2024.08.08 ooya

Joomla パフォーマンスチューニング II: 基本設定

https://magazine.joomla.org/all-issues/december-2021/joomla-performance-tuning-ii-basic-settings

2021 年 12 月 20 日

このシリーズの第 1 部では、哲学的および実用的な理由からサイトのパフォーマンスの調整を行う必要がある理由と、どこから始めるべきかについて説明しました。 その投稿は、必然的に一般的な内容になってしまいました。 このシリーズの第 2 部では、Joomla で実行してパフォーマンスをある程度向上させる基本的な操作をいくつか詳しく説明します。

基本的なシステム設定

サイトを構築するとき、デザインと機能に気を取られすぎて、非常に基本的でかなり単純なシステム設定がサイトのパフォーマンスに大きな影響を与えることを忘れがちです。 サイトを配信する前にグローバル構成でいくつかの簡単なスイッチを変更し、いくつかの簡単なサーバー チェックを行うだけで、大きな違いが生まれます。

キャッシュ

サイトのサーバー側で費やされる時間のほとんどは、訪問者に表示されるページの構築に関係しています。 WordPress とは異なり、Joomla にはキャッシュ システムが組み込まれています。 Joomla 1.0 および 1.5 の劣悪なキャッシュ エクスペリエンスに慣れていたため、人々は Joomla をあまり評価していないように思います。 それは 10 ～ 15 年前のことです。

サイトのグローバル設定に移動し、「システム」 タブから 「キャッシュ」 を 「**オン - プログレッシブ**

キャッシュ」に設定します（＊）。 プログレッシブ キャッシュ オプションは、Joomla 組み込みキャッシュのより優れた実装であり、ページの構築に使用される**各エクステンション**の出力が個別にキャッシュされるようにします。 リクエストが届くと、ページは可能な限りキャッシュされたコンテンツからまとめられます。 これにより、事前に構築された最適化されていないテンプレートで失われるパフォーマンスを軽減することもできます。 これにより、ログインしていない公開ページが確実に高速化されます。 これは、サイトの**検索エンジン ランキング**に最も関連しています。

＊https://docs.joomla.org/Cache#Progressive_Caching

キャッシュ バックエンド（注：キャッシュにハンドラか？）は、ほとんどのサイトでは、ファイル キャッシュを使用しても**問題ありません**。 ファイル キャッシュは、まともな商用共有ホストまたは仮想ホストで実行されている memcached や Redis と同等のパフォーマンスを発揮します。（＊） メモリ使用量ははるかに少なく、したがって実行コストもはるかに低くなります。 技術志向の人ほど「異端だ!」と叫びます。 私も、ある程度はあなたの意見に賛成です。 本当に大規模なサイトや非常に混雑したサイトの場合は、専用の memcached または Redis サーバーをキャッシュ バックエンドとして使用するのが合理的です。 その方が高速になります。 この記事を読んでいる方は、実際にはそのようなサイトではなく、もっとありふれたサイトの高速化を検討している可能性が高いでしょう。 私のビジネス サイトも、毎月数十万から数万のユニーク ビジター数があるにもかかわらず、このカテゴリに分類されます。 専用キャッシュ サーバーを使用したキャッシュからメリットを得られるサイト規模がおわかりいただけると思います。

＊注：ウェブ開発で最も人気のあるインメモリデータストレージソリューションに、Redis と Memcached の 2 つがあります。Memchached はシンプルさと高いパフォーマンスを重視しているため、単純なキャッシュが必要なプロジェクトには有用です。しかし、より高度なキャッシュ機能やそれ以外の機能が必要な場合は、Redis の使用を検討することが有用。 例えば、https://kinsta.com/jp/blog/memcached-vs-redis/

HTML 圧縮 （注：サーバ タブへ）

Joomla で最も見落とされがちなオプションを競うコンテストがあったら、**Gzip ページ圧縮**が断然勝ちます。 まだ有効にしていない場合は、ぜひ**有効に**してください。

このオプションにより、サイトからブラウザに送信される HTML コンテンツが GZip（別名「deflate」）アルゴリズムを使用して圧縮されます。 これにより、クライアントに転送されるデータの合計サイズが大幅に削減されます。 データ転送で節約される時間は、サイトのパフォーマンスに大きな影響を与えます。

サイトの速度は低下しませんか？ いいえ、それはありません。 Joomla によって生成される HTML ページのサイズは数十キロバイトの範囲です。 そのサイズの約 3 分の 1 から半分に圧縮するには、数分の 1 ミリ秒かかります。 ホストと訪問者の間の一般的な転送速度では、これは数ミ

リ秒の時間の節約になります。 この場合、失われる時間よりも 2 桁から 3 桁多くの時間が得られます。

JavaScript と CSS の圧縮

多くのテンプレートやサードパーティのプラグインは、静的ファイル（JavaScript と CSS）をオンザフライ（on-the-fly）（＊）で圧縮することで時間を節約できると主張しています。 このような機能は**使用しないこと**を強くお勧めします。 静的ファイルを圧縮すると転送時間は節約できますが、最終的なパフォーマンスは低下します。

*注）オンザフライ：記録可能な光学メディアへデータを書き込む方式の一つで、あらかじめハードディスクなどにディスクイメージを作成せず、いきなり直接ディスクへ書き込むこと。

この直感に反する結果の理由は、サーバーが静的コンテンツと動的コンテンツを配信する方法について説明する必要があります。 適切に設定された Web サーバーは、頻繁に使用される静的コンテンツをメモリにキャッシュします。 さらに、**ファイルのメモリ マッピング**（＊）などのオペレーティングシステムの高度な機能を使用します。 これにより、静的コンテンツが非常に高速に配信されます。

＊https://httpd.apache.org/docs/2.4/mod/core.html#enablemmap

PHP スクリプトを使用して静的ファイルを圧縮する場合、Web サーバーはリクエストを PHP 実行ファイルに渡す必要があります。 最良のシナリオ（PHP FastCGI Process Manager、別名 **PHP-FPM**、十分な大きさのプロセス プールと PHP OPcache が有効）でも、プロセス間通信と PHP パーサーの状態のリセットに時間がかかります。 スクリプトが変更されていないことを確認し、プリコンパイルされたバイナリ表現をロードして解釈し、PHP バイナリで実行し、静的ファイルを開いてその内容を圧縮し、Apache に送信してクライアントに配信する必要があります。 これらすべてに数十ミリ秒かかります。 数百キロバイトを超えるサイズのファイルを圧縮するのでない限り、失われた時間は、圧縮された小さなファイルを配信することで得られる時間より 1 桁または 2 桁もずっと大きくなります。 したがって、これは**純損失**です。

これを Web サーバー自体で実行することを強くお勧めします。 Apache を使用している場合は、**.htaccess ファイル**に次のコードを追加できます。

```
<IfModule mod_deflate.c>
  AddOutputFilterByType DEFLATE text/plain text/xml text/css application/xml
application/xhtml+xml application/rss+xml application/javascript application/x-javascript
image/svg+xml
</IfModule>

<IfModule mod_gzip.c>
 mod_gzip_on Yes
 mod_gzip_dechunk Yes
 mod_gzip_keep_workfiles No
 mod_gzip_can_negotiate Yes
 mod_gzip_add_header_count Yes
 mod_gzip_send_vary Yes
 mod_gzip_min_http 1000
 mod_gzip_minimum_file_size 300
 mod_gzip_maximum_file_size 512000
 mod_gzip_maximum_inmem_size 60000
 mod_gzip_handle_methods GET
 mod_gzip_item_include file ¥.(html?|txt|css|js|php|pl|xml|rb|py|svg|scgz)$
 mod_gzip_item_include mime ^text/plain$
 mod_gzip_item_include mime ^text/xml$
 mod_gzip_item_include mime ^text/css$
 mod_gzip_item_include mime ^application/xml$
 mod_gzip_item_include mime ^application/xhtml+xml$
 mod_gzip_item_include mime ^application/rss+xml$
 mod_gzip_item_include mime ^application/javascript$
 mod_gzip_item_include mime ^application/x-javascript$
 mod_gzip_item_include mime ^image/svg+xml$
 mod_gzip_item_exclude rspheader ^Content-Encoding:.*gzip.*
 mod_gzip_item_include handler ^cgi-script$
 mod_gzip_item_include handler ^server-status$
 mod_gzip_item_include handler ^server-info$
 mod_gzip_item_include handler ^application/x-httpd-php
 mod_gzip_item_exclude mime ^image/.*
</IfModule>
```

あなたのウェブ サーバーは、静的メディア ファイルの圧縮がはるかに高速になります。 圧縮されたファイルをメモリにキャッシュしておき、次回の配信を高速化できます。

Joomla 4 では、静的ファイルを事前に GZip で圧縮できるようにすることで、さらに高速化しました。 これにより、ウェブ サーバーがオンデマンドで圧縮するのではなく、事前に圧縮されたファイルが配信されます。 仕組みは次のとおりです。

たとえば、JavaScript ファイル media/com_example/js/something.min.js があるとします。 これを GZip で media/com_example/js/something.min.js.gz に圧縮します。 ブラウザーがファイル media/com_example/js/something.min.js を要求すると、ウェブ サーバーは Accepts HTTP ヘッダーをチェックして、GZip で圧縮されたリソースをサポートしているかどうかを確認します。 サポートしている場合は、通常の圧縮されていない media/com_example/js/something.min.js ファイルではなく、media/com_example/js/something.min.js.gz ファイルを配信します。

そのための前提条件は、Joomla に同梱されている htaccess.txt ファイルの名前を .htaccess に変更することです。 または、独自の .htaccess ファイルを管理している場合は、ファイルに次のコードが含まれていることを確認してください。

```
## These directives are only enabled if the Apache mod_headers module is enabled.
## This section will check if a .gz file exists and if so will stream it
##        directly or fallback to gzip any asset on the fly
## If your site starts to look strange after enabling this, and you see
##        ERR_CONTENT_DECODING_FAILED in your browser console network tab,
##        then your server is already gzipping css and js files and you don't need this
##        block enabled in your .htaccess
<IfModule mod_headers.c>
            # Serve gzip compressed CSS files if they exist
            # and the client accepts gzip.
            RewriteCond "%{HTTP:Accept-encoding}" "gzip"
            RewriteCond "%{REQUEST_FILENAME}¥.gz" -s
            RewriteRule "^(.*)¥.css" "$1¥.css¥.gz" [QSA]

            # Serve gzip compressed JS files if they exist
            # and the client accepts gzip.
            RewriteCond "%{HTTP:Accept-encoding}" "gzip"
            RewriteCond "%{REQUEST_FILENAME}¥.gz" -s
            RewriteRule "^(.*)¥.js" "$1¥.js¥.gz" [QSA]
```

```
        # Serve gzip compressed JS files if they exist
        # and the client accepts gzip.
        RewriteCond "%{HTTP:Accept-encoding}" "gzip"
        RewriteCond "%{REQUEST_FILENAME}¥.gz" -s
        RewriteRule "^(.*)¥.js" "$1¥.js¥.gz" [QSA]

        # Serve correct content types, and prevent mod_deflate double gzip.
        RewriteRule "¥.css¥.gz$" "-" [T=text/css,E=no-gzip:1]
        RewriteRule "¥.js¥.gz$" "-" [T=text/javascript,E=no-gzip:1]

        <FilesMatch "(¥.js¥.gz|¥.css¥.gz)$">
                # Serve correct encoding type.
                Header append Content-Encoding gzip

                # Force proxies to cache gzipped &
                # non-gzipped css/js files separately.
                Header append Vary Accept-Encoding
         </FilesMatch>
</IfModule>
```

Joomla 4 は、配布時にすべての静的 JavaScript および CSS ファイルに対して事前に圧縮された .gz ファイルを提供します。 この .htaccess トリックを有効にすると、最小限の労力でサイトがさらに高速化されます。 すばらしいと思いませんか?

静的メディアのキャッシュ

圧縮によって静的メディアのサイズを縮小すれば、戦いの半分は終わります。 これは主に、初めてサイトを訪れた訪問者にとって重要です。 誰かがサイトに戻ってきたときに、ネットワークにまったくアクセスせずに、ブラウザのキャッシュから静的メディアを提供するのは理にかなっています。 Apache を使用している場合は、**.htaccess ファイル**で次のコードを使用できます。

```
<IfModule mod_expires.c>
 # Enable expiration control
 ExpiresActive On

# CSS and JS expiration:
 ExpiresByType text/css "now plus 1 year"
 ExpiresByType application/javascript "now plus 1 year"
 ExpiresByType application/x-javascript "now plus 1 year"

# Image files expiration: 1 month after request
 ExpiresByType image/bmp "now plus 1 year"
 ExpiresByType image/gif "now plus 1 year"
 ExpiresByType image/jpeg "now plus 1 month"
 ExpiresByType image/jp2 "now plus 1 month"
 ExpiresByType image/pipeg "now plus 1 month"
 ExpiresByType image/png "now plus 1 month"
 ExpiresByType image/svg+xml "now plus 1 month"
 ExpiresByType image/tiff "now plus 1 month"
 ExpiresByType image/vnd.microsoft.icon "now plus 1 month"
 ExpiresByType image/x-icon "now plus 1 month"
 ExpiresByType image/ico "now plus 1 month"
 ExpiresByType image/icon "now plus 1 month"
 ExpiresByType image/webp "now plus 1 month"
 ExpiresByType text/ico "now plus 1 month"
 ExpiresByType application/ico "now plus 1 month"
 ExpiresByType image/vnd.wap.wbmp "now plus 1 month"
 ExpiresByType application/vnd.wap.wbxml "now plus 1 month"
 ExpiresByType application/smil "now plus 1 month"

 # Font files expiration: 1 week after request
 ExpiresByType application/vnd.ms-fontobject "now plus 1 week"
 ExpiresByType application/x-font-ttf "now plus 1 week"
 ExpiresByType application/x-font-opentype "now plus 1 week"
 ExpiresByType application/x-font-woff "now plus 1 week"
 ExpiresByType font/woff2 "now plus 1 week"
 ExpiresByType image/svg+xml "now plus 1 week"

# Audio files expiration: 1 month after request
 ExpiresByType audio/ogg "now plus 1 month"
 ExpiresByType application/ogg "now plus 1 month"
 ExpiresByType audio/basic "now plus 1 month"
 ExpiresByType audio/mid "now plus 1 month"
 ExpiresByType audio/midi "now plus 1 month"
```

```
ExpiresByType audio/mpeg "now plus 1 month"
 ExpiresByType audio/mp3 "now plus 1 month"
 ExpiresByType audio/x-aiff "now plus 1 month"
 ExpiresByType audio/x-mpegurl "now plus 1 month"
 ExpiresByType audio/x-pn-realaudio "now plus 1 month"
 ExpiresByType audio/x-wav "now plus 1 month"

# Movie files expiration: 1 month after request
 ExpiresByType application/x-shockwave-flash "now plus 1 month"
 ExpiresByType x-world/x-vrml "now plus 1 month"
 ExpiresByType video/x-msvideo "now plus 1 month"
 ExpiresByType video/mpeg "now plus 1 month"
 ExpiresByType video/mp4 "now plus 1 month"
 ExpiresByType video/quicktime "now plus 1 month"
 ExpiresByType video/x-la-asf "now plus 1 month"
 ExpiresByType video/x-ms-asf "now plus 1 month"
</IfModule>
```

Joomla やサードパーティのエクステンションを更新するとどうなるか、という疑問は当然のものです。静的ファイル（JavaScript、CSS、画像など）は更新の一部として変更されます。 ブラウザーが古いファイルを使用することは望ましくありません。 サイトの見た目がおかしくなるのはよくても、最悪の場合、訪問者にとってサイトが壊れてしまいます。 ここで、クエリ パラメータを使用したメディア バージョン管理が役立ちます。サイトのソース コードを見ると、次のような行があります。

```
<link
href="/media/plg_system_webauthn/css/button.min.css?f15d039055248502c1a41bc
99a31c0f3" rel="stylesheet">
```

?f15d039055248502c1a41bc99a31c0f3 はメディア バージョン管理クエリと呼ばれます。疑問符の後の部分は、静的ファイルが変更されるたびに変更され、重要ではありません。Joomla および適切に記述されたサードパーティのエクステンションは、CSS ファイルと JavaScript ファイルに対してこれを自動的に行います。 記事に画像やビデオなどの他の静的コンテンツを含める場合は、バージョン管理クエリを追加することを忘れないでください。 ?20211205111300（疑問符の後に年、月、日、時、分、秒が続く、つまりクエリを記述した時間）のような単純なもので十分です。

HTTPS と HSTS

HTTPS はサイトのセキュリティ保護に関係しているという誤解がよくあります。 HTTPS は高価で、速度が遅く、e コマースなどを行っていない限り、実際には必要ありません。 もう 1 つの誤解は、HTTPS によってサイトが遅くなるというものです。

これらの誤解は 1990 年代後半に生まれました。 20 年以上前には、明らかに誤りでした。

HTTPS は今日ではほぼ必須です。 HTTPS を使用しないと、サイトには安全でないことを示す大きな赤い警告が表示され、訪問者を怖がらせます。 検索エンジンによってペナルティが課せられます。 この 2 つの問題を解決するためだけでも、HTTPS を使用する必要があります。 貯金箱を壊す必要さえありません。 TLS 証明書は、**Let's Encrypt** のおかげで**無料**になりました。 ほとんどのホスティング コントロール パネルは Let's Encrypt と統合されているため、ホスティング コントロール パネルで無料の TLS 証明書をインストールして自動更新することができます。 ユーザー側でメンテナンスを行う必要はありません。 また、過去 10 年ほどの間にリリースされた最新の CPU には、使用する暗号化操作用のハードウェア アクセラレーションが搭載されているため、HTTPS は超高速です。

ついでに、グローバル構成で「サイト全体に HTTPS を強制」を設定することも忘れないでください。これにより、Joomla サイトは常に HTTPS 経由で配信され、ログインの安全性が高まります。 これを実行した後、HTTPS がサイトで問題なく機能することを確認したら、.htaccess に次のコードを追加します。

```
<IfModule mod_headers.c>
 Header always set Strict-Transport-Security "max-age=31536000" env=HTTPS
</IfModule>
```

これにより、**HSTS (HTTP Strict Transport Security)** と呼ばれる機能が有効になります。簡単に言うと、訪問者が何を指示したかに関係なく、ブラウザにサイトの HTTP バージョンに接続しようとしないように指示します。 これはブラウザ側で行われるため、**https:// プレフィックスなし**でアドレス バーにドメイン名を入力したり、http:// プレフィックスのリンクをクリックしたりする訪問者は、最初にプレーン HTTP バージョンにアクセスして Joomla によってリダイレクトされることなく、常にサイトの HTTPS バージョンにアクセスできます。 これは、モバイルや衛星インターネットなどの高遅延接続では特に高速です。

さらに最適化するには、サイトを **HSTS プリロード リスト（＊）**に送信します。

＊https://hstspreload.org/

HSTS は誰かが最初にサイトにアクセスした後にのみ機能しますが、サイトを HSTS プリロード リストに含めると、訪問者が最初にサイトにアクセスする前に、ブラウザは HSTS を使用してサイトを認識します。 したがって、ブラウザはプレーン HTTP 経由でサイトをロードしようとしません。 繰り返しますが、これは高遅延接続の時間を節約し、簡単かつ無料です。 気に入らない点はありません。

HTTP/2 サーバー プッシュ

以前、Joomla の高速化について話すとき、私は **HTTP/2 サーバー プッシュ（＊）**を有効にしてサイトを高速化する方法を教えていました。

＊https://en.wikipedia.org/wiki/HTTP/2_Server_Push

しかし、Google Chrome の開発者はすでにそのサポートを削除することを提案（＊）しており、HTTP/3 プロトコルにはまったく実装されないと述べています。

＊ https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLvmQUBY/m/vOWBKZGoAQAJ?pli=1

したがって、現時点での私のアドバイスは、気にしないことです。

続く

これは 5 部構成のシリーズの第 2 部です。 第 3 部: 静的メディアの最適化は、Joomla コミュニティ マガジンの 2022 年 1 月号で公開されます。

About the author

Nicholas K. Dionysopoulos

Nicholas は、機械エンジニアから Web 開発者に転身しました。2004 年に Joomla の前身である Mambo を見つけて以来、人生は一変しました。

Joomla の人気エクステンションを数多く作成し、Joomla コアに頻繁に貢献しています。現在は妻、娘、猫 2 匹とともにギリシャのアテネ沿岸に住んでいます。彼は、Joomla コミュニティのおかげでビジネスと家族を持つことができたと考えています。妻とは Joomla カンファレンスで出会ったのですから!

Joomla や拡張機能のコード、ドキュメント、記事を書いていないときは、メカニカル キーボードをいじったり、D&D をプレイしたり、SF シリーズを観たりしています。

コメント

Maurice Molenaar

Maurice Molenaar 2021 年 12 月 21 日火曜日 09:31

素晴らしい!

Nicholas、ありがとう。素晴らしい記事です!

George Ploumakis

George Ploumakis 2021 年 12 月 21 日火曜日 10:38

いつものように素晴らしい

Joomla コミュニティへの、素晴らしい、高レベル、重要、そして長きにわたる貢献に感謝します。

Alan N

Alan N 2022 年 4 月 20 日水曜日 07:29

必読の記事です

Nicholas、とても役に立つ記事です。ありがとう!

Pieter-Jan de Vries

Pieter-Jan de Vries 2022 年 10 月 15 日土曜日 10:52

HTML 圧縮は安全ではないのですか?

ほぼ 1 年前の記事のコメントを読む人がいるかどうかはわかりませんが、わかりません。

最近、動的に生成されたコンテンツの圧縮は安全ではなく、犯罪や侵害の攻撃に対して脆弱であるという情報に遭遇しました。この脆弱性は機密コンテンツを含むページにのみ適用されると思いますが、それでも、「HTML 圧縮」に関する文章に関連して、Nicholas がこの件について何か考えを持っているかどうか興味があります。

Nicholas K. Dionysopoulos

Nicholas K. Dionysopoulos 2022 年 10 月 15 日土曜日 13:56

ここであまり心配しすぎないようにしましょう

Wikipedia の過度に簡略化されたバージョンを読む代わりに、実際の CVE を読んでください

https://www.cve.org/CVERecord?id=CVE-2012-4929

これは、TLS 1.2 以前の実装の問題でした。 そのため、10 年前に報告され修正されたバグがある古いバージョンの TLS (バグ要件があるため、nginx は影響を受けなかったことに注意し

てください) と、非常に正確なタイミング測定で数千のリクエストを開始する中間者攻撃 (これにより、最新のブラウザーの JavaScript とブラウザー拡張機能が排除されます) が必要です。 BREACH には、9 年前のパッチ未適用の gzip ライブラリが必要です。 セキュリティのベストプラクティスを満たしておらず、10 年間更新されておらず、侵害されたネットワーク上にあるものを使用している場合、CRIME 攻撃と BREACH 攻撃は最小限の心配で済むと私は主張します。

いいえ、10 年前の TLS プロトコル実装で解決され、TLS 1.3 (2018 年以降の現在の TLS バージョン) で確実に解決されている問題を「軽減」するために、サーバー レベルで圧縮をオフにすることはしません。

疑心暗鬼になりたいなら、認証を解除した Bluetooth キーボードからのキーストロークを分析したり、シールドされていない (つまり、ほとんど、あるいはすべて) USB キーボード ケーブルの電磁場を測定したり、タイピング音を分析したり、赤外線写真を使用してキーのヒート マップを分析したり (どうやら、多くのキーボードで使用されている ABS キーは、PBT キーキャップよりも熱マッピングの分析がはるかに簡単のようです) する実証済みの攻撃や論文が発表されていることをご存知ですか? マウス入力にもほとんど同じことが当てはまります。オンライン会議中にメガネに映る画面の反射を分析して、約 600 x 400 ピクセルの解像度で画面を表示できます。 コンピューターの電力消費の 1 mW 未満の変動を使用して、何をしているのかを推測したり、エアギャップ コンピューターからデータを抜き出したりできます。 コンピュータに対するあらゆる攻撃の可能性を考えれば、元 CIA 長官が冗談で言った結論にたどり着くでしょう。 コンピュータは、細かく切り刻まれ、溶かされ、粉々にされ、秘密の場所に 6 フィート下に埋められたときに安全です。 しかし、私たち全員がコンピュータを使って極秘資料を扱っているわけではありません。 極秘資料が漏洩すれば、国家の安全保障に悪影響を及ぼしたり、人類を絶滅させる熱核戦争を引き起こしたりする可能性があります。 ですから、あり得ることとあり得ることとを比較して過剰反応する、アルミホイル帽子派に加わらないようにしましょう。

可能性とあり得ることといえば、セキュリティの第一のルールについて話しましょう。 すべてを昨日更新することです。 これらの攻撃はすべてよく知られており、サーバー ソフトウェアと一般的なソフトウェア ライブラリですでに軽減されています。 最新のパッチ レベルで現在メンテナンスされているソフトウェアを使用する最新のサーバー環境を使用している限り、安全に圧縮を使用できます。 既知の攻撃ベクトルから保護されているという意味で、かなり安全です。 確かに、まだ誰も発見していない、または少なくともまだ公表されていないセキュリティ上の問題があるかもしれませんが、攻撃が成功した後の事後分析に役立つ標準的なセキュリティ対策以外にできることは何もありません。 クライアントが、安全でない/侵害されたブラウザ拡張機能を使用せずに、メンテナンスされた最新のオペレーティングシステムで最新のブラウザを使用している限り、クライアント

側も安全です。 どちらかの側がまだ古いソフトウェアを使用している場合、いずれかまたは両方の側が非常に悪い日を迎えることになります。

Pieter-Jan de Vries　2022 年 10 月 15 日土曜日 15:39

訂正します

迅速かつ包括的な返信をありがとうございます。 大変助かりました。 正直に言うと、数日前に同僚が持ち出すまで、私はこのセキュリティ問題とされる問題について知りませんでした。 検索エンジンのランキングに悪影響を与える可能性があるという想定でしたが、今のところその証拠を見つけることができていません。 少なくとも、誰かが再びこの問題を持ち出した場合に何と言えばよいかはわかっていますが、この手の込んだレポートを暗記して再現するのはかなり難しいでしょう。

Nicholas K. Dionysopoulos

Nicholas K. Dionysopoulos 2022 年 10 月 15 日土曜日 21:17

簡単な答え

同僚にできる最も簡単な答えは、「これは 2012 年に発生した問題で、それ以降修正されています。 つまり、10 年以上前のことです。 もし私たちのサーバーが 10 年以上前のソフトウェアを実行しているのであれば、これは私たちにとっては最小限の心配事です。」

以上