Impact of Internet on Business Activities in Serbia and Worldwide

Uticaj Interneta na poslovanje u Srbiji i svetu

DOI: 10.15308/SINTEZA-2014-616-619

ONE SOLUTION FOR PROTECTING PHP SOURCE CODE

Nenad Ristić[1], Aleksandar Jevremović[2]
[1]Sinergija University, BIH
[2]Singidunum University, Serbia

Abstract:
Protecting PHP scripts from unwanted using, copying and modifications is a vast problem today. Present solutions on source code level are mostly working as obfuscators, are free, and are not providing any serious level of protection. Solutions that are based on encoding opcode are more secure but are commercial and require closed-source proprietary PHP interpreter's extension. Furthermore, encoded opcode is not compatible with future versions of interpreters which involve re-buying encoders from authors. Finally, if extension source-code is compromised, all script encoded by that solution are compromised too. In this paper we present a novel model for free and open-source PHP script protection solution.

Key words:
PHP,
interpreted languages,
source code,
protection,
encryption.

INTRODUCTION

Within the last few years, PHP has established to be the most pervasive web platform round the world, operating in more than a third of the web servers across the world. PHP's development has not only been quantitative but also qualitative. Every day more and more businesses rely on PHP to run their applications which are critical for their business. This creates new jobs opportunities and increases the demand for PHP developers. While the difficulty of starting with PHP remains unchanged and very low, the possibilities offered by PHP today allow developers to extend far beyond simple HTML applications. The reviewed object model allows for large-scale developments to be written more efficiently by using standard object-oriented methodologies.

New XML support makes PHP the best language available for processing XML and, coupled with new SOAP support, an ideal platform for creating and using Web Services [1]. PHP is one of the most popular languages for Web development. By December 2013, PHP was being used by a remarkable 81.7% of sites according to W3Techs - World Wide Web Technology Surveys [2]. One of the really significant problems for PHP developers today is lack of free and high-quality solutions for protecting source code of PHP Web applications. By "protecting source code" usually two things are considered: 1) protecting source code to be viewed/modified by others and 2) limiting protected application execution to specific Internet domain or time period.

At this time, there are some solutions for protecting PHP source code which, generally, belong to two key groups. First group contains PHP source code obfuscators, which are usually free, work with source code and provide very low level protection. Second group contains PHP encoders, which work with PHP opcode, thus provide higher level of protection, but are commercial and require using proprietary closed-source PHP extension in production environment.

Even if PHP encoders provide higher level of protection, there are two main problems when using them. First problem is limited lifetime of encoded product because source code is converted to opcode by current version of PHP interpreter, and then opcode is encoded. This means that encoded solution becomes unusable with future versions of PHP interpreters that include some important change of how source code is transformed to opcode. Because of this, developers are forced to buy new versions of encoders and to recompile source code whenever PHP version on Web server is upgraded. Frequent replacements of application files in production environment are usually a painful task.

Second problem with using PHP encoders is dependency of "third trusted part" - author of encoder. This means that whole security of application depends of company that develops encoder. If encoder or extension is compromised (source code is revealed to public), all solutions encoded with that encoder are compromised, too. Additionally, encoded PHP scripts are not protected from encoder authors.

In this paper we are analysing possibilities and issues with creating solution for high-quality protection of PHP scripts. Standard cryptology models are used for this analysis. Based on the results of this analysis, we propose a novel model for solution that provides solid protection of PHP source code on both source code and opcode levels and is not based on trusted third party.

CURRENT SOLUTIONS

Obfuscating source code

Obfuscation is a technique that transforms original source code to its far more complex, confusing and unreadable variant, while preserving code semantics [3]. This technique is used to prevent or decrease efficiency of reverse engineering while providing the same functionality with equal or similar performance. Obfuscating is usually done by replacing variables and user defined functions names to meaningless ones, by removing comments and formatting, and by encoding source code with some of built-in or user-defined encoding functions. Source code obfuscating is very similar to optimization of source code, with difference, that with obfuscation of code we are trying to maximize obscurity while trying to minimize execution time.

In exploration of approaches to obfuscate source code conclusions from [4], [5] make following statement of the code obfuscation code. Given a set of obfuscating transformations $O_t = \{O_{t1}, O_{t2} \dots O_{tn}\}$ and program P consisting of source code objects (classes, methods, statements, etc.) $\{S_{o1}, S_{o2} \dots S_{ok}\}$, find new program P' = $\{\dots, S_{o'j} = O_{ti}(S_{oj}) \dots\}$ such that:

- P' has the same functionality as P, for example the conversions are maintaining semantic
- The indistinctness of P' must be at prime level so that understanding and reverse engineering of P' would be more time consuming than understanding P
- The resilience of every obfuscating transformation $O_{ti}(S_{oj})$ is maximized, for example it will be to complex and difficult to construct an automated tool to undo the obfuscating transformations or applying a tool would be extremely time consuming.
- The stealth of each obfuscating transformation $O_{ti}(S_{oj})$ is maximized, for example the statistical properties of $S_{o'j}$ are similar to those of S_{oj}.
- The execution time delay of P' because of obfuscating transformation must be minimized.

Using obfuscating can give good results when developer wants to prevent software crackers from understanding parts of code and then illegally including them to other Web applications. However, this technique shows poor results when used for restricting usage of protected solution to specific domain name or time period, because, in this case, pirates are not required to understand large portion of code, but only to identify place where limitations are defined. Having in mind that opcode modifications of this type are not compatible with original interpreter, this technique is limited to use with source code only.

Encoding/encrypting

Both encoding and encrypting are reversible data transformation techniques that, however, contain essential differences. Encoding implies using of publicly known transformation algorithm with, if used, also publicly known parameters. In other words, it is assumed that anyone can decode encoded data if informed what encoding algorithm was used for encoding. Encryption, on the other side, is based on secret parameter (key) used in transformation procedure. It is usually assumed that transformation procedure algorithm is publicly known, but it needs not to be.

Most of major PHP encoders today are actually acting as encrypters because are assuming some secret component that prevents encoded source code or opcode to be decoded. This component is usually algorithm (sometimes combined with some encryption parameter like project id or so) which explains why PHP interpreter extensions for these encoders are closed source. That, however, as said before, creates unwanted dependency of encoder provider.

Protecting source code vs protecting opcode

There are significant differences between protection on source code level and protection on opcode level. Main advantage of protection on source code level is compatibility with future versions of PHP interpreter, while main disadvantage of that approach is possibility to reveal original source code if protection is broken.

On the other side, main advantage of protecting on the opcode level is lack of possibility to reveal original source code, while main disadvantage of that approach is limited lifetime of protected scripts. In case of protecting opcode, source code is interpreted by current version of PHP interpreter before it is encoded/encrypted. However, opcode compatibility with future versions of PHP interpreters is much lower than it's the case with source code. This means that developers often need to re-encode original source code whenever hosting provider upgrades to next major version of PHP interpreter. Having in mind that process of replacing encoded scripts is happening in production environment, it is naturally to want to do this as rarely as possible. Also, re-encoding source code for new PHP version usually implies buying new version of encoder.

ANALYSIS WITH CRYPTOGRAPHY MODELS

From cryptology aspect source code protection could be seen as an establishing secure communication channel between developer and PHP interpreter. Instructions, in a form of source code, which represent secret message, are encrypted and can only be decrypted by final interpreter. Using standard characters for representing different roles in secret communication, Alice and Bob are developer and PHP interpreter, while Eve is everyone else - including Web server administrator. Encryption algorithm is considered to be publicly available in all wide used systems.

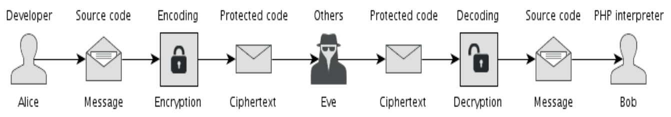

Fig. 1. Cryptology model of secured communication channel

Following presented model, main question that arises is how to manage key(s) that is used for encryption/decryption procedures. This question leads to another question: is symmetric or asymmetric cryptography more appropriate using in this case? Additionally, trust in Bob's integrity must be reconsidered.

Symmetric or asymmetric cryptography

When using symmetric cryptography same key is used for encryption and for decryption. This opens a questions - who generates the key (Alice or Bob), and how is key distributed to the other part? If PHP interpreter generates the key, that key can be stored locally, maybe even inside interpreter's binaries. However, how can this key securely be distributed to developer, with no one else access to it, even system administrator? The same problem, even more accented, remains in case when developer generates the key and needs to distribute that to interpreter.

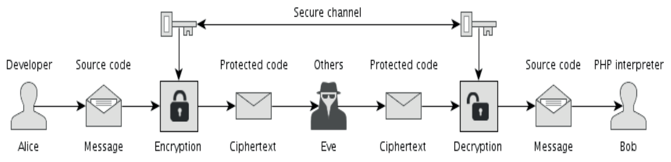

Fig. 2. Architecture using symmetric cryptography

By using asymmetric cryptography, the need for secure channel is eliminated. Developer can encrypt PHP source code by using PHP interpreter's public key. That means that encrypted source code can only be decrypted by using PHP interpreter's private key, which is stored on secure location. Additionally, developer can digitally sign source code with his protected key so interpreter can be sure that it's coming from developer. This is useful when limiting application to work only with files developed by the same developer.

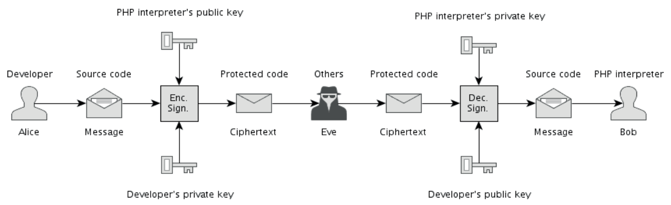

Fig. 3. Architecture using asymmetric cryptography

Lifetime of protected solution in this case is limited by source code compatibility with future PHP interpreters versions, or by digital certificate lifetime (which can be unlimited), whatever comes first. However, problem of location where PHP interpreter's private key is stored, and how it's used, remains. Potential solution is storing private key within interpreter's binary, so only reverse engineering attack is possible. However, behaviour of interpreter is not guaranteed because its source code can be changed and then modified interpreter could be used.

Main problem

As we can see from previous examples, there is no difference if we use symmetric or asymmetric encryption, place for storing key that is used for source code decryption remains the main problem with protecting source code. Another part of this problem is the fact that PHP interpreter is open source, so it can be modified to expose decrypted source code before executing it. That implies that we cannot trust in Bob's integrity, which means that we can consider PHP interpreter on Web server as Eve, too.

Reverse engineering

Reverse engineering goal is at gaining high-level representations of software systems from known low-level objects, such as binaries, source code, execution traces or historical information. Reverse engineering methods and technologies play an important role in many software engineering tasks and quite often are the only way to get an understanding of large and complex software systems [6].

When evaluated from cryptology aspect, reverse engineering process is analogy to cryptanalysis. By this we mean that pirate is trying to read or modify message that is not intended to be seen or modified by him. Idyllic solution for this problem would be one that can't be reverse engineered even if reverse engineering is tried on CPU level.

Question that arises is how deep we need to go in order to provide another trusted part in secret communication - component that will securely implement our programs in environments being controlled and eavesdropped a by potential pirates? And also, is it possible to have such a component as open-source, without relying on secret possessed by disputed "trusted third part" - author of that component? And finally, even if the solution for this problem exists, will its price and complexity be appropriate for using in cheap shared hosting environments?

For the purpose of this paper we set our goal to make protected PHP scripts as safe as if they were typed in some compiled language (like C, for example). This also means that protection from assembler lever reverse engineering is not included in proposed solution.

PROPOSED SOLUTION

Based on exposed results and insights from cryptology based analysis, we propose a novel solution model that provides protection of PHP scripts on both source code and opcode levels, and is not based on trusted third party. Protection level of proposed solution is equal to currently available commercial solutions, based on closed-source components.

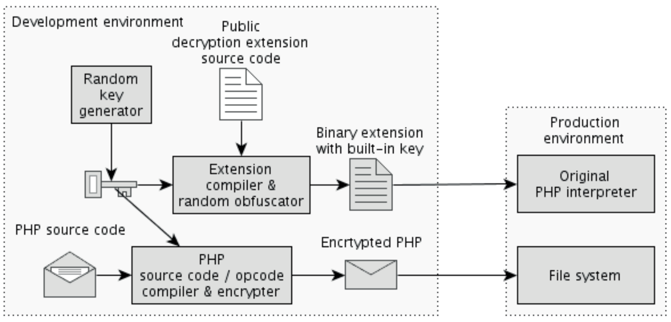
Development environment
Public decryption extension source code
Random key generator
Extension compiler & random obfuscator
Binary extension with built-in key
Production environment
Original PHP interpreter
PHP source code
PHP source code / opcode compiler & encrypter
Encrtypted PHP
File system

Fig. 4. Model of proposed solution

Architecture of proposed solution is explained on Fig 4. Two main components of proposed solution are PHP source code compiler/encrypter and open-source extension for original PHP interpreter. Additional component is random key generator, but for this purpose any (pseudo) random generator can be used.

PHP source code compiler works as a regular interpreter - converts source code to opcode - with exception that result (opcode) is encoded/encrypted with freshly generated key (which is known only to developer). That encoded opcode can be executed only with PHP interpreter that knows the secret key. Also, in order to increase encoded scripts' lifetime, encoder can encode source code directly (with obfuscation if selected), without transforming it to opcode. However, protection level in this case will be significantly lower because potential pirate will be able to catch (obfuscated) source code as a result of extension execution. Even if working with source code instead opcode is supported, this is not recommended because it could be revealed by PHP interpreter modified by the eavesdropper.

Another component of proposed solution is open-source (publicly available) extension for decoding previously explained encoded PHP scripts. However, this extension is completely unusable without having key which is used for encoding PHP scripts. That's why extension is compiled to binary by developer, and during that process key is built-in binary result. However, in order to hide location where key is stored in extension binary, extension compiler, before compiling, is randomly obfuscating extension's source code by adding random code snippets and false keys as variables, that have no impact on extension behaviour. This means that two results of independent compiling's of extension, even with the same key, will give completely different results.

Next step for developer is to upload encoded PHP scripts and compiled extension to share Web server and to enable it when executing his protected scripts. Downside of proposed solution is requirement for server administrator to allow users to load their own binary extensions.

CONCLUSION

In this paper we analysed problem of protecting intellectual property in a form of interpreted languages source code. PHP, as the most prevalent interpreted language for Web development is used as an example. Our main analysis is based on using of standard cryptology models and which are used for analysing existing solutions, as well for search for ideal theoretical model.

Essential problem for protecting PHP scripts, analysed as cryptology model, is lack of another trusted part in secured communication. PHP interpreter, in a role of another part in secured communication, is an open-source software which behaviour is publicly know and can be modified by potential eavesdroppers/pirates.

Source code obfuscation is identified as computationally secure protection, while (human-based) breaking it is analogy to cryptanalysis. However, the need for source code to be understandable by interpreter eliminates obfuscation as a serious standalone protection.

On the other side, source code or opcode encryption requires trusted decryption part, at least as a secured space where key used for decryption is stored and used. This is not possible with using completely open-source solution for PHP interpreter. Using closed-source components for decryption, which is case with existing PHP encoders, creates security and commercial dependency of encoder provider.

Solution's model presented in this paper proposes hybrid approach where all components that provide scripts protection (or secure communication, from cryptology aspect) are publicly available and open-source. However, decryption component is realized as PHP interpreter's extension and is obfuscated and compiled by developer. Key, which is used for PHP scripts encryption, is integrated within aforementioned extension during the compiling procedure.

REFERENCES

[1] Gutmans, Andi, Stig Bakken, and Derick Rethans. „PHP 5 Power Programming (Bruce Perens' Open Source Series)". Prentice Hall PTR, 2004.

[2] http://w3techs.com/technologies/overview/programming_language/all (available on 03.01.2013)

[3] Cho, Seongje and Chang, Hyeyoung and Cho, Yookun, "Implementation of an Obfuscation Tool for C/C++ Source Code Protection on the XScale Architecture", Proceedings of the 6th IFIP WG 10.2 international workshop on Software Technologies for Embedded and Ubiquitous Systems, 2008, ISBN: 978-3-540-87784-4, DOI: 10.1007/978-3-540-87785-1_36

[4] Collberg, C., Thomborson, C., & Low, D. (1998, January). Manufacturing cheap, resilient, and stealthy opaque constructs. In Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (pp. 184-196). ACM.

[5] Collberg, C., Thomborson, C., & Low, D. (1998, May). Breaking abstractions and unstructuring data structures. In Computer Languages, 1998. Proceedings. 1998 International Conference on (pp. 28-38). IEEE.

[6] Pinzger, M., & Antoniol, G. (2013). Guest editorial: reverse engineering. Empirical Software Engineering, 18(5), 857-858.