

Enforcing Privacy in Web Applications

Ariel Futoransky
Corelabs, Core Security Technologies,
Florida 141, 7mo piso, 2do cpo.
Bs. As. (C1405AAC), Argentina

Ariel Waissbein
Corelabs, Core Security Technologies
and
Depto. Matemáticas, FCEyN,
Universidad de Buenos Aires,
Argentina.

Abstract

The development of web applications is typically done oblivious to privacy precautions. Largely, this is due to lack of technical knowledge and appropriate tools for enforcing privacy. As a result, web users' personal information is constantly at risk. We introduce a solution that protects arbitrary web applications from several dangerous privacy threats. It is easy to install, usable (e.g., in terms of expressiveness and appropriate enforcement), and requires no changes to the web application's code. Its main functionality is given by modules that are situated between web server and user, and between web server and APIs. These modules assign a privacy tag to inbound data according to its origin (e.g., received through a web form field or stored in an internal database); and block outbound data according to its privacy tag, what is its destination, and its syntax. This allows us, for example, to block private data from being exposed to users. We have implemented a prototype for the PHP platform that is efficient in terms of CPU and memory usage. We also verified that it enforces privacy in several potentially-vulnerable scenarios.

Furthermore, we provide a plug-in for web browsers that lets users visualize the privacy policy in action.

1 Introduction

Web applications are services delivered to end users by a web server, typically over the Internet and accessed through web browsers. For example, Gmail and Amazon.com. Web applications handle large amounts of data concerning different web users while enforcing diverse privacy policies.

In the context of web applications, a “privacy policy” is an agreement, between web users and the organization that hosts the web application, on how should users' information be handled. For example, web users will enter their credit

card numbers in a web form, if they are ensured that it will be kept private; and they will share their medical records for research purposes, if they are guaranteed privacy.

We consider the privacy problem from the perspective of both parties involved. Web users on the one side, and the person responsible for privacy enforcement (we call him the security officer, or SO for short) together with the underlying web-development group. Web applications pose a difficult problem to web users, that must rely on the application's owner commitment to privacy, and to security officers that must enforce privacy policy efficiently with limited resources ([16]). In most cases, the web users' personal information is seldom protected conclusively, sometimes knowingly, and sometimes by negligence. Attackers are constantly targeting web applications, mostly by accessing them as standard users and sending malformed input ([15]). A successful attack may expose private information, cause severe damage to the attacked organization ([12]), and to the exposed web users ([21]).

The only possible protection is good privacy enforcement. As with most development projects, and without loss of generality, we may describe the privacy enforcement processes in three stages —specification, development, and quality assurance¹. First, the security officer defines the requirements (ideally during the web application's specification), and forwards this document to the developers for its implementation.

Developers use one of several languages for web application design, from complex and powerful platforms (e.g., Java, .NET) to easy-to-use scripting languages (e.g., PHP, Perl, ASP). These languages help developers to produce dynamically-generated pages quickly by way of high-level programming capabilities that reduce the difficulty of accessing complex functionalities, such as mail agents and database back ends. However, these facilities are obli-

¹Specific experiences might deviate from our description, but that has no implications on our results; and most of the deficiencies that we of privacy enforcement named here can be generalized to other contexts.

ous to security and privacy (see [15], [2]); for example, one cannot establish in a simple way that data received through a certain channel must not be exposed to users.

Since scripting languages have a low learning threshold, many projects are commissioned to inexperienced web developers, with little security insight, that cannot implement privacy policies properly with the available technologies. Novel application security products (e.g., [19], [1]) are of no avail, because they are aimed at reducing security risks but not at enforcing privacy (see Section 2). The only choice left is that (careful) developers implement the policy enforcement requirements ([14]), systematically modifying the code associated to web users' data throughout the complete application ([15]); e.g., introducing security checks and policy-validation commands. Needless to say, this is a time-consuming job, it requires special technical skills (cf. [17]), and its success depends on the coverage of this process (i.e., one mistake can be decisive).

After development has ended, the SO may hire a security audit team to identify and help him to remove weaknesses in the web application; thus improving the overall *strength* of the privacy enforcement. Optionally, he publishes the alleged privacy policy (see, e.g., [8], [7]). (Web users should be informed of the privacy policy, and be guaranteed that it is up-to-date and being enforced.) Finally, the web application is deployed.

On their side, web users can take privacy and security precautions when accessing web applications; e.g., review SSL configuration, cookies configuration, check certificates (See [10]), and read the organization's privacy policy (see [7])). But they have no protection against many privacy threats ([21]). Users must decide if they trust that the alleged policy is being enforced before submitting personal information, or exit the site.

In general, evidence shows that privacy enforcement in web applications suffers from great problems that are not being addressed today. This paper is tasked with providing security officers with a usable tool for enforcing privacy policies.

Organization: The next section describes related work. In Section 3 we describe the architecture for our solution, and we lay the model that describes what are the privacy policies that the solution enforces, and how are these defined by the security officer. In Subsection 3.4 we analyze experiences with a prototype that we implemented, and in Section 5 we summarize our results. In Section 4 we introduce a users' privacy tool that can be used in conjunction with our solution. Finally we discuss future work.

2 Related work

Application security solutions, such as [1], [19], [11], [3], and [4] cannot be used for many privacy tasks. They

protect against (security) attacks that might expose privacy, but do not let the security officer configure privacy policy; this damages their expressiveness and identifiability as privacy systems ([20]). The solution of [16], while interesting in another privacy scenario, cannot be used to solve the problems considered in this paper, since it relies in security requirements (i.e., code security) that cannot be ensured in our (more general) setting.

One must name the *TrustBar* as a usable web-users' tool that protects against spoofing ([10]). We also want to mention *Privacy Bird* ([7]) as prior art and inspirational work in privacy tools for web users, which is a web user plug-in that matches user's (privacy) preferences with the alleged privacy policy of websites using P3P specifications ([6]).

The technique for "augmentation of the execution environment" that we present in this paper has very interesting applications in web-application security. This is the subject of a companion paper ([9]).

3 Solution

In this section we describe the architecture for our solution. From this description readers will understand the underlying model, how does the solution work, and what does it do. We assume that readers have an elementary knowledge of web development². Our description should suffice for an experienced developer to implement it, however, it is not our intention to give full implementation details.

Let us fix a web application platform (e.g., Java, .NET, PHP, Perl, ASP). Assume that our tool is available for protecting web applications developed for this platform, and assume given one such web application, i.e., running code is available. Assume moreover, that the infrastructure is set, i.e., the hardware components are placed, and the software platform and our solution have been installed.

3.1 The augmented execution environment

We call **execution environment** (or **EE** for short) to the (platform-specific) virtual machine/interpreter that performs the actions specified in the web application's source code. The **augmented execution environment** (**AEE** for short), which is the core of our solution, is an enhancement of the execution environment. It includes every functionality from the former but its data structures (e.g., internal objects) and the data-manipulation rules (e.g., defined interface layer) are modified to allow policy enforcement.

We refer to those (instantiated) data structures handled by the EE or the AEE —e.g., numbers, characters, or strings, in local or global variables, and function

²See [13], [18] for definitions and elementary notions in web development.

parameters— as **objects**. Let us assume, without loss of generality, that the structure for any object in the EE is represented by a 2-tuple consisting of *type* and *value*; e.g., the number 63 is represented by the 2-tuple (integer, 63). Objects in the AEE are augmented, and represented by 3-tuples consisting of type, value, and *privacy tag*. A **privacy tag** consists in a sequence of 8-tuples of the following boolean variables: *public*, *store allowed*, *plaintext*, *owner only*, *free₄*, ..., *free₈*—where *free₄*, ..., *free₈* are left to be defined by the SO arbitrarily. Tagging is done according to *operations*.

Every manipulation with objects is called an **operation**. We consider three different types of operations: **inbound operations**, which are the operations that accept input directly from web users, back-end storage, or web services and result in the creation of an object in the AEE; **outbound operations**, which are the operations where the AEE “sends” information (e.g., objects) to APIs or web users; and **internal operations**, which consist in computations with objects within the AEE.

On executing an inbound operation the AEE assigns a privacy tag to each of the associated objects according to privacy policy, and depending exclusively on its source. On executing an outbound operation the AEE can block the associated object from being sent away, and/or log this action, according to the privacy policy, and depending on its privacy tag, destination, and syntax of the associated object/s. The relationship that inbound and outbound operations have with privacy policies is described in detail in Section 3.2.

Internal operations either create a new object or reassign values to existing objects. The AEE tags these objects according to **privacy propagation rules**, which are implemented with respect to a lattice structure on the set of all possible privacy tags (they establish what is the tag for new or reassigned objects).

For simplicity, and without loss of generality, we give details for binary operations. Let A denote a binary operation, and let $c := (t_c, v_c) := A(a, b)$ be the result of applying operation A to the objects $a := (t_a, v_a)$ and $b := (t_b, v_b)$ in the execution environment. Let $a' = (t_a, v_a, \text{tag}_a)$ and $b' = (t_b, v_b, \text{tag}_b)$ be defined by adding privacy tags to a and b . Then this operation is modified by the AEE, so that it produces the augmented object $A(a', b') := c' := (t_c, v_c, \text{tag}_c)$ where t_c, v_c are those defined above and tag_c is computed as $\text{tag}_c := \text{tag}_a \text{ AND } \text{tag}_b$ (where the AND operation is calculated boolean by boolean). For example, if *public*=*true* in tag_a and *public*=*false* in tag_b , then *public* is set to (*true* AND *false*) = *false* in tag_c —which expresses the fact that object c contains information that is private. These rules define the lattice structure mentioned above.

Internal operations manipulating strings receive exceptional treatment because they tag results with sequences of 8-tuples, one per character. For example, string concatena-

tion is modified in order to construct the privacy tags for the new object accordingly. That is, given the string “hello ” with every character tagged with the variable *public*=*true* and given the string “john” with every character tagged with the variable *public*=*false*, their concatenation will return the string “hello john”, where the first six characters are tagged with the variable *public* set to *true* and the last four characters are tagged with *public* set to *false*. To avoid storage and performance penalties, the tool uses a single tag for the string in case all the characters are tagged alike.

Internal operations that compute cryptographic functions, e.g., encrypt or hash, are also exceptional and return a single object tagged with the variable *encrypted* set to *true*.

3.2 The privacy policy

In order to define privacy policy we need to fix the following notions. An **input descriptor** consists of a pair **input type** and **target URL**. The former describes the method in the platform executing an inbound operation, e.g., http-post, http-get or mysql. The latter describes the parameter for that method, e.g., a field in a web form, a field in a database table, a file or directory; it may include wild-cards or be empty in order to specify the default behavior. An **output descriptor** consists of a pair **output type** and **target URL**. Likewise, the former describes the component in the platform (or back end) executing the outbound operation, while the latter describes the elements affected. Privacy rules establish what outbound operations should be allowed or blocked, logged or not, according to the privacy tags and syntax of the object associated to the outbound operation. A **privacy rule** consists of an output descriptor, a set of privacy tags, a syntactic description (e.g., only numbers, not in {/, ”, &, -}), and an **action** to be executed (accept, block, remove sensitive information and accept, and/or log).

During configuration, the SO executes the following steps: i) he lists the *input descriptors*, enumerating every possible inbound operation; ii) he assigns a privacy tag to each input descriptor; iii) he lists the *output descriptors* enumerating every possible outbound operation; and iv) he defines the *privacy rules*. Steps i) and ii) configure the AEE so that it adds a privacy tag to every object associated to an inbound operation. After this is done, the AEE parses the code for the web application and adds HTML tags to the portion of code associated to an input descriptor. When the web application is running, and as users are requested to enter information, this lets them know what are the privacy rules associated to that piece of data. Steps iii) and iv) let the SO define what action to follow for every different outbound operations. Configuration details are stored in a readable file that is accessible by the AEE. The AEE evaluates the privacy rules in order, and applies the first matching rule.

3.3 The Security Officer Interface

The security officer can write privacy policies and perform administrative tasks using a special-purpose interface. He may review the active privacy policy, modify and publish a new one, or review the logs generated by this tool.

The logs are listed in an editor, where each log entry includes the time and type of privacy violation attempt, the rule affected, the module function that tried to execute the action, a *stack trace*, the offending command, and the tags attached to each of its arguments.

3.4 Experience results

We implemented the solution for PHP. Benchmarks according to a standard PHP testing suite indicate a minor performance penalty: the processor usage augmented by %10 and memory only by %18. To test privacy and security enforcement we ran popular open-source content management systems (postnuke and mambo), first without external protection and next with our solution. We developed exploits for the 15 latest publicly known (web application) attacks reported in Bugtraq ([5]) that could be used to violate privacy policies. We inserted errors in the authentication process for these web applications that allowed attackers to violate policy; we discovered that they could only be exploited to this end in the unprotected case. We witnessed that while the exploits worked in the former case, they stopped working when the application was protected.

Several users received a trial copy and reported good experience results. In one case, the logging capability helped us to identify a SQL-injection vulnerability that could have been exploited to reveal sensitive information.

4 The web user's experience

The web user of a protected application can optionally install a browser plug-in, a policy system ([14]), that continuously lets him visualize details for the privacy policy in place; plus it allows him to enforce his own policy. For example, do not submit data to my webmail provider if it will be treated as public.

Every input field in webpages rendered in the user's browser is marked by a privacy icon, depending on the HTML tag associated to it (see Fig. 1). If the user's cursor lies in this field, the toolbar (Fig. 2) displays the privacy category associated to this field, a description of this category, and its privacy icon. The web user can access descriptions for the exact rules used to validate the privacy of operations, and other additional information, using some of the plug-in buttons.

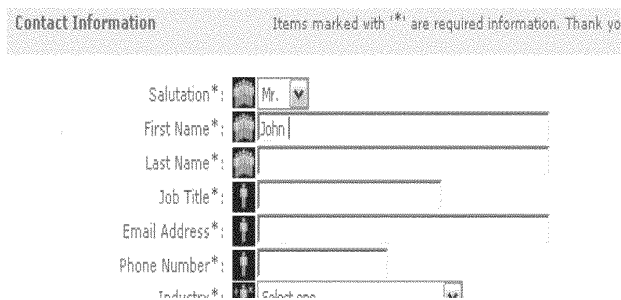


Figure 1. The plug-in working

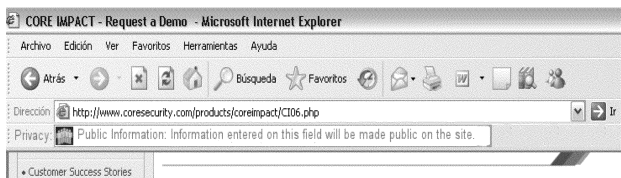


Figure 2. The plug-in's toolbar

5 Results and Remarks

Our main outcome is a technique that allows the web application platform to keep track of the privacy requirements associated to every piece of data. The security officer for a web application can use these tracking capability to specify privacy policy with good expressiveness and appropriate boundaries (see [20]); and, our solution implements this policy efficiently and deterministically.

Our solution lets the security officer define (during configuration) all the inbound and outbound operations with the granularity he requires of the solution. He may attach privacy tags to every inbound piece of data according to its origin, and establish rules for preventing misuse of this data. This simple model allows us to deter most privacy-related attacks, that is, one can express and deter most of the privacy threats.

Protection is deterministic and starts immediately after configuration. Deployment requires neither modifications to the network configuration nor customization of the code of the web application.

The tool can be ported to the platforms of most of the web-application architectures in use today. We chose to implement the solution over PHP, since it is the open-source platform with the largest number of users. However, the details we give will allow a development team to implement the tool *mutatis mutandis* in other platforms—including Java, ASP, Perl, Python, .NET. We have estimated a tolerable performance penalty in terms of processor and memory usage for the implementation of these tools (see Section 3.4 for supporting evidence). Security audits confirmed that the implementation respects the model. As a result, our solu-

tion enforces privacy with very low rates of false positives and false negatives.

We provide a tool for web users: a browser plug-in which presents them with privacy information and allows them to make decisions in terms of their personal privacy needs (cf. [7]).

We wish to mention the following advantages of our approach:

- Security officers can use this tool to implement privacy policy for arbitrary web applications. This fact allows the community of security officers and web developers to experience with this tool and audit its performance and privacy strength —as opposed to *ad hoc* solutions.
- The fact that privacy policy is defined in a separate process to that of web development is very promising on its own; in fact, it requires minimal knowledge in web development and the work needed to do this is proportional to the semantics of the web application (and not the size and clarity of the code).
- There is a very good correlation between the policy configured by the security officer and that enforced.
- In terms of user experience, the information displayed by the users' plug-in is a complete and direct representation of the privacy policy, and not just the SO's privacy requirements write up.

We leave open the problem of studying different privacy tags and preservation rules. An important aspect to be considered is devising functionalities for our solution that allow the access to statistical indicators for private data (but do not harm privacy). This would lead to secure realizations for some privacy information retrieval functionalities, e.g., for use in medical record privacy.

A second problem that we leave for future work is researching implementations that enable us to store data in the back end databases including their privacy tags —of course doing this in our setting where we require no modifications to the source code of the web application.

References

- [1] Airlock. Web-application security, January 2005. URL: <http://www.seclutions.com/>.
- [2] V. Anupam and A. Mayer. Security of web browser scripting languages: Vulnerabilities, attacks, and remedies. In *Proc. 7th USENIX Secur. Symp.*, pages 187–199, January 1998.
- [3] E. G. Barrantes, D. H. Ackley, S. Forrest, T. S. Palmer, D. Stefanović, and D. D. Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *10th ACM Conf. on Comput. and Commun. Secur.*, Washington DC, USA, 2003.
- [4] S. Boyd and A. Keromytis. SQLrand: Preventing SQL Injection Attacks. In M. Jakobsson, M. Yung, and J. Zhou, eds., *Proc. of the 2nd Appl. Cryptogr. and Netw. Secur. Conf.*, volume 3089 of *LNCSS*, Yellow Mountain, China, June 2004. Springer.
- [5] Bugtraq. Mailing list, May 2005. URL: <http://www.securityfocus.com>.
- [6] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. WWW Consortium Recommendation, April 2002.
- [7] L. F. Cranor, M. Arjula, and P. Guduru. Use of a P3P user agent by early adopters. In *WPES '02: Proc. of the 2002 ACM Workshop on Privacy in the Electronic Society*, pages 1–10. ACM Press, 2002.
- [8] L. F. Cranor, M. Langheinrich, and M. Marchiori. A P3P Preference Exchange Language 1.0 (APPEL1.0), 2002. URL: <http://www.w3.org/TR/2002/WD-P3P-preferences-20020415>.
- [9] A. Futransky and A. Waissein. A Dynamic Technique for Enhancing Security and Privacy in Web-based Solutions. Tech. Report, Corelabs, Core Security Technologies, 2005.
- [10] A. Herzberg et al. Trustbar, May 2005. URL: <http://trustbar.mozdev.org>.
- [11] Imperva Software. Total Application Security, January 2005. URL: <http://www.imperva.com>.
- [12] J. Krim. Credit Data Firm Might Close After Database Hacked, Consumers Cancel Contract. *Washington Post*. July 22, 2005.
- [13] L. Lopuck. *Web Design for Dummies*. For Dummies, 2001.
- [14] P. McDaniel. Useless metaphors? Why Specifying Security Is So Hard, 2004. DIMACS Workshop on Usable Privacy and Secur. Software.
- [15] OWASP Organization. The ten most critical web-application security vulnerabilities. OWASP technical report. URL: <http://www.owasp.org>, 2003.
- [16] A. Rezugui, M. Ouzzani, A. Bouguettaya, and B. Medjahed. Preserving privacy in web services. In *The 4th Int. ACM Workshop on Web Inf. and Data Manag.*, Virginia, USA, 2002.
- [17] R. S. Sandhu. Good-enough security: Toward a pragmatic business-driven discipline. *IEEE Internet Computing*, 7(1), 2003.
- [18] J. Scambray and M. Shema. *Web Applications (Hacking Exposed)*. McGraw-Hill Osborne Media, 2002.
- [19] Teros. Web-application security and performance, January 2005. URL: <http://www.teros.com>.
- [20] K.-P. Yee. User interaction design for secure systems. In *ICICS '02: Proc. of the 4th Int. Conf. on Inf. and Commun. Secur.*, pages 278–290. Springer-Verlag, 2002.
- [21] T. Zeller Jr. Black market in credit cards thrives on web, 2005. *New York Times* (late edition). June 21, 2005.

Acknowledgments

The authors want to express their gratitude to D. Tiscornia for improving the system and implementing the benchmark model, and J.P. Martinez Kuhn for auditing

the prototype implementation and analyzing security implications. They want to thank B. Acselrad from <http://www.sleepyminds.com> and Martín Coco for testing and preliminary feedback, and Ivan Arce for his helpful remarks. They are grateful to the anonymous referees for suggestions that helped to improve the presentation of the results of this paper.