Detect Hacked Files via CRON/PHP

By David K. "DK" Lynn
aka "dklynn"

As a Certified Ethical Hacker, I searched for a script which would help me to detect unauthorized file changes. I found a script (probably in the User Contributed Notes at php.net) which I modified to have working very nicely on my "test server" (Windows) as well as on my "production" server (Linux).

The logic is simple: "Build a database of hashed values for vulnerable files (those which hackers will modify to execute code on your server) and compare those values to the actual hashes on a regular basis and report added, changed and deleted files."

Obviously, the code to traverse a server's directory structure and provide hash values is far more complex than the statement above. I will go through the code for the production server.

Database Setup

For security, use a separate database for this which does not share access credentials with any other database. Use cPanel to create the new database and the new user with a strong password (I recommend a 16 character password generated by strongpasswordgenerator.com) and an innocuous name like baseline. Then use PHPMyAdmin's SQL to create two tables:

```
CREATE TABLE baseline (
        file_path VARCHAR(200) NOT NULL,
        file_hash CHAR(40) NOT NULL,
        acct VARCHAR(40) NOT NULL
        PRIMARY KEY (file_path)
);

CREATE TABLE tested (
        tested DATETIME NOT NULL,
        account VARCHAR(40) NOT NULL
        PRIMARY KEY (tested)
);
```

The first table, "baseline," contains a large field for your path/to/filenames, a fixed field for the file_hash (40 characters are required for SHA1) and acct to allow me to monitor accounts or domains separately. Set the file_path as the Primary Key.

The "tested" table will hold the DATETIME of every scan and the account is the same as baseline's acct field so it will allow you to scan various accounts or domains and keep their data separate.

Initialize the PHP File:

First, DEFINE several constants

- PATH is the physical path to the start of your scan which is usually the DocumentRoot. Just remember not to use Windows' backslashes because both Apache and PHP will be looking for forward slashes.
- Database access constants SERVER ('localhost'), USER, PASSWORD and DATABASE.

and several variables

- An array of the file extensions to examine. Because not all files are executable on the server, I only scan .php, .htm, .html and .js files and these need to be specified in an array. Note than an empty array will force ALL files to be scanned (best for security but uses the most server resources).
- Directories to exclude. Iif you have a directory containing malware, shame on you! In any event, if you need to exclude a directory for any reason, you have the opportunity to list them in an array. Don't omit any directories just because you only stored images or pdf files, though, there because a hacker can put his files in there, too!
- Initialize the variables you're about to use: The $file array as an empty array(), the $report string as an empty string and the $acct string (use the account/acct name from your database tables) need to be initialized.

Let's get started!

```
<?php
//	initialize
//	extensions to fetch, an empty array will return all extensions which is best for real security
$ext = array("php","html","js");
//	directories to ignore, an empty array will check all directories
$skip = array("protected");

//	use define statements or enter values directly in the mysqli_connect
define('SERVER','localhost');
define('USER','your user name');
define('PASS','your password');
define('DATABASE','database name');

//	get database handle
$db = mysqli_connect(SERVER,USER,PASS,DATABASE);

$dir = new RecursiveDirectoryIterator(PATH);
$iter = new RecursiveIteratorIterator($dir);
while ($iter->valid())
{
	//	skip unwanted directories
	if (!$iter->isDot() && !in_array($iter->getSubPath(), $skip))
	{
		//	get specific file extensions
		if (!empty($ext))
```

```
                {
                        //      PHP 5.3.4: if (in_array($iter->getExtension(), $ext))
                        if (in_array(pathinfo($iter->key(), PATHINFO_EXTENSION), $ext))
                        {
                                $files[$iter->key()] = hash_file("sha1", $iter->key());
                        }
                } else {
                        //      ignore file extensions
                        $files[$iter->key()] = hash_file("sha1", $iter->key());
                }
        }
        $iter->next();
}
```

What we've just done is use the RecursiveIteratorIterator() function (a function used to iterate through recursive iterators) on the directory ($dir) as it iterates through the directory structure. The first thing it does is check whether a directory has been banned from the iteration then branch depending upon whether file extensions had been specified. The result is a two-dimensional matrix of files, $files), with path/name.ext as the index and corresponding SHA1 hash value.

I'll note here that the commented echo statements were used on my Windows test server without linking to the SMTP server but you'll need to uncomment them if you need to verify the correct functionality.

The file count can be provided immediately by the files array:

```
$report .= "Files has " . count($files) . " records.\r\n";
```

The output, whether to your test monitor or e-mail, has just been given it's first non-empty value: The hashed file count.

Last Hash Scan

The next thing to do is fetch the data/time the last hash scan was accomplished and get the stored path/file and hash set from the database.

```
$results = mysqli_query($db,"SELECT tested FROM tested WHERE acct = '$acct'
        ORDER BY tested DESC LIMIT 1");
if ($results)
{
        while($result=mysqli_fetch_array($results))
        {
                $tested = $result['tested'];
        }
$report .= "Last tested $tested.\r\n";
}
```

Compare Hashed Files with Database Records

So far, we've only learned the current file count and datetime of the last scan. The value we're looking for is to identify the changed files, i.e., those added, changed or deleted. Let's create an array of the differences.

```
//	identify differences
if (!empty($files))
{
	$result = mysqli_query($db,"SELECT * FROM baseline");
	if (!empty($result))
	{
		foreach ($result as $value)
		{
			$baseline[$value["file_path"]] = $value["file_hash"];
		}
		$diffs = array_diff_assoc($files, $baseline);
		unset($baseline);
	}
}

//	sort differences into Deleted, Altered and Added arrays
if (!empty($files))
{
	$results = mysqli_query($db,"SELECT file_path, file_hash FROM baseline WHERE acct =
'$acct'");
	if (!empty($results))
	{
		$baseline = array();	//	from database
		$diffs = array();	//	differences between $files and $baseline
					//	$files is current array of file_path => file_hash
		while ($value = mysqli_fetch_array($results))
		{
			if (!array_key_exists($value["file_path"], $files))
			{
				//	Deleted files
				$diffs["Deleted"][$value["file_path"]] = $value["file_path"];
				$baseline[$value["file_path"]] = $value["file_hash"];
			} else {
				//	Altered files
				if ($files[$value["file_path"]] <> $value["file_hash"])
				{
					$diffs["Altered"][$value["file_path"]] = $value["file_path"];
					$baseline[$value["file_path"]] = $value["file_path"];
				} else {
					//	Unchanged files
					$baseline[$value["file_path"]] = $value["file_hash"];
				}
			}
		}
		if (count($baseline) < count($files))
		{
			//	Added files
			$diffs["Added"] = array_diff_assoc($files, $baseline);
		}
```

```
                unset($baseline);
        }
}
```

When completed, the $diffs array will either be empty or it will contain any discrepancies found in the multi-dimensional array sorted by Deleted, Altered and Added along with the path/file and associated hash pairs for each.

E-Mail Results

You will need to add the discrepancies to the report and e-mail.

```
//      display discrepancies
if (!empty($diffs)) {
$report .= "The following discrepancies were found:\r\n\r\n";
foreach ($diffs as $status => $affected)
{
        if (is_array($affected) && !empty($affected))
        {
                ($test) ? echo "<li>" . $status . "</li>" : $report .= "* $status *\r\n\r\n";
                ($test) ? echo "<ol>" : ";
                foreach($affected as $path => $hash) $report .= " • $path\r\n";
        }
}
} else {
        $report .= "File structure is intact.\r\n";
}

$mailed = mail('you@example.com', $acct . ' Integrity Monitor Report',$report);
```

Update the Database

You're not finished yet!

```
//      update database
//      clear old records
mysqli_query($db,"DELETE FROM baseline WHERE acct = '$acct'");

//      insert updated records
foreach ($files as $path => $hash)
{
        mysqli_query($db,"INSERT INTO baseline (file_path, file_hash, acct)
                VALUES ('$path','$hash', '$acct')");
}

mysqli_query($db,"INSERT INTO tested (tested, acct) VALUES (NOW(), '$acct')");

mysqli_close($db);
?>
```

On the first pass, there will be nothing in the database's baseline table and ALL files will display as Added so don't be alarmed.

Now that you have the code, where do you upload it? Don't even consider placing this code in your webspace (under the DocumentRoot) as that will mean that anyone can access your file and delete the saved information to invalidate your hash scans. For simplicity, put it in the same directory of your account which holds public_html (or similar) directory.

Activate

Now that you have the code, you need to have it activated on a regular basis. That's where the CRON function of the server excels! Simply use your cPanel to create a new CRON job, set the time in the middle of the night when your server should be nearly idle (you don't want to interfere with or delay visitors' activities which also means you should limit yourself to a single scan per day) and use the following directive:

```
/usr/local/bin/php -q /home/account/hashscan.php
```

where `/usr/local/bin/php` is the location of the server's PHP executable and `/home/account/hashscan.php` is the path to your hashscan.php script (or whatever name you gave it).

Wrap-Up

We have created a new database with two tables, one to hold the dates and one to hold the baseline hashes. We have initiated every scan by identifying the file types (by extension) that we need to track and identified the start point (DocumentRoot) for our scan. We've scanned the files avoiding the unwanted directories and compared the hashes against the baseline in the database. Closing the process, we've updated the database tables and either displayed (on a test server) or e-mailed (from the production server) the results. Our CRON job will then activate your hash scan on a regular basis.

This ZIP file contains the above CreateTable.sql, hashscan.php and CRON.txt files.

This is but one part of securing your website, though, as it will only inform you of changes to the types of files you've specified. Before you get this far, you must ensure that your files are malware free (maldet scans established by your host can do this but be sure that you keep a clean master copy off-line), ensured that no one but you can upload via FTP (by using VERY strong passwords) and keeping "canned apps" up to date (because their patches are closing vulnerabilities found and exploited by hackers and their legions of "script kiddies").

In summary, BE PARANOID! There may be no one out to get you but there *are* those out for "kicks" who are looking for easy prey. Your objective is to avoid that classification.