[--] [writing a php fuzzer to self-discover web vulnerabilities ....] [--]

Modificación por Pesst para Webhostingforo.com

Fuzzers are tools which can audit code and probe systems for generic vulnerabilities. For the purpose of this article, we will write several functions for a PHP script which will fuzz the GET parameters of a URL to trigger error codes and discover potential vulnerabilities. We will then explore possibilities of expanding the functionality to become a broader all-emcompassing web vulnerability auditing tool.

Our web fuzzer works by taking a URL and manipulating each GET variable to make every possible combination of requests with an array of malicious characters designed to generate errors. Consider the following array which contains a large selection of common requests which often generate errors and could open scripts up to security holes.

```
// malicious web requests
$vulnchars[0] = array("%00","%2527%252esasdf","%u0000", "%u5c00%u2700","/","../","./.../.","/%2e/", "%2e","%5C","%s", "'","''''","\"", "%%
%%%%","!!!!!!!!!!!!!!!!!!!","#", "%5C27","%%5C%56" , '\'', '\\',';',";a", "|", '\?>', "%a0");
```

```
// malicious sql requests
$vulnchars[1] = array(" OR 1=1", "' OR '!'='!");
```

```
// malicious xss requests
$vulnchars[2] = array("javascript:alert(String.fromCharCode(65,66,67))", "<script>alert('cookies, yo: ' + document.cookie);</script>");
```

We would then make all possible combinations of web requests and analyze the output. Scan the results for an array of common error code output and generate a list of 'flagged' URLs to be later reviewed for auditing purposes. We have put together the following array which contains a list of common web, sql, and xss errors.

```
$flags[0] = array("<b>warning</b>:", "warning:", "<b>fatal error</b>", "failed to open stream:", "internal server error", "there was an error
when processing this directive.", "http/1.1 400", "http/1.1 403", "http/1.1 500", "gateway error", "command not found", "file not found");
$flags[1] = array("[obdc", "mysql error", "you have an error in your sql syntax", "odbc drivers error", "[microsoft sql", );
$flags[2] = array("javascript:alert(string.fromcharcode(65,66,67))", "<script>alert('cookies, yo: ' + document.cookie);</script>");
```

Now that we know what kind of requests to make and what we should be parsing the output for, we can write some PHP code which will query the HTTP server for our requests. In this example, we are only making GET requests, but it can be easily modified ti include other HTTP methods.

```
function MakeRequest($url, $method="GET") {
  $url = str_replace(" ", "%20", $url);
  if ($method=="GET") {
    $host = substr($url, strpos($url, "://") + 3);$host=substr($host, 0,strpos($host, "/"));
  $request = substr($url, strpos($host, "/"));

  $fp = @fsockopen($host, 80, $errno, $errstr, 10);
  if (!$fp) {
     echo "    ERROR . $url $errstr ($errno)$newline";
  } else {
     $out  = "GET $request HTTP/1.1\r\n";
     $out .= "Host: $host\r\n";
     $out .= "Connection: Close\r\n\r\n";
     fwrite($fp, $out);
     while (!feof($fp)) {
         $buf.= fgets($fp);
     }
     fclose($fp);
    }
  }
  return $buf;
}
```

Now that we can get results from the HTTP server for our malicious requests, we need to run it through a function to scan it for the error codes listed above. The following function returns true if the $result has any matches from the $flags array.

```
function TestResult ($result) {
  global $flags;
  $result = strtolower($result);
  for ($i=0;$i < count($flags);$i++) {
    for ($o=0;$o < count($flags);$o++) {
      if (!(strpos($result, $flags[$i][$o]) === false)) {
        return 1;
      }
    }
  }
  return 0;
}
```

Having all the pieces we need, it's time to write some code to tie everything together. The following code uses the array $lists to contain all URLs to probe. It first parses the URL for all GET parameters to fuzz and starts a loop to test all possible combinations of unique URLs. It goes through each GET variable and tries each malicious character while using the default value of all other GET parameters. The total number of requests should be around N ^ N for each url in $list where N is the number of GET parameters in each URL). It then MakesRequest for each unique URL and passes the results off to TestResult, announcing if a match against one of the error codes from $flag.

```
for ($inc=0;$inc<count($list);$inc++) {
  if ($localonly == true AND (substr($list[$inc], 0, 17) != "http://localhost/" AND substr($list[$inc], 0, 17) != "http://127.0.0.1/")) die("This script can only be tested against localhost.");
            // SetUpParameters parses and stores each GET paramater from a URL into the array $get and $getvalues
            $url = SetUpParameters($list[$inc]);
            if (trim($url) != "") {
            echo "$newline$url$newline";
   // go through each kind of vulnerability we are testing
            for ($vulni=0;$vulni<count($vulnchars);$vulni++) {
                        switch ($vulni) {
                          case 0: echo "  * General web vulnerabilities$newline"; break;
                          case 1: echo "  * SQL vulnerabilities$newline"; break;
                          case 2: echo "  * XSS vulnerabilities$newline"; break;
                        }
                        // go through each GET parameter in the URL
                        for ($o=0;$o < count($get);$o++) {
                          for ($i=0;$i<count($vulnchars[$vulni]);$i++) {
                                    // generate url from list of vulnerable characters
                                    $whichparam = $get[$o];
                                    $testing = $url . "?";
                                    // put together the default values for all the other parameters in the script
                                    for ($z=0;$z<count($get);$z++) {
                                      if ($get[$z] != $whichparam) $testing.="&".$get[$z]."=".$getvalue[$z];
                                    }
                                    $testing .= "&" . $whichparam . "=" . $vulnchars[$vulni][$i];

                                    $fun = MakeRequest($testing);
                                    if ($parseforlinks == true) ParseForLinks($fun);
                                    $error = TestResult($fun);
                                    if ($error != 0)
                                      echo "    FLAG! .. $testing$newline";
                                    if ($error == 0 and $verbose == true)
                                      echo "    OK    .. $testing $newline";
                           }
                          }
             }
            }
}
```

This code is the bare essentials to writing a web GET request fuzzer. There are loads of features which can expand this script to be a more encompassing web auditing tool. For starters, the script can be written to read the output of a URL and spider it for additional URLs in <a href="http://$host/"> tags to be added to the $list array. It can also be expanded to include other methods including POST, SSL, cookies, and file upload vulnerabilities. Writing a web fuzzer is a rewarding programming exercise where the possibilities are endless.