SPHERES: An Efficient Server-side Web Application Protection System

Abstract: While the web attacks grow in number and manner, the current web protection methods fail to follow this evolution. This paper introduces a new design of a Web application protection method called SPHERES. The main idea behind SPHERES is that it is placed in the application server, it intercepts the decrypted traffic, and checks it against a set of filtering rules specific to the requests. This design allows SPHERES to have the most accurate picture of the exchanged traffic, the websites structures and workflows, the user sessions and their states, and the system states. This accurate picture of the total system allows SPHERES to build a protection sphere around the website and checks several types and levels of protections efficiently. In addition to the detection of known attacks, SPHERES is able to detect zero-day attacks at runtime. The performance study of SPHERES shows that it is much better than two famous existing web protection tools.

Keywords: Web application security; Protection method; Web application firewall; Owasp; Xss; Csrft; Sql injection.

1 Introduction

While the number of websites in the world increases exponentially and approximates the one billion according to the Internet Live Stats (2014), security companies claim that attackers are moving faster more than defenses. The attackers are upgrading their techniques, while companies struggle to fight old tactics, notices the Symantec 2015 Internet Security Threat Report (2015). According to the same report, advanced attackers continue to favor zero-day vulnerabilities and had discovered 24 zero-day vulnerabilities in 2014. Thus, the need for innovative solutions to protect websites against known and unknown attacks are increasingly in demand. This paper introduces an innovative design of Web Application Firewall that overcomes the current challenges. The firewall, called SPHERES, it is a server-side protection system that acts in real-time to protect websites against any intrusion caused by vulnerable pages or functional flaws.

The paper is organized of the following: Section 2 studies the related works. Section 3 discusses the main points that the web protection is based on and makes some decisions and recommendations in the design of SPHERES. Section 4 introduces the design and the workflow of SPHERES. Section 5 discusses the prototype of SPHERES developed as a module of the PHP application server. Section 6 introduces the filtering method used by SPHERES and introduces the structure of the filtering rules. Section 7 shows how SPHERES acts to protect against the OWASP top ten web application flaws OWASP Top Ten Project
Figure 1: The taxonomy of the web application protection methods.

2 Related works

This section presents the state of the art methods used to protect websites.

Mainly, the web application protection methods could exist at one or more of the three phases of the software development life cycle: the development phase, the testing phase, and the production phase. Figure 1 shows the taxonomy of these methods. During the production/deployment phase, the protection method could be implemented either in the client (the browser), the network, the server, or the operating system.

During the development phase, in order to produce secure web applications, developers usually make use of good design and programming practices. The developer should define the security requirements of the application which depends on the application, but they summed up mainly in the requirements of user management, authentication, authorization, data confidentiality, Integrity, accountability, session management, and standards compliance. Through this phase, developers usually refer one or more security framework such as the general frameworks ISO/IEC 27000-series information security standards, or the more specific ones such as the Payment Card Industry (PCI) Data Security Standard (2013) (PCI-DSS), a set of rules for cardholder data security that applies to any business dealing with credit card transactions. The Open Web Application Security Project (OWASP) has released the version 4 of the its Testing Guide by Meucci et al. (2014) which includes a best-practice testing framework which users can implement in their own organizations and a low-level testing guide that describes techniques for testing most common web application and web service security issues. SANS also continuously updates its SANS Securing Web Application Technologies [SWAT] Checklist (2015). According to the OWASP Code Review Guide (2008), code review is probably the most effective technique for identifying security flaws. A good verification of the cited requirements and checklists disable most known vulnerabilities and insure a protection against most known attacks. However, developers should deal with side-effect of code updates and they usually cannot predict zero-day attacks.

During the testing phase, once a version of the web application is ready, it might be assumed that most known issues have been caught. During the testing phase, a penetration testing provides a last check to ensure that nothing has been missed. The penetration testing could be done manually or automatically. Manual way is perfect when the application is small. It guarantees high effectiveness especially when the tester is a specialist in the security field. However, some web application requires months of security testing. In this case,
security tester can use a web application security scanner which scan web applications to check for known security vulnerabilities automatically. Scanners simulate a malicious user by attacking and probing, and checking what results are not part of the expected result set. As a dynamic testing tool, web scanners are language independent and thus can scan any web application. Moreover, using web security scanners avoids the vulnerabilities found by the attackers using the same tools. However, this kind of method cannot find logical flaws such as information leakage or the use of weak cryptographic functions or inconsistency of complex web transactions. There are dozens of web security scanners most of then with commercial license but also powerful open-source projects that offer good results like Metasploit (2015) and the OWASP Zed Attack Proxy (ZAP) (2015). The penetration testing tools could reveal several vulnerabilities and misconfiguration problems in the web application especially when accompanied with good user assistance. However, they perform predefined tests based on known vulnerabilities and exposures. They fails to protect against zero-day attacks.

In production phase, when the web application is deployed in the Internet, the application will be faced to other kind of attacks and additional vulnerabilities would be raised. The attackers could test many variants of the same known attack, they could search for hidden vulnerabilities, or discover new vulnerabilities and build corresponding zero-day attacks. The web application could be faced to large scale attacks like brute-force attacks, DOS, or DDOS. So that, run-time protection methods are imposed in order to deal with the eventual issues of the new deployment environment, the internet. Note that most of the web applications adopt a multi-tiers architecture. Figure 2 shows the details of the web architecture; The web client is often a browser, or a command-line interface (CLI) for attackers. The front end is the web server which serves static content and potentially few cached dynamic content. For dynamic contents, the web server forwards the request to the application server in order to process and to generate the response on the fly using a server-side language like PHP, JSP, .NET, and Ruby. The server-side script might connect to a database server (the last tier).

The protection method could be implemented in any level of the architecture. First, at the browser level, some kind of simple and generic protection method could exist. However researchers do not focus on the client-side security since attackers usually use
the browser only to scan the web application, to know the functional flow and to perform some probing requests. The large scale scanning and sophisticated attacks are often done with a CLI. A representative work dealing with web application security at the browser level is introduced by Bugliesi et al. (2014) which proposed a security model of a web browser that provides an enforcement of web session integrity. Tateishi and Tabuchi (2007) proposed a new calculus representing browser behavior that prevents information from leakage by means of language-based information flow. The work focuses on the JavaScript language which is widely used to retrieve private information from the browser. Zeng et al. (2010) also work on the monitoring of the behavior of JavaScript execution on the browser. The approach compares the execution to high-level inspection rules and thus prevents from Cross-Site Scripting (XSS) attacks and some session hijacking ones. Jayaraman et al. (2010) demonstrated that the prevailing protection model used in the browsers is inefficient. It is based on the same-origin policy that identifies an application’s origin as a unique combination of <protocol, domain, port> to protect private data like the cookies and the browser state, the DOM, and the native code APIs. The authors propose a new browser protection model that improves the access control of objects and thus mitigates the attacks of access control based attacks such as session hijacking and Cross-site request forgery (CSRF). Dong et al. (2013) proposed a behavior model for diagnosing attacks in Ajax applications, the authors extended the behavior model of a browser in order to capture the contexts where actions take place and the dependency between events. The model is then compared by the normal behavior of the application, that should be provided by the security analyst. When a deviation is noticed, the request is captured and tagged as intrusion. Note that browser-based solutions commonly require the alteration of the source code of the browsers and prevent from few categories of web attacks. Moreover, attackers do not use the browsers in most steps of their intrusions.

The second level of run-time protection of web application is the network level. They are of three categories: network firewall, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS). The network firewall analyzes the network traffic at the first three layers of the OSI reference model. The network firewall rules filter the packet on a protocol/port number basis. The network firewall is stateless and does not record neither analyzes connections, applications, users, etc. The most known network firewalls are the Iptables firewall (2015) for Linux, the IPFW firewall (2015) for FreeBSD and PF (2010) for OpenBSD and Mac OS X. The Intrusion Detection Systems perform deeper analysis that the network firewall, they can inspect contents of the packets to detect known Trojans and viruses when it is a signature-based IDS like Snort IDS (1999), or they can detect abnormal behaviors when it is an anomaly-based IDSs like Bro IDS (1998). When the system is reactive, it detects and reacts against the attacks by closing the connection or by reconfiguring the firewall, it is called Intrusion Prevention System. The network firewall and the IDS/IPS see a given web application as spread packets, they cannot differentiate it from the traffic of another application using the same TCP port. They cannot detect many application layer properties such as application users, user sessions, and service traffic. Moreover, they cannot inspect encrypted data.

The operating system protection level is another alternative that could be used to protect web applications. Good representatives of this category are SELinux (2015) and AppArmor (2015). AppArmor associates with each application a security profile that restricts the capabilities of that application. A profile is a set of privileges (read, write, execute, link,...) and actions for a given path. In addition to manually specifying profiles, AppArmor could run in learning mode in order to build a program’s typical behavior. AppArmor provides
a module for Apache web server that helps in defining a profile for a specific website. However, this kind of security protection sees the application as a process launched with a given user and accessing some resources. It cannot track different users of the same web application, it cannot check the application work-flow, and it cannot analyze the users inputs, the application outputs, and the connections to other servers like a database server.

The most relevant runtime protection method for web applications is the server-side method which is also called Web application firewall (WAF). A WAF is a specialized firewall that handles the traffic of a specific web server. It has a detailed view of the web application including the users, the requests, the responses, the headers and the contents, the decrypted data, the web pages, the website workflow, the server states and its data, and the logs. In general, the application firewalls have high overhead cost since they analyze all the ingoing and outgoing traffic on any OSI layer. However, WAFs focus only on web applications which guarantee a trade-off between overhead cost and deep analysis. The literature includes several works that try to mitigate and prevent from web attacks at server level. For example, there are many detection and prevention methods of SQL injection, they are either signature-based such as proposed by Shanmughaneethi et al. (2009), behavior-based Pinzán et al. (2013), grammar-based Bisht et al. (2010); Liu et al. (2009); Kemalis and Tzouramanis (2008), or taint-based Jan et al. (2010); Alazab et al. (2011); Papagiannis et al. (2011). The XSS detection and prevention methods are also categorized in the same way as the SQL injection detection and prevention methods, they are either signature-based Shanmughaneethi et al. (2009), or behavior-based Sundareswaran et al. (2012), or grammar-based Chandra et al. (2011), or taint-based Avancini and Ceccato (2010). In order to protect against CSRF attacks (also known as XSRF, "Sea Surf", Session Riding, Cross-Site Reference Forgery, Hostile Linking, and One-Click attack), OWASP developed a server-side CSRF protection mechanism for Apache (called mod_csrfprotector), JAVA (called CSRFGuard) and PHP (called CSRF-Protection-PHP). The idea behind the OWASP mechanism is to create a CSRF token to be transmitted between the browser and the server, the server will verify that the page from which the request originated is an authorized page based on the validation of the token. Rocchetto et al. (2014) used the formal language ASLan++ Oheimb and Mödersheim (2011) to write a model for each web application, thus allowing the identification of CSRF at design phase and then attach CSRF tokens to specific points to avoid such attacks. Padmaja (2014) uses a random number generator to create a restricted system thus dealing with broken sessions, insecure direct object references, sensitive data expose, and missing access control.

There also general methods that try to detect web attacks. Devdatta et al. (2010) suggests a formal model for web security including three parts: concepts, threat models, and security goals. The model is general and could not detect application-specific attacks. However, it would only detect simple new vulnerabilities of the application. Christopher et al. (2005) is one of the first anomaly-based general method that detects web attacks. The method performs a learning phase on a training data set containing web queries to extract the normal models of nine features of the analyzed traffic: attribute length using the The Chebyshev inequality ter (2001), attribute character distribution, structural inference for irregular attribute values using a Markov model, token finder to determine whether the values of a certain query attribute are drawn from a limited set of possible alternatives, attribute presence, attribute order, access frequency for each request and for each client using the same method as the attribute length model, interrequest time delay using the Pearson$\chi^2$ test, and invocation order using a Markov model. A global probability will
be then computed based on the probabilities of each cited models, which consists on the anomaly score of the request.

Faced to this diversity of web protection methods that exist in the state of the art, we note a big gap between the literature methods and the trivial signature-based methods used by most WAFs deployed in reality.

3 Discussion

In this section, we will discuss the current state of the art of the web application protection methods. Their advantages and disadvantages in order to deduce the outline of a new efficient protection method. First, whatever the phase and the level of the protection, the methods presented earlier are either signature based or anomaly based, the grammar-based and the taint-based methods could be seen as anomaly-based methods since the grammar-based methods observe the deviation of the queries from their normal behaviors and the taint-based methods observe the deviation of the inputs from their normal uses. The signature-based methods require a continuous update of new attack signatures and they usually cannot detect new threats and attacks. Generic IDSs like Snort may include more than five thousands of signatures to be checked with every packet at run-time, application-specific protection tools like ModSecurity may include more than 250 signatures. The anomaly-based IDS could detect new attacks if their corresponding traffic is different from the defined normal behavior, but the methods in this category are generally characterized by a high false positive rate. This is due to the gap between the defined normal behavior and the effective normal behavior of the protected subject. The more the gap becomes bigger, the more the false positive rate is. Nevertheless, the anomaly-based methods could be better than the signature-based methods if the gap is negligible and the normal behavior domain is very restricted.

Regarding the protection along the software engineering phases, the offline protection methods (performed during the development phase and the testing phase) have similar advantages and drawbacks. They can help to identify known threats but they usually fail to protect against new attacks. The efficiency of the online protection methods depends on the protection level; in general, the more the protection method is close to the web application, the more the efficiency of the method goes up. For example, the network intrusion detection system and the OS firewall cannot deal with encrypted data transmitted between the web client and the web server. WAFs actually perform good results as they intercept real traffic and scan it on the fly, they have also access to all the details of the web applications without any encryption. Moreover, the system overhead of WAF process is less than other systems like the OS firewall or the NIDS because the WAF process tracks only the web server process. However, the current implementations of the WAF modSecurity (2015); ironBee (2015); PHPIDS (2015) are reduced to some trivial filtering techniques of the inputs of the web applications. Some advanced WAFs add some filtering techniques of the outputs and the log of the web applications. These implementations are usually signature-based and provide some simple anomaly detection rules.

Finally, there is a big gap between the trivial methods used by the existing WAFs and the sophisticated methods proposed in the literature. The current paper tries to bridge this gap and propose a new design of a WAF that overcomes the cited problems and helps the research committee to easily integrate their work in a flexible and efficient protection framework called SPHERES.
4 Design Of SPHERES

From now on, we focus on WAFs as they have good impact in the protection of web applications. In this section we introduce SPHERES, a web application firewall designed to protect websites against known and new attacks using a flexible and efficient methods, yet with low overhead. SPHERES puts the protected website into a sphere in a way that every entry point and every exit point of the website could be easily checked without overwhelming the network or the web server with unnecessary work. In contract to regular WAFs, which they sits between the user and the web server, SPHERES surrounds the website with a spherical wall that could be defined with simple filtering rules.

In the literature, a WAF could operate in several distinct modes: a reverse proxy, a layer 2 bridge, a network monitor, or a web server module. In the first two modes (see figure 3), the WAF works in line between the web server and the web users, considers the web server as a black box, checks the traffic against general filtering rules and blocks unwanted traffic. Using these two modes, the WAF could not check specific rules such as the website workflow, the programming language statements, and the per-user actions. The network monitor mode has the same characteristics as the first two modes but the WAF works in passive mode with few reactions and it removes the possible point of failure. In the last mode, the WAF runs as a module to the web server or the application server. SPHERES adopts this mode since it has a total control of the websites and their properties.

The figure 4 shows where SPHERES is placed and how it interacts with the other components of the web system. SPHERES sits behind the web server in order to handle dynamic web pages which have full privilege to read, write, or delete any assets of the servers. The static web pages could be also handled by SPHERES with an easy configuration in the web server. Since SPHERES works close to the application server and the websites assets, it have full control not only of the inputs and the outputs of the websites but also the detailed structure of them, allowing SPHERES to check the normal workflows of the websites, the behavior of each user separately, the accesses to the file system (FS), to the databases (DBMS), and the network, which constitute the main difference between SPHERES and the other WAFs.
SPHERES puts each website in a protection bulwark called sphere. The sphere controls not only the inputs and the outputs of web pages but also the transactions of the website with the server itself (the file system, the databases, and the network) and the normal workflow of the website (see figure 4). SPHERES can control the execution of each website script and tracks the code that could change the state of the server or access sensitive data. During the execution, SPHERES differentiates between the processing code and the Input/Output code. If the web script file is authorized to run by SPHERES, its processing code including for example logic statements, loops, and encoding conversions, does not access to or change the state of the server, thus SPHERES does not check it. However, the threats, when exist, are caused by the I/O code that communicates with the web client and the components of the server.

The protection sphere of a given website defines its normal usage. If the web user deviates from the normal usage, SPHERES considers it as an attack, and raises an alert. Despite other protection methods which they work either in a prevention mode or in a detection mode, SPHERES can work in both modes simultaneously. For example, it could block any traffic attempting to create new script files and it only reports that the user is asking an image that does not exists in the file system.

The protection sphere consists of the following properties:

1. website structure: normal domain of the structure of each website;
2. inputs domains: normal domains of the scripts arguments;
3. scripts outputs: normal domain of each script output;
4. workflows: normal workflow of the website;
5. scripts privileges: normal function of each script;

A well defined sphere is a global and solid bulwark that could detect any unusual behavior. The first property of the protection sphere is the website structure which specifies mainly the front-end scripts called by the web users and eventually the back-end scripts used by inclusion or import. For example, considering a website based on the famous Wordpress
CMS, the front-end script used to consult the posts and search in them is only one script, in order to add a new entry in the blog or leave a comment, the user may use only four scripts, all the remaining scripts are used in back-end. A direct access to a script that does not belong to the front end scripts would be considered as an attack.

The second property is the domains of the inputs which defines the legitimate values of each parameter of each web page. The domains could be defined using the positive security model or the negative security model (see section 6 for more details). The domains could be defined using regular expression. If SPHERES knows the domain of a given parameter, it will block any URL that includes abnormal values such as SQL injection patterns or XSS patterns. The definition of the domains could be tricky, but it will be defined one time. The number of parameters could be high for a given website but the number of patterns is usually low including for example number identifiers, passwords, enumerations, and general texts that should not include JavaScript code or SQL statement termination characters without escaping.

The third property is the scripts outputs which specify the normal output of the web pages. This property should be used in special cases only, since it requires the scan of the whole output including its dependencies such as JavaScript files and CSS files which may take a lot of time and degrades the performance of the server.

The fourth property of the protection sphere is the normal workflow of the website. This is an original property of SPHERES that allows to track the normal order of the script calls. The normal workflow of a given website could be defined with a directed graph where the nodes are the pages and the links are the call precedences. SPHERES gives the administrator an easy way to define such a graph; for a given web script, the administrator have to specify only the inward edges and outward edges of the nodes. The inward edges are the referrers of the page and the outward edges are the redirections of that page. A quick scan of the links and the web forms actions of the script is enough to know both information. The use of regular expressions in order to define the referrers and the redirections allow the aggregation of several values into one pattern. A good definition of the website workflow and a good verification of that workflow would prevent from many dangerous attacks such as CSRF and session hijacking especially for E-commerce websites.

The last property of the website protection sphere includes the scripts privileges. This property is used by SPHERES while intercepting the bidirectional traffic between the web application server and the assets of the server itself including the server file system, the databases, and the network communications. The property tells SPHERES what kind of accesses are allowed or not allowed between the website and the server itself based on three folds: the functions provided by the application server to the developer (for example, PHP functions, JSP methods...), the arguments of the functions, and the returned result of the functions. A focus was made on the accesses that could change the state of the server. For example, in a PHP web application server, the administrator can configure SPHERES in a way that the whole website scripts (/var/www/*) cannot use MySQL, thus blocking any function starting with mysql (mysql.*). SPHERES allows also to restrict the result of any function. For example, the MySQL query function (mysqli_query) in a login page should not return more than one record. SPHERES can check even the arguments of any function. This is useful for example to check the account used in the database connection phase or to analyze the effective SQL request sent to the database server.

Figure 5 summarizes the main execution steps of SPHERES. As soon as the new web script is called, SPHERES checks if the script is allowed, then checks the parameters of that script. The current supported parameters are the GET, POST and COOKIES. If the values
of the parameters belong to their normal domains, then SPHERES checks the validity of the referrer of the request. Then, SPHERES controls the execution of each function specified in the filtering rules (the function call, its arguments, and its result). The final check is the output of the web script. Note that some checks could be skipped according to the filtering rules.

If SPHERES detects an attack, it can block the traffic by canceling the request, logging the details of the attack, and changing the configuration of the network firewall in order to block the IP of the attacker for example.

5 SPHERES for PHP-based websites: a case study

In this section we will introduce the prototype of SPHERES developed on top of the PHP stack.

Figure 6 shows the PHP stack. The Web server abstraction layer called SAPI simplifies the task of adding native support for new Web servers. SAPI gets the PHP requests from the web server and forwards them to the PHP core component. The PHP core parses the called PHP script with the help of the Zend Engine, executes it, and sends back the output to SAPI. SAPI forwards the output to the registered web servers. The PHP stack offers an advanced and flexible API to the PHP extensions. As shown in the figure, the extensions in PHP have full access to all components of the stack. The extensions may add extra features and additional functions to PHP. SPHERES is implemented as an extension that does not
provide additional functions to PHP thus it guarantees the portability of the website code. However, SPHERES intercepts the traffic according to the filtering rules and checks it.

SPHERES intercepts the web requests from SAPI in order to check the URL request parameters, the cookies, and the server variables. In this context, SPHERES insures that the called script is allowed, its variables has normal values and correct sizes, and they do not include suspicious character like the null character that indicates the end of the parameter string. The web page referrers are also checked at this stage since PHP considers them as a server variables. The current prototype can check the parameters of the uploaded files only. Future versions of the prototype of SPHERES should check also the uploaded files and their effective mime types.

SPHERES tracks the function calls from the Zend component by intercepting and rewriting the Zend function called zend_execute_internal. As mentioned earlier, for each function specified in the filtering rules, SPHERES can check the function call, its parameters, and its result. At this stage, SPHERES checks also the included files since PHP includes them by calling the inclusion functions like require_once and include_once. If a rule specifies the prevention mode for some functions, SPHERES will skip the execution of that functions when they look suspicious and returns empty strings as results.

Note that all the cited checks performed by SPHERES are simple with small overhead. However, in case of attack, SPHERES should perform Input/Output calls in order to log and eventually to change the configuration of the network firewall.

6 Filtering Rules: How Should They Be Defined?

This section discusses one of the main component of WAFs which is the filtering rules. Whatever their method, their phase and their level, the web application protection solutions include a common component called filtering rules. All the other components of the WAFs try just to apply efficiently those rules.

A filtering rule could be based on a positive security model (whitelisting) or a negative security model (blacklisting). In the first case the rule defines what is allowed, and rejects everything else. The benefit of using a positive model is that old and new attacks will be prevented. On the other hand, the negative model will prevent only the old attack defined in the rule. The number of negative signatures usually grows fast and has to be maintained and updated. For example the Snort IDS includes more than five thousands of rules. It is clear that the positive model is preferred, but in some cases it is hard sometimes impossible
to define a whitelist for a given subject. In this case, the negative security model is the only remedy. For example, if we want to describe a rule to protect a web application input called ID which refers to the blog post number, it will be easy to use the positive security model in order to define a regular expression describing a number. So that, any value of ID that is not a number is considered incorrect. However, it is hard to apply a positive pattern that describes a form text field because the domain of the field is very large. In this case, the negative model is an alternative solution. Nevertheless, it would be better to use another protection technique such as taint control to enforce a normal manipulation of that kind of user inputs. In conclusion, to protect web applications, both positive and negative security models are required. In the same context, PCI DSS recommends to enforce both positive and negative security models Payment Card Industry (PCI) Data Security Standard, requirement 6.6 (2008).

Another important property of the filtering rules is the scope of the rule which could be general or specific. General rules will be checked for every entry. The more the rule is general, the more it is involved in the check step, the more the overhead increases. For example, PHPIDS version 0.7 comes with more than seventy general rules that would be used to check every request (GET, POST and/or COOKIE requests depending on the general configuration of PHPIDS). ModSecurity includes more than 250 rules but a rule would be placed in one specific phase among five possible phases (Request headers, Request body, Response headers, Response body, Logging). Moreover, the current WAFs usually work as a reverse proxy or as a module to a web server thus the rules could not handle specific programming languages or specific website workflows. However, specific rules requires more work to be defined and is adapted to the stable websites that does not change its sources frequently. Automatic generators of the rules are good helpers to the administrators to describe the rules of a given website.

Based on the previous discussion and the cited recommendations, we introduce a flexible and efficient filtering rule design. The rules in SPHERES are hierarchic in a way that inner rules can inherit from outer rules. Since the overall structure of the websites is hierarchic, the rules will be written in a XML document that has a tree structure by essence. The hierarchy of the rules is as follow: the rules is a set of filters, each filter is composed of six elements starting by the element PATH which can hold a regular expression value that encompasses several effective paths of the web server root. The second element is MAINFILES which also can be expressed as a regular expression pattern. MAINFILES specifies the front-end files that could be requested by end users which excludes the other files used in back-end and might be the entries of some intrusions. The third element is the actions that SPHERES would take in case of intrusion. The user can ignore this element and its value will be inherited from outer elements. The action includes the alert type (from ERROR to INFO), the log type (file, database, email), and the action mode (either to prevent the current traffic or no). The fourth element is the REQUEST HEADER which specifies the normal values of the GET, the POST, the COOKIES and the REFERRER parameters. The elements REQUEST BODY, and RESPONSE BODY checks the payloads of the request and the response. The element RESPONSE HEADER specifies the values of the response tags.

The last filtering element is FUNCTIONS which is one of the most important and original property in SPHERES. This property tracks the behavior of the functions of the web script while executing. The administrator should add the critical functions that could change the state of the servers or could read sensitive data. Inside this element, many functions could be specified with their values patterns of its arguments and the pattern of its output.
For example, using the hierarchical structure of the filters file, the access to the database could be restricted to a specific user with a specific query pattern for all the website scripts.

As mentioned earlier, any rule describing the protection sphere could be configured either in detection mode or in prevention mode. This configuration could be also specified for the whole website in general and the inner rules inherit the actions from the outer ones. For example, the actions could be specified just one time at the website level. However, some rules could have specific actions that override the inherited ones.

There is an extra element called disallowothers that could be included in any of the previous elements. When enabled, SPHERES will alert the call of any other script (when included in the FILTER element), other parameter (when included in the REQUESTHEADER element), other function not specified in the rule and whose corresponding script belongs to the path. This is very important to prevent inclusion attacks at many levels. However, it should be used only when all the existing scripts, parameters, or functions used in the corresponding context are added in the filtering rule.

Figure 8 shows an example of filtering rules written for the WackoPicko vulnerable website (see section 8 for more details). The first filtering rule target the main file high_quality.php of the directory /var/www/html/wackopicko/pictures/ and specifies that the script takes one number and should come from a link in the page confirm.php. The second part of the rule specifies that the script cannot call any function of the MYSQLI module. The rule also specifies that the script could open the files existing in the folders of /var/www/html/wackopicko/upload/* only for reading. The tag skip is turned on to enforce the prevention mode. There is another element in the filtering rule called disallownullchar when turned on SPHERES will consider the null character in the parameters as an intrusion.

The rules will be formatted in XML which allows to validate it offline against the XSD and so that to decrease the overhead of the online system. The filtering rules file seems to be hard to write. However, in practice it could be easily prepared and deployed. Infact, there are two types of websites: websites based on Open source code and websites based one closed course code. In the first case, the website uses a CMS developed by a large committee and used world wide. The development committee could easily create the filtering file one time.
<filter>
  <id>1</id>
  <path>/var/www/html/wackopicko/pictures/</path>
  <mainfiles>high_quality.php</mainfiles>
  <requestheader>
    <disallownullchar>1</disallownullchar>
    <disallowothers>1</disallowothers>
    <logtype>1</logtype>
    <alerttype>5</alerttype>
    <skip>1</skip>
    <get>
      <var><name>picid</name><maxlen>3</maxlen>
      <valuepattern>^[0-9]+$</valuepattern>
      </valuepattern>
      </get>
      <server>
        <var><name>referer</name>
        <valuepattern>http://localhost/wackopicko/cart/confirm.php</valuepattern>
        </valuepattern>
      </server>
  </requestheader>
  <functions>
    <action><logtype>1</logtype>
    <alerttype>4</alerttype>
    <skip>1</skip>
    <func>
      <authorized>0</authorized>
      <fname>mysqli.*</fname>
    </func>
    <func>
      <authorized>1</authorized>
      <fname>fopen</fname>
      <args>
        <arg><position>1</position>
        <valuepattern>/var/www/html/wackopicko/upload/.*</valuepattern>
      </arg>
      <arg><position>2</position>
      <valuepattern>r</valuepattern>
    </args>
  </func>
  </functions>
</filter>

Figure 8  Example of filtering rules
for each version for all the users. In the second case, the website is written by specialists in web development generally in large companies which have not any problem of expertise or time or budget to write the filtering file. Moreover, it will be very easy and feasible to build a user interface helper that provides trivial categorization of the regular expressions and the critical functions and that helps to generate the rules. For example the rules written for the famous WackoPicko vulnerable website include only twenty unique regular expressions. Most of them are simple and trivial to write.

7 How SPHERES could prevent web attacks

In this section we will discuss how could SPHERES prevent the web attacks. For that purpose, we will introduce the most critical web risks and intrusions and we will show how SPHERES can prevent their corresponding attacks. The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted OWASP foundation (2015). One major product of OWASP is the OWASP top ten which presents the most critical web application security flaws to be updated every three years. The most recent top ten web security risks are:

A1 â‚¬ Injection: Is the most critical risk that a web application could be faced to. Attackers target either the commands or the queries in order to inject untrusted data. It mainly includes four types of attacks: SQL injection, Code injection, command injection, and XML injection. The impact of these flaws is severe and could lead to code execution on the web server, code execution on the client-side such as JavaScript, Denial of Service (DoS), and Sensitive Information Disclosure. The injections comes from any source of data. If the data source of a given web script is the script parameters, then good patterns of the script parameters could be enough to protect the system. But if the data comes from other sources, the filtering rules should focus on the functions that use the input data.

A2 â‚¬ Broken Authentication and Session Management: Since the HTTP protocol treats every user request as a new connection and does not keep the state of the user session, the web APIs propose some workarounds to make the web application statefull using session mechanisms. The mechanism allows to create a session token for each user (generally after a successful log in) which is sent to the browser using cookies and communicated with every request. However, this design of session mechanism introduces several risks and gives to the attackers many opportunities to impersonate the web users by stealing the session tokens (session hijacking) or predicting it (session fixation) or doing a brute force attack. The same attacks could be applied for the authentication credentials as well. Since the implementations of the authentication and session mechanisms are different from website to another and from developer to another, it will be difficult to have common solution for their flaws. However, if the developer decide to use a solution that enhance the security of user sessions such as Sessionlock Adida (2008), Sec$ess De Ryck et al. (2015), One-time cookies Dacosta et al. (2012), Origin-Bound Certificates Dietz et al. (2012), Http integrity header Hallam-Baker (2012), or BetterAuth Johns et al. (2012), it will be very easy to define their normal behavior using the request properties cited in the protection sphere above.

A3 â‚¬ Cross-Site Scripting (XSS): During this type of attack, the attacker sends a malicious script to an unsuspecting user. The user launches the script in the browser and executes it. Depending on the malicious script, it can access any data in the user browser like cookies, session tokens, or other sensitive information used with that site. These scripts
can even modify the content of the output page. There are three types of XSS: stored XSS that stores the malicious code in the server and resent in later pages to other users, reflected XSS that forces the server to reflect immediately the malicious code without storing it, and DOM XSS targets the pages that modify itself in the browser without even contacting the server and injects the malicious code in the modified parts. Since the reflected XSS and the stored XSS attacks use the request parameters to inject the code, SPHERES can easily prevent these attacks by specifying the normal values of each request parameter. The stored XSS could be also detected while calling the server-side functions. Regarding DOM XSS, since the attacker tries to inject the code in the URL or the referrer for example, SPHERES can detect these kind of injections since it checks these page properties (URL, Referrer) before the page is sent to the browser.

**A4 âž¢ Insecure Direct Object References**: This risk is caused when accepting the value of a parameter without validating it. The attacker manipulates the parameter’s value to refer to an object that should not be accessed by the current user. SPHERES protects this kind of attack by specifying a normal pattern for that parameter. However, the value specified by the attacker could verify the pattern or not. When the the value does not verify the pattern, SPHERES will easily detect such attacks. This is the case of path traversal attack for example, where the attacker inserts the path traversal characters like (../) in a parameter that accepts a file name for example.

The existing WAFs could not detect the attack when the manipulated value matches the pattern. In this case, the pattern is for example a three-digits number (\[0 \- 9\]1, 3$); the regular value of the current user is for example 1, and the attacker change it to 2 which is also a number to access information related to another user. SPHERES cannot detect such intrusion when checking the parameter’s pattern. However, it could be detected while parsing the function arguments. The privileges for the objects could be specified in the FUNCTION element of the filtering rules. In the WackoPicko example above, a rule is build for the function fopen in order to restrict the read mode of the images that exist only in the directories (/var/www/html/wackopicko/upload/.*). The problem grows more when the objects have per-user privileges. Also when the reference to the object is an indirect reference. In these cases, some rules could be specified at the functions level or the pages outputs.

SPHERES introduces an elegant method to specify privileges on objects and functions where the roles are easy to describe. The idea is to write a rule that includes a variable (a user role for example) which is evaluated at runtime. The variable could be included in the regular expressions used later in the function level or in the headers. For example, the following two rules could be created to call the same page (products.php) with two different roles each have different privileges on some records in the database:

1. the user can call the mysql_query function only with a SQL SELECT statement targeting a specific subset of a database table if it is a regular user:

2. the user can call the mysql_query function using any SQL statement targeting a specific subset of a database table if it is an administrator of the website:

These rules uses the variables (@@user) that could be evaluated at runtime. The value of the variable will be injected on the fly in the regular expressions depending on it.

**A5 âž¢ Security Misconfiguration**: In general, this flaw exists outside the website itself such as the configuration files of the web server, the application server, and the database. Our proposed WAF is designed to protect only the website and its flows. Nevertheless,
Figure 9 Example of filtering rules specifying per-user roles
**Figure 10** Example of filtering rules specifying per-user roles
some websites use some frameworks that come with default settings such as the Content Management Systems and the web-based database clients. Since these frameworks are web based, it will be feasible and useful to write their filtering rules and thus any misconfiguration will be detected by SPHERES.

A6 a. **Sensitive Data Exposure**: This risk leads to steal or modify sensitive data using SQL injection attacks, eavesdropping the traffic if it is not or not well encrypted, or hijacking the user’s session. All of these attacks are discussed above. While writing the web scripts, the sensitive data should be handled with care, stored using advanced encryption algorithms, and the cryptographic keys are exchanged with strong mechanisms. Many of these prevention methods could be described using our filtering rules and thus checked at runtime by SPHERES.

A7 a. **Missing Function Level Access Control**: Missing function level access control is close to the risk of insecure object references. Instead of accessing unauthorized functions, the attacker tries to access unauthorized functions by manipulating the URLs. Some examples of attacks that exploit this risk is forced browsing, path traversal, and path manipulation. The same protection methods used with the insecure direct object references could be used here. The problem remains when the functions are provided on a per-user basis. This case requires a role management at web script level. However, if the role privileges are easy to describe, they could be defined in the filters by defining different function requirements depending on different requests such as restricting the objects in figures 9 and 10. Rather than restricting objects, the rules would restrict functions and services.

A8 a. **Cross-Site Request Forgery (CSRF)**: In order to perform such attack, the attacker provokes a victim to click on a link (by sending it by email or posting it in a forum for example) which invokes directly or indirectly a sensitive action in a website. The action will be performed using the victim’s credentials in favor to the hacker. In contrast to other attacks, CSRF traffic seems legitimate since it is generated by the victim with his credentials and with his browser.

This attack has high impacts and could lead to critical transactions on the targeted website. An easy way to detect such attacks is to analyze the referrer of the web pages which should belong to the website’s domain. If the developer of the website uses a token library such those introduced by Rocchette et al. (2014) and Padmaja (2014), SPHERES will easily insure the normal execution of their mechanisms since they usually inject checkable tokens in the cookies, in the forms or in the links.

A9 a. **Using Components with Known Vulnerabilities**: The use of non web-based components is out of the scope of the WAF which aims to secure the websites against using components with known vulnerabilities as sources of the attacks. However, when the used components are web-based, the filtering rules for these components should be written using the SPHERES logic. Therefore, the websites that uses the vulnerable components will be protected.

A10 a. **Unvalidated Redirects and Forwards**: This application flaw might exit if the functions of redirect and forward are used and involve user parameters while calculating their destination URL. SPHERES could detect such behavior at two levels; while checking the user parameters and while calling the functions of redirect and forward. The list of URL that the website might use are usually limited, could be generated using simple parsers, and could be easily described with SPHERES rules.

This section discussed the flaws according to the OWASP top ten. We show that SPHERES can detect and prevent almost all the corresponding attacks that could exploit these flaws. Moreover, SPHERES can detect several attacks not cited above such as the
shell upload attack, file inclusion, and any other attack that contraries the normal behavior of a website.

8 Performance Analysis

The prototype of SPHERES is implemented as a module of the PHP web application server. The first part of the performance study is to compute the overhead of SPHERES. For this aim, we used the benchmarking script *bench.php* that comes with the source of PHP used to test the overhead of the PHP modules against dozens of simple and time-consuming methods. The tests are launched in a regular DELL laptop with an Intel Core I3 CPU M330 @2.13 GHZ using 6 GB RAM running a Ubuntu 14.04 LTS 32 bits distribution as a VirtualBox virtual machine. SPHERES is developed as an extension of PHP version 5.6.6. A set of 18 filtering rules is written for the vulnerable website WackoPicko Doupé et al. (2010) is fed to SPHERES. First, *bench.php* is executed without loading SPHERES which gives the baseline values of the durations of each method. Then, the same benchmarking file is executed after loading SPHERES. The table 1 summarizes the overhead results. The results shows that SPHERES adds an extra overhead to PHP close to 15% which is considered low.

<table>
<thead>
<tr>
<th>Bench method</th>
<th>without SPHERES</th>
<th>SPHERES</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>0.852</td>
<td>0.776</td>
</tr>
<tr>
<td>simplecall</td>
<td>0.826</td>
<td>1.268</td>
</tr>
<tr>
<td>simpleucall</td>
<td>0.879</td>
<td>1.222</td>
</tr>
<tr>
<td>simpleudcall</td>
<td>0.847</td>
<td>1.141</td>
</tr>
<tr>
<td>mandel</td>
<td>1.583</td>
<td>1.994</td>
</tr>
<tr>
<td>mandel2</td>
<td>2.067</td>
<td>2.111</td>
</tr>
<tr>
<td>ackermann(7)</td>
<td>0.784</td>
<td>0.892</td>
</tr>
<tr>
<td>ary(50000)</td>
<td>0.175</td>
<td>0.168</td>
</tr>
<tr>
<td>ary2(50000)</td>
<td>0.136</td>
<td>0.177</td>
</tr>
<tr>
<td>ary3(2000)</td>
<td>1.188</td>
<td>1.289</td>
</tr>
<tr>
<td>fibo(30)</td>
<td>2.066</td>
<td>2.820</td>
</tr>
<tr>
<td>hash1(50000)</td>
<td>0.258</td>
<td>0.371</td>
</tr>
<tr>
<td>hash2(500)</td>
<td>0.277</td>
<td>0.276</td>
</tr>
<tr>
<td>heapsort(200000)</td>
<td>0.803</td>
<td>0.766</td>
</tr>
<tr>
<td>matrix(20)</td>
<td>0.746</td>
<td>0.779</td>
</tr>
<tr>
<td>nestedloop(12)</td>
<td>1.241</td>
<td>1.142</td>
</tr>
<tr>
<td>sieve(30)</td>
<td>0.737</td>
<td>0.735</td>
</tr>
<tr>
<td>strcat(200000)</td>
<td>0.121</td>
<td>0.119</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>15.587</strong></td>
<td><strong>18.045</strong></td>
</tr>
</tbody>
</table>

Table 1 Overhead analysis of SPHERES: column 1 is the benchmarking method as in (bench.php), column 2 is the duration of the method without loading SPHERES, and column 3 is the duration of the same method when SPHERES is loaded.

In order to study the performance of SPHERES from a classification point-of-view, we developed a web security benchmarking tool (that will be introduced in a separate paper) called WebSecBench. The figure 11 shows the architecture of WebSecBench. WebSecBench is composed of three components:
Figure 11: The architecture of the web security benchmarking tool WebSecBench.

• a vulnerable website called WackoPicko2 which is a modified version of the vulnerable website WackoPicko DoupÉ© et al. (2010);

• attack scenarios: each attack scenario is a complex test case that may include several steps to gain the attack objective. Each scenario is classified in one or more OWASP to ten risks. The scenarios include also a set of legitimate requests; The scenarios is written specifically to WackoPicko in order to know exactly if the attack has succeeded or failed.

• The attack traffic generator that reads the attack scenarios, generates the corresponding requests, send them to the WackoPicko server, wait for the responses, analyzes them, and reports the statistics.

The attack scenarios exploits the vulnerabilities of WackoPicko2 in order to perform 218 complex attacks and 22 legitimate scenarios. Each attack scenario may include several steps and the intrusion result. WebSecBench generates the traffic of each scenario, if the intrusion result is received it increments the False Positive ratio, Otherwise it increments the True Positive ratio. For legitimate scenarios, WebSecBench increments either the True Negative ratio in case that the security tool did not intercept the traffic or the False Positive ratio otherwise. At the end of the benchmarking, WebSecBench reports for each OWASP top ten risk and for the legitimate scenarios the overall ratios.

The famous ModSecurity modSecurity (2015) WAF and PHPIDS PHPIDS (2015) are installed separately and tested with WebSecBench in order to compare their performance results with those of SPHERES. The performance results of the three protection solutions are summarized in the confusion matrices below.

Each confusion matrix includes two classes (attack and legitimate) and shows the following classification metrics:

• **True positives** (TP): These refer to the attack scenarios that were correctly labeled by the security solution.
• **True negatives** (TN): These are the legitimate scenarios that were correctly labeled by the security solution.

• **False positives** (FP): These are the legitimate scenarios that were incorrectly labeled as attacks.

• **False negatives** (FN): These are the attack scenarios that were mislabeled as legitimate.

• Given an element $n_{ij}$ of the confusion matrix, $n_{ij}$ is the number of scenarios of class $c_j$ predicted in the class $c_i$. **Precision**: is the proportion of the true positives against all the positive results (both true positives and false positives), the precision for the class $i$ is:

$$Precision_i = \frac{n_{ii}}{n_{ii} + FP}$$

• **Recall**: called also the true positive rate, is the proportion of actual positives which are predicted positive (the sum of true positives and false negatives), the recall for the class $i$ is:

$$Recall_i = \frac{n_{ii}}{n_{ii} + FN}$$

• **The overall accuracy**: the overall fraction of correctly classified examples (both true positives and true negatives):

$$Overall\ Accuracy = \frac{\sum_i n_{ii}}{\sum_i \sum_j n_{ji}} = \frac{TP + TN}{\sum_i \sum_j n_{ji}}$$

The classifier (the security tool under benchmarking) should maximize precision and recall. However, these goals are incompatible. A classifier can maximize the precision by only making positive predictions it is sure about. But this decision worsens recall. To maximize recall, the classifier would maximize the risk while making predictions, which makes the precision smaller.

<table>
<thead>
<tr>
<th>PHPIDS Truth</th>
<th>Attacker</th>
<th>Legitimate</th>
<th>Prediction overall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prediction</strong></td>
<td>TP: 105</td>
<td>FP: 5</td>
<td>110</td>
<td>95.45%</td>
</tr>
<tr>
<td><strong>Legitimate</strong></td>
<td>FN: 113</td>
<td>TN: 17</td>
<td>130</td>
<td>13.08%</td>
</tr>
<tr>
<td><strong>Truth Overall</strong></td>
<td>218</td>
<td>22</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>48.17%</td>
<td>77.27%</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Overall Accuracy</strong></td>
<td>50.83%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

According to the confusion matrix of PHPIDS in table 2, among 218 attack scenarios, PHPIDS recognizes only 105 one, leading to a very low value of the recall (48.17%) for the attacks. For legitimate scenarios, the recall value is better (77.27%) but stills low. The overall accuracy of PHPIDS is 50.83%. PHPIDS demonstrates good results when the attacks are simple SQL injections or XSS, but it fails with sophisticated attacks exploiting sessions flaws, misconfiguration, CSRF, and abnormal website workflows.
Table 3 The confusion matrix of ModSecurity.

<table>
<thead>
<tr>
<th>ModSecurity</th>
<th>Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Attack</td>
</tr>
<tr>
<td>Prediction</td>
<td>TP: 179</td>
</tr>
<tr>
<td></td>
<td>FN: 39</td>
</tr>
<tr>
<td>Truth Overall</td>
<td>218</td>
</tr>
<tr>
<td>Recall</td>
<td>82.11%</td>
</tr>
</tbody>
</table>

Overall Accuracy 80.42%

ModSecurity (see table 3) is better than PHPIDS. It classified correctly 179 attacks among 218 thus having about 82% as a recall. However, this good result with the attacks is not the same with the legitimate scenarios. It considered 8 normal requests as attacks which is worse than PHPIDS. The overall accuracy of ModSecurity is 80.42%. ModSecurity has hundreds of rules that may suspect more attacks but also more legitimate scenarios. Moreover, the execution overhead would be larger since each request will be checked against those hundreds of rules.

Table 4 The confusion matrix of SPHERES.

<table>
<thead>
<tr>
<th>SPHERES</th>
<th>Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Attack</td>
</tr>
<tr>
<td>Prediction</td>
<td>TP: 216</td>
</tr>
<tr>
<td></td>
<td>FN: 2</td>
</tr>
<tr>
<td>Truth Overall</td>
<td>218</td>
</tr>
<tr>
<td>Recall</td>
<td>99.08%</td>
</tr>
</tbody>
</table>

Overall Accuracy 99.17%

The table 4 summarizes the performance results of SPHERES. Among 218 attack scenarios, SPHERES was able to detect 216 of them and raised 2 false negatives. The undetected attack scenarios include one attack classified under the category A2-Broken Authentication and Session Management and one attack under A8-Cross-Site Request Forgery (CSRF). These attack scenarios try to purchase a product from the WackoPicko website. The purchase operation includes five consecutive steps. SPHERES uses the purchase workflow to check such attacks. However, since WackoPicko does not implement a good method to check the precedence orders of the purchase steps, SPHERES checks this normal flow using the page referrer provided by the browser which is actually a weak method; the attacker has appended the requests with customized referrers that seem to be normal. In order to avoid such an attack, the website should implement a good workflow checker and describes it using SPHERES filtering rules. Similarly, the session-based attack was undetected by SPHERES because the attack requests uses a stolen valid cookies. Since the cookies are valid, SPHERES cannot predict such abnormal behavior. However, if the website includes an advanced session management method, it will be easy to describe it in the filtering rules and to detect the session inconsistencies. The precision of SPHERES for the class attack is 100% and the recall for the same class is 99.08% which are considered very good results.

Concerning the legitimate requests, SPHERES did not confuse them with any attacks at all. All the legitimate scenario was correctly classified as legitimate thus producing a recall
value for the class legitimate 100%. The precision of the class legitimate is 91.67% which is a good result but is not as excellent as the other metrics because of the few number of legitimate scenarios considered in the benchmarking; the two incorrectly classified attacks affect greatly the precision value. The overall accuracy of SPHERES is 99.17% which is very good result.

Note that SPHERES detects easily and efficiently the injection attacks such as the SQL injection, code injection and XSS. In addition, it provides an elegant way to protect from session-based and workflow attacks. Compared to the other security methods, SPHERES demonstrates much better results.

9 Conclusions and Future works

This paper has introduced a new design of a Web application protection method called SPHERES. The main idea behind SPHERES is that it intercepts the traffic coming to the web server and the application server and checks it against a set of filtering rules specific to the requests. This design allows SPHERES to have the most accurate picture of the traffic, the websites, and the system states. So that, in contrast to other protection methods like the Network Intrusion Systems, SPHERES handles the traffic without any encryption, it has a detailed idea about the traffic generated by each user and each session, and a detailed picture of the websites workflows and its corresponding functions. This accurate picture of the total system allows SPHERES to build a solid protection sphere around the website and checks several types and levels of protections. However, SPHERES requires a set of filtering rules for each website describing the normal usage of the website. We show that the filtering rules would be written one time for each website version. In addition to the detection of known attacks, the normal usage description allows SPHERES to detect new attacks at runtime. The performance of SPHERES is very good compared to two famous existing web protection tools. Nevertheless, SPHERES cannot protect totally a website, the development phases of the web application are major steps of the security of the systems.

SPHERES is a web protection framework that opens many perspectives of the low-level web protection. The knowledge of the structure and the normal workflow of the website, the state of the system, and the users traffics make easy the security controls. But, how this picture could be minimized? How could SPHERES avoid double checks of the same request at different levels? How the normal usage of the websites and their filtering rules could be generated automatically and efficiently?

References


26


AppArmor security project: [ONLINE] Available at: http://apparmor.net. 2015.


Marco Rocchetto, MartÁn Ochoa, Mohammad Torabi Dashti: Model-Based Detection of CSRF. ICT Systems Security and Privacy Protection (SEC) 2014: 30-43


SPHERES: An Efficient Server-side Web Application Protection System


