POSTER—CryptServer: Strong Data Protection in Commodity LAMP Servers

Zhaofeng Chen*
Institute of Computer Science and Technology
Peking University
chenzhaofeng@pku.edu.cn

Xinshu Dong, Prateek Saxena, Zhenkai Liang
Department of Computer Science
National University of Singapore
{xdong,prateeks,liangzk}@comp.nus.edu.sg

ABSTRACT

Modern web applications store sensitive data on their servers. Such data is prone to theft resulting from exploits against vulnerabilities in the server software stacks. In this work, we propose a new architecture for web servers, called CryptServer, in which we pre-determine and fix a small amount of application code that can compute over sensitive data. By encrypting sensitive data before making it available to the rest of untrusted application code, CryptServer provides strong defense against all malicious code that an attacker may run in the server software stack. As a step towards making this approach practical, we develop an assistance tool to identify the portion of server-side logic that requires computation over sensitive data. Our preliminary results show that the size of such logic is small in six popular web applications we study. To the extent of our evaluation, converting these applications to a CryptServer architecture requires modest developer effort.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

Keywords

Web security, server security, data protection

1. INTRODUCTION

Web servers are prone to a large variety of attacks at the application layer — SQL injection, server misconfiguration, data-to-code attacks, OS command injection, and so on [11]. These attacks can be used to exfiltrate financially sensitive or user's private data. A recent study shows that 54% of data breaches involved compromised servers [10].

Several research works have started investigating mechanisms to protect sensitive data on web servers, such as applying encryption scheme to databases [3, 6] or by partitioning web application code into different trust levels [4, 5]. However, these techniques do not protect the sensitive data comprehensively throughout its processing lifetime in the web application code, and trust a large TCB.

Our Solution. In this work, we fortify web servers with a new second line of defense to secure sensitive data on commodity cloud-hosted LAMP servers. Conceptually, our solution protects sensitive data by encrypting the sensitive data and splitting the PHP engine logic into two strands of computation logic: sensitive and non-sensitive. Only certain trusted functions that compute on sensitive data in the PHP engine are executed in the sensitive strand. We call such functions *Pieces of Sensitive Logic (PSLs)*. We isloate the PSLs in an isolated exectuion enviroment (trusted VM) to provide rich computations on the encrypted data. All of the rest of web application logic only has access to sensitive data in encrypted form. With this new architecture, which we call CryptServer, we significantly reduce sensitive data's direct exposure to untrusted code on commodity cloud-hosted server stacks.

To migrate existing web applications to this new architecture, we take a two-phase approach to help the adoption: an *analysis phase* and an *instrumentation phase*. In the analysis phase, we automatically identify PSLs in the PHP application using an assistance tool we have built. When web developers run this tool with their application test harness, it tracks the flow of sensitive data, and automatically identifies functions that compute on the sensitive data.

Once all the PSLs for the application are identified, we manually instrument the original PHP application to invoke the corresponding PSL functionality running in an isolated environment (e.g. in a VM, a PAL [1], or a hypapp [9]). CryptServer enforces that all data input and output from PSLs are cryptographically sealed, i.e., protected with authenticated encryption. All data exchanged between the transformed PSLs and the non-PSLs is encrypted using authenticated encryption. Thus, the vulnerable web server, even if compromised, can only observe the encrypted flows of sensitive data.

Our main empirical finding is that PSL logic for the real-world applications we test is small. It consists of only 30K lines of C code of the PHP interpreter logic that runs in the trusted isolated environment. In contrast, previous works that partition servers, such as CLAMP [5] and SilverLine [7], have reported either much lareger TCB [5] or do not permit any computation on sensitive data [7]. Our evaluation

*Research done when visiting National University of Singapore

CCS'13, November 4–8, 2013, Berlin, Germany.
ACM 978-1-4503-2477-9/13/11.
http://dx.doi.org/10.1145/2508859.2512525.

reports that CryptServer is expressive enough to support real-world flows of sensitive data without directly leaking information about sensitive data through explicit channels. We also find that the developer effort to migrate existing applications to this new architecture is on the order of a few hours, with the help of our assistance tool.

In summary, to our knowledge, CryptServer is the first to propose an architecture that enables significant TCB reduction in web server stacks with authenticated encryption; our analysis tool and preliminary results verify its applicability to existing web applications. CryptServer complements server platforms with encrypted databases [6].

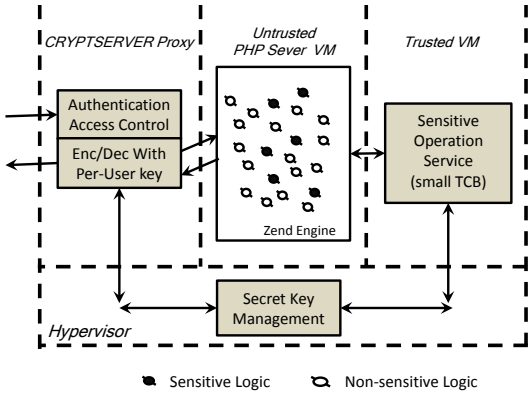

Figure 1: Architecture of CryptServer

2. CryptServer ARCHITECTURE

Figure 1 shows the CryptServer architecture. In our setting, the web server stack is built on top of a trusted hypervisor. The entire web server stack in the server VM is not trusted and is assumed to be vulnerable.

CryptServer **Proxy.** Our target is to secure both the confidentiality and integrity of sensitive data with authenticated encryption. In our approach, a lightweight CryptServer Proxy is introduced to encrypt sensitive data based on developer annotations before the data flows into the PHP application server VM. Similar to CLAMP [5], we have an HTTPS front end that authenticates the users, and performs key management and access control. On receiving each request from the client, the proxy authenticates the user and invokes the underlying hypercalls to encrypt sensitive data in annotated HTTP request fields with the per-user secret key accordingly. Then the encrypted data is delivered to the untrusted server VM. An alternative approach is that the client (e.g. user browser) helps encrypt the sensitive data instead of the proxy.

Untrusted PHP Server VM. When the encrypted sensitive data flows into the vulnerable server, the pre-compiled PHP opcodes (see details in Section 3) start execution in the PHP engine. Only parts of the opcode functions handle the sensitive data. We treat a function as PSL if and only if it receives sensitive data as arguments, and returns a value. To enable PSLs to operate on the encrypted sensitive data, we manually instrument the PSLs to make remote call to a trusted VM for computation on encrypted data.

Trusted VM. We isolate PSLs into a separate trusted VM (DOM0 in Xen Hypervisor [2]) that provides a Sensitive Operation Service (SOS) component to facilitate access to the sensitive data. When a sensitive operation is requested from the PHP server VM, the SOS first decrypts the sensitive data and checks the data integrity. If the check passes, the SOS component calls the requesting PSL to compute on the decrypted data. Finally, the SOS component encrypts the return value with authenticated encryption scheme and sends it back to the PHP server VM.

If all arguments in the request are encrypted, an attacker cannot tamper with them without failing the integrity check by the SOS component. However, there are cases where both sensitive (encrypted) and non-sensitive (unencrypted) data appear in the argument list. For example, PHP applications frequently search for special characters in user input strings. In this case, the input string is sensitive while the special character set is a constant and non-sensitive variable. An attacker can manipulate the constant argument value to obtain information on the sensitive data. We prevent such information leakage channels by applying the analysis as we detail in Section 3 to identify constant values that flow into function arguments, and encrypt them to prevent tampering from malicious code.

Summary. In the CryptServer architecture, any sensitive data flowing into the vulnerable server is encrypted. For ease of implementation, we are currently not encrypting boolean return values. This could leak certain control flow information to attackers. However, by design, PSLs do not allow attackers to control the argument values, as they take encrypted sensitive values and pre-encrypted constant and static variables as arguments. This achieves reduced indistinguishability [8]. In addition, as we show in Section 4, the pre-encrypted values allow little information leakage to attackers, as their numbers are very small for each web application we study. Although our solution tightly limits explicit information leakage, side channels are still possible, such as timing, length, and power monitoring attacks, etc. Our current solution does not address these issues.

3. ANALYSIS ENGINE

To aid developers to migrate their existing web applications to the CryptServer architecture, we build a dynamical analysis engine to automatically identify PSLs in the PHP engine. It takes developer annotations of sensitive data and a test harness of the web application as input. It then performs dynamic taint analysis to identify the operations that require access to the plaintext of the sensitive data, and finally marks them as PSLs as output. Note that not all operations that process sensitive data require decryption of the data. For example, the ASSIGN operation copies the encrypted data, but it does not require access to the plaintext of the data. Thus this kind of operation is not marked as PSL.

Before execution, PHP code is pre-compiled into PHP opcodes. Each opcode corresponds to a set of handling functions the PHP engine. Variables in PHP scripts are compiled into an internal presentation (the zval structure). Besides, all the constant and static variables mentioned in Section 2 can be automatically identified at the compilation time as zvals. Presently, we identify PSLs at the granularity of the PHP opcode level. Our analysis engine intercepts all operations on data in zvals and converts them into a standard "source->dest" propagation formula. It dynamically tracks

Application	#Pages w/ PSLs	#PSLs vs. Total #OPs	#Constant/Static PSL Arguments vs. Total #PSL Arguments	#Uniq PSLs vs. #Uniq Total OPs	Names of Sensitive Fields
phpBB3	16	342/131808 (0.26%)	20/635 (3.15%)	41/234 (17.52%)	subject, message, keywords
OpenEMR	6	46/59474 (0.08%)	3/86 (3.49%)	17/137 (12.41%)	reason, form_ss, form_body, note, issues
AjaxRPG	1	28/2044 (1.37%)	2/61 (3.28%)	9/86 (10.47%)	inputText
Roundcubemail	13	72/68541 (0.11%)	0/161 (0.00%)	30/237 (12.66%)	_subject, _message
Wordpress	15	111/440504 (0.03%)	1/209 (0.48%)	38/236 (16.10%)	post_title, content
HotCRP	8	97/76980 (0.13%)	4/131 (3.05%)	13/153 (8.50%)	paperSummary, commentsToAuthor, commentsToPC

Table 1: Percentage of PSLs in PHP applications

the data flow of sensitive data and marks operations in this flow that requires arguments' plaintext information. Finally the analysis engine outputs all marked operations as PSLs.

We implement this engine as a PHP extension in 9K lines of C code. All these identified PSLs are inspected manually and instrumented to request operations provided by the SOS.

4. EVALUATION

To evaluate the applicability, adoption effort, and TCB reduction of our solution, we apply our solution to 6 open-source PHP applications. Our preliminary results demonstrate that our proposed solution applies to these popular applications with moderate adoption cost. The experiments also show that the reduction in TCB that has access to sensitive data is also significant.

For each application, we manually annotate sensitive fields, shown in the last column in Table 1. For analysis we fill in these fields in the web pages, and submit the sensitive data to the server. For each web application we study, it takes one author about 3 hours to understand the functionality of the application, and further annotate the sensitive field. In real-world deployment, this identification effort can be reduced with developer assistance.

Column 2 in Table 1 shows the number of pages containing PSLs in each application. The PSLs mainly consist of computation opcodes (e.g. arithmetic operation opcode) and PHP functions (e.g. operations for string, array and regular expression). We calculate the number of opcodes executed during the execution, compared with number of identified PSLs. As shown in Column 3 of Table 1, the number of executed PSLs is much smaller than total executed opcodes, especially for large PHP applications (less than 1%). This indicates that the CryptServer is applicable to existing PHP applications. After eliminating the duplicate operations, the number of unique PSLs (Column 5) accounts for a very small portion (less than 18%). In fact, the PSLs code isolated in the SOS is small, with 30K lines of PHP interpreter logic for corresponding PHP applications.

For instrumentation, we identify the unique opcodes from dynamic runs with the test harness, which need to be tunneled back to the trusted SOS VM. Given the identified unique opcodes for PSLs, it takes less than 5 minutes to instrument each of them to request the corresponding operation from the SOS. In addition, once an opcode, for example ZEND_CONCAT, is instrumented in one application, it can be directly applied to all other applications. In all, our manually instrumented work only requires one-time effort for developers to migrate the existing application to CryptServer.

5. ACKNOWLEDGMENTS

This research is partially supported by research grant R-252-000-495-133 from Ministry of Education (MOE), Singapore. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of MOE, Singapore.

6. REFERENCES

[1] Portableapps.com launcher. `http://portableapps.com/apps/development/portableapps.com_launcher`.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, SOSP '03, 2003.

[3] Adrienne Porter Felt, Matthew Finifter, Joel Weinberger, and David Wagner. Diesel: applying privilege separation to database access. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, 2011.

[4] Taesoo Kim and Nickolai Zeldovich. Making linux protection mechanisms egalitarian with userfs. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[5] Bryan Parno, Jonathan M. McCune, Dan Wendlandt, David G. Andersen, and Adrian Perrig. Clamp: Practical prevention of large-scale data leaks. In *Proceedings of the 2009 IEEE Symposium on Security and Privacy*, 2009.

[6] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, SOSP '11, 2011.

[7] Krishna P. N. Puttaswamy, Christopher Kruegel, and Ben Y. Zhao. Silverline: toward data confidentiality in storage-intensive cloud applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, 2011.

[8] Shruti Tople, Shweta Shinde, Prateek Saxena, and Zhaofeng Chen. Autocrypt: Enabling homomorphic server computation to protect sensitive web content. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, CCS '13, 2013.

[9] Amit Vasudevan, Sagar Chaki, Limin Jia, Jonathan M. McCune, James Newsome, and Anupam Datta. Design, implementation and verification of an extensible and modular hypervisor framework. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, 2013.

[10] Verizon. 2013 data breach investigation report. `http://www.verizonenterprise.com/DBIR/2013/`.

[11] Wei Xu, Sandeep Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks. In *Proceedings of the 15th USENIX Security Symposium*, 2006.