УДК 004.056:004.031.4

Д. М. Самойленко
Національний університет кораблебудування імені адмірала Макарова,
кафедра ЕОС та інформаційної безпеки

СТВОРЕННЯ БЕЗПЕЧНИХ ШАРУВАТИХ СТРУКТУР МОВОЮ PHP/ SECURE LAYERS CONSTRUCTION ON PHP

Запропоновано методи реалізації архітектурних програмних шарів засобами мови PHP з дотриманням вимог безпеки щодо їх створення, функціонування та знищення. Наведено приклад створення первинного шару диспетчера доступу з використанням як функцій PHP, так і сервера. Висвітлено принципи конструювання внутрішніх шарів. Додержання запропонованих принципів програмування може покращити показники безпеки мережевих інформаційних ресурсів.

Ключові слова: інформаційна безпека, мережеві ресурси, PHP мережеве програмування, захист інформації.

This paper deals with the demonstration of secure layering principles with the usage of PHP language. It shows the methods for layers creation, exploitation and destruction in compliance with data protection requirements. The way for access manager creation is carried out as an example of primary layer. For other layers the concept of programming is illustrated. The usage of principles shown in the work could improve secure indicators of network information resource.

Key words: information security, network resource, PHP, network programming, data protection.

Вступ

Задачі захисту мережевого інформаційного ресурсу (МІР) – сайту, порталу, веб-банку тощо еволюціонують з розвитком та ускладненням мережевих технологій. На цьому етапі необхідно розділяти МІР як програмний твір та середовище його функціонування (СФ) – серверне апаратне та програмне забезпечення, що реалізує клієнт-серверну комунікацію, здійснює попереднє оброблення запитів та ініціює виконання коду МІР і є (чи може бути) спільним для багатьох МІР на одному сервері. Найчастіше основні зусилля щодо захисту спрямовуються саме на удосконалення безпеки СФ – оновлення ліній зв'язку, комутаційного обладнання, операційних систем тощо.

Варто відзначити, що розділення МІР і СФ можна відстежити і на спрямованості мережевих атак. Наприклад, атаки типу "відмови в обслуговуванні" (DOS) спрямовані на СФ, тоді як атаки ін'єкції кодів – на МІР. Відтак заходи щодо виявлення та захисту від зазначених атак повинні реалізовуватись у різних об'єктах різними способами.

Синхронно поділяються і аспекти адміністративної відповідальності за недоліки безпеки. Згідно з нормами законодавства відповідальність пов'язана з питаннями власності. Очевидно, що власник СФ (серверу, комутаційного обладнання, файєрволу тощо) і власник МІР можуть бути різними особами. У такому разі суперечки щодо компенсацій витрат, що сталися через недоліки захисту, можуть вилитись в окрему проблему, яка може бути посилена розширенням кола власників інтелектуальних прав, зокрема на операційну систему, антивіруси тощо.

У цій роботі основна увага зосереджена на аспектах захисту самого МІР. Вважатимемо, що власник МІР розміщає (хостить) його у СФ, що є власністю іншої особи. Відповідно усі питання щодо забезпечення роботи МІР мають бути максимально пов'язані у ньому без надмірних сподівань на якість захисту СФ.

Аналіз досліджень та публікацій

Певні напрямки побудови захисних рішень, що враховують згаданий поділ, ґрунтуються на створенні додаткового "прошарку" між МІР та СФ [1]. Хоча існує певне розмаїття подібних прошарків, їх доволі часто у неформальному спілкуванні узагальнюють терміном "докер", тоді як Docker – це лише один з зазначених програмних продуктів. Головна ідея таких програм полягає у створенні віртуального СФ для кожного МІР, зокрема повторно запущеного. Здебільшого створюється навіть віртуальне апаратне забезпечення (hardware virtualization). Таким рівнем віртуалізації відрізняється, наприклад, комплекс Vagrant [1].

Віртуальне СФ, з одного боку, не дає можливості кодам МІР здійснювати вплив на параметри реального СФ та, з іншого, – дещо стабілізує роботу МІР під час зовнішніх впливів на реальне СФ. Також реалізується додаткова незалежність різних МІР чи різних запусків одного МІР, що одночасно є активними на сервері. Як недолік можна зазначити, що віртуалізація є доволі ресурсомісткою процедурою і значно збільшує завантаженість як процесора, так і пам'яті сервера. Останнє може стати причиною погіршення доступності МІР за великої інтенсивності звернень користувачів, тобто уразливості до DOS-атак.

Також потрібно зазначити, що віртуальне СФ не покращує захищеності МІР до атак, спрямованих безпосередньо на його коди. Якщо SQL-ін'єкція є можливою, то неважливо, з якої бази даних (БД) буде вилучена інформація – з реальної чи віртуальної. Пошкодження віртуальної БД не заподіє великої шкоди, проте витік службової інформації зебезпечити віртуалізацією не вдасться.

Класично вважається, що найдоцільнішим вбачається шлях, за якого МІР створюється з максимально незалежними структурними елементами. Подібний підхід до концепції захисту МІР встановлюється нормативною документацією [2, 3] та міжнародними стандартами [4]. Стосовно аспектів, що стосуються безпосередньо МІР, можна сформулювати такі вимоги. По-перше, структурною одиницею МІР виділяється комплексна система захисту інформації (КСЗІ), що є "непроникливою оболонкою", яка захищає внутрішні об'єкти МІР [2, р. 8]. По-друге, висувається вимога максимальної інкапсуляції даних всередині об'єктів, відмови від оголошення глобальних змінних. По-третє, встановлюється необхідність виявлення та реєстрації спроб несанкціонованого доступу (НСД) [3, р. 6.1.2.4]. Зміст вжитого у наведеному означенні терміна "оболонка" пропонується уточнити за допомогою міжнародного стандарту [4, р. 5.2] як виділену архітектурну складову МІР (шар, layer), яка унеможливлює передачу запитів до внутрішніх об'єктів, окрім, як через себе.

З практичного погляду, МІР, як програмний комплекс, створюють за допомогою серверних мов програмування. Доволі часто для програмної реалізації оболонкового принципу використовується об'єктно-орієнтована парадигма програмування (ООП), яка на рівні ідеології забезпечує як інкапсуляцію даних, так і непроникливість у розумінні недоступності об'єктних кодів. Необхідно зазначити, що якісний супровід ООП, тобто можливість утворення справді "непроникливих" оболонок, має забезпечуватись, зокрема, засобами мови програмування. Для конкретики потрібно визначитись з мовою, яка буде використовуватись під час створення МІР.

Як серверну мову програмування для дослідження вибрано мову оброблення гіпертексту – PHP. Критерієм вибору виступила популярність мови PHP [5], динаміка її розвитку та підтримка фактично усіма хост-сервісами. Незважаючи на окремі негативні висловлювання убік PHP, її поширеність свідчить про недоцільність відмови від цієї мови, актуалізуючи поставлене завдання створення КСЗМІР вибраними засобами.

Основною проблемою, що виникає під час побудови КСЗМІР мовою PHP, полягає у відсутності надійного забезпечення мовою інкапсуляції даних. Усі оголошені функції стають доступними як глобальні, незалежно від місця їх оголошення (зокрема, всередині об'єктів чи інших функцій). Додатково недоцільність використання об'єктно-орієнтованого підходу (ООП), реалізованого мовою PHP, розкрита у [6], з прикладом атаки розкриття значень захищених об'єктних полів запитами "ззовні" об'єкта.

Вимоги щодо додаткової функціональності КСЗІ МІР, деталізації її будови вимагають складання моделей атак, способів їх виявлення та протидії. У цьому напрямку потрібно відзначити

досягнення, що дали змогу формалізувати ознаки атак [7], провести аналіз нейромережевих засобів виявлення атак [8], сформулювати алгоритми проектування механізмів захисту з урахуванням індивідуальних наборів атрибутів [9], а також відзначити аспекти ранжування загроз за ризиками [10]. Висновки реферованих досліджень повинні бути враховані під час складання конкретних реалізацій КСЗІ МІР.

Враховуючи відзначені особливості атак і принципів захисту, скажемо про необхідність розроблення об'єднаної концепції створення МІР. Ідеї, закладені у докер-подібні системи у поєднанні з принципами "оболонки", інкапсуляції і архітектурного розшарування, разом можуть стати основою побудови КСЗІ МІР.

Мета роботи – розробити принципи програмування мовою PHP, що дають змогу структурувати код, спрямований на покращення інкапсуляції, для реалізації КСЗІ МІР.

Реалізація диспетчера доступу

У найпростішому випадку об'єднану концепцію непроникливої оболонки і докер-систем можна вважати як такий спосіб організації кодів МІР, за якого виокремлюється програмний код, що стає проміжною ланкою між СФ та безпосереднім функціоналом МІР. З погляду програмування це означає, що жодна з основних програмних функцій МІР не є доступною, доки не виконаються успішно функції КСЗІ. Це, зокрема, принципово обмежить потенціал атак ін'єкції кодів, оскільки самі коди доступу до істотних функцій МІР на етапі оброблення запиту користувача просто не існують.

Щодо PHP, то недоступність функцій може бути забезпечена лише відсутністю їх попереднього оголошення. Відповідно, на момент запуску МІР оголошеними повинні бути виключно функції КСЗІ, а усі інші оголошуватимуться згодом. З огляду на те, що процеси створення нових програмних структур під час виконання у програмуванні називаються динамічними, називатимемо поступове розширення структури МІР динамічним структуруванням.

За наявності необхідності додаткового розшарування кодів: багатокрокової автентифікації чи розмежування рівнів доступу можна говорити про каскадне динамічне структурування, тобто поетапне оголошення функцій наступного шару за позитивних результатів роботи функцій попереднього шару.

Першим етапом відокремлення МІР і СФ можна вважати реалізацію диспетчера доступу (ДД) – створення єдиної точки проходження усіх запитів [2]. Стосовно МІР це завдання має певні особливості. Здебільшого МІР складається з кількох різних файлів, що розміщаються у різних каталогах. Частина з них містить програмні інструкції, що можуть бути активовані при зверненні до файлів в обхід стартової сторінки. Типовим вирішенням цієї проблеми є створення окремого каталогу для відповідних файлів та обмеження доступу до нього. Проте, як вже було зазначено, обмеження доступу до файлової системи здійснюється засобами СФ, на надійність яких можна не покладатись.

Контролювання процесів запуску розподілених програм має реалізовуватись як складова частина цих програм. Перевагу потрібно надати розробленню інтерфейсів взаємодії, що визначатимуть структури даних, які передаються між різними фрагментами кодів і свідчать про правильний механізм запуску. У найпростішому випадку таким інтерфейсом може бути одна змінна, що оголошується у стартовому файлі і перевіряється в усіх відокремлених файлах. Зазначений підхід можна спостерігати у якісних проектах з відкритим кодом. Посилений захист забезпечуватиметься розрізненням інформації, що передається у різні файли. За такого підходу дискредитація одного з файлів не погіршить захищеність інших.

Другою особливістю ДД МІР є попереднє оброблення запитів засобами СФ. Вони здійснюєть аналіз структури запитів, відокремлюють заголовки, формують глобальні масиви з відомостями про деталі запиту і параметри, що додані до нього. У процесі оброблення може виникнути помилка або бути змінений стартовий (індексний) файл. З метою упередження цих ситуацій, потрібно перезначити функціонал СФ, наскільки це можливо.

Наприклад, для одного з найпопулярніших серверів Apache зазначені інструкції оформляються у файлі з іменем .htaccess, що має бути створений у кожному каталозі. Перехоплення помилок забезпечується інструкцією ErrorDocument, вплив на оброблення запитів – директивою

RewriteRule. За допомогою цих засобів можна спрямувати усі запити, зокрема і помилкові, до одного єдиного файла, що виконуватиме роль ДД. Зокрема, інструкція “RewriteRule .* dispdosp.php” спрямує усі звернення до файла “dispdosp.php”, навіть якщо у запиті вказане ім’я існуючого файла чи каталогу.

Для додаткового контролю правильної роботи впроваджених засобів необхідно створити звичайний індексний файл, запуск якого свідчитиме про порушення у роботі СФ. На вибір розробника засобами індексного файла можна або спробувати передати керування до ДД, або спровокувати аварійну зупинку МІР. Перший варіант потрібно вважати небезпечнішим, оскільки відмова окремих функцій СФ відкриває потенціал атак “руху по директоріях” (DT-directory traversal). У будь-якому разі з міркувань безпеки недоцільно створювати інші файли у стартовому каталозі МІР, залишаючи лише дві можливості запуску, – через ДД або через індексний файл.

Реалізація реєстрації, як послуги безпеки [2, 3], цілком логічно має бути забезпечена ДД. Оскільки довільний запит, що надходить до МІР є керуючим, тобто таким, що супроводжується змінами у роботі МІР, усі запити підпадають під критерії необхідної реєстрації [2]. У будь-якому разі реєстрація усіх запитів без винятку є простішою з погляду програмної реалізації. Зростання збережених обсягів даних буде цілком помірним і компенсованим виграшем у часі, потрібному для аналізу запиту на критерії щодо обов’язкової реєстрації.

На практиці, здебільшого для збереження інформації, що надходить до МІР, використовуються системи управління базами даних (СУБД). Переваги та недоліки вибору СУБД є предметом окремої дискусії у межах цієї роботи і обговорюватись не будуть. Проте сам факт з’єднання з БД у ДД додатково свідчитиме про правильний спосіб запуску МІР і унеможливить доступ до функцій БД в обхід диспетчера, тобто безпосередньо відповідатиме концепції шаруватої архітектури.

Внутрішні шари

За результатом успішної роботи ДД потрібно активувати коди наступного шару – аналізатора запитів. До завдань цього шару зараховуватимемо прийняття вирішення щодо самого запиту, – чи є він правильним, помилковим, чи містить ознаки атак чи розвідки [12].

Типовою для мови PHP є практика, коли за результатами попереднього аналізу запиту (у цьому разі – після ДД) запускається механізм сесій [13], через який встановлюється певна ознака (сесійна змінна), що, по-перше, свідчить про нормальний запуск МІР та, по-друге, встановлює певну прив’язку до користувача. Після цього МІР перезапускається і в подальшому прив’язується до сесії.

Підключення кодів наступного шару відповідно до ідеї динамічного структурування коду може бути виражене у такий спосіб. Насамперед потрібно відмовитись від ідеї ініціації сесії та перезапуску МІР, оскільки це фактично є поверненням до попереднього шару. Також передача ідентифікатора сесії робить комунікацію вразливою до прослуховування каналу чи окремих атак “крадіжки сесій” [14].

Далі необхідно встановити певний інтерфейс взаємодії шарів. У найпростішому випадку ним може бути певна змінна, що встановлюється в одному шарі та перевіряється у наступному (але не передається у канал комунікації). Як вже зазначалось, роль інтерфейсу може полягати у відкритому з’єднанні з БД, але, якщо передбачається можливість роботи МІР у статичному режимі (за відсутності зв’язку з БД), інформації про з’єднання може виявитись недостатньо. За посилених вимог безпеки інтерфейсна змінна може містити кодове значення, що буде використовуватись для дешифрування фрагментів коду наступного шару.

Врешті-решт функції шару повинні уможливлювати як динамічне створення, так і динамічне знищення. Саме остання вимога потребує підвищеної уваги під час використання мови PHP. Ця особливість PHP стосується технології оголошення функцій: якщо функція була оголошена, то вона стає “видимою” в усіх областях видимості програми та унеможливлює знищення [11].

Зауваження можна ілюструвати таким прикладом. На певному етапі роботи користувачу передається форма авторизації, де він вводить логін та пароль. Функції перевірки автентичності даних на сервері ще не активовані, тобто атаки ін’єкції кодів щодо доступу неможливі принципово. Користувач вводить та передає дані на сервер і зазначені функції оголошуються, але у разі помилки

авторизації переходу до наступного шару не відбудеться, користувачу буде відображене попередження неправильної спроби та поля повторного введення. А серверні функції залишаться оголошеними. Тобто у разі помилки першої спроби авторизації з'являється потенційна небезпека ін'єкції кодів через оголошені функції. (У прикладі передбачається асинхронний режим роботи MIP, оскільки відправлення даних з перезапуском MIP суперечить архітектурному принципу "руху" від шару до шару як в один, так і в інший бік).

Виходом із описаної ситуації може бути використання анонімних функцій (функціональних вказівників). Оголошені функції не мають імені, тобто не вносяться у глобальну таблицю оголошених функцій PHP. Доступ до функцій забезпечується збереженням їх адрес (вказівників) у змінних. А змінні унеможливлюють як оголошення, так і знищення, а також обмеження області видимості. Існують два способи задання анонімних функцій, безпосередньо:

$f = function($xy){тіло функції}

та за допомогою виклику вбудованої функції create_function:

$f = create_function('$x','тіло').

У будь-якому разі створена функція не реєструється у переліку оголошених користувачем функцій та унеможливлює знищення як змінної стандартною директивою мови PHP unset:

```
$analyze = function($request){echo $request,"<br/>";}; // Оголошення функції
  $analyze("str"); // Виклик функції з параметром
  unset($analyze); // Знищення функції – подальший її виклик неможливий
```

Варто зауважити, що відсутність реєстрації анонімних функцій у стандартному переліку оголошених функцій додатково покращує якісні показники безпеки MIP через ускладнення розвідки доступних для виклику кодів.

У той самий час процедура оголошення анонімних функцій, хоча і погіршує читабельність коду, ускладнюючи роботу розробника, проте надає елементарні заходи обфускації програми, що також є позитивним з погляду інформаційної безпеки.

Також використання анонімних функцій покращують "стійкість" програми до внутрішніх помилок. Так, помилки повторного оголошення звичайних функцій є фатальними і унеможливлюють перехоплення як виключення (exception) засобами безпечного випробування (try-catch). Якщо з якихось причин копія шару буде активована повторно, то фатальна помилка припинить роботу MIP. Анонімні оголошення не призводять до помилок, оскільки перезначити змінну можна багато разів. За потреби перевірки на наявність активної копії шару можна скористатись інструкцією isset, яка надає інформацію щодо встановлення змінної:

```
if ( isset($analyze) ) {
        реагування на помилку повторного оголошення
}
else{
        створення анонімної функції з вказівником $analyze
}
```

Наведена технологія має універсальний характер і може бути застосована для створення довільної кількості шарів, обмеження стосуються тільки логіки функціонування.

Висновки

Розглянуто принципи проектування програмної складової мережевих інформаційних ресурсів з дотриманням міжнародних стандартів ISO/IEC щодо розшарування.

Показано спосіб створення первинного шару – диспетчера доступу з використанням як засобів мови PHP, так і серверного механізму оброблення. Враховано можливість перевірки відмови роботи диспетчера.

Наведено спосіб конструювання внутрішніх шарів з можливістю як створення, так і знищення функціональності у динамічному режимі. Основу підходу становить використання анонімних функцій. Зазначені переваги, недоліки та особливості зазначеного способу.

Перспективи подальших розвідок вбачаються у практичному випробуванні запропонованих методів порівняно з класичними з погляду показників інформаційної безпеки.

1. Lockhart J. Docker for PHP Developers. Efficient Nginx, PHP, and MySQL development environments with Docker. [Електронний ресурс]/ Josh Lockhart / New Media Campaigns. – Режим доступу http://www.newmediacampaigns.com/blog/docker-for-php-developers (дата звернення: 24.09.16). – Назва з екрана. 2. НД ТЗІ 1.1-002-99 Загальні положення щодо захисту інформації в комп'ютерних системах від несанкціонованого доступу. [Текст] / НД ТЗІ, затверджений наказом ДСТСЗІ СБ України від 28.04.1999. № 22. 3. НД ТЗІ 2.5-010-2003 Вимоги до захисту інформації WEB – сторінки від несанкціонованого доступу. [Текст] / НД ТЗІ, затверджений наказом ДСТСЗІ СБ України від 02.04.2003. № 33. 4. ISO/IEC 7498-1:1994(E) Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. [Text] / ISO/IEC 7498-1:1994(E). Second edition. Corrected and reprinted 1996. Switzerland: ISO/IEC Copyright Office. – 68 p. 5. Рейтинг языков программирования в 2016 году [Електронний ресурс]/ Блог компании King Servers. – Режим доступу https://habrahabr.ru/company/kingservers/blog/307012/ (дата звернення: 24.09.16). – Назва з екрана. 6. Самойленко Д. М. Проектування захищених ресурсів за об'єктно-орієнтованим принципом мовою PHP 5. [Текст] / Самойленко Д. М. // Зб. наук. пр. НУК, 2014. – № 3. – С. 83–87. 7. Азарсков В. Параметры прогнозирования и идентификации атак в информационно-коммуникационных системах. [Текст] / В. Азарсков, А. Гизун, А. Грехов, С. Скворцов // Захист інформації. – 2014. – Т. 16, № 1. – С. 89–95. 8. Корченко О. Сучасні нейромережеві методи та моделі оцінки параметрів безпеки ресурсів інформаційних систем. [Текст] / О. Корченко, І. Терейковський, А. Дзюбаненко // Захист інформації. – 2014. – Т. 16, № 3. – С. 223–232. 9. Іванченко І. С. Захист інформаційних ресурсів на основі атомарної концепції безпеки. [Текст] / І. С. Іванченко // Інформаційна безпека. – 2013. – № 3 (11). – С. 22–28. 10. Архипов А. Практические аспекты оценивания рисков реализации угроз в информационных системах. [Текст] / А. Архипов, А. Скиба // Захист інформації. – 2014. – Т. 16, № 3. – С. 215–222. 11. Руководство по PHP [Електронний ресурс] / Группа документирования PHP. – Режим доступу http://php.net/manual/ru/ (дата звернення: 26.12.14). – Назва з екрана. 12. Самойленко Д. М. Гнучке упередження мережних атак [Текст] / Д. М. Самойленко // Вісник національного університету "Львівська політехніка" "Автоматика, вимірювання та керування". – 2015. – № 821. – С. 84–89. 13. Шлосснейгл Д. Профессиональное программирование на PHP [Текст] / Джордж Шлосснейгл. – СПб.: Вильямс, 2005. – 624 с. 14. Лободенко Д. И. Безопасное программирование на PHP. Кража сессии. [Електронний ресурс] / Д. И. Лободенко // Информационный портал "SoftTime-INFO". – Режим доступу http://www.softtime.ru/info/articlephp.php?id_article=36 (дата звернення: 19.04.15). – Назва з екрана.