digital whisper
Technologic Papers

Mage lvl 90 - The Magento RCE

מאת נתנאל רובין

הקדמה

אני לא בטוח מי חשב שזה רעיון טוב לערבב את PHP עם כרטיסי אשראי, אבל אין ספק שהוא עשה לכלל חוקרי האבטחה שירות גדול.

כידוע PHP היא אחת השפות הכי לא קונסיסטנטיות שיש. אם שפות תכנות נותנות לך רובה ציידים ואת האפשרות לירות לעצמך ברגל, PHP דואגת להסתיר את ההדק ולכוון בשבילך את הרובה אוטומטית לראש.

במאמר הזה אני אפרט על תהליך המחקר שביצעתי על מערכת עגלת הקניות הווירטואלית 'Magento' ועל ההריסה השיטתית של רוב מנגנוני האבטחה בה, עד להרצת קוד ללא אותנטיקציה. המאמר לא יפרט את כל ה-flow, אלא יפרט את תהליך המחשבה שלי כשחקרתי את המערכת.

בשביל התיאור הטכני המלא (באנגלית) ניתן להיכנס לכאן:

http://blog.checkpoint.com/2015/04/20/analyzing-magento-vulnerability/

למי שלא מכיר, Magento היא מערכת עגלת הקניות הכי פופולרית שיש כיום בעולם - היא חולשת על 30% מהשוק ובסך הכול מגלגלת בסביבות ה-60 מיליארד דולר לשנה בקוד ה-PHP הסבוך שלה.

בנוסף, היא נרכשה ע"י eBay ב-2011 ב-180 מיליון דולר.

Diving In

השתמשתי ב-Apache ו-MySQL כדי להריץ את המערכת ואחרי תהליך התקנה קצרצר פתחתי את Zend Studio כדי לחקור את הקוד.

כמו כל חוקר טוב הדבר הראשון שעשיתי היה לעבור בזריזות על פונקציות שונות שנוכל לנצל, כגון 'popen()', 'system()', 'include|require' ופונקציות שמתעסקות עם ה-file system של השרת. כמצופה מכל מערכת שמכבדת את עצמה לא נתקלתי בשום דבר שניתן לנצל בלי הרשאות אדמין.

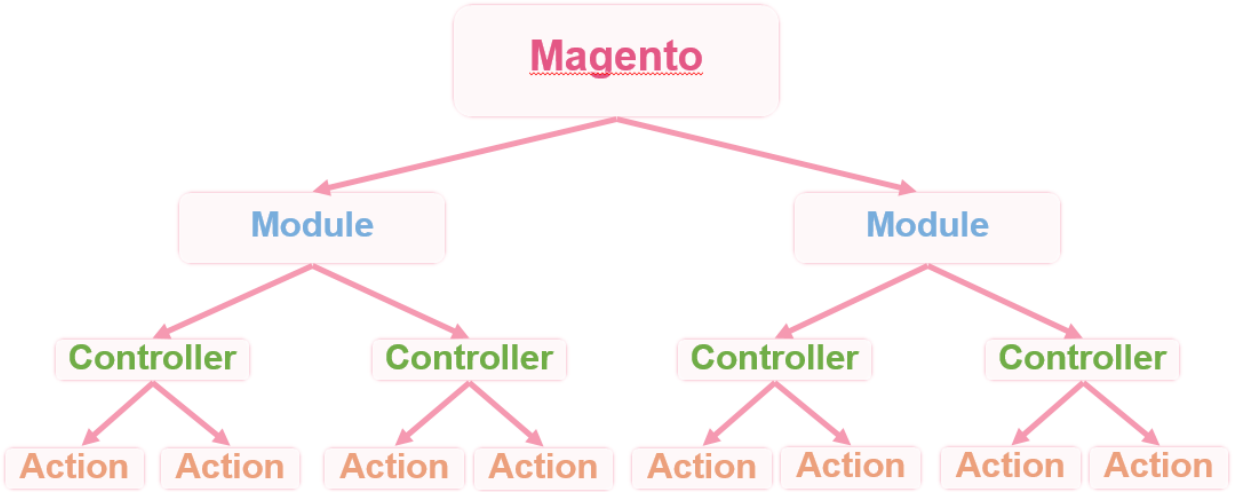
הדבר הכי חשוב שאפשר להבין מלעבור על פונקציות מסוכנות במערכת זה איך היא מתפקדת - איך היא טוענת controllers שונים, איך היא מבצעת אותנטיקציה, איך היא מתקשרת עם ה-DB, לכן, הדבר הראשון שקפץ לי לעין היה הדרך שבה המערכת טוענת את החלקים השונים בה.

Magento עושה הרבה שימוש בטעינה דינאמית של מחלקות ו-Controllers. למעשה, המערכת מורכבת מ-Modules, שמורכבים מ-Controllers, שמכילים Actions.

מודול הוא למעשה תיקייה על ה-file system שמכילה קבצי PHP שמספקים פונקציונליות מסוימת-למשל, יש מודול שאחראי על אותנטיקציה, או מודול שאחראי על ניהול פרטי הלקוח.

כל קובץ PHP כזה מכיל מחלקה שמתפקדת כ-Controller. מחלקה זו מכילה מתודות שאליהן מתייחסים כ-Actions.

אם נרצה להראות זאת בתרשים:



המרכיבים האלה נטענים בצורה דינאמית כאשר משתמש כלשהו מבקש אותם ע"י הציון שלהם ב-URI של הבקשה. למעשה, ההבדל היחידי בין בקשה של משתמש רגיל לבין בקשה של אדמין היא המחרוזת 'admin' שתופיע בתחילת כל URL של האחרון.

לדוגמא, זוהי בקשה של משתמש רגיל שניגש למודול 'downloadable' ולקונטרולר 'file':

```
GET /index.php/downloadable/file/ HTTP/1.1
```

וזוהי בקשה של אדמין לאותו מודול וקונטרולר:

```
GET /index.php/admin/downloadable/file/ HTTP/1.1
```

אבל איך למעשה המערכת מוודאת שהערכים שהכנסנו לה תקינים? במקרה של המודולים היא עושה את הדבר הנכון ושומרת white list של כל המודולים הקיימים, ובמידה והמשתמש הכניס מודול שלא קיים, היא זורקת שגיאה.

במקרה של הקונטרולים הדבר קצת יותר מסובך.

ציינו שקונטרולרים הם למעשה קבצי PHP שמכילים מחלקה בתוך תיקיית המודול. אם, לדוגמא, לקונטרולר שלנו קוראים 'file' והוא נמצא במודול שנקרא 'downloadable', אזי הקובץ שלו יקרא 'fileController.php', הוא ימצא בתוך תיקייה שנקראת 'downloadable' והוא יכיל מחלקה שנקראת 'Mage_Downloadable_FileController'.

כשהמערכת מקבלת קונטרולר מהמשתמש היא מנסה לחפש את קובץ ה-PHP שלו. מכיוון שראינו ששם הקובץ למעשה מורכב משם הקונטרולר, המערכת מרכיבה ומנסה לאנקלד path שמורכב (חלקית) מהמשתמש.

לצערי, חוץ ב-CVE המתקפה הזו לא הייתה שווה הרבה. הקובץ שהמערכת תאנקלד תמיד יסתיים ב-'Controller.php', ובלי Null Byte אין לנו יותר מדי קבצים לאנקלד עם זה. אבל, זה גרם לי לחשוב שאולי יש עוד מקום שבו המערכת מבצעת הנחה לוגית לא נכונה.

המקום ההגיוני הבא לחפש בו הוא כמובן האותנטיקציה-איך המערכת יודעת שמשתמש אדמין מחובר כרגע? זה הקוד שנקרא כשמשתמש מוסיף את התחילית 'admin' לבקשה שלו:

```
$requestedActionName = $request->getActionName();
$openActions = array(
    LIST_OF_OPEN_ACTIONS (login, reset password, etc.)
);

if (in_array($requestedActionName, $openActions)) {
    $request->setDispatched(true); // An open actions
} else {
    if($user) { // A user exist
        $user->reload(); // Check validity of user
    }
    if (!$user || !$user->getId()) { // No user or no user ID
        if ($request->getPost('login')) { // Try to login
            TRY_TO_LOGIN
        }
         // Fail point
        if (!$request->getParam('forwarded')) {
```

```
            if ($request->getParam('isIframe')) {
                NO_ACCESS
            } elseif($request->getParam('isAjax')) {
                NO_ACCESS
            } else {
                NO_ACCESS
            }
            return false;
        }
    }
}

Mage_Admin_Model_Observer::actionPreDispatchAdmin()
```

ניתן לראות שהקוד בודק האם המשתמש כבר מחובר בתור אדמין או לחילופין מנסה להתחבר כאחד. במידה והוא לא, הקוד בודק האם פרמטר שנקרא 'forwarded' קיים, ואם לא הוא פשוט מסיים את הריצה של הסקריפט.

אבל מה זה בכלל הפרמטר הזה 'forwarded'? למעשה הוא דגל שקונטרולרים יכולים לקבוע שמציין שאת תהליך האותנטיקציה הם מעוניינים לעשות בעצמם. לדוגמא, קונטרולר שאחראי על OAuth מן הסתם יבצע אותנטיקציה בעצמו.

הבעיה טמונה בכך שהפרמטר נקבע בתוך המשתנה '$request'. המשתנה הזה מכיל למעשה את כל המידע שהמשתמש שלח לשרת בבקשת ה-HTTP, כולל את כל הפרמטרים שהוא שלח ב-GET וב-POST.

בפועל, 'forwarded' אמנם יכול להיקבע ע"י המערכת, אך הוא יכול להיקבע גם ע"י המשתמש ע"י שליחה שלו כפרמטר HTTP רגיל.

יש, אני אדמין!

אז זהו, שלא. אנחנו אמנם יכולים לעקוף את האותנטיקציה הראשונית, שבסך הכול בודקת אם אנחנו מחוברים כאדמין או לא, אבל רוב הקונטרולרים בודקים בנוסף האם יש לנו הרשאות מסוימות. בגלל שאנחנו אפילו לא מחוברים למערכת, מן הסתם אין לנו אפילו הרשאה אחת.

לצערנו, הקונטרולרים המעניינים באמת, אלה שנותנים לנו להעלות קבצים, לערוך theme-ים, לגשת למסד וכו' מבצעים בדיקת הרשאות נוספת. זה הציב בפניי אתגר חדש-למצוא קונטרולר שלא דורש שום הרשאה שנותן לנו לעשות משהו מעניין.

אחרי שסיננתי את כל הקונטרולרים שדורשים הרשאה כלשהי נשארתי עם בערך חמישה קונטרולרים מאוד זניחים. 3 מהם היו אחראים ל-GUI של המערכת, אחד היה הקונטרולר הדיפולטיבי שהציג את הדף הראשי, ואחד הדפיס תמונה למסך ע"י קלט מהמשתמש.

בא נסתכל על הקוד של הקונטרולר שאחראי להדפיס תמונה:

```
// Get the ___directive parameter
$directive = $this->getRequest()->getParam('___directive');

// This function does bade64_deocde to the input
 $directive = Mage::helper('core')->urlDecode($directive);

// Filter(?) the input
$url = Mage::getModel('cms/adminhtml_template_filter')->filter($directive);

// Try to load the image
try {
    $image = Varien_Image_Adapter::factory('GD2');
    $image->open($url);
    $image->display();
}

Mage_Adminhtml_Cms_WysiwygController::directiveAction()
```

ניתן לראות שהערך של המשתנה '$directive' נקבע ע"י '()getParam' , שמחזירה ערך של פרמטר HTTP. לאחר מכן, הוא עובר פרסור ע"י הפונקציה '()filter' מהמחלקה 'adminhtml_template_filter', ולבסוף הקונטרולר מתייחס אליו כנתיב לתמונה.

כשראיתי את הקוד מיד בדקתי האם אנחנו יכולים להדפיס כל קובץ מהשרת, גם אם הוא לא תמונה, אך לצערי GD2 מוודא שהקובץ שביקשנו הוא תמונה וולידית, ובמידה והוא לא הסקריפט יוצא ומחזיר שגיאה.

אבל למה בעצם מפרסרים את המשתנה לפני שמתייחסים אליו כנתיב? ובכן, מכיוון שהקונטרולר הזה אמור להיקרא אוטומטית ע"י המערכת, לפעמים הנתיב יכול להכיל מחרוזות שאמורות לייצג נתיבים שונים במערכת. לדוגמא, הנתיב יכול להתחיל ב-'BASEDIR', מה שיגרום לפרסור להחליף את המחרוזת בתיקיית האם של המערכת.

כפי שראינו, הפרסור מתבצע באמצעות המחלקה 'adminhtml_template_filter', שאחראית בין היתר על פרסור קבצי טמפלייט של פאנל האדמין.

זאת אומרת שבנוסף ליכולת המרשימה של טעינת נתיבים, אנו בעצם יכולים להשתמש בכל תג שטמפלייט של פאנל האדמין יכול!

זה מרחיב משמעותית את משטח התקיפה שלנו. כעת, אנו יכולים להתחזות לטמפלייט אדמין סטנדרטי ולנסות לטעון דברים דינאמית. Smarty למשל, אחד מהפרסרים הכי פופולרים שיש, מאפשר לטמפלייטים להריץ קוד באמצעות eval.

יש, אני טמפלייט!

אחרי שבדקתי בדיוק מה אנחנו יכולים לעשות בתור טמפלייט, נשארתי רק עם תג אחד שאני יכול להשתמש בו, זאת מכיוון שרוב התגים האחרים בטמפלייט דורשים משתנים חיצוניים שהפרסר אמור להחליף, ומכיוון שאנו נקראנו ישירות, אין לנו כאלה.

התג שאנו יכולים להשתמש בו נקרא 'blockDirective', והוא אחראי על טעינה של בלוקים לטמפלייט. בלוקים הם למעשה מחלקות PHP שאחראיות להציג דברים שונים ב-GUI. למשל, יש בלוק שאחראי להציג את החדשות, בלוק שאחראי להציק מוצרים אחרונים שנרכשו וכו'.

מערכת הפרסור נותנת לנו לשלוט על סוג הבלוק שאנו רוצים לטעון, על חלק מהפרמטרים שלו, ועל המתודה שאנו רוצים להריץ ע"מ לטעון את הפלט של הבלוק.

זהו הקוד שאחראי על הפעולה:

```
public function blockDirective($const)
{ // We're controlling $const!
    $blockParams = $this->_getIncludeParameters($const[2]);
    if (isset($blockParams['type'])) { // Create a new block
        $type = $blockParams['type'];
        $block = $layout->createBlock($type,null, $blockParams);
    }
    if ($block) { // Set the block properties
        $block->setBlockParams($blockParams);
        foreach ($blockParams as $k => $v) {
            $block->setDataUsingMethod($k, $v);
        }
    }
    if (isset($blockParams['output'])) { // Get method to call
        $method = $blockParams['output'];
    }
    return $block->$method(); // Call it!
}
```

כאמור, בלוקים אחראים להצגת דברים שונים ב-GUI. חלק מאותם דברים מגיעים מה-DB, כמו מוצרים, לקוחות וכו'. חלק מהפונקציונאליות שאותם בלוקים מציעים כוללת פילטור על עמודות ב-DB ע"י הוספת 'WHERE' לשאילתת ה-SQL. ניתן ליצור מספר פילטורים שונים, כמו חיפוש של מחרוזת טקסט או ID ספציפי.

ע"מ לאפשר את הפונקציונליות הזו המערכת מחפשת את הפרמטר 'filter' בבקשת ה-HTTP, ובמידה והוא נמצא היא מנסה לפרסר אותו.

אחד מהאופציות השונות לפילטור היא האופציה לשלוף על range מסוים של IDs. המערכת מאפשרת פילטור כזה ע"י קבלת מערך מהמשתנה שמכיל את המפתח 'from' ואת המפתח 'to'. אם המערך מכיל את המפתחות האלה, הוא מועבר אל פונקציה שמפרסרת את השאילתה לשליפה. ניתן לראות את הקוד שלה כאן:

```
$conditionKeyMap = array(
    // A dictionary containing operation and their matching SQL
    CONDITION_DICTIONARY_MAP
);

$query = '';
// If the condition is an array
if (is_array($condition)) {
    // If there's a 'field_expr' field, assign it to $fieldName
    if (isset($condition['field_expr'])) {
        $fieldName = str_replace('#?', $this->quoteIdentifier($fieldName),
$condition['field_expr']);
        unset($condition['field_expr']);
    }
    …
    // Add the start condition
    if (isset($condition['from'])) {
        $from = $this->_prepareSqlDateCondition($condition, 'from');
        $query = $this->_prepareQuotedSqlCondition($conditionKeyMap['from'], $from,
$fieldName);
    }
    // Add the end condition
    if (isset($condition['to'])) {
        $query .= empty($query) ? '' : ' AND ';
        $to = $this->_prepareSqlDateCondition($condition, 'to');
        $query = $this->_prepareQuotedSqlCondition($query . $conditionKeyMap['to'],
$to, $fieldName);
    }
    …
}
Varien_Db_Adapter_Pdo_Mysql::prepareSqlCondition ()
```

ניתן לשים לב שהדבר הראשון שהקוד עושה זה בודק האם התנאי שהכנסנו מערך. מכיוון שאנו שולפים על range, הוא אכן כזה.הדבר השני שהפונקציה עושה זה לבדוק האם המפתח 'field_expr' קיים, ובמידה וכן היא קובעת את המשתנה '$fieldName' על פי הערך במערך.

'$fieldName' הוא למעשה העמודה עליה אנו שולפים, ערך שנחשב מאובטח ולכן לא מתבצע עליו escaping. מכיוון שאנו שולטים על כל מערך ה-'$condition', אנו יכולים ליצור בנוסף גם את המפתח הזה ובכך לשלוט על העמודה, מה שמאפשר לנו למעשה SQLI. מכיוון שמג'נטו משמשת ב-PDO, אנו יכולים להשתמש בכמה שאילתות SQL במקום בשאילתה אחת, מה שמאפשר לנו להריץ איזו שאילתה שנרצה מאשר להריץ רק 'union select'.

מפה הנתיב נראה כבר די ברור, נוכל להוסיף משתמש אדמין משלנו ולהשתלט על המערכת. אבל בתור תוקפים מתוחכמים לא נרצה להשאיר עקבות כל כך ברורות, ולכן חיפשתי משהו יותר נסתר מזה.

יש, אני אדמין (?)

מג'נטו היא מערכת שחוסכת במשאבי השרת, לכן כשאדמין מעלה תמונה לשרת הוא קודם כל נשמר ב-DB. כאשר משתמש מסוים מנסה לגשת לאותה תמונה, המערכת כותבת אותו גם על ההארד דיסק, על מנת לחסוך במקום כאשר תמונה עלתה אך אין בה שימוש.

זה למעשה מאפשר לנו ליצור קובץ משלנו על השרת ואז לייצא אותו אל ה-file system. הבעיה היא שהקובץ יכתב בתיקייה השמורה לתמונות, ובנוסף ישנו קובץ 'htaccess' שמבטל הרצת CGI בתיקייה.

לכן המטרה שלנו היא לכתוב קובץ תמונה תקין, עם סיומת תקינה, שיכיל קוד PHP שאיכשהו ירוץ, מה שכמובן מוביל אותנו לחפש LFI.

כאמור אנחנו יכולים ליצור כל מחלקת בלוק שנרצה, ואז לקרוא לאיזה מתודה שנרצה בה.

מחלקה מעניינת אחת נקראת 'Mage_Core_Block_Template_Zend', שאחראית, כפי שהשם מרמז, לטעינה של טמפלייטים. המחלקה קרצה לי, ואחרי כמה שורות קוד הגעתי ל:

```
$this->_run($this->_file); // Includes $this->_file
```

הבעיה היא ש-'_file' למעשה אמור להיות תיקייה (פירוט מופיע במסמך הטכני המלא) ולכן הוא תמיד יכיל סיומת '/'.

כידוע לא ניתן לאנקלד תיקייה ולכן הייתי צריך לחשוב על פתרון יצירתי שיאפשר לנו לאנקלד קובץ למרות הסיומת. מכיוון שאנו שולטים בכל המחרוזת '_file' פרט ל-'/' הסורר הנ"ל, אנו גם שולטים ב-wrapper של ה-stream.

בגרסאות PHP ישנות (5.2 ומטה) הייתי יכול להשתמש ב-'http://' כדי לגרום לשרת לאנקלד קוד שנמצא אצלי על השרת ובכך לפתור את הבעיה, מכיוון שמדובר כבר ב-HTTP כבר לא מיוחסת ל-'/' משמעות מיוחדת, ולמעשה אני יכול להגדיר את השרת שלי כך שיתעלם ממנו.

בגרסאות חדשות יותר של PHP האופציה הזו כבויה בדיפולט, ולמעשה כמעט אף שרת בעולם לא מאפשר אותה יותר, לכן זוהי לא אופציה טובה עבורנו.

חיפוש ברחבי ה-wrappers השונים ש PHP מציעה הוביל אותי אל - 'phar://'.

'Phar' הוא למעשה סוג של 'Jar' לקבצי PHP-הוא מתפקד כארכיון שמכיל קבצי PHP שניתן לאנקלד מבלי למקם אותם אינדיבידואלית על ה-file system.

מכיוון ש-'Phar' הוא ארכיון, אינקלוד קובץ בתוכו תראה כך:

```
include 'Phar://somefile.phar/somefile.php';
```

כבר ניתן לראות שניגשים לקבצים בתור ה-phar עם '/', אך מה יקרה אם נקרא לקובץ כך:

```
include 'Phar://somefile.phar/';
```

במקרה הזה קוד ה-stub של הארכיון יקרא וירוץ. קוד ה-stub הוא למעשה קוד שנקרא אוטומטית כשמאנקלדים את הארכיון עצמו, על מנת לבצע אתחול במקרה הצורך.

למעשה, הפיצ'ר הזה מושלם עבורנו! כך נוכל לאנקלד את הקובץ שהעלנו עם סיומת ה-'/' המציקה. השאלה היחידה שנשארה היא איך אנחנו גורמים לקובץ להתנהג גם כקובץ תמונה תקין?

ובכן, קוד ה-stub שדיברנו עליו הוא למעשה המחרוזת הראשונה בקובץ ה-phar. למעשה, נוכל להוסיף לו את התוכן של תמונת jpg שלמה ורק לאחר מכן את הקוד שלנו.

מכיוון שהפורמט של jpg למעשה קורא את הקובץ עד שהוא מגיע למחרוזת הסיום שמוגדרת לו, הקובץ שיצרנו יקרא גם כ-jpg תקין וגם כ-phar תקין!

סיכום

אחרי שעקפנו אותנטיקציה, הוספנו קובץ למסד עם SQL Injection, ייצאנו אותו ל-file system, והשתמשנו ב-RFI כדי לאנקלד קובץ jpg/phar, הצלחנו להריץ קוד. ואם לטעמכם לא היינו חשאיים מספיק, phar גם תומך ב-tar, zip, gzip וב-bzib2 כאפשרויות דחיסה, סתם בשביל הקטע ☺