{

"title": "Session Management Response Identified",

"description": "The given response has been identified as containing a session management token. The 'Other Info' field contains a set of header tokens that can be used in the Header Based Session Management Method. If the request is in a context which has a Session Management Method set to \"Auto-Detect\" then this rule will change the session management to use the tokens identified.",

"simplified\_description": "This is like a security scanner noticing that your website uses a specific key (session token) to remember users. It's not a problem - just the scanner documenting how your site tracks logged-in users. Think of it as a researcher taking notes on which key you use to identify visitors, not finding anything wrong with the lock itself.",

"solution": "This is an informational alert rather than a vulnerability and so there is nothing to fix.",

"simplified\_solution": "No action is required. This is informational only - the scanner is simply noting that it detected how your application manages user sessions.",

"code\_solution\_sample": {

"solution\_description": "Since this is informational and not a vulnerability, the code example demonstrates best practices for secure session management implementation. The sample shows how to properly configure session tokens with security attributes like HttpOnly, Secure, and SameSite flags to prevent common session-related attacks such as XSS-based session theft and CSRF.",

"affected\_files": "Session configuration files such as web.xml (Java), settings.py (Django), session middleware configuration files (Express.js), or application startup files where session management is initialized",

"code": "// Node.js/Express - Secure Session Configuration Example\nconst express = require('express');\nconst session = require('express-session');\nconst app = express();\n\n// Configure session middleware with security best practices\napp.use(session({\n name: 'SessionID', // Custom session cookie name (matches detected token)\n secret: process.env.SESSION\_SECRET, // Strong secret from environment variable\n resave: false,\n saveUninitialized: false,\n cookie: {\n httpOnly: true, // Prevents client-side JavaScript access (XSS protection)\n secure: true, // Ensures cookie is only sent over HTTPS\n sameSite: 'strict', // Prevents CSRF attacks by restricting cross-site requests\n maxAge: 3600000, // Session expires after 1 hour (3600000ms)\n domain: 'pentest-ground.com' // Restrict cookie to specific domain\n },\n rolling: true // Reset expiration on each request (sliding window)\n}));\n\n// Session regeneration after authentication to prevent session fixation\napp.post('/login', (req, res) => {\n // Validate credentials (authentication logic here)\n if (authenticateUser(req.body)) {\n // Regenerate session ID after successful login\n req.session.regenerate((err) => {\n if (err) {\n return res.status(500).json({ error: 'Session error' });\n }\n // Store user info in new session\n req.session.userId = req.body.userId;\n res.json({ message: 'Login successful' });\n });\n }\n});\n\n// Destroy session on logout\napp.post('/logout', (req, res) => {\n req.session.destroy((err) => {\n if (err) {\n return res.status(500).json({ error: 'Logout failed' });\n }\n res.clearCookie('SessionID');\n res.json({ message: 'Logged out successfully' });\n });\n});\n\nmodule.exports = app;"

}

}

    {

        "title": "Missing Anti-clickjacking Header",

        "description": "The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.",

        "simplified\_description": "Your website can be tricked into appearing inside a hidden frame on a malicious website, like putting your house window inside someone else's window so they can trick people into clicking on your stuff without knowing it.",

        "solution": "Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.\nIf you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's \"frame-ancestors\" directive.",

        "simplified\_solution": "Add a security header to your web pages that tells browsers whether your site can be displayed inside frames on other websites, preventing clickjacking attacks.",

        "code\_solution\_sample": {

            "solution\_description": "This code adds the X-Frame-Options header to all HTTP responses, which instructs browsers to either deny framing entirely (DENY) or only allow framing from the same origin (SAMEORIGIN). This prevents malicious websites from embedding your pages in hidden frames.",

            "affected\_files": "This is usually found in server configuration files, web server settings (like Apache/Nginx), or in application middleware where HTTP security headers are configured.",

            "code": "// Node.js/Express example\nconst express = require('express');\nconst app = express();\n\n// Add X-Frame-Options header to prevent clickjacking\napp.use((req, res, next) => {\n  // DENY: Prevents any domain from framing the content\n  // SAMEORIGIN: Only allows framing from the same domain\n  res.setHeader('X-Frame-Options', 'DENY');\n  next();\n});\n\n// Alternative: Using Content Security Policy\napp.use((req, res, next) => {\n  // frame-ancestors 'none' is equivalent to X-Frame-Options: DENY\n  res.setHeader('Content-Security-Policy', \"frame-ancestors 'none'\");\n  next();\n});"

        }

    }

{

"title": "Re-examine Cache-control Directives",

"description": "The cache-control header has not been set properly or is missing, allowing the browser and proxies to cache content. For static assets like css, js, or image files this might be intended, however, the resources should be reviewed to ensure that no sensitive content will be cached.",

"simplified\_description": "Think of caching like making photocopies of documents. Your website isn't telling browsers and servers whether they can make copies of pages. This is fine for public information like images, but dangerous for sensitive pages like login forms or user dashboards - someone could access old 'photocopies' with private information stored in the browser or network.",

"solution": "For secure content, ensure the cache-control HTTP header is set with \"no-cache, no-store, must-revalidate\". If an asset should be cached consider setting the directives \"public, max-age, immutable\".",

"simplified\_solution": "Configure your server to tell browsers not to save copies of sensitive pages (like login or account pages), but allow caching of public resources like images and stylesheets to improve performance.",

"code\_solution\_sample": {

"solution\_description": "This code implements middleware that automatically sets appropriate cache-control headers based on the type of content being served. Sensitive routes (login, edit, user-specific content) receive strict no-cache directives to prevent browsers and proxies from storing sensitive data, while static assets and public pages can be cached for better performance. The solution addresses all identified URLs by applying context-aware caching policies.",

"affected\_files": "Application middleware configuration files, web server configuration files (Apache httpd.conf/.htaccess, Nginx nginx.conf), application startup files (app.js, server.js, main.py), or framework-specific configuration files where HTTP headers are set globally",

"code": "// Node.js/Express - Secure Cache-Control Middleware Implementation\nconst express = require('express');\nconst app = express();\n\n// Middleware to set cache-control headers based on route sensitivity\nconst secureCache = (req, res, next) => {\n const path = req.path.toLowerCase();\n \n // Sensitive routes - prevent all caching\n const sensitiveRoutes = [\n '/login',\n '/logout', \n '/dashboard',\n '/profile',\n '/account',\n '/admin',\n '/edit',\n '/api/user'\n ];\n \n // Check if current path matches sensitive route patterns\n const isSensitive = sensitiveRoutes.some(route => path.includes(route));\n \n if (isSensitive) {\n // Strict no-cache for sensitive content (login, edit pages, user data)\n res.setHeader('Cache-Control', 'no-cache, no-store, must-revalidate, private');\n res.setHeader('Pragma', 'no-cache'); // HTTP 1.0 backward compatibility\n res.setHeader('Expires', '0'); // Proxies\n } \n // Static assets - allow aggressive caching\n else if (path.match(/\\.(css|js|jpg|jpeg|png|gif|ico|woff|woff2|ttf|svg)$/)) {\n // Cache static assets for 1 year with immutable flag\n res.setHeader('Cache-Control', 'public, max-age=31536000, immutable');\n }\n // Public pages - moderate caching\n else if (['/about', '/services', '/blog', '/contact'].some(route => path === route)) {\n // Cache public pages for 1 hour but allow revalidation\n res.setHeader('Cache-Control', 'public, max-age=3600, must-revalidate');\n }\n // Dynamic content (posts) - short-term caching\n else if (path.match(/^\\/post\\/\\d+$/)) {\n // Cache blog posts for 10 minutes\n res.setHeader('Cache-Control', 'public, max-age=600, must-revalidate');\n }\n // Default for homepage and other routes - no cache\n else {\n res.setHeader('Cache-Control', 'no-cache, no-store, must-revalidate');\n res.setHeader('Pragma', 'no-cache');\n res.setHeader('Expires', '0');\n }\n \n next();\n};\n\n// Apply cache middleware globally to all routes\napp.use(secureCache);\n\n// Example route definitions\napp.get('/', (req, res) => {\n res.send('Homepage');\n});\n\napp.get('/login', (req, res) => {\n // This route automatically gets no-cache headers\n res.send('Login page');\n});\n\napp.get('/about', (req, res) => {\n // This route gets moderate caching headers\n res.send('About page');\n});\n\napp.get('/post/:id', (req, res) => {\n // Blog posts get short-term caching\n res.send(`Blog post ${req.params.id}`);\n});\n\napp.get('/:id/edit', (req, res) => {\n // Edit pages get strict no-cache headers\n res.send('Edit page');\n});\n\n// For API responses containing sensitive data\napp.get('/api/user/profile', (req, res) => {\n res.setHeader('Cache-Control', 'no-store, private');\n res.json({ username: 'user', email: 'user@example.com' });\n});\n\nmodule.exports = app;"

}

}

{

"title": "Content Security Policy (CSP) Header Not Set",

"description": "Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.",

"simplified\_description": "Imagine your website is a building without a security policy about who can enter and what they can bring inside. Without Content Security Policy, attackers can inject malicious code (like hidden scripts) into your pages, which can steal user data or deface your site. CSP acts like a strict guest list and security checkpoint, telling browsers exactly which sources of content (scripts, images, styles) are trusted and allowed to run on your website.",

"solution": "Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.",

"simplified\_solution": "Configure your server to send a Content-Security-Policy header with every response, which tells browsers exactly which sources of content are allowed to load and execute on your website pages.",

"code\_solution\_sample": {

"solution\_description": "This code implements a middleware that sets a comprehensive Content-Security-Policy header for all HTTP responses. The CSP policy defines trusted sources for scripts, styles, images, fonts, and other content types. It blocks inline scripts and styles by default (preventing XSS attacks), only allows content from the same origin and specified trusted domains, and includes additional security directives like upgrade-insecure-requests and frame-ancestors to prevent clickjacking. The policy can be adjusted based on the application's specific needs while maintaining security best practices.",

"affected\_files": "Application middleware configuration files (e.g., app.js, server.js, index.js for Node.js), web server configuration files (Apache httpd.conf/.htaccess, Nginx nginx.conf), framework-specific security configuration files (Spring Security config, Django settings.py), or load balancer configuration where HTTP headers are injected",

"code": "// Node.js/Express - Content Security Policy Implementation\nconst express = require('express');\nconst app = express();\n\n// CSP Middleware - Sets Content-Security-Policy header on all responses\nconst setCSPHeader = (req, res, next) => {\n // Build a comprehensive CSP policy\n const cspDirectives = [\n // Default fallback - only allow resources from same origin\n \"default-src 'self'\",\n \n // Script sources - prevent inline scripts and eval() to block XSS\n // Only allow scripts from same origin\n \"script-src 'self'\",\n \n // Style sources - allow same origin styles\n // Add 'unsafe-inline' only if absolutely necessary (reduces security)\n \"style-src 'self'\",\n \n // Image sources - allow same origin and data URIs for inline images\n \"img-src 'self' data: https:\",\n \n // Font sources - allow same origin fonts\n \"font-src 'self'\",\n \n // Connection sources for AJAX, WebSocket, EventSource\n \"connect-src 'self'\",\n \n // Media sources (audio/video)\n \"media-src 'self'\",\n \n // Object sources (Flash, Java applets) - block all\n \"object-src 'none'\",\n \n // Frame sources - control where site can be embedded\n \"frame-src 'self'\",\n \n // Frame ancestors - prevent clickjacking by controlling who can embed this site\n \"frame-ancestors 'self'\",\n \n // Base URI - restrict base tag URLs\n \"base-uri 'self'\",\n \n // Form action - restrict form submission targets\n \"form-action 'self'\",\n \n // Upgrade insecure requests - automatically upgrade HTTP to HTTPS\n \"upgrade-insecure-requests\",\n \n // Block mixed content\n \"block-all-mixed-content\"\n ];\n \n // Join all directives with semicolons\n const cspHeader = cspDirectives.join('; ');\n \n // Set the Content-Security-Policy header\n res.setHeader('Content-Security-Policy', cspHeader);\n \n // Optional: Set report-only mode for testing (doesn't block, only reports violations)\n // Uncomment below for testing phase, then switch to enforcement mode\n // res.setHeader('Content-Security-Policy-Report-Only', cspHeader);\n \n next();\n};\n\n// Apply CSP middleware to all routes\napp.use(setCSPHeader);\n\n// Example: More permissive CSP for specific routes if needed\napp.get('/interactive-page', (req, res, next) => {\n // Override CSP for pages that need external resources\n const permissiveCSP = [\n \"default-src 'self'\",\n \"script-src 'self' https://cdnjs.cloudflare.com\",\n \"style-src 'self' 'unsafe-inline' https://fonts.googleapis.com\",\n \"font-src 'self' https://fonts.gstatic.com\",\n \"img-src 'self' data: https:\",\n \"connect-src 'self' https://api.example.com\",\n \"frame-ancestors 'none'\"\n ].join('; ');\n \n res.setHeader('Content-Security-Policy', permissiveCSP);\n next();\n});\n\n// Alternative: Using helmet middleware (recommended for production)\n// const helmet = require('helmet');\n// app.use(helmet.contentSecurityPolicy({\n// directives: {\n// defaultSrc: [\"'self'\"],\n// scriptSrc: [\"'self'\"],\n// styleSrc: [\"'self'\"],\n// imgSrc: [\"'self'\", \"data:\", \"https:\"],\n// connectSrc: [\"'self'\"],\n// fontSrc: [\"'self'\"],\n// objectSrc: [\"'none'\"],\n// mediaSrc: [\"'self'\"],\n// frameSrc: [\"'self'\"],\n// frameAncestors: [\"'self'\"],\n// formAction: [\"'self'\"],\n// baseUri: [\"'self'\"],\n// upgradeInsecureRequests: []\n// }\n// }));\n\n// Route handlers\napp.get('/', (req, res) => {\n res.send('<h1>Homepage with CSP protection</h1>');\n});\n\napp.get('/login', (req, res) => {\n res.send('<h1>Login page with CSP protection</h1>');\n});\n\napp.get('/about', (req, res) => {\n res.send('<h1>About page with CSP protection</h1>');\n});\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on port ${PORT} with CSP enabled`);\n});\n\nmodule.exports = app;"

}

}

{

"title": "Cookie No HttpOnly Flag",

"description": "A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.",

"simplified\_description": "Imagine your session cookie is like a house key. Without the HttpOnly flag, it's like leaving that key on a table where anyone (including malicious scripts) can grab it. If an attacker injects malicious JavaScript into your website, they can steal this cookie and impersonate users, potentially taking over their accounts. The HttpOnly flag is like putting the key in a locked box that only the server can access.",

"solution": "Ensure that the HttpOnly flag is set for all cookies.",

"simplified\_solution": "Add the HttpOnly flag to all cookies, especially session cookies, so that client-side JavaScript cannot access them and steal sensitive session information.",

"code\_solution\_sample": {

"solution\_description": "This code demonstrates how to properly set the HttpOnly flag on cookies, specifically the SessionID cookie identified in the vulnerability. The implementation sets HttpOnly along with other critical security flags including Secure (HTTPS-only), SameSite (CSRF protection), and appropriate expiration times. This prevents JavaScript from accessing the cookie through document.cookie, effectively blocking XSS-based session hijacking attacks.",

"affected\_files": "Session management configuration files, application middleware setup files (e.g., app.js, server.js for Node.js), session configuration modules, authentication controllers, or web server configuration files where cookies are set (Apache httpd.conf/.htaccess, Nginx nginx.conf)",

"code": "// Node.js/Express - Secure Cookie Configuration with HttpOnly Flag\nconst express = require('express');\nconst session = require('express-session');\nconst cookieParser = require('cookie-parser');\nconst app = express();\n\n// Cookie parser middleware\napp.use(cookieParser());\n\n// Session configuration with HttpOnly and other security flags\napp.use(session({\n name: 'SessionID', // Match the cookie name from vulnerability report\n secret: process.env.SESSION\_SECRET || 'your-strong-secret-key-here',\n resave: false,\n saveUninitialized: false,\n cookie: {\n httpOnly: true, // PRIMARY FIX: Prevents JavaScript access to cookie\n secure: true, // Ensures cookie is only sent over HTTPS\n sameSite: 'strict', // Prevents CSRF attacks\n maxAge: 1800000, // Cookie expires after 30 minutes (in milliseconds)\n domain: 'pentest-ground.com', // Restrict cookie to your domain\n path: '/' // Cookie available across entire site\n }\n}));\n\n// Alternative: Manually setting cookies with HttpOnly flag\napp.get('/set-secure-cookie', (req, res) => {\n // Example of setting a custom cookie with all security flags\n res.cookie('SessionID', 'secure-session-value-here', {\n httpOnly: true, // Prevents XSS attacks by blocking JavaScript access\n secure: true, // Only transmitted over HTTPS\n sameSite: 'strict', // Prevents CSRF attacks\n maxAge: 1800000, // 30 minutes expiration\n signed: true // Cookie signing for integrity verification\n });\n \n res.send('Secure cookie set with HttpOnly flag');\n});\n\n// Login route example with secure session cookie\napp.post('/login', (req, res) => {\n // Authentication logic here (validate credentials)\n const userId = authenticateUser(req.body.username, req.body.password);\n \n if (userId) {\n // Regenerate session ID to prevent session fixation\n req.session.regenerate((err) => {\n if (err) {\n return res.status(500).json({ error: 'Session error' });\n }\n \n // Store user data in session\n req.session.userId = userId;\n req.session.username = req.body.username;\n \n // Session cookie automatically includes HttpOnly flag from config\n res.json({ \n success: true, \n message: 'Login successful with secure session' \n });\n });\n } else {\n res.status(401).json({ error: 'Invalid credentials' });\n }\n});\n\n// Logout route - properly destroy session\napp.post('/logout', (req, res) => {\n req.session.destroy((err) => {\n if (err) {\n return res.status(500).json({ error: 'Logout failed' });\n }\n \n // Clear the cookie from client\n res.clearCookie('SessionID', {\n httpOnly: true,\n secure: true,\n sameSite: 'strict'\n });\n \n res.json({ message: 'Logged out successfully' });\n });\n});\n\n// Middleware to check if session is valid and not expired\nconst requireAuth = (req, res, next) => {\n if (!req.session || !req.session.userId) {\n return res.status(401).json({ error: 'Unauthorized - Please login' });\n }\n next();\n};\n\n// Protected route example\napp.get('/dashboard', requireAuth, (req, res) => {\n res.json({ \n message: 'Protected dashboard',\n user: req.session.username \n });\n});\n\n// Helper function placeholder\nfunction authenticateUser(username, password) {\n // Implement your authentication logic here\n // Return userId if valid, null if invalid\n return username === 'testuser' ? 12345 : null;\n}\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on https://localhost:${PORT}`);\n console.log('All cookies configured with HttpOnly flag for XSS protection');\n});\n\nmodule.exports = app;"

}

}

{

"title": "Cookie without SameSite Attribute",

"description": "A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.",

"simplified\_description": "Think of the SameSite attribute like a security guard that checks whether requests are coming from your own website or from external sites. Without this guard, a malicious website can trick a user's browser into sending your session cookie along with a harmful request, allowing attackers to perform actions on behalf of the user without their knowledge (like changing passwords or making purchases). This is called Cross-Site Request Forgery (CSRF).",

"solution": "Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.",

"simplified\_solution": "Add the SameSite attribute to all cookies, setting it to 'strict' for maximum security or 'lax' for a balance between security and functionality, preventing browsers from sending cookies with cross-site requests.",

"code\_solution\_sample": {

"solution\_description": "This code configures the SessionID cookie with the SameSite attribute set to 'strict', which prevents the browser from sending the cookie with any cross-site requests. The implementation also includes other essential security attributes: HttpOnly (prevents JavaScript access), Secure (requires HTTPS), and appropriate expiration settings. Using 'strict' provides the strongest protection against CSRF attacks, while 'lax' can be used if the application needs to support top-level navigation from external sites while still maintaining CSRF protection for POST requests.",

"affected\_files": "Session management configuration files, application middleware setup (e.g., app.js, server.js for Node.js), session configuration modules, cookie-setting controllers, authentication handlers, or web server configuration files (Apache httpd.conf/.htaccess, Nginx nginx.conf) where session cookies are configured",

"code": "// Node.js/Express - Secure Cookie Configuration with SameSite Attribute\nconst express = require('express');\nconst session = require('express-session');\nconst cookieParser = require('cookie-parser');\nconst app = express();\n\n// Apply cookie parser middleware\napp.use(cookieParser());\n\n// Configure session middleware with SameSite and comprehensive security attributes\napp.use(session({\n name: 'SessionID', // Match the cookie name from the vulnerability report\n secret: process.env.SESSION\_SECRET || 'replace-with-strong-secret-key',\n resave: false,\n saveUninitialized: false,\n cookie: {\n httpOnly: true, // Prevents JavaScript access (XSS protection)\n secure: true, // Requires HTTPS transmission\n sameSite: 'strict', // PRIMARY FIX: Prevents cross-site cookie transmission\n // Options: 'strict' (most secure), 'lax' (balanced), 'none' (requires secure: true)\n maxAge: 1800000, // Cookie expires after 30 minutes (milliseconds)\n domain: 'pentest-ground.com', // Restrict to specific domain\n path: '/' // Cookie available across entire site\n },\n rolling: true // Reset expiration on each request (sliding session)\n}));\n\n// Alternative approach: Manually setting cookies with SameSite attribute\napp.get('/set-secure-cookie', (req, res) => {\n // Set cookie with all security attributes including SameSite\n res.cookie('SessionID', 'generated-session-token-value', {\n httpOnly: true, // XSS protection\n secure: true, // HTTPS only\n sameSite: 'strict', // CSRF protection - strict mode\n maxAge: 1800000, // 30 minutes\n signed: true // Cookie signature for integrity\n });\n \n res.send('Secure cookie with SameSite attribute set');\n});\n\n// Example: Using 'lax' mode for cookies that need to work with external navigation\napp.post('/login', (req, res) => {\n // Authenticate user (validation logic here)\n const authenticated = authenticateUser(req.body.username, req.body.password);\n \n if (authenticated) {\n // Regenerate session to prevent session fixation\n req.session.regenerate((err) => {\n if (err) {\n return res.status(500).json({ error: 'Session creation failed' });\n }\n \n // Store user information in session\n req.session.userId = authenticated.userId;\n req.session.username = authenticated.username;\n \n // For scenarios requiring 'lax' mode (e.g., supporting external links)\n // Uncomment and modify session config to use sameSite: 'lax'\n // This allows cookies on top-level GET requests from external sites\n // but still blocks them on POST/PUT/DELETE and embedded requests\n \n res.json({ \n success: true, \n message: 'Login successful with CSRF-protected session' \n });\n });\n } else {\n res.status(401).json({ error: 'Invalid credentials' });\n }\n});\n\n// Logout handler - clear session and cookie\napp.post('/logout', (req, res) => {\n req.session.destroy((err) => {\n if (err) {\n return res.status(500).json({ error: 'Logout failed' });\n }\n \n // Clear cookie with matching attributes\n res.clearCookie('SessionID', {\n httpOnly: true,\n secure: true,\n sameSite: 'strict',\n path: '/'\n });\n \n res.json({ message: 'Logged out successfully' });\n });\n});\n\n// Protected route requiring valid session\napp.get('/dashboard', (req, res) => {\n if (!req.session || !req.session.userId) {\n return res.status(401).json({ error: 'Unauthorized' });\n }\n \n res.json({ \n message: 'Protected content',\n user: req.session.username,\n securityNote: 'Session protected with SameSite=strict against CSRF' \n });\n});\n\n// Authentication helper function (placeholder)\nfunction authenticateUser(username, password) {\n // Implement your authentication logic here\n // Return user object if valid, null/false if invalid\n if (username === 'testuser' && password === 'correct-password') {\n return { userId: 12345, username: 'testuser' };\n }\n return null;\n}\n\n// Configuration note for different SameSite modes:\n// - 'strict': Most secure. Cookie not sent on any cross-site request.\n// Use for sensitive operations (banking, admin panels).\n// - 'lax': Balanced security. Cookie sent on top-level GET navigation from external sites,\n// but not on POST/embedded requests. Good default for most applications.\n// - 'none': No CSRF protection. Requires secure: true. Only use if absolutely necessary\n// for cross-site functionality (e.g., embedded widgets).\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on https://localhost:${PORT}`);\n console.log('SessionID cookie configured with SameSite=strict for CSRF protection');\n});\n\nmodule.exports = app;"

}

}  
{

"title": "Cookie Without Secure Flag",

"description": "A cookie has been set without the secure flag, which means that the cookie can be accessed via unencrypted connections.",

"simplified\_description": "Imagine sending a secret password through regular mail instead of a secure courier. Without the Secure flag, your session cookie (which acts like a key to your account) can be sent over unencrypted HTTP connections. This means attackers on the same network (like public Wi-Fi) could intercept and steal this cookie, then use it to impersonate the user and hijack their session.",

"solution": "Whenever a cookie contains sensitive information or is a session token, then it should always be passed using an encrypted channel. Ensure that the secure flag is set for cookies containing such sensitive information.",

"simplified\_solution": "Set the Secure flag on all session cookies to ensure they are only transmitted over encrypted HTTPS connections, preventing attackers from intercepting them over insecure networks.",

"code\_solution\_sample": {

"solution\_description": "This code configures the SessionID cookie with the Secure flag enabled, ensuring the cookie is only transmitted over HTTPS connections. The implementation includes comprehensive security settings: Secure flag prevents transmission over HTTP, HttpOnly prevents JavaScript access (XSS protection), SameSite prevents CSRF attacks, and appropriate expiration settings. This configuration protects the session token from man-in-the-middle attacks and network sniffing on insecure connections.",

"affected\_files": "Session management configuration files, application middleware setup files (e.g., app.js, server.js, index.js for Node.js), session initialization modules, authentication controllers, web server configuration files (Apache httpd.conf/.htaccess, Nginx nginx.conf), or framework-specific configuration files where cookies are set (Django settings.py, Spring Security configuration, PHP session configuration)",

"code": "// Node.js/Express - Secure Cookie Configuration with Secure Flag\nconst express = require('express');\nconst session = require('express-session');\nconst cookieParser = require('cookie-parser');\nconst app = express();\n\n// Middleware to enforce HTTPS (redirect HTTP to HTTPS)\napp.use((req, res, next) => {\n // Check if request is over HTTP (not HTTPS)\n if (req.header('x-forwarded-proto') !== 'https' && process.env.NODE\_ENV === 'production') {\n // Redirect to HTTPS\n return res.redirect(`https://${req.header('host')}${req.url}`);\n }\n next();\n});\n\n// Apply cookie parser middleware\napp.use(cookieParser());\n\n// Configure session with Secure flag and comprehensive security settings\napp.use(session({\n name: 'SessionID', // Match the cookie name from vulnerability report\n secret: process.env.SESSION\_SECRET || 'replace-with-cryptographically-strong-secret',\n resave: false,\n saveUninitialized: false,\n cookie: {\n secure: true, // PRIMARY FIX: Cookie only sent over HTTPS\n httpOnly: true, // Prevents JavaScript access (XSS protection)\n sameSite: 'strict', // Prevents CSRF attacks\n maxAge: 1800000, // Session expires after 30 minutes (milliseconds)\n domain: 'pentest-ground.com', // Restrict to specific domain\n path: '/' // Cookie available across entire site\n },\n rolling: true // Reset expiration on each request\n}));\n\n// Alternative: Manually setting cookies with Secure flag\napp.get('/set-secure-cookie', (req, res) => {\n // Example of manually setting a secure cookie\n res.cookie('SessionID', 'secure-session-token-value', {\n secure: true, // Only transmitted over HTTPS\n httpOnly: true, // No JavaScript access\n sameSite: 'strict', // CSRF protection\n maxAge: 1800000, // 30 minutes expiration\n signed: true // Cookie signing for integrity\n });\n \n res.send('Secure cookie has been set with Secure flag');\n});\n\n// Login route with secure session management\napp.post('/login', (req, res) => {\n // Authentication logic (validate credentials)\n const user = authenticateUser(req.body.username, req.body.password);\n \n if (user) {\n // Regenerate session ID to prevent session fixation attacks\n req.session.regenerate((err) => {\n if (err) {\n return res.status(500).json({ error: 'Session creation failed' });\n }\n \n // Store user data in secure session\n req.session.userId = user.userId;\n req.session.username = user.username;\n req.session.loginTime = Date.now();\n \n // SessionID cookie automatically includes Secure flag from config\n res.json({ \n success: true, \n message: 'Login successful with secure session cookie' \n });\n });\n } else {\n res.status(401).json({ error: 'Invalid credentials' });\n }\n});\n\n// Logout route - properly destroy session\napp.post('/logout', (req, res) => {\n req.session.destroy((err) => {\n if (err) {\n return res.status(500).json({ error: 'Logout failed' });\n }\n \n // Clear the cookie with matching security attributes\n res.clearCookie('SessionID', {\n secure: true,\n httpOnly: true,\n sameSite: 'strict',\n path: '/'\n });\n \n res.json({ message: 'Logged out successfully' });\n });\n});\n\n// Protected route example\napp.get('/dashboard', (req, res) => {\n // Verify session exists and is valid\n if (!req.session || !req.session.userId) {\n return res.status(401).json({ error: 'Unauthorized - Please login' });\n }\n \n res.json({ \n message: 'Protected dashboard content',\n user: req.session.username,\n sessionSecure: true\n });\n});\n\n// Helper function for authentication (placeholder)\nfunction authenticateUser(username, password) {\n // Implement your authentication logic here\n // Return user object if valid, null if invalid\n if (username === 'validuser' && password === 'correct-password') {\n return { userId: 12345, username: 'validuser' };\n }\n return null;\n}\n\n// IMPORTANT: For production deployment\n// 1. Ensure your application is served over HTTPS only\n// 2. Configure SSL/TLS certificates properly\n// 3. Use environment variables for sensitive configuration\n// 4. Set NODE\_ENV=production in production environments\n\nconst PORT = process.env.PORT || 443; // Use port 443 for HTTPS\nconst HTTPS\_ENABLED = process.env.HTTPS\_ENABLED === 'true';\n\nif (HTTPS\_ENABLED) {\n const https = require('https');\n const fs = require('fs');\n \n // HTTPS server configuration\n const httpsOptions = {\n key: fs.readFileSync(process.env.SSL\_KEY\_PATH || './ssl/private.key'),\n cert: fs.readFileSync(process.env.SSL\_CERT\_PATH || './ssl/certificate.crt')\n };\n \n https.createServer(httpsOptions, app).listen(PORT, () => {\n console.log(`Secure HTTPS server running on port ${PORT}`);\n console.log('SessionID cookie configured with Secure flag enabled');\n });\n} else {\n // Development mode warning\n console.warn('WARNING: Running without HTTPS. Secure flag requires HTTPS in production.');\n app.listen(PORT, () => {\n console.log(`Development server running on port ${PORT}`);\n });\n}\n\nmodule.exports = app;"

}

}  
  
{

"title": "Cross-Domain Misconfiguration",

"description": "Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server.",

"simplified\_description": "Think of CORS like security guards at a building entrance. Your website currently has a policy that says 'allow anyone from anywhere to access our data' (Access-Control-Allow-Origin: \*). This is like telling security guards to let everyone in without checking credentials. Malicious websites can now request and receive your sensitive data, potentially stealing user information or performing unauthorized actions on behalf of your users.",

"solution": "Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the \"Access-Control-Allow-Origin\" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.",

"simplified\_solution": "Replace the wildcard '\*' in the Access-Control-Allow-Origin header with specific trusted domain names, or remove CORS headers entirely to prevent unauthorized websites from accessing your application's data.",

"code\_solution\_sample": {

"solution\_description": "This code implements a secure CORS configuration middleware that replaces the dangerous wildcard ('\*') configuration with a whitelist of trusted domains. It validates incoming requests against an approved list of origins and only sets the Access-Control-Allow-Origin header for authorized domains. The solution includes additional security measures such as credentials handling, appropriate HTTP methods, and secure headers. For public CDN resources (like CSS/JS libraries), the code demonstrates when wildcard CORS might be acceptable, but for application endpoints containing sensitive data, it enforces strict origin validation.",

"affected\_files": "Application middleware configuration files (e.g., app.js, server.js for Node.js), web server configuration files (Apache httpd.conf/.htaccess, Nginx nginx.conf), API gateway configuration, reverse proxy settings, or framework-specific CORS configuration files (Express CORS middleware, Spring Security configuration, Django CORS settings)",

"code": "// Node.js/Express - Secure CORS Configuration\nconst express = require('express');\nconst app = express();\n\n// Define whitelist of trusted domains that are allowed to access your API\nconst ALLOWED\_ORIGINS = [\n 'https://pentest-ground.com',\n 'https://www.pentest-ground.com',\n 'https://app.pentest-ground.com'\n // Add other trusted domains as needed\n // NEVER use 'http://' origins in production (use HTTPS only)\n // NEVER include '\*' in this list\n];\n\n// Secure CORS Middleware\nconst secureCorsMiddleware = (req, res, next) => {\n const origin = req.headers.origin;\n \n // Check if the requesting origin is in the whitelist\n if (origin && ALLOWED\_ORIGINS.includes(origin)) {\n // Set CORS headers only for whitelisted origins\n res.setHeader('Access-Control-Allow-Origin', origin);\n \n // Allow credentials (cookies, authorization headers) only for trusted origins\n res.setHeader('Access-Control-Allow-Credentials', 'true');\n \n // Specify allowed HTTP methods\n res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');\n \n // Specify allowed headers\n res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization, X-Requested-With');\n \n // Cache preflight requests for 1 hour (3600 seconds)\n res.setHeader('Access-Control-Max-Age', '3600');\n } else {\n // For requests from non-whitelisted origins, do NOT set CORS headers\n // This allows the browser's Same-Origin Policy to block the request\n // Log unauthorized access attempts for security monitoring\n if (origin) {\n console.warn(`CORS request blocked from unauthorized origin: ${origin}`);\n }\n }\n \n // Handle preflight OPTIONS requests\n if (req.method === 'OPTIONS') {\n // If origin is whitelisted, respond with 204 No Content\n if (origin && ALLOWED\_ORIGINS.includes(origin)) {\n return res.status(204).end();\n } else {\n // Block preflight from unauthorized origins\n return res.status(403).json({ error: 'CORS request not allowed' });\n }\n }\n \n next();\n};\n\n// Apply CORS middleware globally to all routes\napp.use(secureCorsMiddleware);\n\n// Alternative: Route-specific CORS configuration\n// For public APIs that don't contain sensitive data\napp.get('/api/public/data', (req, res) => {\n // Public endpoints might allow broader access if truly public\n // However, still avoid wildcard '\*' when possible\n res.json({ message: 'Public data accessible with restricted CORS' });\n});\n\n// For sensitive endpoints, ensure CORS is strictly enforced\napp.get('/api/user/profile', (req, res) => {\n // Verify authentication\n if (!req.headers.authorization) {\n return res.status(401).json({ error: 'Unauthorized' });\n }\n \n // CORS middleware already applied - only whitelisted origins can access\n res.json({ \n userId: 12345, \n username: 'user',\n email: 'user@example.com' \n });\n});\n\n// Alternative using cors npm package (recommended for production)\n/\*\nconst cors = require('cors');\n\nconst corsOptions = {\n origin: function (origin, callback) {\n // Allow requests with no origin (like mobile apps, Postman, curl)\n if (!origin) return callback(null, true);\n \n if (ALLOWED\_ORIGINS.includes(origin)) {\n callback(null, true);\n } else {\n console.warn(`CORS blocked: ${origin}`);\n callback(new Error('Not allowed by CORS policy'));\n }\n },\n credentials: true,\n methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],\n allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With'],\n maxAge: 3600\n};\n\napp.use(cors(corsOptions));\n\*/\n\n// For completely removing CORS (strictest approach)\n// Simply don't set any Access-Control-\* headers\n// The browser will enforce Same-Origin Policy automatically\nconst noCorsMiddleware = (req, res, next) => {\n // Remove any CORS headers if they were set elsewhere\n res.removeHeader('Access-Control-Allow-Origin');\n res.removeHeader('Access-Control-Allow-Credentials');\n res.removeHeader('Access-Control-Allow-Methods');\n res.removeHeader('Access-Control-Allow-Headers');\n \n // For OPTIONS requests, respond with 403 Forbidden\n if (req.method === 'OPTIONS') {\n return res.status(403).end();\n }\n \n next();\n};\n\n// Apply no-CORS middleware if your application doesn't need cross-origin access\n// app.use(noCorsMiddleware);\n\n// Security best practices:\n// 1. NEVER use Access-Control-Allow-Origin: \* with credentials\n// 2. Always use HTTPS origins in production (never http://)\n// 3. Regularly audit and minimize the whitelist\n// 4. Implement authentication for sensitive endpoints\n// 5. Use Content Security Policy (CSP) headers as additional defense\n// 6. Monitor and log blocked CORS requests for security analysis\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on port ${PORT}`);\n console.log('Secure CORS configuration active - wildcard origin blocked');\n console.log('Allowed origins:', ALLOWED\_ORIGINS.join(', '));\n});\n\nmodule.exports = app;"

}

}  
{

"title": "Cross-Domain JavaScript Source File Inclusion",

"description": "The page includes one or more script files from a third-party domain.",

"simplified\_description": "Your website is loading JavaScript code from an external CDN (cdnjs.cloudflare.com). While CDNs are convenient, this is like letting someone else write part of your security system's code. If that external source is compromised or serves malicious code, attackers could steal user data, hijack sessions, or deface your site. You're trusting a third party with the security of your entire application.",

"solution": "Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.",

"simplified\_solution": "Either host JavaScript libraries on your own servers or use Subresource Integrity (SRI) hashes to verify that external scripts haven't been tampered with, and implement a strict Content Security Policy to control which domains can serve scripts.",

"code\_solution\_sample": {

"solution\_description": "This solution implements multiple layers of defense: (1) Subresource Integrity (SRI) hashes that verify the exact content of external scripts, preventing execution if they've been modified, (2) A Content Security Policy (CSP) that whitelists only trusted script sources, and (3) Guidance for self-hosting libraries as the most secure option. The SRI hash acts like a digital fingerprint - the browser will only execute the script if it matches exactly what you expect. The CSP prevents any unauthorized scripts from running, even if injected by attackers.",

"affected\_files": "HTML template files, view templates (e.g., EJS, Handlebars, Pug, JSP, Thymeleaf), base layout files, header/footer partials, React/Vue component files that reference external scripts, and server configuration files for setting CSP headers (app.js, server.js, middleware configuration, Apache .htaccess, Nginx configuration)",

"code": "<!-- HTML - Secure External Script Loading with SRI and CSP -->\n\n<!-- SOLUTION 1: Add Subresource Integrity (SRI) hash to external scripts -->\n<!-- SRI ensures the script hasn't been tampered with -->\n<script \n type=\"text/javascript\" \n src=\"https://cdnjs.cloudflare.com/ajax/libs/OwlCarousel2/2.2.1/owl.carousel.min.js\"\n integrity=\"sha384-6+gF8R0FVN/Tq3Sf2xLKnlGPrEe3+2Lw6rRe7QIUE9TfYM/OwLgqEaEMQxX0q3zX\"\n crossorigin=\"anonymous\">\n</script>\n\n<!-- To generate SRI hash, use: https://www.srihash.org/ -->\n<!-- Or use command line: curl -s URL | openssl dgst -sha384 -binary | openssl base64 -A -->\n\n<!-- SOLUTION 2 (RECOMMENDED): Self-host the library -->\n<!-- Download owl.carousel.min.js and host it on your own domain -->\n<script type=\"text/javascript\" src=\"/js/vendor/owl.carousel.min.js\"></script>\n\n<!-- SOLUTION 3: Implement Content Security Policy (CSP) -->\n<!-- Server-side implementation (Node.js/Express) -->\n\n// Node.js/Express - CSP Middleware with Whitelisted Script Sources\nconst express = require('express');\nconst crypto = require('crypto');\nconst app = express();\n\n// Middleware to set Content Security Policy\nconst setSecureCSP = (req, res, next) => {\n // Define allowed script sources\n const scriptSources = [\n \"'self'\", // Allow scripts from same origin\n \"https://cdnjs.cloudflare.com\" // Whitelist only trusted CDN\n // Add other trusted domains as needed\n // NEVER use 'unsafe-inline' or 'unsafe-eval' in production\n ];\n \n // Build CSP header\n const cspHeader = [\n `default-src 'self'`,\n `script-src ${scriptSources.join(' ')}`,\n `style-src 'self' 'unsafe-inline' https://cdnjs.cloudflare.com`,\n `img-src 'self' data: https:`,\n `font-src 'self' https://cdnjs.cloudflare.com`,\n `connect-src 'self'`,\n `frame-ancestors 'self'`,\n `base-uri 'self'`,\n `form-action 'self'`,\n `object-src 'none'`\n ].join('; ');\n \n res.setHeader('Content-Security-Policy', cspHeader);\n next();\n};\n\n// Apply CSP middleware globally\napp.use(setSecureCSP);\n\n// Template rendering with SRI hash generation\napp.get('/', (req, res) => {\n // If using template engine (EJS, Pug, etc.)\n res.render('index', {\n // Pass SRI hash to template\n owlCarouselSRI: 'sha384-6+gF8R0FVN/Tq3Sf2xLKnlGPrEe3+2Lw6rRe7QIUE9TfYM/OwLgqEaEMQxX0q3zX'\n });\n});\n\n// Helper function to verify external script integrity\nconst verifyScriptIntegrity = async (url, expectedHash) => {\n const https = require('https');\n \n return new Promise((resolve, reject) => {\n https.get(url, (response) => {\n let data = '';\n response.on('data', chunk => data += chunk);\n response.on('end', () => {\n // Calculate SHA-384 hash of the script content\n const hash = crypto.createHash('sha384').update(data).digest('base64');\n const integrity = `sha384-${hash}`;\n \n if (integrity === expectedHash) {\n resolve(true);\n } else {\n console.error(`Script integrity check failed for ${url}`);\n console.error(`Expected: ${expectedHash}`);\n console.error(`Received: ${integrity}`);\n resolve(false);\n }\n });\n }).on('error', reject);\n });\n};\n\n// BEST PRACTICE: Script self-hosting configuration\nconst path = require('path');\n\n// Serve self-hosted vendor libraries from a dedicated directory\napp.use('/js/vendor', express.static(path.join(\_\_dirname, 'public/js/vendor'), {\n // Enable caching for better performance\n maxAge: '1y',\n // Prevent directory listing\n index: false,\n // Set secure headers\n setHeaders: (res, filePath) => {\n if (filePath.endsWith('.js')) {\n res.setHeader('Content-Type', 'application/javascript; charset=utf-8');\n res.setHeader('X-Content-Type-Options', 'nosniff');\n }\n }\n}));\n\n// Template helper for generating script tags with SRI\nconst generateSecureScriptTag = (src, integrity, crossorigin = 'anonymous') => {\n if (src.startsWith('http://') || src.startsWith('https://')) {\n // External script - require SRI\n if (!integrity) {\n throw new Error('External scripts must include SRI integrity hash');\n }\n return `<script src=\"${src}\" integrity=\"${integrity}\" crossorigin=\"${crossorigin}\"></script>`;\n } else {\n // Internal script - no SRI needed\n return `<script src=\"${src}\"></script>`;\n }\n};\n\n// Security monitoring: Log external script loading\napp.use((req, res, next) => {\n // Monitor CSP violations\n if (req.path === '/csp-report') {\n console.warn('CSP Violation:', req.body);\n return res.status(204).end();\n }\n next();\n});\n\n// Add CSP report-uri for monitoring violations\nconst cspWithReporting = (req, res, next) => {\n const csp = res.getHeader('Content-Security-Policy');\n res.setHeader('Content-Security-Policy', `${csp}; report-uri /csp-report`);\n next();\n};\n\n// DEPLOYMENT CHECKLIST:\n// 1. Generate SRI hashes for all external scripts using https://www.srihash.org/\n// 2. Add integrity and crossorigin attributes to all external script tags\n// 3. Implement CSP header with whitelisted script sources only\n// 4. Prefer self-hosting libraries when possible\n// 5. Regularly update library versions and regenerate SRI hashes\n// 6. Monitor CSP violation reports for security incidents\n// 7. Never allow user-controlled script sources\n// 8. Use 'nonce' or 'hash' for inline scripts instead of 'unsafe-inline'\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on port ${PORT}`);\n console.log('CSP and SRI protection enabled for external scripts');\n});\n\nmodule.exports = app;\n\n/\* \n \* ALTERNATIVE: Complete self-hosting approach (MOST SECURE)\n \* \n \* Steps:\n \* 1. Download owl.carousel.min.js from CDN\n \* 2. Place in /public/js/vendor/ directory\n \* 3. Update all HTML templates to reference local path:\n \* <script src=\"/js/vendor/owl.carousel.min.js\"></script>\n \* 4. Remove CDN from CSP whitelist\n \* 5. Implement versioning for cache busting:\n \* <script src=\"/js/vendor/owl.carousel.min.js?v=2.2.1\"></script>\n \*/"

}

}  
  
{

"title": "Modern Web Application",

"description": "The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.",

"simplified\_description": "This is not a vulnerability or security issue. The scanner has simply detected that your website uses modern web technologies (like dynamic JavaScript content loading). This is purely informational - it's like a mechanic noting that your car has a modern engine. The scanner is suggesting that if you want to test the application thoroughly, you should use specialized testing tools designed for modern web apps.",

"solution": "This is an informational alert and so no changes are required.",

"simplified\_solution": "No action is needed. This is just informational feedback indicating that your application uses modern web development patterns, which may require different testing approaches for security assessment.",

"code\_solution\_sample": {

"solution\_description": "Since this is an informational alert with no security risk, no code changes are required. However, the sample below demonstrates best practices for modern web applications including proper accessibility, SEO optimization, and security considerations that should be followed when building dynamic web applications. The code shows how to properly implement links with meaningful href attributes (avoiding empty href=''), use semantic HTML, implement proper ARIA labels for accessibility, and add security headers that are important for modern web applications.",

"affected\_files": "This informational alert does not indicate vulnerable files. However, for modern web application best practices, review: HTML template files, JavaScript files handling dynamic content, frontend framework components (React, Vue, Angular), routing configuration files, and server response header configuration files",

"code": "// Node.js/Express - Modern Web Application Best Practices\nconst express = require('express');\nconst path = require('path');\nconst helmet = require('helmet');\nconst app = express();\n\n// Security headers for modern web applications\napp.use(helmet({\n contentSecurityPolicy: {\n directives: {\n defaultSrc: [\"'self'\"],\n scriptSrc: [\"'self'\", \"'unsafe-inline'\"],\n styleSrc: [\"'self'\", \"'unsafe-inline'\"],\n imgSrc: [\"'self'\", \"data:\", \"https:\"],\n connectSrc: [\"'self'\"],\n fontSrc: [\"'self'\"],\n objectSrc: [\"'none'\"],\n mediaSrc: [\"'self'\"],\n frameSrc: [\"'none'\"]\n }\n },\n hsts: {\n maxAge: 31536000,\n includeSubDomains: true,\n preload: true\n }\n}));\n\n// Serve static files securely\napp.use('/static', express.static(path.join(\_\_dirname, 'public'), {\n maxAge: '1d',\n etag: true,\n lastModified: true\n}));\n\n// Modern web application route handler example\napp.get('/', (req, res) => {\n res.render('index');\n});\n\napp.get('/about', (req, res) => {\n res.render('about');\n});\n\napp.get('/services', (req, res) => {\n res.render('services');\n});\n\n// API endpoint for modern SPAs (Single Page Applications)\napp.get('/api/posts', (req, res) => {\n // Return JSON data for AJAX requests\n res.json({\n posts: [\n { id: 1, title: 'Post 1', content: 'Content 1' },\n { id: 2, title: 'Post 2', content: 'Content 2' }\n ]\n });\n});\n\n// Example: Proper link implementation (addressing empty href in evidence)\napp.get('/best-practices-example', (req, res) => {\n res.send(`\n <!DOCTYPE html>\n <html lang=\"en\">\n <head>\n <meta charset=\"UTF-8\">\n <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">\n <title>Modern Web App Best Practices</title>\n </head>\n <body>\n <!-- BAD PRACTICE (found in evidence): Empty href -->\n <!-- <a href=\"\">Read More</a> -->\n \n <!-- GOOD PRACTICE: Proper href with meaningful destination -->\n <a href=\"/blog/article-1\" aria-label=\"Read more about article 1\">\n <span>Read More</span>\n <img src=\"/static/images/arrow.png\" alt=\"Arrow icon\" class=\"ml-2\" />\n </a>\n \n <!-- For JavaScript-handled links, use button elements -->\n <button \n type=\"button\" \n onclick=\"handleReadMore()\" \n aria-label=\"Read more about this article\"\n class=\"link-button\">\n <span>Read More</span>\n <img src=\"/static/images/arrow.png\" alt=\"Arrow icon\" class=\"ml-2\" />\n </button>\n \n <!-- For anchor links within the same page -->\n <a href=\"#section-details\" aria-label=\"Jump to details section\">\n <span>Read More</span>\n </a>\n \n <!-- External links should indicate they open in new window -->\n <a \n href=\"https://example.com/article\" \n target=\"\_blank\" \n rel=\"noopener noreferrer\"\n aria-label=\"Read more on external site (opens in new window)\">\n <span>Read More</span>\n </a>\n \n <script>\n // Modern JavaScript for handling dynamic content\n function handleReadMore() {\n // Proper event handling for modern web apps\n fetch('/api/article-details')\n .then(response => response.json())\n .then(data => {\n // Update DOM with fetched content\n document.getElementById('content').innerHTML = data.content;\n })\n .catch(error => {\n console.error('Error loading content:', error);\n });\n }\n \n // Progressive enhancement example\n if ('serviceWorker' in navigator) {\n // Register service worker for offline functionality\n navigator.serviceWorker.register('/sw.js')\n .then(reg => console.log('Service Worker registered'))\n .catch(err => console.log('Service Worker registration failed'));\n }\n </script>\n </body>\n </html>\n `);\n});\n\n// Modern routing for Single Page Applications (SPA)\napp.get('\*', (req, res) => {\n // For React/Vue/Angular apps, serve index.html for all routes\n // Let client-side routing handle the navigation\n res.sendFile(path.join(\_\_dirname, 'public', 'index.html'));\n});\n\n// Best practices for modern web applications:\n// 1. Use meaningful href attributes (not empty strings)\n// 2. Implement proper ARIA labels for accessibility\n// 3. Use semantic HTML5 elements\n// 4. Implement CSP headers for XSS protection\n// 5. Enable HSTS for forcing HTTPS\n// 6. Use proper HTTP methods (GET for retrieval, POST for mutations)\n// 7. Implement proper error handling\n// 8. Use progressive enhancement principles\n// 9. Optimize for performance (lazy loading, code splitting)\n// 10. Follow WCAG accessibility guidelines\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Modern web application running on port ${PORT}`);\n console.log('Security headers and best practices enabled');\n});\n\nmodule.exports = app;\n\n/\* \n \* HTML Template Best Practices (for template engines like EJS, Pug, Handlebars)\n \* \n \* <!-- Instead of empty href -->\n \* <a href=\"\">\n \* \n \* <!-- Use proper destination -->\n \* <a href=\"/blog/<%= post.id %>\">\n \* \n \* <!-- Or use button for JavaScript interactions -->\n \* <button type=\"button\" onclick=\"loadMore()\">\n \* \n \* Accessibility considerations:\n \* - Always provide alt text for images\n \* - Use semantic HTML (nav, main, article, section)\n \* - Ensure sufficient color contrast\n \* - Make interactive elements keyboard accessible\n \* - Provide skip navigation links\n \* - Use ARIA labels when necessary\n \*/"

}

}  
  
{

"title": "Server Leaks Version Information via \"Server\" HTTP Response Header Field",

"description": "The web/application server is leaking version information via the \"Server\" HTTP response header. Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.",

"simplified\_description": "Your web server is publicly announcing its exact make and model (nginx/1.29.1) in every response, like putting a detailed label on your front door saying 'Protected by XYZ Security System v2.3'. Attackers can use this information to quickly look up known vulnerabilities for that specific version and target your server with exploits designed for nginx 1.29.1, making their job much easier.",

"solution": "Ensure that your web server, application server, load balancer, etc. is configured to suppress the \"Server\" header or provide generic details.",

"simplified\_solution": "Configure your Nginx server to either hide the version number completely or replace it with a generic server name, preventing attackers from knowing exactly which software version you're running.",

"code\_solution\_sample": {

"solution\_description": "This configuration removes or obscures the server version information from the HTTP response headers. The 'server\_tokens off' directive removes the version number from the Server header and error pages, while the 'more\_set\_headers' directive (using the headers-more-nginx-module) either completely removes the Server header or replaces it with a generic value. This prevents attackers from identifying the specific Nginx version and researching version-specific vulnerabilities.",

"affected\_files": "Nginx configuration files such as nginx.conf (main configuration file), site-specific configuration files in /etc/nginx/sites-available/ or /etc/nginx/conf.d/, server block configuration files, or included configuration files referenced within the main nginx.conf",

"code": "# Nginx Configuration - Suppress Server Version Information\n\n# SOLUTION 1: Hide version number (Basic approach)\n# This removes the version number but still shows 'nginx'\n# Add this to the http block in nginx.conf\n\nhttp {\n # Hide nginx version number from Server header and error pages\n server\_tokens off;\n \n # Additional security headers\n add\_header X-Frame-Options \"SAMEORIGIN\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n add\_header X-XSS-Protection \"1; mode=block\" always;\n add\_header Referrer-Policy \"strict-origin-when-cross-origin\" always;\n \n server {\n listen 81 ssl http2;\n server\_name pentest-ground.com;\n \n # SSL configuration\n ssl\_certificate /etc/nginx/ssl/certificate.crt;\n ssl\_certificate\_key /etc/nginx/ssl/private.key;\n ssl\_protocols TLSv1.2 TLSv1.3;\n ssl\_ciphers HIGH:!aNULL:!MD5;\n \n # Root and index configuration\n root /var/www/html;\n index index.html index.htm;\n \n # Static files location\n location /static/ {\n alias /var/www/html/static/;\n expires 30d;\n access\_log off;\n }\n \n # Application routes\n location / {\n try\_files $uri $uri/ =404;\n }\n }\n}\n\n# SOLUTION 2: Completely remove or replace Server header (Recommended)\n# Requires nginx-extras package or headers-more-nginx-module\n# Install with: apt-get install nginx-extras (Debian/Ubuntu)\n# Or compile nginx with: --add-module=/path/to/headers-more-nginx-module\n\nhttp {\n # Hide version number\n server\_tokens off;\n \n # Option A: Completely remove the Server header\n more\_clear\_headers 'Server';\n \n # Option B: Replace with generic value (choose one approach)\n # more\_set\_headers 'Server: WebServer';\n # more\_set\_headers 'Server: ';\n \n # Security headers\n more\_set\_headers 'X-Frame-Options: SAMEORIGIN';\n more\_set\_headers 'X-Content-Type-Options: nosniff';\n more\_set\_headers 'X-XSS-Protection: 1; mode=block';\n \n server {\n listen 81 ssl http2;\n server\_name pentest-ground.com;\n \n ssl\_certificate /etc/nginx/ssl/certificate.crt;\n ssl\_certificate\_key /etc/nginx/ssl/private.key;\n \n root /var/www/html;\n \n location / {\n try\_files $uri $uri/ =404;\n }\n }\n}\n\n# SOLUTION 3: Custom error pages without version info\n# Create custom error pages that don't leak server information\n\nhttp {\n server\_tokens off;\n \n server {\n listen 81 ssl http2;\n server\_name pentest-ground.com;\n \n # Custom error pages (prevents default nginx error pages with version)\n error\_page 404 /404.html;\n error\_page 500 502 503 504 /50x.html;\n \n location = /404.html {\n root /var/www/html/errors;\n internal;\n }\n \n location = /50x.html {\n root /var/www/html/errors;\n internal;\n }\n \n location / {\n root /var/www/html;\n try\_files $uri $uri/ =404;\n }\n }\n}\n\n# VERIFICATION STEPS:\n# After applying configuration changes:\n# 1. Test configuration: nginx -t\n# 2. Reload nginx: systemctl reload nginx (or: service nginx reload)\n# 3. Verify with curl: curl -I https://pentest-ground.com:81/\n# 4. Check that Server header shows either:\n# - 'Server: nginx' (without version) with server\_tokens off\n# - 'Server: WebServer' (custom value) with more\_set\_headers\n# - No Server header at all with more\_clear\_headers\n\n# INSTALLATION NOTES:\n# If headers-more-nginx-module is not available:\n# \n# For Debian/Ubuntu:\n# sudo apt-get install nginx-extras\n# \n# For CentOS/RHEL:\n# sudo yum install nginx-mod-http-headers-more\n# \n# Or compile from source:\n# wget https://github.com/openresty/headers-more-nginx-module/archive/v0.34.tar.gz\n# tar -xzvf v0.34.tar.gz\n# cd nginx-source-directory\n# ./configure --add-module=/path/to/headers-more-nginx-module-0.34\n# make\n# sudo make install\n\n# ADDITIONAL HARDENING:\n# Consider also implementing:\n# - Disable unnecessary modules\n# - Limit request methods\n# - Rate limiting\n# - Fail2ban integration\n# - Regular security updates\n\n# Example of additional security directives:\nhttp {\n server\_tokens off;\n more\_clear\_headers 'Server';\n \n # Limit request methods\n map $request\_method $bad\_method {\n default 1;\n GET 0;\n HEAD 0;\n POST 0;\n PUT 0;\n DELETE 0;\n }\n \n server {\n listen 81 ssl http2;\n server\_name pentest-ground.com;\n \n # Block requests with disallowed methods\n if ($bad\_method) {\n return 405;\n }\n \n # Rate limiting zone\n limit\_req\_zone $binary\_remote\_addr zone=general:10m rate=10r/s;\n limit\_req zone=general burst=20 nodelay;\n \n location / {\n root /var/www/html;\n try\_files $uri $uri/ =404;\n }\n }\n}"

}

}  
  
{

"title": "Strict-Transport-Security Header Not Set",

"description": "HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL). HSTS is an IETF standards track protocol and is specified in RFC 6797.",

"simplified\_description": "Without HSTS, your website is like a building that has a secure entrance but doesn't require people to use it. Users might accidentally visit the insecure HTTP version of your site, or attackers could intercept the first connection and downgrade it to HTTP before encryption kicks in. This 'SSL stripping' attack lets them steal passwords, session cookies, and other sensitive data. HSTS forces browsers to always use the secure HTTPS entrance, even if someone types 'http://' in the address bar.",

"solution": "Ensure that your web server, application server, load balancer, etc. is configured to enforce Strict-Transport-Security.",

"simplified\_solution": "Add the Strict-Transport-Security header to all HTTPS responses, telling browsers to only connect via HTTPS for a specified period (typically 1-2 years), and optionally include all subdomains.",

"code\_solution\_sample": {

"solution\_description": "This configuration adds the Strict-Transport-Security (HSTS) header to all HTTPS responses from the Nginx server. The 'max-age' directive tells browsers to remember this policy for one year (31536000 seconds), 'includeSubDomains' extends the policy to all subdomains, and 'preload' enables inclusion in browser HSTS preload lists for maximum protection. The configuration also includes HTTP to HTTPS redirection to ensure all traffic uses encryption, and the 'always' parameter ensures the header is sent even with error responses.",

"affected\_files": "Web server configuration files such as nginx.conf, site-specific configuration in /etc/nginx/sites-available/ or /etc/nginx/conf.d/, Apache httpd.conf or .htaccess files, load balancer configuration, or application-level middleware configuration files (Node.js app.js, Django settings.py, Spring Security configuration)",

"code": "# Nginx Configuration - Implement HSTS (Strict-Transport-Security)\n\n# HTTP server block - Redirect all HTTP traffic to HTTPS\nserver {\n listen 80;\n listen [::]:80;\n server\_name pentest-ground.com www.pentest-ground.com;\n \n # Redirect all HTTP requests to HTTPS\n return 301 https://$server\_name$request\_uri;\n}\n\n# HTTPS server block - Main configuration with HSTS\nserver {\n listen 81 ssl http2;\n listen [::]:81 ssl http2;\n server\_name pentest-ground.com www.pentest-ground.com;\n \n # SSL/TLS Configuration\n ssl\_certificate /etc/nginx/ssl/certificate.crt;\n ssl\_certificate\_key /etc/nginx/ssl/private.key;\n ssl\_protocols TLSv1.2 TLSv1.3;\n ssl\_ciphers HIGH:!aNULL:!MD5:!3DES;\n ssl\_prefer\_server\_ciphers on;\n ssl\_session\_cache shared:SSL:10m;\n ssl\_session\_timeout 10m;\n \n # HSTS Header - PRIMARY FIX\n # Forces browsers to use HTTPS for 1 year (31536000 seconds)\n # includeSubDomains: Apply HSTS to all subdomains\n # preload: Allows inclusion in browser HSTS preload lists\n add\_header Strict-Transport-Security \"max-age=31536000; includeSubDomains; preload\" always;\n \n # Additional Security Headers\n add\_header X-Frame-Options \"SAMEORIGIN\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n add\_header X-XSS-Protection \"1; mode=block\" always;\n add\_header Referrer-Policy \"strict-origin-when-cross-origin\" always;\n add\_header Content-Security-Policy \"default-src 'self'; script-src 'self' https://cdnjs.cloudflare.com; style-src 'self' 'unsafe-inline'; img-src 'self' data: https:; font-src 'self'; object-src 'none'; frame-ancestors 'self';\" always;\n \n # Document root\n root /var/www/html;\n index index.html index.htm;\n \n # Static files with caching\n location /static/ {\n alias /var/www/html/static/;\n expires 30d;\n access\_log off;\n add\_header Cache-Control \"public, immutable\";\n # HSTS header is inherited from server block\n }\n \n # Images location\n location /images/ {\n alias /var/www/html/images/;\n expires 30d;\n access\_log off;\n }\n \n # Application routes\n location / {\n try\_files $uri $uri/ =404;\n }\n \n # Login page - extra security\n location /login {\n # Ensure HSTS is set (inherited from server block)\n try\_files $uri $uri/ =404;\n }\n \n # Error pages\n error\_page 404 /404.html;\n error\_page 500 502 503 504 /50x.html;\n location = /50x.html {\n root /var/www/html/errors;\n }\n}\n\n# HSTS PRELOAD SUBMISSION REQUIREMENTS:\n# Before submitting to https://hstspreload.org/, ensure:\n# 1. Valid SSL/TLS certificate\n# 2. All HTTP redirects to HTTPS\n# 3. All subdomains support HTTPS (if using includeSubDomains)\n# 4. HSTS header on base domain with:\n# - max-age >= 31536000 (1 year)\n# - includeSubDomains directive\n# - preload directive\n\n# PROGRESSIVE HSTS DEPLOYMENT STRATEGY:\n# Step 1: Start with short max-age for testing (1 week = 604800)\n# add\_header Strict-Transport-Security \"max-age=604800\" always;\n\n# Step 2: After successful testing, increase to 1 month (2592000)\n# add\_header Strict-Transport-Security \"max-age=2592000\" always;\n\n# Step 3: Add includeSubDomains after verifying all subdomains support HTTPS\n# add\_header Strict-Transport-Security \"max-age=2592000; includeSubDomains\" always;\n\n# Step 4: Increase to 1 year and add preload for maximum protection\n# add\_header Strict-Transport-Security \"max-age=31536000; includeSubDomains; preload\" always;\n\n# VERIFICATION COMMANDS:\n# Test HSTS header:\n# curl -I https://pentest-ground.com:81/ | grep -i strict\n\n# Expected output:\n# Strict-Transport-Security: max-age=31536000; includeSubDomains; preload\n\n# Test HTTP to HTTPS redirect:\n# curl -I http://pentest-ground.com/ \n# Should return: 301 Moved Permanently with Location: https://...\n\n# IMPORTANT CONSIDERATIONS:\n# 1. HSTS only works over HTTPS - ensure valid SSL certificate first\n# 2. Once browser receives HSTS header, it caches the policy for max-age duration\n# 3. To remove HSTS, set max-age=0 and wait for browser cache to expire\n# 4. includeSubDomains affects ALL subdomains - verify they support HTTPS\n# 5. Preload is permanent - only add after thorough testing\n# 6. Use 'always' parameter to send header even with error responses (4xx, 5xx)\n\n# ALTERNATIVE: Application-level HSTS (Node.js/Express example)\n# If you cannot modify server config, implement in application code:\n\n/\*\nconst express = require('express');\nconst app = express();\n\n// HSTS Middleware\napp.use((req, res, next) => {\n if (req.secure) {\n res.setHeader(\n 'Strict-Transport-Security',\n 'max-age=31536000; includeSubDomains; preload'\n );\n }\n next();\n});\n\n// Redirect HTTP to HTTPS\napp.use((req, res, next) => {\n if (!req.secure && req.get('x-forwarded-proto') !== 'https') {\n return res.redirect(301, 'https://' + req.headers.host + req.url);\n }\n next();\n});\n\n// Or use helmet middleware (recommended)\nconst helmet = require('helmet');\napp.use(helmet.hsts({\n maxAge: 31536000,\n includeSubDomains: true,\n preload: true\n}));\n\*/\n\n# MONITORING AND MAINTENANCE:\n# 1. Monitor SSL certificate expiration\n# 2. Regularly update TLS protocols and ciphers\n# 3. Test HSTS implementation: https://hstspreload.org/\n# 4. Check browser compatibility\n# 5. Document HSTS policy in security documentation"

}

}

{

"title": "X-Content-Type-Options Header Missing",

"description": "The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.",

"simplified\_description": "Without the X-Content-Type-Options header, browsers may try to 'guess' what type of file they're receiving instead of trusting what your server tells them. This is like a security guard ignoring the label on a package and opening it to see what's inside. Attackers can exploit this by uploading a malicious file disguised as an image - the browser might 'sniff' it and execute it as JavaScript, leading to cross-site scripting (XSS) attacks that steal user data or hijack accounts.",

"solution": "Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.",

"simplified\_solution": "Add the X-Content-Type-Options: nosniff header to all HTTP responses, forcing browsers to strictly follow the declared content type and preventing them from guessing or reinterpreting file types.",

"code\_solution\_sample": {

"solution\_description": "This configuration adds the X-Content-Type-Options: nosniff header to all HTTP responses from the Nginx server. The 'nosniff' directive prevents browsers from MIME-sniffing responses away from the declared Content-Type, which blocks attacks where malicious content is disguised with an incorrect content type. The 'always' parameter ensures the header is included even in error responses (4xx, 5xx). The configuration also demonstrates proper Content-Type settings for different file types to work correctly with the nosniff directive.",

"affected\_files": "Web server configuration files such as nginx.conf, site-specific configuration in /etc/nginx/sites-available/ or /etc/nginx/conf.d/, Apache httpd.conf or .htaccess files, load balancer configuration, or application-level middleware configuration files (Node.js app.js, Django settings.py, Spring Security configuration, PHP configuration)",

"code": "# Nginx Configuration - Add X-Content-Type-Options Header\n\nhttp {\n # MIME types configuration\n include /etc/nginx/mime.types;\n default\_type application/octet-stream;\n \n # Global security headers applied to all responses\n # X-Content-Type-Options prevents MIME-sniffing\n add\_header X-Content-Type-Options \"nosniff\" always;\n \n # Additional security headers (best practice)\n add\_header X-Frame-Options \"SAMEORIGIN\" always;\n add\_header X-XSS-Protection \"1; mode=block\" always;\n add\_header Referrer-Policy \"strict-origin-when-cross-origin\" always;\n add\_header Strict-Transport-Security \"max-age=31536000; includeSubDomains; preload\" always;\n \n server {\n listen 81 ssl http2;\n listen [::]:81 ssl http2;\n server\_name pentest-ground.com;\n \n # SSL configuration\n ssl\_certificate /etc/nginx/ssl/certificate.crt;\n ssl\_certificate\_key /etc/nginx/ssl/private.key;\n ssl\_protocols TLSv1.2 TLSv1.3;\n ssl\_ciphers HIGH:!aNULL:!MD5;\n \n root /var/www/html;\n index index.html index.htm;\n \n # HTML pages - explicitly set Content-Type\n location / {\n try\_files $uri $uri/ =404;\n # Inherit X-Content-Type-Options from http block\n # Content-Type is set by nginx based on mime.types\n }\n \n # Static CSS files\n location ~\* \\.css$ {\n # Ensure correct Content-Type for CSS\n add\_header Content-Type \"text/css; charset=utf-8\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n expires 30d;\n access\_log off;\n }\n \n # Static JavaScript files\n location ~\* \\.js$ {\n # Ensure correct Content-Type for JavaScript\n add\_header Content-Type \"application/javascript; charset=utf-8\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n expires 30d;\n access\_log off;\n }\n \n # Image files (PNG, JPG, JPEG, GIF, SVG)\n location ~\* \\.(png|jpg|jpeg|gif|svg|ico)$ {\n # Content-Type set automatically by nginx mime.types\n add\_header X-Content-Type-Options \"nosniff\" always;\n expires 30d;\n access\_log off;\n }\n \n # Static files directory\n location /static/ {\n alias /var/www/html/static/;\n # X-Content-Type-Options inherited from http block\n expires 30d;\n access\_log off;\n \n # Ensure proper Content-Type for common file types\n location ~\* /static/.\*\\.css$ {\n add\_header Content-Type \"text/css; charset=utf-8\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n }\n \n location ~\* /static/.\*\\.js$ {\n add\_header Content-Type \"application/javascript; charset=utf-8\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n }\n }\n \n # Images directory\n location /images/ {\n alias /var/www/html/images/;\n add\_header X-Content-Type-Options \"nosniff\" always;\n expires 30d;\n }\n \n # Application routes with security headers\n location ~ ^/(about|services|blog|contact|login)$ {\n try\_files $uri $uri/ =404;\n # X-Content-Type-Options inherited from http block\n }\n \n # Dynamic routes (posts and edit pages)\n location ~ ^/(post/[0-9]+|[0-9]+/edit)$ {\n try\_files $uri $uri/ =404;\n # Security headers inherited\n }\n \n # Favicon\n location = /favicon.ico {\n add\_header Content-Type \"image/x-icon\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n access\_log off;\n expires 30d;\n }\n \n # Error pages\n error\_page 404 /404.html;\n error\_page 500 502 503 504 /50x.html;\n location = /50x.html {\n root /var/www/html/errors;\n add\_header X-Content-Type-Options \"nosniff\" always;\n }\n }\n}\n\n# IMPORTANT CONSIDERATIONS:\n# 1. Always set correct Content-Type headers for all resources\n# 2. Use 'always' parameter to include header in error responses\n# 3. X-Content-Type-Options: nosniff works with properly set Content-Type\n# 4. Test with different file types to ensure proper rendering\n# 5. Monitor for any issues with file downloads after implementation\n\n# MIME TYPES REFERENCE (should be in /etc/nginx/mime.types):\n# text/html html htm shtml;\n# text/css css;\n# application/javascript js;\n# image/png png;\n# image/jpeg jpeg jpg;\n# image/gif gif;\n# image/svg+xml svg svgz;\n# image/x-icon ico;\n# application/json json;\n# application/xml xml;\n# text/plain txt;\n\n# VERIFICATION COMMANDS:\n# Test header is present:\n# curl -I https://pentest-ground.com:81/ | grep -i x-content-type\n\n# Expected output:\n# X-Content-Type-Options: nosniff\n\n# Test for different file types:\n# curl -I https://pentest-ground.com:81/static/css/style.css\n# curl -I https://pentest-ground.com:81/static/js/bootstrap.js\n# curl -I https://pentest-ground.com:81/static/images/logo.png\n\n# ALTERNATIVE: Node.js/Express Implementation\n/\*\nconst express = require('express');\nconst helmet = require('helmet');\nconst app = express();\n\n// Use helmet to set X-Content-Type-Options and other security headers\napp.use(helmet.noSniff());\n\n// Or set manually\napp.use((req, res, next) => {\n res.setHeader('X-Content-Type-Options', 'nosniff');\n next();\n});\n\n// Ensure correct Content-Type for routes\napp.get('/api/data', (req, res) => {\n res.type('application/json');\n res.json({ data: 'example' });\n});\n\napp.get('/page', (req, res) => {\n res.type('text/html');\n res.send('<html><body>Content</body></html>');\n});\n\n// Static files with correct Content-Type\napp.use('/static', express.static('public', {\n setHeaders: (res, path) => {\n res.setHeader('X-Content-Type-Options', 'nosniff');\n // Express automatically sets Content-Type based on file extension\n }\n}));\n\*/\n\n# ALTERNATIVE: Apache .htaccess Implementation\n/\*\n<IfModule mod\_headers.c>\n # Add X-Content-Type-Options header to all responses\n Header always set X-Content-Type-Options \"nosniff\"\n \n # Ensure correct Content-Type for specific file types\n <FilesMatch \"\\.(css)$\">\n Header set Content-Type \"text/css; charset=utf-8\"\n </FilesMatch>\n \n <FilesMatch \"\\.(js)$\">\n Header set Content-Type \"application/javascript; charset=utf-8\"\n </FilesMatch>\n \n <FilesMatch \"\\.(png|jpg|jpeg|gif|svg)$\">\n Header set X-Content-Type-Options \"nosniff\"\n </FilesMatch>\n</IfModule>\n\*/\n\n# SECURITY BENEFITS:\n# 1. Prevents MIME confusion attacks\n# 2. Blocks execution of disguised malicious content\n# 3. Protects against XSS via content type manipulation\n# 4. Enforces proper Content-Type handling\n# 5. Reduces attack surface for file upload vulnerabilities\n\n# TESTING CHECKLIST:\n# ✓ Verify header present on HTML pages\n# ✓ Verify header present on CSS files\n# ✓ Verify header present on JavaScript files\n# ✓ Verify header present on image files\n# ✓ Verify header present on error pages (404, 500)\n# ✓ Test with browser developer tools\n# ✓ Scan with security tools (OWASP ZAP, Burp Suite)\n# ✓ Ensure no functionality breaks after implementation"

}

}  
  
{

"title": "Retrieved from Cache",

"description": "The content was retrieved from a shared cache. If the response data is sensitive, personal or user-specific, this may result in sensitive information being leaked. In some cases, this may even result in a user gaining complete control of the session of another user, depending on the configuration of the caching components in use in their environment. This is primarily an issue where caching servers such as \"proxy\" caches are configured on the local network. This configuration is typically found in corporate or educational environments, for instance.",

"simplified\_description": "Think of caching like a photocopier storing copies of documents. When your application doesn't specify caching rules, shared proxies (like those in office networks) may store copies of responses. If these responses contain sensitive user data (like account information or session tokens), the next person using that network might see someone else's private data from the cache. It's like accidentally getting someone else's photocopied bank statement from a shared office copier.",

"solution": "Validate that the response does not contain sensitive, personal or user-specific information. If it does, consider the use of the following HTTP response headers, to limit, or prevent the content being stored and retrieved from the cache by another user: Cache-Control: no-cache, no-store, must-revalidate, private; Pragma: no-cache; Expires: 0. This configuration directs both HTTP 1.0 and HTTP 1.1 compliant caching servers to not store the response, and to not retrieve the response (without validation) from the cache, in response to a similar request.",

"simplified\_solution": "For pages containing sensitive user data, configure your server to send cache control headers that tell browsers and proxy servers never to cache the content, ensuring each user gets fresh, private data directly from your server.",

"code\_solution\_sample": {

"solution\_description": "This code implements context-aware cache control headers based on content sensitivity. Public static resources (CSS, JS, images from CDNs) are allowed to cache for performance, while sensitive application pages (login, user profiles, dashboards, API responses with personal data) receive strict no-cache directives. The middleware automatically detects the route type and applies appropriate caching policies: 'no-cache, no-store, must-revalidate, private' for sensitive content and 'public, max-age' for static assets. This prevents sensitive data from being stored in shared caches while maintaining performance for public resources.",

"affected\_files": "Application middleware configuration files (app.js, server.js for Node.js), web server configuration files (nginx.conf, Apache httpd.conf/.htaccess), API gateway or reverse proxy configuration, framework-specific middleware (Express middleware, Django middleware.py, Spring interceptors), and controller or route handler files where HTTP headers are set",

"code": "// Node.js/Express - Context-Aware Cache Control Implementation\nconst express = require('express');\nconst app = express();\n\n// Middleware to set appropriate cache control headers based on content type\nconst secureCacheControl = (req, res, next) => {\n const path = req.path.toLowerCase();\n const method = req.method;\n \n // List of sensitive routes that should NEVER be cached\n const sensitiveRoutes = [\n '/login',\n '/logout',\n '/dashboard',\n '/profile',\n '/account',\n '/settings',\n '/admin',\n '/api/user',\n '/api/account',\n '/api/session',\n '/edit',\n '/checkout',\n '/payment'\n ];\n \n // Check if current path contains sensitive route patterns\n const isSensitivePath = sensitiveRoutes.some(route => path.includes(route));\n \n // Check if this is an authenticated session\n const hasAuthToken = req.headers.authorization || req.cookies.SessionID;\n \n // CASE 1: Sensitive data or authenticated content - NO CACHING\n if (isSensitivePath || hasAuthToken || method !== 'GET') {\n // Prevent all caching for sensitive content\n res.setHeader('Cache-Control', 'no-cache, no-store, must-revalidate, private');\n res.setHeader('Pragma', 'no-cache'); // HTTP 1.0 compatibility\n res.setHeader('Expires', '0'); // Proxies\n \n // Additional security: prevent caching in browser back/forward\n res.setHeader('Surrogate-Control', 'no-store');\n }\n // CASE 2: CDN resources (external libraries) - Allow caching\n else if (path.includes('cdnjs.cloudflare.com') || \n path.match(/\\.(css|js|woff|woff2|ttf|eot)$/) && \n path.includes('/static/')) {\n // Public static assets can be cached aggressively\n res.setHeader('Cache-Control', 'public, max-age=31536000, immutable');\n }\n // CASE 3: Public static images - Moderate caching\n else if (path.match(/\\.(jpg|jpeg|png|gif|svg|ico|webp)$/)) {\n res.setHeader('Cache-Control', 'public, max-age=86400'); // 1 day\n }\n // CASE 4: Public pages without user data - Short-term caching\n else if (['/about', '/services', '/blog', '/contact'].some(route => path === route)) {\n res.setHeader('Cache-Control', 'public, max-age=3600, must-revalidate'); // 1 hour\n }\n // CASE 5: Default for HTML pages - No caching (conservative approach)\n else {\n res.setHeader('Cache-Control', 'no-cache, no-store, must-revalidate, private');\n res.setHeader('Pragma', 'no-cache');\n res.setHeader('Expires', '0');\n }\n \n next();\n};\n\n// Apply cache control middleware globally\napp.use(secureCacheControl);\n\n// Example: API endpoint with sensitive user data\napp.get('/api/user/profile', (req, res) => {\n // Middleware already applied no-cache headers\n if (!req.headers.authorization) {\n return res.status(401).json({ error: 'Unauthorized' });\n }\n \n // Additional explicit cache prevention for API responses\n res.setHeader('Cache-Control', 'no-store, private');\n \n res.json({\n userId: 12345,\n username: 'user',\n email: 'user@example.com',\n sensitiveData: 'private information'\n });\n});\n\n// Example: Login page with strict no-cache\napp.get('/login', (req, res) => {\n // Cache headers already set by middleware\n res.send('<html><body><form>Login Form</form></body></html>');\n});\n\n// Example: Public blog page with moderate caching\napp.get('/blog', (req, res) => {\n // Public content - cache headers allow moderate caching\n res.send('<html><body>Public Blog Content</body></html>');\n});\n\n// Example: Post-authentication redirect with session data\napp.post('/login', (req, res) => {\n // POST requests automatically get no-cache headers\n // Validate credentials...\n \n if (validCredentials) {\n // Set session cookie with secure flags\n res.cookie('SessionID', 'secure-token', {\n httpOnly: true,\n secure: true,\n sameSite: 'strict'\n });\n \n // Explicitly prevent caching of redirect response\n res.setHeader('Cache-Control', 'no-store, no-cache, must-revalidate, private');\n res.redirect('/dashboard');\n }\n});\n\n// Static file serving with appropriate caching\napp.use('/static', express.static('public', {\n setHeaders: (res, path) => {\n if (path.endsWith('.css') || path.endsWith('.js')) {\n // Static assets - long-term caching\n res.setHeader('Cache-Control', 'public, max-age=31536000, immutable');\n } else if (path.match(/\\.(jpg|jpeg|png|gif|svg)$/)) {\n // Images - moderate caching\n res.setHeader('Cache-Control', 'public, max-age=86400');\n }\n }\n}));\n\n// Alternative: Using helmet middleware for enhanced security\n/\*\nconst helmet = require('helmet');\n\n// Disable caching globally (most secure, but impacts performance)\napp.use(helmet.noCache());\n\n// Or use custom cache control per route\napp.get('/sensitive-route', \n helmet.noCache(),\n (req, res) => {\n res.json({ sensitiveData: 'private' });\n }\n);\n\*/\n\n// Nginx Configuration Alternative (for reference)\n/\*\n# Nginx - Cache Control Configuration\n\nserver {\n listen 81 ssl http2;\n server\_name pentest-ground.com;\n \n # Default: No caching for HTML pages\n location / {\n add\_header Cache-Control \"no-cache, no-store, must-revalidate, private\" always;\n add\_header Pragma \"no-cache\" always;\n add\_header Expires \"0\" always;\n }\n \n # Sensitive routes: Strict no-cache\n location ~ ^/(login|dashboard|profile|account|admin|.\*edit.\*) {\n add\_header Cache-Control \"no-cache, no-store, must-revalidate, private\" always;\n add\_header Pragma \"no-cache\" always;\n add\_header Expires \"0\" always;\n add\_header Surrogate-Control \"no-store\" always;\n }\n \n # Static assets: Aggressive caching\n location /static/ {\n location ~\* \\.(css|js|woff|woff2|ttf|eot)$ {\n add\_header Cache-Control \"public, max-age=31536000, immutable\";\n expires 1y;\n access\_log off;\n }\n \n location ~\* \\.(jpg|jpeg|png|gif|svg|ico|webp)$ {\n add\_header Cache-Control \"public, max-age=86400\";\n expires 1d;\n }\n }\n \n # API endpoints: No caching\n location /api/ {\n add\_header Cache-Control \"no-store, private\" always;\n add\_header Pragma \"no-cache\" always;\n add\_header Expires \"0\" always;\n }\n}\n\*/\n\n// IMPORTANT CONSIDERATIONS:\n// 1. NEVER cache responses containing:\n// - Session tokens or authentication data\n// - Personal user information (names, emails, addresses)\n// - Financial data (credit cards, account balances)\n// - Health information or PII\n// - CSRF tokens or security-related data\n//\n// 2. Safe to cache:\n// - Public static assets (CSS, JS, images)\n// - Public marketing pages without user data\n// - CDN-hosted libraries\n//\n// 3. Cache control headers hierarchy:\n// - Cache-Control (HTTP 1.1) - primary directive\n// - Pragma (HTTP 1.0) - backward compatibility\n// - Expires (HTTP 1.0/1.1) - fallback for old proxies\n//\n// 4. Testing:\n// - Verify headers with: curl -I https://yourdomain.com/sensitive-page\n// - Test with browser DevTools Network tab\n// - Check \"Age\" header in responses (should be absent for no-cache)\n// - Test from corporate proxy environments if possible\n\n// VERIFICATION CHECKLIST:\n// ✓ Login pages have no-cache headers\n// ✓ User dashboards have no-cache headers\n// ✓ API responses with user data have no-cache headers\n// ✓ POST/PUT/DELETE requests have no-cache headers\n// ✓ Static assets have appropriate max-age\n// ✓ Public pages have reasonable cache duration\n// ✓ All cache headers include 'private' for user-specific content\n// ✓ CDN resources are excluded from no-cache restrictions\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on port ${PORT}`);\n console.log('Context-aware cache control enabled');\n console.log('Sensitive routes protected from caching');\n});\n\nmodule.exports = app;"

}

}  
  
{

"title": "Vulnerable JS Library",

"description": "The identified library appears to be vulnerable.",

"simplified\_description": "Your website is using outdated versions of JavaScript libraries (Bootstrap 3.0.2 and jQuery 3.4.1) that contain known security vulnerabilities. This is like using old locks on your doors that burglars know how to pick. Attackers can exploit these weaknesses to inject malicious code, steal user data, or compromise your website. These specific versions have documented security flaws that hackers actively target.",

"solution": "Upgrade to the latest version of the affected library.",

"simplified\_solution": "Update Bootstrap and jQuery to their latest stable versions to eliminate known security vulnerabilities and ensure your application is protected against documented exploits.",

"code\_solution\_sample": {

"solution\_description": "This solution demonstrates how to upgrade the vulnerable JavaScript libraries to their latest secure versions. Bootstrap 3.0.2 (released 2013) contains multiple XSS vulnerabilities and should be upgraded to Bootstrap 5.3.x or later. jQuery 3.4.1 (released 2019) has prototype pollution vulnerabilities (CVE-2019-11358, CVE-2020-11022, CVE-2020-11023) and should be upgraded to jQuery 3.7.x or later. The code shows proper implementation using CDN links with Subresource Integrity (SRI) hashes for security, along with fallback mechanisms and proper placement in HTML. It also includes package.json configuration for projects using npm/yarn, and verification steps to ensure successful migration.",

"affected\_files": "HTML template files, layout/master page files (index.html, base.html, layout.ejs, header.pug), static asset directories (/static/js/, /public/js/, /assets/js/), package.json or bower.json for dependency management, build configuration files (webpack.config.js, gulpfile.js), and any JavaScript files that reference or import these libraries",

"code": "<!-- HTML - Upgrade Vulnerable JavaScript Libraries -->\n\n<!-- BEFORE (VULNERABLE): -->\n<!-- <script src=\"/static/js/jquery-3.4.1.min.js\"></script> -->\n<!-- <script src=\"/static/js/bootstrap.js\"></script> -->\n\n<!-- AFTER (SECURE): Update to latest stable versions -->\n\n<!DOCTYPE html>\n<html lang=\"en\">\n<head>\n <meta charset=\"UTF-8\">\n <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">\n <title>Secure Application</title>\n \n <!-- Bootstrap 5.3.3 CSS (Latest stable as of 2024) -->\n <!-- Upgrade from Bootstrap 3.0.2 to 5.3.3+ -->\n <link \n href=\"https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css\" \n rel=\"stylesheet\" \n integrity=\"sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH\" \n crossorigin=\"anonymous\">\n</head>\n<body>\n <!-- Your content here -->\n \n <!-- JavaScript libraries at end of body for performance -->\n \n <!-- jQuery 3.7.1 (Latest stable version) -->\n <!-- Upgrade from jQuery 3.4.1 to 3.7.1+ -->\n <!-- Fixes CVE-2019-11358, CVE-2020-11022, CVE-2020-11023 -->\n <script \n src=\"https://code.jquery.com/jquery-3.7.1.min.js\" \n integrity=\"sha384-1H217gwSVyLSIfaLxHbE7dRb3v4mYCKbpQvzx0cegeju1MVsGrX5xXxAvs/HgeFs\" \n crossorigin=\"anonymous\"></script>\n \n <!-- Fallback to local copy if CDN fails -->\n <script>\n if (typeof jQuery === 'undefined') {\n document.write('<script src=\"/static/js/jquery-3.7.1.min.js\"><\\/script>');\n }\n </script>\n \n <!-- Bootstrap 5.3.3 JS Bundle (includes Popper.js) -->\n <!-- Bootstrap 5 has breaking changes from v3 - migration required -->\n <script \n src=\"https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js\" \n integrity=\"sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz\" \n crossorigin=\"anonymous\"></script>\n \n <!-- Fallback for Bootstrap -->\n <script>\n if (typeof bootstrap === 'undefined') {\n document.write('<script src=\"/static/js/bootstrap.bundle.min.js\"><\\/script>');\n }\n </script>\n</body>\n</html>\n\n<!-- ============================================ -->\n<!-- PACKAGE.JSON CONFIGURATION (for npm projects) -->\n<!-- ============================================ -->\n\n/\*\n{\n \"name\": \"secure-web-app\",\n \"version\": \"1.0.0\",\n \"description\": \"Web application with updated dependencies\",\n \"dependencies\": {\n \"jquery\": \"^3.7.1\",\n \"bootstrap\": \"^5.3.3\",\n \"popper.js\": \"^2.11.8\"\n },\n \"scripts\": {\n \"audit\": \"npm audit\",\n \"audit-fix\": \"npm audit fix\",\n \"update-deps\": \"npm update\",\n \"check-updates\": \"npx npm-check-updates\"\n }\n}\n\*/\n\n<!-- ============================================ -->\n<!-- SELF-HOSTING LIBRARIES (Recommended for production) -->\n<!-- ============================================ -->\n\n<!-- Step 1: Download latest versions -->\n<!-- \n jQuery: https://jquery.com/download/\n Bootstrap: https://getbootstrap.com/docs/5.3/getting-started/download/\n-->\n\n<!-- Step 2: Place in /static/js/ directory -->\n<!-- Step 3: Update HTML references -->\n\n<!--\n<script src=\"/static/js/jquery-3.7.1.min.js\"></script>\n<script src=\"/static/js/bootstrap.bundle.min.js\"></script>\n-->\n\n<!-- ============================================ -->\n<!-- MIGRATION NOTES: Bootstrap 3 to Bootstrap 5 -->\n<!-- ============================================ -->\n\n/\*\nBootstrap 5 Breaking Changes (Major updates required):\n\n1. jQuery is NO LONGER REQUIRED for Bootstrap 5\n - Bootstrap 5 uses vanilla JavaScript\n - Keep jQuery only if your custom code depends on it\n\n2. Class name changes:\n - .pull-left → .float-start\n - .pull-right → .float-end\n - .hidden → .d-none\n - .show → .d-block\n - .ml-\* → .ms-\* (margin-left → margin-start)\n - .mr-\* → .me-\* (margin-right → margin-end)\n - .pl-\* → .ps-\* (padding-left → padding-start)\n - .pr-\* → .pe-\* (padding-right → padding-end)\n\n3. Form controls:\n - .form-control-\* sizing classes changed\n - .custom-select → .form-select\n - .custom-file → removed (use .form-control with type=\"file\")\n\n4. JavaScript API changes:\n - Data attributes: data-toggle → data-bs-toggle\n - Events: 'hidden.bs.modal' instead of 'hidden'\n - Modal: $('#myModal').modal('show') → new bootstrap.Modal(myModal).show()\n\n5. Removed components:\n - Wells, panels, thumbnails removed\n - Use cards instead\n\n6. Glyph icons removed:\n - Use Bootstrap Icons, Font Awesome, or other icon libraries\n\nRefer to official migration guide:\nhttps://getbootstrap.com/docs/5.3/migration/\n\*/\n\n<!-- ============================================ -->\n<!-- SECURITY VERIFICATION CHECKLIST -->\n<!-- ============================================ -->\n\n/\*\nPost-Update Verification Steps:\n\n1. Check library versions:\n - Open browser DevTools → Console\n - Run: jQuery.fn.jquery (should show 3.7.1+)\n - Run: bootstrap.Tooltip.VERSION (should show 5.3.3+)\n\n2. Verify SRI hashes:\n - Generate new hashes: https://www.srihash.org/\n - Or use: curl -s URL | openssl dgst -sha384 -binary | openssl base64 -A\n\n3. Test functionality:\n - Test all interactive components (modals, dropdowns, tooltips)\n - Verify responsive behavior\n - Check console for JavaScript errors\n\n4. Run security audit:\n - npm audit (for npm projects)\n - Check https://snyk.io/vuln/ for known vulnerabilities\n - Use OWASP ZAP or similar tools to rescan\n\n5. Update documentation:\n - Document library versions in README.md\n - Update deployment scripts\n - Inform team of breaking changes\n\*/\n\n<!-- ============================================ -->\n<!-- AUTOMATED DEPENDENCY MANAGEMENT -->\n<!-- ============================================ -->\n\n/\*\nNode.js/npm - Automated Updates and Monitoring:\n\n1. Install dependency checking tools:\n npm install -g npm-check-updates\n npm install -g snyk\n\n2. Check for updates:\n npx npm-check-updates\n npx npm-check-updates -u # Update package.json\n\n3. Security scanning:\n npm audit\n npm audit fix\n npm audit fix --force # For breaking changes\n \n # Or use Snyk:\n snyk test\n snyk monitor\n\n4. Automated CI/CD security checks (GitHub Actions example):\n\*/\n\n# .github/workflows/security-scan.yml\n# name: Security Audit\n# on: [push, pull\_request]\n# jobs:\n# security:\n# runs-on: ubuntu-latest\n# steps:\n# - uses: actions/checkout@v3\n# - uses: actions/setup-node@v3\n# with:\n# node-version: '18'\n# - run: npm ci\n# - run: npm audit --production\n# - run: npx snyk test --severity-threshold=high\n\n<!-- ============================================ -->\n<!-- KNOWN VULNERABILITIES FIXED -->\n<!-- ============================================ -->\n\n/\*\njQuery 3.4.1 → 3.7.1 fixes:\n- CVE-2019-11358: Prototype pollution vulnerability\n- CVE-2020-11022: XSS vulnerability in htmlPrefilter\n- CVE-2020-11023: XSS vulnerability in selector logic\n\nBootstrap 3.0.2 → 5.3.3 fixes:\n- CVE-2016-10735: XSS in data-target attribute\n- CVE-2018-14040: XSS in collapse data-parent attribute\n- CVE-2018-14041: XSS in tooltip and popover data-template\n- CVE-2018-14042: XSS in sanitizer bypass\n- CVE-2019-8331: XSS in tooltip/popover with sanitizer\n- Multiple other XSS vulnerabilities in versions 3.x-4.x\n\*/\n\n<!-- ============================================ -->\n<!-- MAINTENANCE BEST PRACTICES -->\n<!-- ============================================ -->\n\n/\*\n1. Regular updates:\n - Review dependencies quarterly minimum\n - Subscribe to security advisories (GitHub Security Alerts)\n - Monitor CVE databases for your stack\n\n2. Version pinning:\n - Use exact versions in production: \"jquery\": \"3.7.1\"\n - Use caret for development: \"jquery\": \"^3.7.1\"\n - Lock file: commit package-lock.json\n\n3. Testing strategy:\n - Maintain comprehensive test suite\n - Test after every dependency update\n - Use staging environment for validation\n\n4. Rollback plan:\n - Keep previous working versions\n - Document rollback procedures\n - Use version control tags\n\n5. Security monitoring:\n - Enable Dependabot/Renovate for automated PRs\n - Set up Snyk or similar continuous monitoring\n - Subscribe to library security mailing lists\n\*/\n\n<!-- ============================================ -->\n<!-- IMMEDIATE ACTION ITEMS -->\n<!-- ============================================ -->\n\n/\*\nCRITICAL - DO IMMEDIATELY:\n\n1. Update jQuery from 3.4.1 to 3.7.1 or later\n2. Update Bootstrap from 3.0.2 to 5.3.3 or later\n3. Test all functionality after updates\n4. Deploy to production after thorough testing\n5. Set up automated dependency monitoring\n\nTIMELINE:\n- Day 1: Update in development environment\n- Day 2-3: Thorough testing and bug fixes\n- Day 4: Deploy to staging\n- Day 5: Production deployment\n- Ongoing: Monitor for new vulnerabilities\n\*/"

}

}

{

"title": "Information Disclosure - Suspicious Comments",

"description": "The response appears to contain suspicious comments which may help an attacker.",

"simplified\_description": "Your website's code contains comments and notes (like 'TODO', 'username' references, or developer notes) that are visible to anyone inspecting the page source. This is like leaving sticky notes with hints about your security system on your front door. While these comments help developers during coding, they can reveal system information, planned features, database field names, or security weaknesses that attackers can exploit to plan more sophisticated attacks.",

"solution": "Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.",

"simplified\_solution": "Remove or sanitize all developer comments, TODO notes, and debugging information from production code before deployment, especially those containing sensitive field names, system details, or unfinished security features.",

"code\_solution\_sample": {

"solution\_description": "This solution demonstrates a comprehensive approach to removing suspicious comments from production code. It includes: (1) a build process that strips comments from HTML, JavaScript, and CSS files before deployment, (2) server-side HTML minification middleware that removes comments dynamically, (3) template configuration to prevent comment output, and (4) a pre-commit hook to detect sensitive patterns in comments. The code uses popular tools like html-minifier, terser, and cssnano to automatically clean production assets while preserving readable code in development environments.",

"affected\_files": "HTML template files (\*.html, \*.ejs, \*.pug, \*.hbs), JavaScript source files (\*.js), CSS/SCSS files (\*.css, \*.scss), server-side view templates, build configuration files (webpack.config.js, gulpfile.js, package.json), and any files containing developer comments or TODO notes that are served to clients",

"code": "// Node.js/Express - Remove Suspicious Comments in Production\n\nconst express = require('express');\nconst minify = require('html-minifier').minify;\nconst path = require('path');\nconst fs = require('fs');\nconst app = express();\n\n// Environment configuration\nconst isProduction = process.env.NODE\_ENV === 'production';\n\n// Middleware to minify HTML and remove comments in production\nconst htmlMinifyMiddleware = (req, res, next) => {\n if (!isProduction) {\n return next();\n }\n \n // Store original send function\n const originalSend = res.send;\n \n // Override send function to minify HTML responses\n res.send = function(data) {\n // Only process HTML content\n if (typeof data === 'string' && \n (res.get('Content-Type') || '').includes('text/html')) {\n try {\n // Minify and remove comments\n data = minify(data, {\n removeComments: true, // Remove ALL HTML comments\n collapseWhitespace: true, // Remove extra whitespace\n removeAttributeQuotes: true, // Clean attributes\n minifyJS: true, // Minify inline JavaScript\n minifyCSS: true, // Minify inline CSS\n removeRedundantAttributes: true,\n removeScriptTypeAttributes: true,\n removeStyleLinkTypeAttributes: true\n });\n } catch (err) {\n console.error('HTML minification error:', err);\n // Continue with original data if minification fails\n }\n }\n \n // Call original send with processed data\n originalSend.call(this, data);\n };\n \n next();\n};\n\n// Apply middleware globally in production\nif (isProduction) {\n app.use(htmlMinifyMiddleware);\n}\n\n// Configure view engine to not render comments\napp.set('view engine', 'ejs');\napp.set('views', path.join(\_\_dirname, 'views'));\n\n// Example routes\napp.get('/contact', (req, res) => {\n // BAD: Contains TODO comment visible in source\n // <!-- TODO: Add CAPTCHA for spam prevention -->\n \n // GOOD: Use environment variables or config files for notes\n const pageConfig = {\n title: 'Contact Us',\n // Developer notes kept server-side only\n devNotes: process.env.NODE\_ENV === 'development' \n ? 'Remember to add CAPTCHA' \n : null\n };\n \n res.render('contact', pageConfig);\n});\n\napp.get('/login', (req, res) => {\n // BAD: HTML comment revealing field name\n // <!-- Username field for authentication -->\n \n // GOOD: No sensitive information in client-side code\n res.render('login', {\n csrfToken: req.csrfToken(),\n // Keep implementation details server-side\n });\n});\n\n// Static files configuration for production\nif (isProduction) {\n // Serve pre-minified, comment-free static files\n app.use('/static', express.static(path.join(\_\_dirname, 'dist'), {\n maxAge: '1y',\n etag: true\n }));\n} else {\n // Development: serve unminified files with comments\n app.use('/static', express.static(path.join(\_\_dirname, 'public')));\n}\n\n// ============================================\n// BUILD PROCESS CONFIGURATION (package.json scripts)\n// ============================================\n\n/\*\n{\n \"scripts\": {\n \"build\": \"npm run clean && npm run build:js && npm run build:css && npm run build:html\",\n \"clean\": \"rm -rf dist/\*\",\n \"build:js\": \"terser public/js/\*\*/\*.js -o dist/js/bundle.min.js --compress --mangle --comments false\",\n \"build:css\": \"cssnano public/css/style.css dist/css/style.min.css --no-comments\",\n \"build:html\": \"node scripts/minify-html.js\",\n \"lint:comments\": \"node scripts/check-suspicious-comments.js\",\n \"precommit\": \"npm run lint:comments\"\n },\n \"devDependencies\": {\n \"html-minifier\": \"^4.0.0\",\n \"terser\": \"^5.19.0\",\n \"cssnano\": \"^6.0.0\",\n \"cssnano-cli\": \"^1.0.5\"\n }\n}\n\*/\n\n// ============================================\n// AUTOMATED COMMENT DETECTION SCRIPT\n// scripts/check-suspicious-comments.js\n// ============================================\n\n/\*\nconst fs = require('fs');\nconst path = require('path');\nconst glob = require('glob');\n\n// Patterns to detect suspicious comments\nconst suspiciousPatterns = [\n /TODO/i,\n /FIXME/i,\n /HACK/i,\n /XXX/i,\n /BUG/i,\n /username/i,\n /password/i,\n /api[\_-]?key/i,\n /secret/i,\n /token/i,\n /admin/i,\n /database/i,\n /sql/i,\n /\\bdb\\b/i,\n /config/i,\n /vulnerable/i,\n /security/i,\n /temporary/i,\n /debug/i,\n /test[\_-]?data/i\n];\n\nfunction checkFile(filePath) {\n const content = fs.readFileSync(filePath, 'utf8');\n const lines = content.split('\\n');\n const issues = [];\n \n lines.forEach((line, index) => {\n // Check for HTML comments\n const htmlCommentMatch = line.match(/<!--(.+?)-->/g);\n if (htmlCommentMatch) {\n htmlCommentMatch.forEach(comment => {\n if (suspiciousPatterns.some(pattern => pattern.test(comment))) {\n issues.push({\n file: filePath,\n line: index + 1,\n content: comment.trim()\n });\n }\n });\n }\n \n // Check for JavaScript comments\n const jsCommentMatch = line.match(/\\/\\/(.+?)$|/\\\*(.+?)\\\*\\//g);\n if (jsCommentMatch) {\n jsCommentMatch.forEach(comment => {\n if (suspiciousPatterns.some(pattern => pattern.test(comment))) {\n issues.push({\n file: filePath,\n line: index + 1,\n content: comment.trim()\n });\n }\n });\n }\n });\n \n return issues;\n}\n\n// Scan all relevant files\nconst files = [\n ...glob.sync('views/\*\*\\/\*.{html,ejs,pug,hbs}'),\n ...glob.sync('public/\*\*\\/\*.{js,css,html}'),\n ...glob.sync('src/\*\*\\/\*.{js,jsx,ts,tsx}')\n];\n\nconst allIssues = [];\nfiles.forEach(file => {\n const issues = checkFile(file);\n allIssues.push(...issues);\n});\n\nif (allIssues.length > 0) {\n console.error('\\n❌ Suspicious comments detected:\\n');\n allIssues.forEach(issue => {\n console.error(`${issue.file}:${issue.line}`);\n console.error(` ${issue.content}\\n`);\n });\n console.error(`Found ${allIssues.length} suspicious comment(s).`);\n console.error('Please remove or sanitize these comments before deployment.\\n');\n process.exit(1);\n} else {\n console.log('✅ No suspicious comments detected.\\n');\n}\n\*/\n\n// ============================================\n// HTML MINIFICATION BUILD SCRIPT\n// scripts/minify-html.js\n// ============================================\n\n/\*\nconst fs = require('fs');\nconst path = require('path');\nconst { minify } = require('html-minifier');\nconst glob = require('glob');\n\nconst inputDir = 'views';\nconst outputDir = 'dist/views';\n\nconst minifyOptions = {\n removeComments: true,\n collapseWhitespace: true,\n removeAttributeQuotes: true,\n minifyJS: true,\n minifyCSS: true,\n removeRedundantAttributes: true,\n removeScriptTypeAttributes: true,\n removeStyleLinkTypeAttributes: true,\n useShortDoctype: true\n};\n\n// Ensure output directory exists\nif (!fs.existsSync(outputDir)) {\n fs.mkdirSync(outputDir, { recursive: true });\n}\n\n// Process all HTML template files\nconst files = glob.sync(`${inputDir}/\*\*\\/\*.{html,ejs,pug}`);\n\nfiles.forEach(file => {\n const content = fs.readFileSync(file, 'utf8');\n const minified = minify(content, minifyOptions);\n \n const relativePath = path.relative(inputDir, file);\n const outputPath = path.join(outputDir, relativePath);\n const outputDirPath = path.dirname(outputPath);\n \n if (!fs.existsSync(outputDirPath)) {\n fs.mkdirSync(outputDirPath, { recursive: true });\n }\n \n fs.writeFileSync(outputPath, minified);\n console.log(`Minified: ${file} → ${outputPath}`);\n});\n\nconsole.log(`\\nMinified ${files.length} file(s).`);\n\*/\n\n// ============================================\n// TEMPLATE BEST PRACTICES\n// ============================================\n\n/\*\nEJS Template Example (views/contact.ejs):\n\nBAD - Contains suspicious comments:\n<!DOCTYPE html>\n<html>\n<head>\n <title>Contact</title>\n <!-- TODO: Add CAPTCHA for spam prevention -->\n <!-- Username field uses 'email' as identifier -->\n</head>\n<body>\n <!-- FIXME: Validate form on server side -->\n <form action=\"/contact\" method=\"POST\">\n <input type=\"text\" name=\"email\" />\n <button type=\"submit\">Submit</button>\n </form>\n</body>\n</html>\n\nGOOD - No comments, clean production code:\n<!DOCTYPE html>\n<html>\n<head>\n <title>Contact</title>\n</head>\n<body>\n <form action=\"/contact\" method=\"POST\">\n <input type=\"text\" name=\"email\" required />\n <button type=\"submit\">Submit</button>\n </form>\n</body>\n</html>\n\nDeveloper notes should be in:\n- Separate documentation files (README.md, CONTRIBUTING.md)\n- Issue tracking systems (Jira, GitHub Issues)\n- Code review comments (not in source)\n- Server-side configuration files (not served to clients)\n\*/\n\n// ============================================\n// GIT PRE-COMMIT HOOK\n// .git/hooks/pre-commit\n// ============================================\n\n/\*\n#!/bin/bash\n\necho \"Checking for suspicious comments...\"\nnpm run lint:comments\n\nif [ $? -ne 0 ]; then\n echo \"Commit rejected: Suspicious comments detected.\"\n echo \"Please remove sensitive comments before committing.\"\n exit 1\nfi\n\necho \"Pre-commit checks passed.\"\nexit 0\n\*/\n\n// ============================================\n// PRODUCTION DEPLOYMENT CHECKLIST\n// ============================================\n\n/\*\nBefore deploying to production:\n\n✓ Run build process to minify and remove comments\n✓ Set NODE\_ENV=production\n✓ Verify no TODO/FIXME comments in deployed files\n✓ Check HTML source in browser for leaked information\n✓ Review JavaScript console for debug statements\n✓ Scan with security tools (OWASP ZAP, Burp Suite)\n✓ Remove or obfuscate field names in comments\n✓ Clear developer notes from client-side code\n✓ Test that functionality works without comments\n✓ Document important notes in secure wiki/docs instead\n\*/\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on port ${PORT}`);\n console.log(`Environment: ${process.env.NODE\_ENV}`);\n if (isProduction) {\n console.log('✓ HTML comment removal enabled');\n console.log('✓ Asset minification enabled');\n }\n});\n\nmodule.exports = app;"

}

}

{

"title": "Secure Pages Include Mixed Content",

"description": "The page includes mixed content, that is content accessed via HTTP instead of HTTPS.",

"simplified\_description": "Your secure HTTPS website is loading some resources (like images, scripts, or links) over insecure HTTP connections. This is like having a secure vault with an encrypted door, but leaving a regular unlocked window open. Even though your main page uses encryption, the unencrypted HTTP content can be intercepted by attackers who can then inject malicious code, steal data being transmitted, or trick users into visiting phishing sites. Browsers will also display security warnings that damage user trust.",

"solution": "A page that is available over SSL/TLS must be comprised completely of content which is transmitted over SSL/TLS. The page must not contain any content that is transmitted over unencrypted HTTP. This includes content from third party sites.",

"simplified\_solution": "Replace all HTTP URLs with HTTPS URLs throughout your website, including images, scripts, stylesheets, and links to third-party resources, ensuring every element on secure pages uses encrypted connections.",

"code\_solution\_sample": {

"solution\_description": "This solution provides multiple layers of protection against mixed content vulnerabilities. It includes: (1) automated URL rewriting to convert HTTP URLs to HTTPS, (2) Content Security Policy (CSP) headers that block mixed content and enforce HTTPS, (3) middleware to scan and validate outgoing HTML for mixed content issues, (4) strict HTTPS enforcement with HSTS headers, and (5) automated tools to detect and fix mixed content during development. The code also demonstrates proper handling of external resources and implements the upgrade-insecure-requests CSP directive to automatically upgrade HTTP requests to HTTPS.",

"affected\_files": "HTML template files (\*.html, \*.ejs, \*.pug, \*.hbs), view components, CSS files with url() references, JavaScript files loading external resources, server-side rendering components, content management system templates, email templates, and any files containing hardcoded URLs or resource references to external sites",

"code": "// Node.js/Express - Fix Mixed Content Vulnerabilities\n\nconst express = require('express');\nconst cheerio = require('cheerio');\nconst url = require('url');\nconst app = express();\n\n// Middleware to enforce HTTPS and prevent mixed content\nconst secureContentMiddleware = (req, res, next) => {\n // Ensure connection is HTTPS\n if (!req.secure && req.get('x-forwarded-proto') !== 'https' && process.env.NODE\_ENV === 'production') {\n return res.redirect(301, `https://${req.hostname}${req.url}`);\n }\n \n // Set security headers to prevent mixed content\n // Content-Security-Policy with upgrade-insecure-requests directive\n res.setHeader(\n 'Content-Security-Policy',\n \"default-src 'self' https:; \" +\n \"script-src 'self' https://cdnjs.cloudflare.com; \" +\n \"style-src 'self' 'unsafe-inline' https:; \" +\n \"img-src 'self' https: data:; \" +\n \"font-src 'self' https:; \" +\n \"connect-src 'self' https:; \" +\n \"media-src 'self' https:; \" +\n \"object-src 'none'; \" +\n \"frame-src 'self' https:; \" +\n \"upgrade-insecure-requests; \" + // Automatically upgrade HTTP to HTTPS\n \"block-all-mixed-content\" // Block any remaining mixed content\n );\n \n // HSTS header to enforce HTTPS\n res.setHeader(\n 'Strict-Transport-Security',\n 'max-age=31536000; includeSubDomains; preload'\n );\n \n next();\n};\n\n// Middleware to scan and fix mixed content in HTML responses\nconst mixedContentScannerMiddleware = (req, res, next) => {\n const originalSend = res.send;\n \n res.send = function(data) {\n // Only process HTML responses\n if (typeof data === 'string' && \n (res.get('Content-Type') || '').includes('text/html')) {\n try {\n const $ = cheerio.load(data);\n let mixedContentFound = false;\n const issues = [];\n \n // Check and fix all resource URLs\n const resourceSelectors = [\n { selector: 'img[src]', attr: 'src' },\n { selector: 'script[src]', attr: 'src' },\n { selector: 'link[href]', attr: 'href' },\n { selector: 'iframe[src]', attr: 'src' },\n { selector: 'video[src]', attr: 'src' },\n { selector: 'audio[src]', attr: 'src' },\n { selector: 'source[src]', attr: 'src' },\n { selector: 'embed[src]', attr: 'src' },\n { selector: 'object[data]', attr: 'data' },\n { selector: 'a[href]', attr: 'href' }\n ];\n \n resourceSelectors.forEach(({ selector, attr }) => {\n $(selector).each((i, elem) => {\n const originalUrl = $(elem).attr(attr);\n \n if (originalUrl && originalUrl.startsWith('http://')) {\n mixedContentFound = true;\n const issue = `${selector}: ${originalUrl}`;\n issues.push(issue);\n \n // Attempt to upgrade to HTTPS\n const secureUrl = originalUrl.replace(/^http:\\/\\//, 'https://');\n $(elem).attr(attr, secureUrl);\n \n console.warn(`[Mixed Content Fixed] ${req.path}: ${originalUrl} → ${secureUrl}`);\n }\n });\n });\n \n // Check inline styles for HTTP URLs\n $('[style]').each((i, elem) => {\n const style = $(elem).attr('style');\n if (style && style.includes('http://')) {\n mixedContentFound = true;\n const fixedStyle = style.replace(/http:\\/\\//g, 'https://');\n $(elem).attr('style', fixedStyle);\n console.warn(`[Mixed Content Fixed] Inline style in ${req.path}`);\n }\n });\n \n // Log mixed content issues in development\n if (mixedContentFound && process.env.NODE\_ENV !== 'production') {\n console.error(`\\n⚠️ Mixed Content Detected on ${req.path}:`);\n issues.forEach(issue => console.error(` - ${issue}`));\n console.error('');\n }\n \n data = $.html();\n } catch (err) {\n console.error('Mixed content scanner error:', err);\n }\n }\n \n originalSend.call(this, data);\n };\n \n next();\n};\n\n// Apply middleware\napp.use(secureContentMiddleware);\napp.use(mixedContentScannerMiddleware);\n\n// Helper function to ensure URLs are HTTPS\nfunction ensureHttps(urlString) {\n if (!urlString) return urlString;\n \n // Skip data URLs, relative URLs, and protocol-relative URLs\n if (urlString.startsWith('data:') || \n urlString.startsWith('//') || \n urlString.startsWith('/') ||\n urlString.startsWith('#')) {\n return urlString;\n }\n \n // Convert HTTP to HTTPS\n if (urlString.startsWith('http://')) {\n return urlString.replace(/^http:\\/\\//, 'https://');\n }\n \n // Already HTTPS or other protocol\n return urlString;\n}\n\n// Template helper function\napp.locals.secureUrl = ensureHttps;\n\n// Example route with mixed content fixed\napp.get('/about', (req, res) => {\n // BAD - Contains HTTP URL (mixed content)\n // const externalLink = 'http://www.pleasedontmixhttpwithhttps.com';\n \n // GOOD - Uses HTTPS\n const externalLink = 'https://www.pleasedontmixhttpwithhttps.com';\n \n // Or use protocol-relative URL (automatically matches page protocol)\n // const externalLink = '//www.pleasedontmixhttpwithhttps.com';\n \n res.render('about', {\n externalLink: ensureHttps(externalLink),\n imageUrl: ensureHttps('https://example.com/image.jpg'),\n scriptUrl: ensureHttps('https://cdn.example.com/script.js')\n });\n});\n\n// Content validation function for user-generated content\nfunction sanitizeUserContent(htmlContent) {\n const $ = cheerio.load(htmlContent);\n \n // Fix all HTTP URLs to HTTPS\n $('img[src], script[src], link[href], a[href]').each((i, elem) => {\n const tagName = elem.tagName.toLowerCase();\n const attr = tagName === 'a' || tagName === 'link' ? 'href' : 'src';\n const url = $(elem).attr(attr);\n \n if (url && url.startsWith('http://')) {\n $(elem).attr(attr, ensureHttps(url));\n }\n });\n \n return $.html();\n}\n\n// API endpoint example\napp.post('/api/content', (req, res) => {\n let content = req.body.content;\n \n // Sanitize content to remove mixed content\n content = sanitizeUserContent(content);\n \n // Save sanitized content\n res.json({ content, message: 'Content saved with secure URLs' });\n});\n\n// ============================================\n// AUTOMATED MIXED CONTENT DETECTION SCRIPT\n// scripts/check-mixed-content.js\n// ============================================\n\n/\*\nconst fs = require('fs');\nconst path = require('path');\nconst glob = require('glob');\nconst cheerio = require('cheerio');\n\nfunction checkFileForMixedContent(filePath) {\n const content = fs.readFileSync(filePath, 'utf8');\n const issues = [];\n \n // Check for HTTP URLs in various formats\n const httpRegex = /http:\\/\\/[^\\s\"'<>)]+/gi;\n const matches = content.match(httpRegex) || [];\n \n matches.forEach(match => {\n // Skip localhost and development URLs\n if (!match.includes('localhost') && !match.includes('127.0.0.1')) {\n const lineNumber = content.substring(0, content.indexOf(match)).split('\\n').length;\n issues.push({\n file: filePath,\n line: lineNumber,\n url: match\n });\n }\n });\n \n // Additional check for HTML files\n if (filePath.endsWith('.html') || filePath.endsWith('.ejs')) {\n try {\n const $ = cheerio.load(content);\n \n $('img[src^=\"http://\"], script[src^=\"http://\"], link[href^=\"http://\"], a[href^=\"http://\"]').each((i, elem) => {\n const url = $(elem).attr('src') || $(elem).attr('href');\n if (url && !url.includes('localhost')) {\n issues.push({\n file: filePath,\n element: elem.tagName,\n url: url\n });\n }\n });\n } catch (err) {\n // Skip if not valid HTML\n }\n }\n \n return issues;\n}\n\n// Scan project files\nconst files = glob.sync('views/\*\*\\/\*.{html,ejs,pug}');\nconst allIssues = [];\n\nfiles.forEach(file => {\n const issues = checkFileForMixedContent(file);\n allIssues.push(...issues);\n});\n\nif (allIssues.length > 0) {\n console.error('\\n❌ Mixed Content Issues Detected:\\n');\n allIssues.forEach(issue => {\n console.error(`${issue.file}:${issue.line || '?'}`);\n console.error(` URL: ${issue.url}`);\n if (issue.element) console.error(` Element: ${issue.element}`);\n console.error('');\n });\n console.error(`Total: ${allIssues.length} mixed content issue(s) found.\\n`);\n process.exit(1);\n} else {\n console.log('✅ No mixed content issues detected.\\n');\n}\n\*/\n\n// ============================================\n// TEMPLATE EXAMPLES\n// ============================================\n\n/\*\nEJS Template (views/about.ejs):\n\nBAD - Mixed Content:\n<!DOCTYPE html>\n<html>\n<head>\n <title>About</title>\n <link rel=\"stylesheet\" href=\"http://example.com/style.css\">\n</head>\n<body>\n <img src=\"http://example.com/image.jpg\">\n <a href=\"http://www.pleasedontmixhttpwithhttps.com\">External Link</a>\n <script src=\"http://cdn.example.com/script.js\"></script>\n</body>\n</html>\n\nGOOD - All HTTPS:\n<!DOCTYPE html>\n<html>\n<head>\n <title>About</title>\n <link rel=\"stylesheet\" href=\"https://example.com/style.css\">\n</head>\n<body>\n <img src=\"https://example.com/image.jpg\" alt=\"Description\">\n <a href=\"<%= secureUrl(externalLink) %>\">External Link</a>\n <script src=\"https://cdn.example.com/script.js\"></script>\n</body>\n</html>\n\nBEST - Protocol-relative URLs (matches page protocol):\n<!DOCTYPE html>\n<html>\n<head>\n <title>About</title>\n <link rel=\"stylesheet\" href=\"//example.com/style.css\">\n</head>\n<body>\n <img src=\"//example.com/image.jpg\" alt=\"Description\">\n <a href=\"//www.pleasedontmixhttpwithhttps.com\">External Link</a>\n <script src=\"//cdn.example.com/script.js\"></script>\n</body>\n</html>\n\*/\n\n// ============================================\n// NGINX CONFIGURATION\n// ============================================\n\n/\*\n# nginx.conf - Force HTTPS and prevent mixed content\n\nserver {\n listen 80;\n server\_name pentest-ground.com;\n \n # Redirect all HTTP to HTTPS\n return 301 https://$server\_name$request\_uri;\n}\n\nserver {\n listen 81 ssl http2;\n server\_name pentest-ground.com;\n \n ssl\_certificate /path/to/cert.pem;\n ssl\_certificate\_key /path/to/key.pem;\n \n # Security headers\n add\_header Strict-Transport-Security \"max-age=31536000; includeSubDomains; preload\" always;\n add\_header Content-Security-Policy \"default-src 'self' https:; upgrade-insecure-requests; block-all-mixed-content\" always;\n add\_header X-Content-Type-Options \"nosniff\" always;\n add\_header X-Frame-Options \"SAMEORIGIN\" always;\n \n location / {\n proxy\_pass http://localhost:3000;\n proxy\_set\_header X-Forwarded-Proto $scheme;\n proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;\n proxy\_set\_header Host $host;\n }\n}\n\*/\n\n// ============================================\n// DEPLOYMENT CHECKLIST\n// ============================================\n\n/\*\nBefore deploying:\n\n✓ Replace all http:// URLs with https://\n✓ Use protocol-relative URLs (//) where appropriate\n✓ Verify all third-party resources support HTTPS\n✓ Enable Content-Security-Policy with upgrade-insecure-requests\n✓ Enable HSTS header\n✓ Test site with browser console open for mixed content warnings\n✓ Run automated mixed content scanner\n✓ Check all external links and resources\n✓ Update CDN and third-party library URLs to HTTPS\n✓ Verify SSL certificate is valid and properly configured\n✓ Test with multiple browsers\n\*/\n\nconst PORT = process.env.PORT || 3000;\napp.listen(PORT, () => {\n console.log(`Server running on port ${PORT}`);\n console.log('✓ Mixed content protection enabled');\n console.log('✓ HTTPS enforcement active');\n});\n\nmodule.exports = app;"

}

}  
  
{

"title": "Absence of Anti-CSRF Tokens",

"description": "No Anti-CSRF tokens were found in a HTML submission form.\nA cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.\n\nCSRF attacks are effective in a number of situations, including:\n    \* The victim has an active session on the target site.\n    \* The victim is authenticated via HTTP auth on the target site.\n    \* The victim is on the same local network as the target site.\n\nCSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.",

"simplified\_description": "Imagine you're logged into your bank's website. A malicious site could trick you into clicking a hidden link that tells your bank to transfer money, and because you're already logged in, the bank trusts the request and does it. This attack works because the website doesn't have a way to verify that the request to perform an action (like a transfer) actually came from you clicking a button on their real website.",

"solution": "Phase: Architecture and Design\nUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.\nFor example, use anti-CSRF packages such as the OWASP CSRFGuard.\n\nPhase: Implementation\nEnsure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.      \n\nPhase: Architecture and Design\nGenerate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).\nNote that this can be bypassed using XSS.\n\nIdentify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.\nNote that this can be bypassed using XSS.\n\nUse the ESAPI Session Management control.\nThis control includes a component for CSRF.\n\nDo not use the GET method for any request that triggers a state change.\n\nPhase: Implementation\nCheck the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.",

"simplified\_solution": "To fix this, implement anti-CSRF tokens. This involves generating a unique, secret token for each user session and embedding it in all forms, so the server can verify that submissions are legitimate and not from a malicious site.",

"code\_solution\_sample": {

"solution\_description": "This code uses the 'csurf' middleware in a Node.js/Express application to implement the Synchronizer Token Pattern. It generates a unique CSRF token for each user session, embeds it in a hidden input field in forms, and then verifies the token on every POST request to ensure the request originated from the application itself, thus preventing CSRF attacks.",

"affected\_files": "This is typically found in application middleware configuration (e.g., app.js or server.js), form rendering templates (e.g., .ejs, .pug, or JSX files), and base controller classes where security features are globally applied.",

"code": "// Node.js/Express - CSRF Protection using 'csurf' middleware\nconst express = require('express');\nconst session = require('express-session');\nconst cookieParser = require('cookie-parser');\nconst csrf = require('csurf');\nconst app = express();\n\n// 1. Setup session and cookie parser middleware (csurf requires it)\napp.use(cookieParser());\napp.use(session({\n secret: 'a-very-strong-and-long-secret-key',\n resave: false,\n saveUninitialized: true,\n cookie: { secure: true, httpOnly: true, sameSite: 'strict' }\n}));\n\n// 2. Use csurf middleware to generate and validate tokens\nconst csrfProtection = csrf({ cookie: true });\napp.use(csrfProtection);\n\n// 3. Middleware to make the token available to templates\napp.use((req, res, next) => {\n res.locals.csrfToken = req.csrfToken();\n next();\n});\n\n// Middleware for parsing form data\napp.use(express.urlencoded({ extended: false }));\n\n// Example: Render a form with the CSRF token\napp.get('/form', (req, res) => {\n // The 'res.locals.csrfToken' is passed to the view engine\n res.send(`\n <h1>CSRF Protected Form</h1>\n <form action=\"/process\" method=\"POST\">\n <input type=\"hidden\" name=\"\_csrf\" value=\"${res.locals.csrfToken}\">\n <label for=\"data\">Enter Data:</label>\n <input type=\"text\" id=\"data\" name=\"data\">\n <button type=\"submit\">Submit</button>\n </form>\n `);\n});\n\n// Example: Process the form submission\n// The csurf middleware will automatically validate the token. \n// If the token is invalid, it will throw an error.\napp.post('/process', (req, res) => {\n console.log('Data processed successfully:', req.body.data);\n res.send('Form submitted successfully!');\n});\n\n// Error handling middleware for invalid CSRF tokens\napp.use((err, req, res, next) => {\n if (err.code !== 'EBADCSRFTOKEN') return next(err);\n\n // Handle CSRF token errors here\n res.status(403).send('Invalid CSRF token: form tampered with or session expired.');\n});\n\nconst PORT = 3000;\napp.listen(PORT, () => {\n console.log(`Server running on http://localhost:${PORT}`);\n});"

}

}