่เว็บไซต์ที่เขียนโค้ดอย่างปลอดภัยก็ยังเสี่ยงถูกโจมตีได้ ช่องโหว่ทั้งคู่เปิดทางให้ผู้โจมตีที่ไม่มีบัญชีผู้ใช้ (unauthenticated) สามารถรันโค้ดได้จากระยะไกล (Remote Code Execution) บนเซิร์ฟเวอร์ที่ให้บริการเว็บผ่าน PHP แทบทุกรูปแบบการติดตั้ง ไม่ว่าจะเป็น Apache mod_php, PHP-FPM, php-cgi, FrankenPHP หรือ LiteSpeed

ความเสี่ยงสูงสุดอยู่ที่การตั้งค่าแบบ Apache + mod_php ซึ่งพิสูจน์แล้วว่าสามารถถูกโจมตีได้ด้วยคำขอ (request) เดียวโดยไม่ต้องยืนยันตัวตน และโค้ดที่ผู้โจมตีส่งมาจะทำงานก่อนที่แอปพลิเคชันจะเริ่มประมวลผลด้วยซ้ำ นอกจากการรันโค้ดแล้วผู้โจมตียังสามารถ อ่านหน่วยความจำ (memory) ของโปรเซส PHP worker ซึ่งอาจมีรหัสผ่าน โทเคนยืนยันตัวตน API key และใบรับรองของคำขออื่นๆ ที่ค้างอยู่ในหน่วยความจำได้โดยไม่ต้องรันโค้ดก่อนด้วยซ้ำ ประเด็นสำคัญที่สุดคือ ไม่มีการตั้งค่าใดที่ป้องกันช่องโหว่ได้ครบถ้วน การอัปเดต PHP เป็นเวอร์ชันที่แก้ไขแล้วเท่านั้นที่เป็นทางแก้ที่สมบูรณ์

รายละเอียดช่องโหว่ทั้งสองรายการ

1. CVE-2026-109618 - RCE ผ่านความเสียหายจากการจัดการหน่วยความจำไม่พอ (OOM corruption)
 - เมื่อการจองหน่วยความจำ (memory allocation) เกินค่า memory_limit PHP จะหยุดการทำงานกลางคัน (fatal) ขณะที่ข้อมูลคำขอภายในยังเริ่มต้นไม่สมบูรณ์ ขั้นตอนการเคลียร์หน่วยความจำหลังจบคำขอ (request cleanup) อาจเข้าใจผิดว่าข้อมูลที่ยังไม่สมบูรณ์นั้นใช้งานได้จริง แล้วนำไปประมวลผลต่อ ซึ่งนำไปสู่การทำให้ข้อมูลในหน่วยความจำเสียหายและเปิดทางให้รันโค้ดได้
2. CVE-2026-109619 - RCE ผ่านสถานะ global ที่ค้างอยู่ (stale global state)
 - สถานะภายในที่เป็นของคำขอหนึ่ง (request-owned internal state) อาจยังคงอยู่ แม้ว่าหน่วยความจำที่รองรับมันจะถูกคืนกลับไปแล้ว ทำให้คำขอถัดไปที่ worker เดียวกันรับมา อาจไปใช้สถานะที่ค้างอยู่นี้ก่อนที่ระบบจะรีเซ็ตค่าตามการเริ่มต้นปกติ

ทั้งสองช่องโหว่เกิดจากรูปแบบความผิดพลาดเดียวกัน ระบบเพิ่มตัวนับหรือประกาศสถานะว่าพร้อมใช้งานก่อน จะทำงานที่อาจล้มเหลวได้ เช่น การจองหน่วยความจำ แล้วค่อยเริ่มต้นค่าจริงทีหลัง หากงานที่อาจล้มเหลวนั้นเกิด ข้อผิดพลาดร้ายแรง (fatal error) ขึ้นกลางคัน การ "ย้อนกลับ" (rollback) หรือ "ยืนยัน" (commit) ที่ควรเกิดขึ้นจะถูกข้ามไป แต่ตัวนับ/สถานะที่ประกาศไว้ล่วงหน้ายังคงอยู่ ทำให้ส่วนอื่นของระบบที่นำข้อมูลไปใช้ต่อเข้าใจผิดว่าข้อมูลนั้นถูกต้องสมบูรณ์

วิธีที่นักวิจัยใช้ค้นพบช่องโหว่

ช่องโหว่ที่ถูกค้นพบโดยทีมวิจัยของ Wiz Research, Calif และทีม Codex Security ของ OpenAI โดยทำงานร่วมกับทีมความปลอดภัยของ PHP และทีม Gemini ของ Google ในขั้นตอนการแก้ไข จุดเริ่มต้นคือการย้อนรอยบั๊กที่ค้นพบครั้งแรกโดยนักวิจัยชื่อ Stefan เมื่อปี 2004 เพื่อค้นหาว่ายังมี "ข้อผิดพลาดลักษณะเดียวกัน" ของรูปแบบเดิมหลงเหลืออยู่ในชุดซอร์สโค้ดหรือไม่ โดยนักวิจัยใช้โมเดล AI ช่วยตรวจสอบโค้ดและสร้าง exploit ซึ่งพวกเขาระบุว่าช่วยให้งานที่ปกติต้องใช้ทีมงานหลายคนนานหลายเดือน ทำสำเร็จได้โดยคนเดียวในเวลาประมาณหนึ่งสัปดาห์ (แม้ยังต้องอาศัยความรู้พื้นฐานเฉพาะทางเกี่ยวกับช่องโหว่เก่าในการค้นหาโมเดล)

ขั้นตอนหลักของการค้นพบและสร้าง exploit มีดังนี้

ขั้นที่ 1 - หาวิธีกระตุ้น OOM (out-of-memory) แบบควบคุมได้

- พบว่า PHP ตรวจสอบค่า `post_max_size` เทียบกับ `Content-Length` ที่ประกาศมาเท่านั้น แต่ไม่ได้บังคับขีดจำกัดเดียวกันกับจำนวนไบต์ที่รับจริง ทำให้คำขอแบบ HTTP/1.1 chunked หรือ HTTP/2 ที่ไม่มีความยาวชัดเจน (multipart) สามารถส่งข้อมูลเกินขีดจำกัดที่ตั้งไว้ได้
- พบวิธีใช้ไฟล์อัปโหลดแบบ RFC1867 ที่มี `filename=""` (ไม่มีข้อมูลไฟล์จริง) ซึ่ง PHP จะไม่นับรวมในขีดจำกัด `max_file_uploads` แต่ยังคงสร้างข้อมูลใน `$_FILES` ถึง 6 รายการต่อไฟล์ หากตั้งชื่อไฟล์แบบซ้อนกันหลายชั้น (เช่น `files[a][b][c]...`) จะทำให้มีการจองหน่วยความจำจำนวนมากจากคำขอขนาดเล็กเพียงคำขอเดียว จนสามารถทำให้โปรเซส `memory_limit` ได้ตามต้องการ

ขั้นที่ 2 - หาจุดที่การถูกขัดจังหวะ (interrupt) จาก OOM ทำให้เกิดสถานะอันตราย นักวิจัยตรวจสอบโค้ดในส่วนการจัดการหน่วยความจำ (ZendMM), การรายงาน ข้อผิดพลาดร้ายแรง (fatal error) และกระบวนการปิดคำขอ (request shutdown) พบรูปแบบอันตรายในฟังก์ชันจัดการ HashTable ภายในของ PHP ซึ่งมีการเพิ่มตัวนับจำนวนสมาชิกก่อน จะมีการจองหน่วยความจำก็จริง หากการจองก็เกิด fatal ระหว่างกลาง ตัวนับจะถูกเพิ่มไปแล้ว ทำให้กระบวนการเคลียร์หน่วยความจำ (teardown) เข้าใจว่ามีข้อมูลสมบูรณ์อยู่ในตำแหน่งนั้น ทั้งที่ยังไม่ได้ถูกเริ่มต้นค่าจริง นำไปสู่การใช้งานหน่วยความจำที่ "ค้าง" (stale) อยู่

บนระบบที่ใช้ Apache + mod_php เมื่อการเริ่มต้นคำขอ (request startup) ล้มเหลว โค้ดที่ตามมายังคงเรียกกระบวนการปิดคำขอตามปกติ ซึ่งจะไปกระตุ้นให้ประมวลผลข้อมูลที่ค้างอยู่ ส่งผลให้สามารถรันโค้ดได้โดยไม่ต้องยืนยันตัวตน ขณะที่บน

PHP-FPM การล้มเหลวของการเริ่มต้นคำขอจะทำให้ worker โพรเซสนั้นถูกปิดไปเลย ไม่เปิดโอกาสให้คำขอถัดไปมาใช้สถานะที่ค้างอยู่ จึงเป็นเหตุผลที่ mod_php เสี่ยงกว่า

ขั้นที่ 3 - สรุปรูปแบบทั่วไป แล้วค้นหาเพิ่มด้วย AI หลังพบบั๊กแรก นักวิจัยให้ AI สรุปรูปแบบข้อผิดพลาดที่พบร่วมกันเป็นชุดคำแนะนำ เพื่อค้นหาโค้ดลักษณะ "ประกาศ/ทำลายสถานะ → ทำงานที่อาจล้มเหลวได้ → เริ่มต้น/ยืนยัน/คืนค่า" ในจุดอื่นๆ ของชุดซอร์สโค้ด ซึ่งนำไปสู่การพบช่องโหว่ที่สองในพีเจอร์ Session Upload Progress

ขั้นที่ 4 - การค้นพบในพีเจอร์ Session Upload Progress พีเจอร์นี้เปิดใช้งานเป็นค่าเริ่มต้น และสามารถเริ่มต้น/อัปเดตเซสชันได้ตั้งแต่ตอนที่ PHP กำลังแยกวิเคราะห์เนื้อหาคำขอก่อนที่จะโมดูลต่างๆ จะถูกเรียกใช้งานอย่างเป็นทางการ หากมีการส่งไฟล์แรกในคำขอ multipart พีเจอร์นี้จะเริ่มเซสชันเร็วกว่าปกติ และหากคำขอนั้นเกิด ข้อผิดพลาดร้ายแรง (fatal error) ระหว่างทาง ขั้นตอนทำความสะอาด (cleanup) ของเซสชันจะถูกข้ามไป แต่หน่วยความจำของคำขอจะถูกคืนกลับไปแล้ว ทำให้ตัวแปร global ของโมดูล Session ยังคงชี้ไปยังหน่วยความจำที่ถูกปล่อยคืน (dangling pointer)

จากนั้นนักวิจัยพบว่าสามารถส่ง คำขอครั้งที่สอง ที่มี Session ID และฟิลด์ PHP_SESSION_UPLOAD_PROGRESS แต่ไม่มีไฟล์แนบ เพื่อกระตุ้นขั้นตอนทำความสะอาดโดยตรง (ข้ามขั้นตอนเริ่มต้นใหม่) ทำให้ระบบไปอ่าน/ทำลายค่าจาก Session ID และข้อมูลเซสชันที่ "ค้าง" (stale) อยู่จากคำขอก่อนหน้านี้ นี่คือการโจมตีแบบ unauthenticated RCE บน mod_php ที่ใช้ได้ตั้งแต่ PHP 5.4.0 เป็นต้นมา

ขั้นที่ 5 - ขยายผลไปถึง PHP-FPM ผ่าน Generic Unserializer เนื่องจาก PHP-FPM จะปิด worker ทันทีหากเกิด fatal หลังจาก startup สำเร็จแล้วแทนพวกเขาพบว่ากลไก unserializer ทั่วไปของ PHP ที่ใช้งานผ่าน unserialize(), session_decode(), หรือฟังก์ชันห่อหุ้มอย่าง maybe_unserialize() ของ WordPress เก็บสถานะที่ใช้ร่วมกันไว้ใน global ของโมดูล Basic ซึ่งจะถูกลบตามปกติเมื่อถอดรหัสสำเร็จ แต่หาก fatal เกิดขึ้นระหว่างถอดรหัส สถานะนี้จะไม่ถูกลบออก ทำให้ worker ที่ยังมีชีวิตอยู่เพราะ fatal เกิดหลัง startup มีสถานะค้างอยู่ที่คำขอถัดไปใช้งานได้ โดยอาศัย Session Upload Progress เป็นช่องทางเข้าถึงสถานะที่ค้างนั้นอีกครั้ง จุดนี้เปิดทางให้ผู้โจมตีควบคุมตำแหน่งหน่วยความจำที่ PHP จะเขียนค่าซ้อนทับ (คล้ายการโจมตีแบบ memory corruption ที่แม่นยำ) ซึ่งนำไปสู่ RCE ได้ทั้งบน Apache mod_php และ PHP-FPM แต่ขึ้นกับว่าแอปพลิเคชันมีจุดเรียกใช้ฟังก์ชันถอดรหัสที่เข้าถึงได้จากภายนอกหรือไม่

ผลการทดสอบการโจมตีจริง

นักวิจัยสาธิตการโจมตีสำเร็จแบบ pre-auth RCE บนแอปพลิเคชันดังต่อไปนี้ WordPress, Drupal, Joomla, MediaWiki, Roundcube, Magento, phpMyAdmin, Nextcloud, Laravel, Symfony, WHMCS รวมถึงสคริปต์ PHP ทั่วไปในบางกรณี และยังสาธิตการอ่านหน่วยความจำของ worker (ซึ่งอาจมีข้อมูลลับของคำขออื่น) ต่อแอปพลิเคชันที่นักวิจัยไม่ได้เป็นเจ้าของ เพื่อยืนยันว่าปัญหานี้ไม่ขึ้นกับโค้ดของแอปพลิเคชันใดแอปพลิเคชันหนึ่ง

ที่น่าสนใจคือ รูปแบบข้อผิดพลาดที่คล้ายกันเคยถูกแพตช์ใน HHVM (Hack VM) มาตั้งแต่ปี 2020 จากการเปลี่ยนแปลงโค้ดหนึ่งของ PHP เอง และพบปัญหากับ "security-audit-2024" ติดอยู่กับการเปลี่ยนแปลงโค้ดที่เกี่ยวข้อง ซึ่งสอดคล้องกับที่

PHP Foundation ยอมรับในภายหลังว่าเคยพบสัญญาณบางอย่างของบั๊กนี้ในการตรวจสอบโค้ดภายนอกเมื่อปี 2024 แต่ตอนนั้นยังสร้างเส้นทางโจมตีจริงไม่สำเร็จ จึงไม่ได้ถูกจัดลำดับความสำคัญสูงสุด

ระบบที่ได้รับผลกระทบ

- ทุกการติดตั้ง PHP ที่ให้บริการเว็บ ได้แก่ PHP-FPM, Apache mod_php, php-cgi, FrankenPHP, LiteSpeed
- มีความเสี่ยงสูงสุด: Apache mod_php (unauthenticated RCE ด้วยคำขอเดียว ก่อนแอปพลิเคชันเริ่มทำงาน)
- CVE-2026-109618: ยืนยันผลกระทบแน่ชัดต่อ mod_php; สำหรับ PHP-FPM มีความเป็นไปได้ตามการวิเคราะห์ของ PHP เอง แต่ยังไม่มีการสาธิต exploit ที่สมบูรณ์
- CVE-2026-109619: สาธิตได้ทั้งบน mod_php และ PHP-FPM แต่เส้นทางบน FPM ขึ้นกับว่าแอปพลิเคชันมีจุดเรียกใช้ฟังก์ชัน deserialize ที่เข้าถึงได้จากภายนอกหรือไม่
- ช่วงเวอร์ชันที่ได้รับผลกระทบ: ย้อนไปถึง PHP 5.4.0 (บางเส้นทาง) จนถึงก่อนเวอร์ชันที่แก้ไข
- จากการสแกนของบริษัท Mallory เมื่อ 20 ตุลาคม พบเซิร์ฟเวอร์ที่เข้าถึงได้ผ่านอินเทอร์เน็ตและรายงานเวอร์ชัน PHP 8.1 หรือเก่ากว่าในฐานะข้อมูล Shodan ประมาณ 884,000 เครื่อง คิดเป็น 59% ของโฮสต์ประมาณ 1.5 ล้านเครื่องที่รายงานเวอร์ชันใดๆ ยังมีอีกราว 740,000 เครื่องที่รายงานว่าใช้ PHP แต่ไม่ระบุเวอร์ชัน

เวอร์ชันที่แก้ไขแล้ว

สายเวอร์ชัน	สถานะก่อนแพตช์	เวอร์ชันที่แก้ไขแล้ว
PHP 8.5	ต่ำกว่า 8.5.12	8.5.12
PHP 8.4	ต่ำกว่า 8.4.27	8.4.27
PHP 8.3	ต่ำกว่า 8.3.36	8.3.36
PHP 8.2	ต่ำกว่า 8.2.35	8.2.35

- PHP 8.1 และเก่ากว่า (8.0, 7.x, 5.x): หมดอายุการสนับสนุน (end-of-life) ไม่มีแพตช์อย่างเป็นทางการจากทีม PHP อย่างไรก็ตาม ทีมความปลอดภัยของ PHP ได้จัดทำ source backport เฉพาะกิจ ให้สำหรับ PHP 7.4, 8.0 และ 8.1 เพื่อช่วยผู้ที่ต้อง build เองหรือรอผู้ให้บริการ
- ดิสโทรลินุกซ์และผู้ให้บริการ LTS หลายรายได้รับแพตช์ล่วงหน้าแล้ว จึงอาจมีเลขเวอร์ชันเดิมที่ดูเหมือนยังไม่อัปเดต แต่ได้รับการนำแพตช์ไปปรับใช้กับเวอร์ชันเดิมแล้ว (backport) ควรตรวจสอบจาก changelog ของแพ็คเกจว่ามีการอ้างอิง CVE นี้หรือไม่ ไม่ควรดูจากเลขเวอร์ชันเพียงอย่างเดียว
- ผู้ที่พึ่งพา repository แบบอาสาสมัคร rpms.remirepo.net ได้รับคำแนะนำให้เปลี่ยนไปใช้แพ็คเกจของดิสโทรหรือ build เองแทน เนื่องจาก repo นี้จะไม่มีการอัปเดตทันที

คำแนะนำการแก้ไขและบรรเทาความเสี่ยง

อัปเดต PHP runtime ที่ให้บริการเว็บจริง (ไม่ใช่แค่แอปพลิเคชันที่รันอยู่บน PHP เช่น WordPress หรือ Laravel) เป็นเวอร์ชัน 8.5.12 / 8.4.27 / 8.3.36 / 8.2.35 ตามสายที่ใช้งาน จากนั้น รีสตาร์ท Apache, PHP-FPM และ PHP worker ที่ทำงานระยะยาว (long-running) ทุกตัว เพราะโปรเซสเก่าที่ยังทำงานอยู่จะไม่ได้รับผลของแพตช์จนกว่าจะรีสตาร์ท

หากใช้ container หรือ image สำเร็จรูป ต้อง build/pull image ใหม่ที่มี PHP เวอร์ชันแก้ไขแล้ว ไม่ใช่แค่แก้โค้ดแอปพลิเคชันภายใน image เดิม

หากยังอัปเดตไม่ได้ทันที (ชั่วคราวเท่านั้น ไม่ใช่วิธีแก้ไขที่สมบูรณ์)

สำหรับ CVE-2026-109618 (OOM corruption)

- ตั้งค่า `memory_limit = -1` โดยใช้วิธีที่แอปพลิเคชันเขียนทับไม่ได้ เช่น `php_admin_value` ในพูลของ FPM หรือในค่าคอนฟิกเซิร์ฟเวอร์ระดับเซิร์ฟเวอร์ และตรวจสอบว่าไม่มีการตั้งค่าขีดจำกัดอื่นที่ไปจำกัดซ้ำในระดับ `pool/directory/application`
- ตั้งค่า `post_max_size` ให้ต่ำ: หากไม่ต้องใช้การอัปโหลดไฟล์ ให้ตั้ง `post_max_size = 8M`; ถ้าจำเป็นต้องอัปโหลดไฟล์ ให้ตั้ง `post_max_size = 32M` และต้องให้ `upload_max_filesize` ต่ำกว่า `post_max_size` เสมอ
- บังคับขีดจำกัดขนาดคำขอ (decoded request size), จำนวนฟิลด์, จำนวนส่วน multipart, จำนวน header และระดับความซ้อนของข้อมูล ที่ระดับเว็บเซิร์ฟเวอร์หรือ proxy ด้านหน้า โดยต้องครอบคลุมคำขอที่ไม่มี Content-Length ที่ใช้งานได้ด้วย
- ใช้ขีดจำกัดระดับ container/cgroup เป็นแนวป้องกันด้าน availability เลือกนโยบายที่ฆ่าเฉพาะ worker ที่มีปัญหา ไม่ควรใช้ `RLIMIT_AS` หรือ `RLIMIT_DATA` แทน เพราะ PHP เตือนว่าค่าเหล่านี้อาจไปกระตุ้นให้เกิดเส้นทางที่เสี่ยงเสียเอง

สำหรับ CVE-2026-109619 (stale global state)

- ตั้งค่า `session.upload_progress.enabled = 0` แล้ว รีสตาร์ท worker ทุกตัว โดยต้องตั้งผ่าน `php.ini`, FPM pool หรือค่าคอนฟิกเซิร์ฟเวอร์ที่มีผลก่อนการแยกวิเคราะห์คำขอ ไม่ควรใช้ `ini_set()` หรือ `.user.ini` ของ FPM เพราะจะไม่มีผลทันเวลา
- บน PHP 8.5 ให้ตั้งค่า `fatal_error_backtraces = 0` เพิ่มเติม แล้วรีสตาร์ท worker
- ลบการตั้งค่าที่ไม่ถูกต้องหรือเลิกใช้แล้ว (deprecated) ที่ทำให้เกิดข้อความแจ้งเตือน (diagnostic) ก่อนขั้นตอนเริ่มต้นจะเสร็จสมบูรณ์ — การปิดไม่ให้แสดงข้อความเหล่านี้ด้วย `error_reporting`, `display_errors` หรือ `@` ไม่ช่วยแก้ปัญหา

- แก้ไข ข้อผิดพลาดร้ายแรง (fatal error) หรือ timeout ที่ผู้โจมตีสามารถกระตุ้นให้เกิดได้ รวมถึงความล้มเหลวใน callback, การประมวลผลไฟล์อัปโหลด/นำเข้าข้อมูล และขั้นตอน cleanup ต่างๆ

ข้อควรระวังสำคัญ: การตั้งค่า session.upload_progress.enabled=0 เพียงอย่างเดียว ป้องกันได้เฉพาะรูปแบบการโจมตีผ่าน session ที่ยืนยันแล้วเท่านั้น ไม่ครอบคลุมรูปแบบอื่น และ ไม่มีผลใดๆ ต่อ CVE-2026-109618 เลย นักพัฒนาหลักของ PHP เองยืนยันชัดเจนว่า "ทางแก้เดียวคือการแพตช์/อัปเดต PHP เท่านั้น"

หากยังไม่สามารถอัปเดตหรือยังใช้เวอร์ชัน end-of-life

- **พิจารณา** ตัดการเชื่อมต่อระบบจากอินเทอร์เน็ตสาธารณะชั่วคราว จนกว่าจะแพตช์ได้ (คำแนะนำจาก CERT-EU)
- **หากปิดระบบไม่ได้** ให้จำกัดการเข้าถึง เช่น วาง VPN, อนุญาตเฉพาะ IP ที่กำหนด (allowlist) หรือกำหนดการยืนยันตัวตนที่ชั้น edge และตรวจสอบว่าไม่มีช่องทางที่ทราฟฟิกจะเข้าถึง origin เซิร์ฟเวอร์โดยตรงโดยข้ามการควบคุมเหล่านี้
- **จำกัดผลกระทบหากผู้โจมตีเข้าถึงระบบได้สำเร็จ:** ลดข้อมูลที่เก็บไว้บนเซิร์ฟเวอร์, ลดสิทธิ์ (privilege) ของโปรเซส, ถอด credential ที่เก็บไว้ (รวมถึง cloud role), และเปิดใช้งาน logging พร้อมเก็บ log ไว้ในที่ปลอดภัยเพื่อใช้สืบสวนย้อนหลัง
- **สำหรับเวอร์ชันเก่ากว่า 8.2 (รวม 8.1, 7.x, 5.x):** อัปเดตเป็นเวอร์ชันที่รองรับ, ถอดออกจากอินเทอร์เน็ต, หรือใช้ source backport เฉพาะกิจที่ทีม PHP จัดทำให้ (สำหรับ 7.4/8.0/8.1) ดิสโทรหรือผู้ให้บริการโฮสติ้งบางรายอาจ backport แพตช์ตามตารางเวลาของตนเอง

มาตรการเสริมอื่นๆ ที่นักวิจัยบางรายเสนอ

- **บล็อกคำขอที่มีเนื้อหาแบบ multipart/form-data ที่ชั้นเว็บเซิร์ฟเวอร์** ก่อนที่จะถึง PHP เนื่องจากช่องโหว่ที่ส่งสัยอยู่ในวิธีที่ PHP อ่านเนื้อหาประเภทนี้ ต้องบล็อกทุก HTTP method ไม่ใช่แค่ POST เพราะ PHP 8.4 เพิ่มฟังก์ชัน request_parse_body() ที่อ่าน multipart body จาก method อื่นได้ด้วย มาตรการนี้มีต้นทุน ฟิเจอร์อัปโหลดไฟล์และฟอร์มบางประเภทจะใช้งานไม่ได้จนกว่าจะปลดบล็อก
- **Web Application Firewall (WAF)** เช่น กฎฉุกเฉินของ Cloudflare เป็นเพียง มาตรการชั่วคราว (stopgap) ไม่ใช่ทางแก้ถาวร ใช้ป้องกันได้เฉพาะทราฟฟิกที่ผ่าน WAF จริง และอาจพลาดเส้นทางโจมตีใหม่ที่ยังไม่รู้จัก Cloudflare เองก็ยืนยันจุดนี้อย่างชัดเจน

สำหรับผู้โฮสติ้งที่ไม่ได้ดูแล PHP เอง

การอัปเดตแอปพลิเคชัน เช่น WordPress, Laravel ไม่ได้อัปเดต PHP runtime เบื้องหลัง ต้องตรวจสอบกับผู้ให้บริการโฮสติ้งหรือดิสโทรว่าจะมีแพ็กเกจ PHP ที่แก้ไขแล้วเมื่อใด และเตรียมทีมงานให้พร้อมดำเนินการทันทีที่แพตช์พร้อมใช้งาน

การตรวจสอบหลังแพตช์และสัญญาณที่อาจบ่งบอกถึงการโจมตี (IOC)

หลังจากอัปเดตแล้ว ควรตรวจสอบเพิ่มเติม

- ทบทวนค่า memory_limit และ post_max_size เทียบกับความต้องการจริงของแอปพลิเคชัน เนื่องจากแพตช์เพิ่มขีดจำกัดหน่วยความจำแบบ hard limit ใหม่ ซึ่งเมื่อถึงขีดจำกัดจะยุติโปรเซสทันทีและ ไม่รัน shutdown handler
- ตรวจสอบ log ย้อนหลังก่อนแพตช์ เพื่อหาสัญญาณการถูกโจมตีที่อาจเกิดขึ้นแล้ว ได้แก่
 - คำขอ multipart ที่ผิดปกติ ตามด้วยข้อความ error เกี่ยวกับ memory limit หรือ HTTP 500 ซ้ำๆ
 - ค่า PHPSESSID ที่มีขนาดใหญ่ผิดปกติในการตอบกลับ
 - การรีสตาร์ท PHP worker โดยไม่ทราบสาเหตุ
 - จากคำแนะนำทั่วไปของ Huntress การสังเกต process ที่ php-fpm, php-cgi, httpd หรือ apache2 เรียกใช้ shell, interpreter หรือเครื่องมือดาวน์โหลดไฟล์ผิดปกติ
- หากพบสิ่งผิดปกติ: แยกเซิร์ฟเวอร์ออกจากระบบ (isolate), เก็บรักษา log ของเว็บเซิร์ฟเวอร์และ PHP ไว้เป็นหลักฐาน, และตรวจหา web shell, ไฟล์ใหม่ใน web root, cron job หรือ scheduled task ใหม่ และบัญชีผู้ใช้ใหม่ที่ไม่ได้รับอนุญาต

ข้อมูลอ้างอิง

Oct 22, 2026, By CSOC UIH

- <https://thehackernews.com/2026/10/php2shell-two-unpatched-critical-php.html>
- <https://thehackernews.com/2026/10/php2shell-what-actually-protects-you.html>
- <https://thehackernews.com/2026/10/php-issues-emergency-patches-for-two.html>
- <https://calif.io/php2shell/#patch-the-planet>
- <https://www.php.net/archive/2026.php#2026-10-21-5>
- <https://github.com/php/php-src/security/advisories/GHSA-6wmf-4rv5-p3rp>
- <https://github.com/php/php-src/security/advisories/GHSA-w8hw-w9fx-j79w>