Analysis of PHP Obfuscation Software

December 7, 2010

Janina Fedjko, Joel Beales, and Karl Campbell

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

jfedjko@gmail.com
joel@xeo.ca
krlcmpbll@gmail.com

***Abstract* — For the purposes of this project we analyzed two obfuscation software products, namely Codelock and Codelock Tracker. These products are used to encrypt applications developed in PHP. The goal of our analysis was to determine the effectiveness of the protection provided by these products. We were able to obtain the keys and decrypt files encrypted by either of the two products. Moreover, we were then able to exploit severe security flaws present in Codelock. This report outlines the details of the analyzed software, summarizes external efforts related to our project, describes our approach and analysis methodology, and provides a discussion of the obtained results as well as the conclusions we came to based on our findings.**

I. INTRODUCTION

Hypertext Preprocessor (PHP) is a scripting language most commonly used for web development [1]. PHP was developed in the middle of the 1990's and has been dramatically gaining popularity over the last decade. Much of the appeal to PHP consists of the ease of integration with open source web servers and database applications. Since PHP is a scripting language, it is never actually compiled into binary. Source code written in PHP is converted on-the-fly to an external format that is executed by a PHP engine, which creates machine-readable binary code. This carries the implication that the source code for a PHP application always has to exist in plaintext format and cannot be packaged into a standalone, executable file. Due to this fact, PHP developers that wish to distribute their applications must do so with at the risk that someone could easily violate licensing restrictions or potentially modify their code and redistribute it under as a brand new product. Conversely, developers using compiled languages, such as C++ or Java, do not have the need for such concerns, as their source code is always directly translated into binary data and, consequently, much more difficult to obtain, modify and redistribute.

In attempts to overcome the issue of potential intellectual property theft for PHP application developers, several commercial products have become available on the market for PHP source code encryption. Encrypting PHP source code allows developers to distribute their code without the fear of attacks from malicious users.

There are two common methods of encrypting PHP source code. The first method is fairly secure, yet expensive and complicated. It involves converting the code into binary format, releasing it in this format to the public, and then installing an extension application on the server, set up to parse this binary data. Besides the aforementioned negatives of this method, this encryption tactic is also problematic due to the fact that hosting service providers are often reluctant to install PHP parsing applications on their servers. The second, less expensive encryption option involves obfuscating the source code. This is achieved through converting the source code into ciphertext using a specified key. The ciphertext is decrypted at runtime, briefly exposing the plaintext source code to the local PHP engine used to interpret it. [2]

For many PHP developers, the only line of defense from theft involves utilizing PHP encryption software designed in accordance with one of the abovementioned methods. As individuals familiar with software development, we were able to recognize the importance of protecting intellectual property and use this as a motivation for carrying out this project, the purpose of which was to determine the level of protection provided by some obfuscation software products. We analyzed two products developed by Codelock, a company in New Zealand that specializes in encryption software development. These products are Codelock and Codelock Tracker. The developers claim that both products provide wholesome protection for PHP applications. Our goal was to evaluate the feasibility of decrypting files that have been encrypted with these products.

This report outlines the details of the analyzed software, summarizes external efforts related to our project, describes our approach and analysis methodology, and provides a discussion of the obtained results as well as the conclusions we came to based on our findings.

II. Analyzed System

As mentioned, two software products were analyzed for the purposes of this project. These are Codelock and Codelock Tracker, both developed by Codelock. The two products are not at all related in design, only by brand.

Codelock is a standalone PHP application that accepts a user-specified key, an optional expiration date for the key and a PHP file in plaintext format as inputs, and produces an encrypted source file. In order to execute the source file, the encryption must be reversed with the same key that was used to encrypt it.

Codelock Tracker is a more advanced product. Its main advantage is the inclusion of a “call-home” feature. Similarly to Codelock, it encrypts a plaintext file with a key, except this key is provided by the home server rather than specified by the user. During execution, the local application contacts the home server, requesting the key needed for decryption. This request contains the encrypted file and information about the local server environment necessary for authentication. Once the home server requests the key, the application uses it in combination with the file serial number to perform the decryption.

III. Related Work

We have performed extensive Internet research on the works done by others in relation to the subject of decrypting files that contain encrypted PHP source code. Our search, however, has been mostly futile. Although somewhat disappointing, the lack of relevant findings is not at all puzzling. Openly publishing results of any attempts to break or exploit commercial software would carry legal implications and may entail serious consequences.

We did, however, notice countless advertisements from people looking for someone capable of decrypting files that have been encrypted with either Codelock or Codelock Tracker. This implies that both of these products have been broken by others in the past, although whether or not the approach was similar to others cannot be determined.

We have also found several instances of software that was advertised as tools for decrypting files encrypted with Codelock. These tools, however, were nothing more than disguised malware and did not function as advertised. No such tools were found for CodLock Tracker.

IV. Analysis Methodology

We began our analysis of Codelock by downloading a fifteen day free trial version of the software from the company website. Then, through trial and error, we began exploring the functionality of the software. Interestingly enough, it was discovered that Codelock was actually developed in PHP and then used to encrypt its own source code.

The source code for Codelock's logic was contained in a file named 'codelock.php' and included with other necessary software components, all of which were encrypted. The task at hand involved reverse engineering this file. Our first clue to accomplishing this task was a long string of data that was being executed with the following command: *eval(base64_decode($data)).* This command performs the actions specified by the string of data passed as a parameter to the *base64_decode($data)* function, which decodes the specified data in base-64. This indicates that the first layer of data has been obfuscated through base-64 encoding. Replacing *eval(...)* with *echo(...)* yielded in the program printing the decoded version of the data. After the first replacement, we searched the decoded data for another instance of *eval(...)* and replaced it with *echo(...)* yet again. This step was performed several times until every obfuscation layer has been removed and the final iteration of *echo(...)* yielded in the reveal of the unencrypted source code that is used to decrypt and execute encrypted files. From here onwards we will refer to this component as the ‘decrypter.’

Every encrypted file contains an encrypted payload. By examining the decrypter, we were able to determine where the payload is parsed, decrypted and executed. It was discovered that the decrypter uses the first three characters of the key to perform a simple replacement to convert ciphertext to plaintext, which is then decoded using *base64_decode($data)*.

Our next goal involved extracting the key from the encrypted file. Through elementary string manipulation and a series of base-64 decoding operations, the key was discovered in the 'codelock.php' file in plaintext format, under several layers of base-64 encoding.

Once we mastered key extraction and source file decryption, we began looking for more vulnerabilities in the Codelock software. One crucial flaw of the decrypter is the presence of *@extract($_REQUEST)* function. $_REQUEST is the global array, which contains all the information gathered from cookies, and the *post()* and *get()* methods for exchanging information. For example, a query such as: *script.php?foo=bar* will make the following entry to the $_REQUEST array: $_REQUEST[*foo*] = *bar*, which can then be easily accessed to gain the corresponding data. The function: *@extract($_REQUEST)* is a simple, yet extremely unsafe method of accessing the information in the $_REQUEST array, turning the otherwise static array into a set of modifiable variables. With this knowledge, a malicious user can set the initial value of any variable in the script, potentially causing security risks. This line has the same effect as enabling *register_globals* in a PHP configuration. Due to the fact that variables that have been initialized during the decryption of the source code maintain these values during execution, scripts that have been encrypted with Codelock gain security risks that the developer would assume to not exist.

Another critical security problem that we discovered is associated with the way Codelock performs decryptions on the Apache server in Microsoft Windows environment. When Codelock decrypts a source file, it places the unencrypted data into a temporary file, which is deleted upon execution of the script. The path for this temporary file is stored in the $_REQUEST array. Because of the aforementioned presence of *@extract($_REQUEST)*, we were able to manually specify the location of the temporary file. Setting the appropriate variable using the URL query string to a filename of a file that already exists on the server causes that file to be overwritten and deleted, posing an obviously severe security issue. This exploit can be used to delete any file within the web root with appropriate permissions. In order to exploit this vulnerability, an attacker needs to know nothing beyond the URL of a file on the server that has been encrypted with Codelock. Fortunately, this vulnerability does not exist on all server types, as the creation and deletion of the temporary file is handled differently by different systems. As mentioned, the vulnerability was present in our install of Apache and PHP on a Microsoft Windows machine; however, on a Ubuntu Linux machine the vulnerability no longer existed. Most web servers utilize Linux as their operating system, so this vulnerability is most likely not widespread, but it does pose a significant security risk to the websites that are vulnerable.

After successfully exploiting Codelock, we turned our attention to Codelock Tracker. We set up the Codelock Tracker server and attempted to intercept the communication between a script requesting its key and the server supplying the key to the script. It was found that for authorization purposes the script sends environment information to the server, namely a hash of the file itself, the IP address of the local host, the URL of the file, and the serial number specific to the script's license. Once the server receives and verifies the script's credentials, it simply sends back the key without any further authentication or challenges. The key exchange takes place using the POST method of the HTTP protocol, which is the same method generally used to submit information to web forms in an Internet browser. After discovering the steps taken during the authorization sequence, introducing a replay attack to retrieve the key for decrypting the script did not present a challenge. To accomplish this, we simply needed to collect the appropriate credentials and place them in our own web form. There was no lockout or warning for the software administrator that somebody was providing false credentials when authentication was failed intentionally.

Upon retrieving the key, we were faced with the challenge of decrypting the source file. Through analysis of the encrypted file, we were able to isolate the function that uses the key to decrypt the ciphertext. This function first performs a *base64_decode(…)* of the key obtained from the server. Then, the ASCII value of each character of the key is bitwise XOR'ed with the ASCII value of each character of the serial number to produce a 'session key.' Similarly, the session key is then bitwise XOR'ed with the ASCII values of ciphertext of the encrypted script, revealing the source code in plaintext format. Based on these findings, the encryption method employed by Codelock Tracker can be categorized as a block cipher with an electronic codebook (ECB) mode of operation. The key returned by the home server is an ASCII string of 255 characters, which translates into bit-strength of 1785 bits. Clearly, attempting to retrieve the key through brute force is unfeasible; however, if the encrypted file is large enough, it may be possible to obtain some portions of the code due to repeated segments created by the ECB mode encryption.

V. Results

We were extremely satisfied with the results obtained through our analysis of Codelock and Codelock Tracker. We were able to obtain the keys and decrypt files encrypted with either software, thereby achieving the main goal of our project. Then, through extensive analysis, we were able to comprehend the decryption algorithm used by both products, which allowed us to perform and reverse encryptions on any PHP application. Furthermore, we were able to discover and exploit severe security issues associated with Codelock, including the enabling of the variables contained in the $_REQUEST array and the ability to delete files that have been encrypted with Codelock from the web root directory.

Moreover, we developed a set of tools for systematic decryption of files and vulnerability exploitation associated with both Codelock and Codelock Tracker. The tool created for Codelock accepts the filename of an encrypted file as a parameter and extracts the key. The key is then entered into another tool that we developed, which also accepts the corresponding filename and produces the source code in plaintext format. We also developed a tool that deletes a user-specified file that has been encrypted with Codelock from a web root directory of the web server that contains this file. The tool that was developed for Codelock Tracker mimics the "call-home" key request to obtain the key for an encrypted file. Then another tool accepts this key, along with the encrypted file and the serial number, and produces the code in plaintext format.

VI. Discussion

It is clear that a PHP script that has been encrypted with either Codelock or Codelock Tracker is not fully protected, as the developers of the products would like the users to believe. At stake are users' confidentiality, integrity and availability of resources. By exploiting vulnerabilities present in Codelock and Codelock Tracker, malicious users are able to retrieve PHP developers' intellectual property and redistribute it as their own. Moreover, if targeted under specific conditions, unsuspecting developers can completely lose the contents of their websites. When using Codelock and Codelock Tracker, developers' applications are threatened by deception, disruption and usurpation.

The encryption method used by Codelock attempts to achieve security through obscurity, not following any recognizable encoding format. The fact that the software only uses the first three characters of the key for encryption leaves the encrypted files vulnerable to brute force attacks. Moreover, even if the key is unknown, the plaintext code derived based on any input for the key is still somewhat readable to anyone familiar with PHP scripting. Codelock Tracker, on the other hand, utilizes every character of the 255-character string to encrypt and decrypt.

Moreover, Codelock creates a major security issue for many web servers that host scripts, which have been encrypted with Codelock by exposing the variables contained in the $_REQUEST array. This flaw is frightening, as otherwise secure scripts are made insecure because of this vulnerability. Even more worrisome is the fact that distributing a fix for this vulnerability is extremely difficult or even not viable. Every script encrypted with Codelock would need to be decrypted and then re-encrypted using a patched version. As Codelock is a commercial application, it should have been thoroughly tested internally before being released for public use. The effects of tampering with publically accessible variables should have been investigated on multiple server configurations prior to the product's release. PHP developers who are trusting Codelock to protect their property are paying for software that contains a vulnerability that likely would not have existed had the product been thoroughly tested.

It was surprising to discover was how much stronger the encryption methods used in Codelock Tracker are than those used by Codelock. In fact, Codelock Tracker would have been a fairly effective product, had it not contained a flaw in its authorization technique.

The first principle of designing secure systems that was violated by the developers of Codelock is the Principle of Open Design. The proprietary encryption method utilized by the software can best be described as providing security through obscurity, rather than through any actually secure techniques. The main method of encryption consists of repeatedly performing base-64 encodings, which is typically intended for representing data in a different format, rather than converting it to ciphertext.

Another design principle violated by Codelock developers is the principle of Layered Defense. It can even be said that Codelock breaks down defenses offered by native PHP settings by its implementation of *@extract($_REQUEST)*. We believe that this line likely exists because Codelock was originally developed when *register_globals* being enabled in PHP was the default setting. Then, rather than rewriting Codelock to support this change, the developers injected a flaw into the system as a simple fix for compatibility issues.

Finally, both Codelock and Codelock Tracker violate the Principle of Questioning Assumptions. Had the developers constantly re-examined their assumptions about threat agents, assets and the environment of the system, they would have likely been able discover the vulnerabilities present in their products and derive viable solutions.

A simple improvement for Codelock involves hashing the key stored in the encrypted file and using every character of the key for encrypting. Another major improvement would be to derive a more sophisticated encryption technique. PHP modules such as Mcrypt are capable of performing two-way encryptions with industry standard encryption algorithms. Codelock developers could layer the defenses by combining a non-proprietary encryption mechanism with already used obfuscation techniques.

An improvement on Codelock Tracker would be to implement a more sophisticated authentication process that is not so simple to imitate. Implementing a nonce feature could greatly reduce the vulnerability of Codelock Tracker. However, such an exchange would increase the response time for the key return. Additionally, the tracker software may want to implement a lock out feature for users providing invalid information when trying to request a key. If hashes or other authentication information is invalid, somebody is likely in the process of attempting to decrypt sources. As the encrypted source is essentially secure without the key, it would be best to block these users before they are successful in their efforts. It may also be worthwhile to use a different mode of operation in the implementation of the block cipher to prevent any risk of deciphering repeated parts created by ECB.

VII. Conclusion

There are two methods of encryption PHP scripts. The first method involves converting code into binary format, whereas the second method involves obfuscation. The former is a much more expensive option, limiting its availability to smaller developers, while the latter is cheaper, but less secure. For the purposes of this project, we analyzed the strength of protection provided by two obfuscation products, namely Codelock and Codelock Tracker. We were able to obtain the keys and decrypt files that were encrypted with either of the two products. Moreover, we were able to exploit serious security flaws present in Codelock. The encryption technology of Codelock was determined to be fairly weak, starting with insufficient password strength and finishing with ineffective encryption mechanism. Codelock Tracker is a much stronger product, with its main vulnerability residing in its authentication process. This vulnerability allows for replay attacks when retrieving the key from the Codelock Tracker home server.

While performing our extensive analysis and research, we noticed that Codelock Tracker was recently reverted into Beta stage and has become unavailable for purchase through the Codelock website. Hopefully, some of its vulnerabilities have been discovered and the company is working on repairing and improving the software. It is recommended that the authentication process is improved in Codelock Tracker and the severe vulnerabilities present in Codelock are patched before unsuspecting developers fall victim to theft of their intellectual property.

References

[1] The PHP Group. "History of PHP and related projects". December 3, 2010. http://ca3.php.net/history December 6, 2010.

[2] Maung, Ei. “Compiled Languages vs. Scripting Languages.” *Ei Maung’s Blog*. http://eimg.wordpress.com/2007/12/31/compiled-languages-vs-scripting-languages/. December 6, 2010.