ProxMon
Automating Web Application Penetration Testing

Jonathan Wilkins <jwilkins[at]isecpartners[dot]com>
http://www.isecpartners.com

March 9, 2007

Abstract

Performing a web application penetration test is full of repetitive but essential tasks. ProxMon is an extensible Python based framework that reduces testing effort, improves consistency and reduces errors. Its use requires limited additional effort as it processes the proxy logs that you're already generating and reports discovered issues. In addition to penetration testing, ProxMon is useful in QA, developer testing and regression testing scenarios.

Key features:

- automatic value tracing of set cookies, sent cookies, query strings and post parameters across sites
- proxy agnostic
- included library of vulnerability checks
- active testing mode
- cross platform
- open source license
- easy to program extensible python framework

1 Introduction

When I'm conducting a web app pen test, I want to get as much as I can done in an automated manner because I know that if I try and do it manually I'm certain to miss something. No matter how dedicated you might be, a few hours of staring at broken HTML and bizarre JSP, ASP, PHP or Perl is going to tax your brain. When testing a large application, unless your notes are meticulous, by day 4 you're going to be wondering if you managed to check every last server for writable directories while authenticated.

Automated tools like Nessus are great for network penetration tests, but if you're working on web applications, they don't have access to all of the pages behind the login form, so their utility is limited. Even if you give them credentials, they have difficulty parsing pages and executing JavaScript.

Other tools leave a lot in the hands of the auditor. If all you want to do is get rid of maxLength on an input field or tweak a header, you have your choice of about a hundred different applications and browser plugins. If you want a little more control and a reasonable log of what you've seen, there are only a handful of proxy tools you can use.

Proxies give you great insight into how an application works and the good ones will support the following features:

- SSL transactions
- Proxy chaining
- Log seen transactions
- Allow for manual editing of transactions
- Permit scripting

Most auditors I know use WebScarab, though Paros, Burp, SpikeProxy, Pantera and WebScarab-NG all support the above features and have varying levels of polish.

The Overview section of the WebScarab web page says:

> There is no shiny red button on WebScarab, it is a tool primarily designed to be used by people who can write code themselves, or at least have a pretty good understanding of the HTTP protocol.

Think of ProxMon as a first step toward that shiny red button. With ProxMon running alongside WebScarab, all you have to do is browse the target website to start seeing areas that merit further investigation or actual vulnerabilities. Using ProxMon while testing parameter tampering will often uncover unrelated flaws that would otherwise require time consuming careful analysis. It doesn't replace the mind of an experienced web application auditor, but it helps you ensure proper coverage when faced with large applications and short schedules.

2 Usage

Basic usage is trivial. If you run ProxMon at the command line, it will automatically scan your most recent WebScarab temporary directory and report any discovered issues. If you are currently running WebScarab it will also be placed in monitor mode and continue to report issues as you browse the site.

The results below are from an online (-o) test, which includes passive tests (those which only examine stored transactions) as well as active tests (which will make outbound network connections to hosts seen in the transactions to perform further tests).

Here's an example of what you can expect to see when running the tool:

```
[*] starting ProxMon v1.0.14 (http://www.isecpartners.com)
[*] Copyright (C) 2007, Jonathan Wilkins, iSEC Partners Inc.
[*] Proxmon comes with ABSOLUTELY NO WARRANTY;
[*] This is free software, and you are welcome to redistribute it
[*] under certain conditions; see accompanying file LICENSE for
[*] details on warranty and redistribution details.
[*] Loading support for: WebScarab
[*] Loading Checks ...
 - Find interesting comments
 - Find cookie values that also are sent on the query string
 - Find HTTP Basic or Digest Authentication usage
 - Identify frameworks and scripts in use by server
 - Find dangerous functions in JavaScript code
 - NOP - Does nothing
 - Find offsite redirects
 - Find cookies with the secure flag that also get sent cleartext
 - Find values set over SSL that later go cleartext
 - Find values sent to other domains
```

```
 - Find common undesirable directories
 - Find files that indicate common vulnerabilities
 - Find directories that allow directory listing
 - Find SSL server configuration issues
 - Find directories writable via PUT
[*] 15 checks loaded
[*] Finding available sessions ...
[*] Processing session test/webscarab in test
[*] Running in monitor mode
[*] Monitoring test/webscarab
[*] Parsing existing conversations ...
[*] Interesting comment: XXX in http://scratch.bitland.net:80/ (TIDs: 35)
[*] Interesting comment: bug in http://www.bitland.net:80/ (TIDs: 532)
[*] Interesting comment: TODO in http://scratch.bitland.net:80/ (TIDs: 35)
[*] Interesting comment: ??? in http://scratch.bitland.net:80/ (TIDs: 35)
[*] Interesting comment: !!! in http://scratch.bitland.net:80/ (TIDs: 35)
[*] Cookie value seen on QS: secret1 (Secure, SSL) (TIDs: 16)
[*] Cookie value seen on QS: secret2 (Secure, SSL) (TIDs: 9)
[*] Digest auth seen: Authorization: Digest username="jwilkins", realm="scratchdigest", nonce="
    OI1LOhsmBAA=5ee56732a14ca51e16fd2af52a1f4c2f6f0492b0", uri="/digestauth", algorithm=MD5,
    response="892592e2a3d58169c6a1f56fd07718e1", qop=auth, nc=00000001, cnonce="082c875dcb2ca740"
    (TIDs: 34)
[*] Basic auth seen: Authorization: Basic andpbGtpbnM6YXNkZmFzZGY= (TIDs: 31, 32)
[*] IDed framework: scratch.bitland.net:80 is using PHP/5.2.1 (http://www.php.net) (TIDs: 35)
[*] IDed framework: www.isecpartners.com:80 is using YUI/1.2.3 (http://developer.yahoo.com/yui) (
    TIDs: 16)
[*] Unsafe JavaScript found: eval at http://scratch.bitland.net:80/:15 (TIDs: 35)
[*] Unsafe JavaScript found: eval at http://scratch.bitland.net:80/:16 (TIDs: 35)
[*] Secure cookie value sent clear: secret2 (TIDs: 7, 9)
[*] Secure cookie value sent clear: secret1 (TIDs: 16, 36)
[*] Value set over SSL sent clear: secret2 as secure2 (TIDs: 7)
[*] Value set over SSL sent clear: secret2 as bar (TIDs: 9)
[*] Value set over SSL sent clear: secret1 as foobar (TIDs: 16)
[*] Value set over SSL sent clear: secret1 as asdf (TIDs: 36)
[*] Value (secret1) sent to multiple domains: bitland.net (TIDs: 5, 6, 36)
[*] Value (secret1) sent to multiple domains: isecpartners.com (TIDs: 16)
[*] Bad directory found: /backup/ on scratch.bitland.net:80 (TIDs: 0)
[*] Bad file found: /environ.pl on scratch.bitland.net:80 (TIDs: 0)
[*] Listing of /listable/ on scratch.bitland.net:80 succeeded (TIDs: 0)
[*] SSL Config issue https://www.bitland.net:443: aNULL null cipher (TIDs: 0)
[*] SSL Config issue https://www.bitland.net:443: Export strength ciphers (TIDs: 0)
[*] SSL Config issue https://www.bitland.net:443: 40 bit Export strength ciphers (TIDs: 0)
[*] SSL Config issue https://www.bitland.net:443: Low strength ciphers (TIDs: 0)
[*] SSL Config issue https://www.bitland.net:443: SSLv2 protocol (TIDs: 0)
[*] Upload to /put/ on scratch.bitland.net:80 succeeded (TIDs: 0)
[*] Parsed 38 existing conversations
[*] Session is not active, no point in monitoring
```

Some checks are based on a single transaction and some are based on tracking more complicated global state.

The http_auth check will trigger any time it sees an Authorization header.

```
[*] Basic auth seen: Authorization: Basic andpbGtpbnM6YXNkZmFzZGY= (TIDs: 31, 32)
```

The secure_cookie_sent_clear check tracks all cookies that are set with the Secure flag and will report if their associated value is ever sent clear. To be clear, this check doesn't care about names. If it ever sees that value in any cleartext context (for example, under a different name and sent on a query string), it will alert you.

```
[*] Secure cookie value sent clear: secret2 (TIDs: 7, 9)
```

Not every result is an actual vulnerability. For instance, the JavaScript checks only point to functions that tend to be used insecurely. However, this is a vast improvement over trying to go through all of the logs manually because you can see interesting events as they occur and investigate further immediately.

```
[*] Unsafe JavaScript found: eval at http://scratch.bitland.net:80/:15 (TIDs: 35)
```

The above check is reporting the server, the function (eval) and line number (15) and the transaction that it was seen in (TID: 35).

Other checks results indicate a definite issue

```
[*] SSL Config issue https://www.bitland.net:443: 40 bit Export strength ciphers (TIDs: 0)
```

40-bit SSL ciphers are something that shouldn't be supported.

Some important command line parameters are:

-o: Perform active tests, which include actions like connecting to hosts

-d: Specify a directory to search for sessions

-c: Summarize cookie information

-q: Summarize query string information

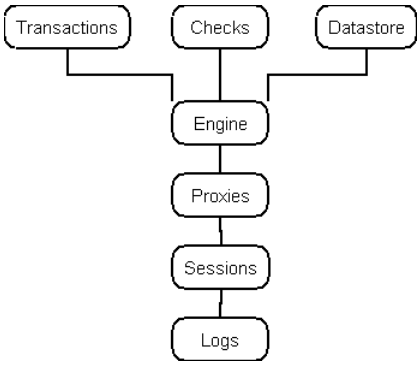
-v: Include verbose information

The verbose mode tells the checks to report a little more information where appropriate. The comment_warn check will output the line that contains the relevant comment tag.

```
[*] Interesting comment: TODO in http://scratch.bitland.net:80/ (TID: 35)
<!-- TODO: - this is a test -->
```

3 Architecture Overview

The main engine in proxmon.py loads the available proxies and checks. The engine then calls the selected proxy's sessions() function to get information on all stored sessions or makes a selection using session_info() if a session is specifiec in the command line parameters. Then it uses the proxy's get() or get_next() method to retrieve logs. The logged requests and responses are then passed to the request and response parsers in transaction.py. Those methods populate the datastore and execute the appropriate check methods. At the end of processing, control is passed back to the engine where checks are called one last time to process the completed transaction (if they have exposed a run() method).



4 Adding to checks

Many checks rely on simple text configuration files that can be easily edited to increase coverage.

For instance, the bad_directory check searches all of the seen directories for accessible subdirectories that are generally undesirable such as 'backup' or 'IISAdmin'. If you want to add a directory to the bad_directory check, all you have to do is edit modules/bad_directory.cfg and add the directory you want the tool to search for.

```
bad_dirs: [ backup, bak, _vti_bin, IISHelp, IISAdmin, old ]
```

The config file format is quite simple but flexible. [1] If you want a simple string, just type it with no quotes. Single quotes will ensure proper escaping. Anything after a # is a comment.

A more complicated example is the bad_files check. This check is run in online mode and checks all detected directories for common problem files such as test scripts, configuration files that might contain sensitive information or known vulnerable scripts.

```
bad_files: [
  {file: 'iisstart.asp',
   text: SERVER_NAME,
   desc: Default page},

  {file: 'environ.pl',
   text: SERVER_NAME,
   desc: "Test script leaks server information in it's output"},

  {file: 'environ.cgi',
   text: SERVER_NAME,
   desc: "Test script leaks server information in it's output"}
]
```

Another useful feature of ProxMon is that it can tell you what frameworks are in use. For instance, the version of PHP[2] or YUI[3]. These tests require a little more sophistication when specifying strings. This module allows the use of regular expressions so you can be fairly accurate.

```
framework_info: [
  {
    name:     dojo,
    url:      'http://dojotoolkit.org',
    body:     'http://dojotoolkit.org/community/licensing.shtml',
    bodyver: 'dojo.version\s([\d+|\.]+)'
  }

  { name:     YUI,
    url:      'http://developer.yahoo.com/yui',
    body:     'http://developer.yahoo.net/yui/license.txt',
    bodyver: 'http://developer.yahoo.net/yui/license.txt\nversion:\s+(.*)\n'
  }

  { name:    PHP,
    url:     'http://www.php.net',
    rhl:     '^Server:\s+.*(PHP)'
    rhlver: '^Server:\s+.*PHP/([\d+|\.]+)'
  }
]
```

[1] ProxMon uses http://www.red-dove.com/python_config.html by Vinay Sajip
[2] http://www.php.net
[3] http://developer.yahoo.com/yui

5 Writing checks

You can add your own checks to allow ProxMon to understand a custom application, or make it aware of a security issue you are particularly concerned about. If your application always enforces a certain page flow, ProxMon could have a custom check that verifies this flow is consistent. It could also attempt to perform actions without sending the correct cookies or modify request values illegally. This can make ProxMon a valuable part of your regression testing process.

All you need to do to have ProxMon incorporate your check is to place the new check in the modules subdirectory and ProxMon will automatically detect it if it's a subclass of pmcheck.

5.1 Passive checks

Here's a module that does nothing.

```
1 from pmcheck import *
2
3 class nop(check):
4   "NOP - Does nothing"
```

Listing 1: ../modules/nop.py

The first line references the file that contains the class definitions for the check classes. The check class is the base class and it is used for most passive checks. The netcheck class is used for active checks, anything that will make outbound network connections should subclass from this. The last basic check type is the postruncheck, which is only run at the end of a session and is used by the cookie_summary check and query_summary checks.

The above check doesn't do anything because it doesn't implement any functions that are automatically run during or after transaction processing.

Most checks are based on the check class and, as mentioned above, run when the full transaction has finished processing and all data is available. To run like this, just declare a function called run()

```
1  "Find cookies with the secure flag that also get sent cleartext"
2  from pmcheck import *
3  from pmutil import *
4
5  class secure_cookie_sent_clear(check):
6    "Find cookies with the secure flag that also get sent cleartext"
7
8    def run(self, pmd):
9      for s in pmd.SetCookieSecureValues:
10       if s in trivial_values: continue
11       if s in pmd.ClearValues:
12         for x in xrange(len(pmd.ClearValues[s])):
13           desc = "[*] Secure cookie value sent clear: %s" % s
14           id = pmd.ClearValues[s][x]['httpparams']['id']
15           self.add_single(desc, id=id)
```

Listing 2: ../modules/secure_cookies_sent_clear.py

Here the module is checking pmd (which is the ProxMon Datastore. This is updated constantly with information on values seen in the various transactions.

The SetCookieSecureValues dictionary is keyed by value. To find out if any value is also seen clear, all you have to do is loop through the keys in the dictionary and compare against those in pmd's ClearValues dictionary.

Knowing that isn't enough to get output though. A call to the check class's add_single() method is needed for the application to report on the issue.

Other methods are available. For instance, the req_hl_parse method is called anytime there's a request header line available to process.

```
1  "Find HTTP Basic or Digest Authentication usage"
2  from pmcheck import *
3  import re
4
5  class http_auth(check):
6    "Find HTTP Basic or Digest Authentication usage"
7
8    def req_hl_parse(self, l, t):
9      if re.search('^Authorization:\sBasic', l, re.IGNORECASE):
10       desc = "[*] Basic auth seen: %s" % (l.strip())
11       self.add_single(desc, id=t['id'])
12     elif re.search('^Authorization:\sDigest', l, re.IGNORECASE):
13       desc = "[*] Digest auth seen: %s" % (l.strip())
14       self.add_single(desc, id=t['id'])
```

Listing 3: ../modules/http_auth.py

Since all this check is concerned with is the request headers, there's no need to implement a run() function. The simple presence of an Authorization header is information enough.

5.2 Check methods

The following is in the same order as the methods would be called during normal transaction processing.

5.2.1 Called from transaction.parserequest

rl_parse() Parses the first line of the request
This contains the method ('GET', 'HEAD', ...), the requested URL ('http://www.isecpartners.com:80/index.html') and the HTTP version ('HTTP/1.0')

req_hl_parse() Called for each header line
This is passed the whole header line ('Host: www.isecpartners.com\r\n')

req_body_parse() Called to parse the full body
This is generally the POST or PUT contents

5.2.2 Called from transaction.parseresponse

sl_parse() Parses the first line of the response. (The status line)
('200 OK HTTP/1.0')

resp_hl_parse() Called for each header line
Similar to the request Headers

resp_body_parse() Called on the uncompressed response body.
The code in transaction.py will handle gzip and both varieties of deflate.

5.2.3 Called from proxmon.scan or proxmon.tail

run() The full transaction has been handled and the transaction dictionary is fully populated. All values will have been added to the datastore.

5.3 Active checks

Checks that do more than just parse the stored transactions are also quite easy to write. These checks allow you to actively test for classes of vulnerabilities that can't be detected by parsing log files alone.

The urltesting module has a number of convenient functions available that will connect out to target hosts and check for or retrieve files. This module is based on curl [4] and pyCurl [5].

If your network check uses the functions in urltesting and you are on a network that requires the use of a proxy (other than WebScarab), it will automatically use the proxy specified in the http_proxy and https_proxy environment variables or you can override these via the -p command line parameter. It will also get access to the appropriate cookies, which are constantly written out as proxmon.cj in Mozilla cookies.txt format.

The expiry on cookies stored to the proxmon.cj file are incorrect. In order to ensure the availability of ephemeral cookies or ones with a very short lifetime, the expiry is changed to 1 year after the current time.

```
1  "Find directories that allow directory listing"
2  from pmcheck import *
3  from pmutil import *
4  try:
5    import urltesting
6  except ImportError:
7    loaderror = True
8
9  class dir_listing(netcheck):
10   "Find directories that allow directory listing"
11   def __init__(self):
12     netcheck.__init__(self)
13     self.checked_dirs_by_server = {}
14
15   def run(self, pmd):
16     end = len(pmd.Transactions)
17     for t in pmd.Transactions[self.lasttransaction:end]:
18       dirs = implied_dirs(t['path'])
19       for d in dirs:
20         if t['server'] in self.checked_dirs_by_server:
21           if d in self.checked_dirs_by_server[t['server']]:
22             continue
23         if urltesting.gives_directory_listing(t['proto']+'://'+t['server']+d):
24           desc = '[*] Listing of %s on %s succeeded' % (d, t['server'])
25           self.add_single(desc)
26         if t['server'] in self.checked_dirs_by_server:
27           self.checked_dirs_by_server[t['server']].append(d)
```

[4] http://curl.haxx.se/
[5] http://pycurl.sourceforge.net/

```
28            else:
29              self.checked_dirs_by_server[t['server']] = [d]
30        self.lasttransaction = end
```

Listing 4: ../modules/dir_listing.py

This module waits until transaction processing is complete and then iterates through any unseen transactions. First, it expands the path to include any implied directories (eg /foo/bar/baz/ implies /, /foo, /foo/bar/). Then it searches through any that it hasn't seen already and sees if they contain strings that imply that the directory contents are viewable.

Important to note is the difference between t['server'] and t['host']. 'server' contains the hostname and port (such as isecpartners.com:80), whereas host just includes the name or IP of the server (simply isecpartners.com). You generally want to be careful when writing checks to reference the former instead of the latter. Different systems are often running on different ports or under different server names. Distinguishing by 'host' alone isn't always sufficient.

5.4 Transaction properties

Given a normal request for http://www.isecpartners.com:80/index.html?a=1&b=2 the values the transaction dictionary would have are given in (). The following is in the order the properties are added.

5.4.1 Assigned by the proxy module

id The transaction identifier (1)

5.4.2 Assigned in transaction.py by parsereqline()

method The HTTP method used in the request ('GET')
url The full URL ('http://www.isecpartners.com:80/index.html?a=1&b=2')
version The HTTP Version ('HTTP/1.0')
proto Either http or https ('http')
hostname Just the host portion of the URL ('www.isecpartners.com')
domain The last 2 or 3 portions of the hostname ('isecpartners.com')
qsf The full query string ('a=1&b=2')
qs A list with the portions of the QS ('[{'name': 'a', 'value': '1'} ...]')
path Server side directory ('/')
port Port the server is listening on ('80')
server The combined hostname and port ('www.isecpartners.com:80')

5.4.3 Assigned in transaction.py by parserequest()

sentcookies A list of cookies sent by the client

host The Host: header value

5.4.4 Assigned in transaction.py by parserequest()

setcookies A list of cookies set by the server, complete with flags
respcontenttype The value of the Content-Type header
respcontentlength The value of the Content-Length header
location The value of the Location header

6 Datastore

Currently the datastore is focused on tracking values that are seen in cookies, on the query string or are sent as post parameters. These are all set by the transaction module.

Each of these values has the associated transaction details (described above) available under the 'http-params' key.

6.1 Lists

Transactions All processed transactions
SetCookies All cookies seen in SetCookie headers
SentCookies All cookies seen in Cookie headers
QueryStrings All values passed on the query string
PostParams All values sent via POST

6.2 Dictionaries (keyed by value)

SetCookieValues All Set-Cookies
SetCookieSSLValues Set-Cookies sent over SSL
SetCookieSecureValues Set-Cookies with the Secure flag
SentCookieValues All values sent in Cookie headers from the client
AllCookieValues All of the cookies
QueryStringValues All values sent via the query string
QuerySPostParamValues All values submitted in POSTs
ClearValues All values sent cleartext
SecureValues All values sent over SSL
AllValues Everything

7 Supporting Other Proxies

Adding support for other proxies is fairly straight forward. The full interface is documented in the appendicies, but partially repeated here so that readers can get an idea of the process.

7.1 Methods

New proxy modules must support the following methods:

sessions(where) Gets a list of available sessions, this is populated via the session_info() function

session_info(where, name) Returns a dict including the session id, whether the session is still active, a list of domains seen in the session, a list of transactions (only required to contain the request and status lines) and dates.

get(session, tinfo) Returns the specified transaction from the selected session
get() and get_next only have to be able to pass the request and response bodies as strings to the engine in order to take advantage of the appropriate processing code in transaction.py.

get_next(session) Returns the next unseen transaction

7.2 Variables

New proxy modules must also define the following variables:

proxy_name The string that will be displayed by the engine

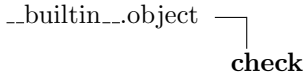
A Module pmcheck

Contains the base check classes (check, netcheck, postruncheck)

A.1 Variables

Name	Description
loaderror	**Value:** `False` *(type=`bool`)*

A.2 Class check



Known Subclasses: cookie_sent_on_qs, http_auth, id_framework_passive, interesting_comments, js_warn, netcheck, nop, offsite_redirect, postruncheck, secure_cookie_sent_clear, ssl_val_sent_clear, value_sent_thirdparty

Check base class

A.2.1 Methods

__init__(*self*)
Overrides: __builtin__.object.__init__

add(*self*, *items*)

Add a list of new unique results

Parameters
`items:` list of results

add_single(*self*, *desc*, *id*=`0`, *verbose*=`None`, *module*=`None`, *value*=`None`)

Add a single result to the datastore

Parameters
`desc:` Description of the issue

clear(*self*)

Clear all results and reset lastreported and lasttransaction

config_loaded(*self*)

Checks if a config file has been loaded and prints an error if not
Call this at the top of each function if a cfg file is required

load_config(*self*)

Auto-loads the appropriate configuration file for the module
Config file should have the same name as the python file, with .cfg instead of .py

report(*self*, *pmd*)

Full report run at the end of session, mainly used by postrunchecks

req_body_parse(*self*, *body*, *info*)

Called by the transaction processor on each complete request body

req_hl_parse(*self*, *line*, *info*)

Called by the transaction processor on all request headers

resp_body_parse(*self*, *body*, *info*)

Called by the transaction processor to parse the whole response body

resp_hl_parse(*self*, *line*, *info*)

Called by the transaction processor on each response header

rl_parse(*self*, *line*, *info*)

Called by the transaction processor on the first line of each request

run(*self*, *pmd*)

Called by scan() or tail() once each transaction has been fully added to pmd

set_verbosity(*self*, *verbosity*)

Sets the level of information displayed by the check"

Parameters

`verbosity:` 0 = default, 1=verbose, 2=debug
(type=int)

show_all(*self*)

Show all results

show_new(*self*)

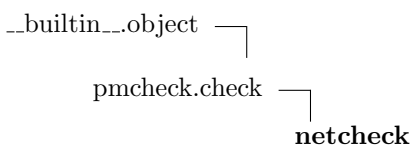
Show only results that haven't been displayed before

sl_parse(*self*, *line*, *info*)

Called by the transaction processor on the first line of each response

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

A.3 Class netcheck



Known Subclasses: bad_directories, bad_files, dir_listing, ssl_config, writable_dir

Base class for all checks that do network stuff

A.3.1 Methods

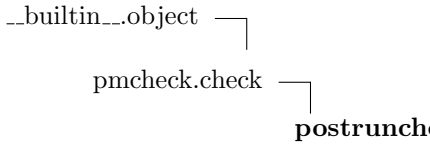
__init__(*self*)
Overrides: pmcheck.check.__init__

set_proxy(*self*, *proxy*)

For those network checks that don't check the environment

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from check: add, add_single, clear, config_loaded, load_config, report, req_body_parse, req_hl_parse, resp_body_parse, resp_hl_parse, rl_parse, run, set_verbosity, show_all, show_new, sl_parse

A.4 Class postruncheck



Known Subclasses: cookie_summary, query_summary

Base class for report modules

A.4.1 Methods

__init__(*self*) Overrides: pmcheck.check.__init__

report(*self*, *pmd*)
Main output method called for postrunchecks
Overrides: pmcheck.check.report

show_all(*self*)
Overriding because postrunchecks shouldn't output until the end
Overrides: pmcheck.check.show_all

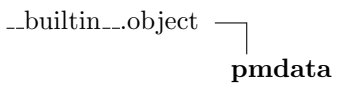
show_new(*self*)
Overriding because postrunchecks shouldn't output until the end
Overrides: pmcheck.check.show_new

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__
Inherited from check: add, add_single, clear, config_loaded, load_config, req_body_parse, req_hl_parse, resp_body_parse, resp_hl_parse, rl_parse, run, set_verbosity, sl_parse

B Module pmdata

Datastore populated by proxy log parsers and consumed by modules

B.1 Class pmdata



B.1.1 Methods

__init__(*self*)
Overrides: __builtin__.object.__init__

add_postparam(*self*, *pp*)

add_querystring(*self*, *qs*)

add_sentcookie(*self*, *c*)

Add a cookie that was sent by a client

Parameters
`c`: A cookie

add_setcookie(*self*, *c*)

Set cookie headers contain a single cookie, parse this and add the contents to the various cookie dictionaries

Parameters
`c`: A cookie

add_transactions(*self*, *trans*)

Adds a list of transactions to Transactions

Parameters
`trans`: The transaction

find_value(*self*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

C Module urltesting

Performs testing on urls to determine if they provide directory listings or allow uploading of data with HTTP PUT and assorted support functions

C.1 Functions

allows_upload(*url*)

file_contains(*url*, *string*)

get_curl()

gives_directory_listing(*url*)

url_exists(*url*)

C.2 Variables

Name	Description
DEBUG	**Value:** `False` *(type=`bool`)*

D Module transaction

Generic Transaction processing routines

D.1 Functions

get_domain(*s*)

Extract the domain portion of a hostname

Parameters

`s:` string with a hostname

get_hostname(*s*)

Extract the hostname from the URL

Parameters

`s:` String containing an URL

get_path(*s*)

get_port(*s*)

get_proto(*s*)

get_querystring(*s*)

Extract the query string portion of an URL

Parameters

`s:` URL in http://host:port/path?qs format

get_querystring_parts(*s*)

Creates a list of value dicts
param s: Full query string

get_sentcookies(*l*, *t*, *pmd*)

get_setcookie(*l*, *t*, *pmd*)

parsereqline(*l*)

Parse the initial line of a request and build a dict with relevant values

Parameters

`l:` string containing the full request line

parserequest(*data*, *checks*, *t*, *pmd*, *urlfilter*)

Parse a request
Needs a full request in the form of a string in data (request line, headers and body)

parserespline(*l*)

Parse the first line of the response and return a dict with relevant values

Parameters

`l:` status line string

parseresponse(*data*, *checks*, *t*, *pmd*, *urlfilter*)

Parse a response
Needs a full response in the form of a string in data (status line, headers and body)

D.2 Variables

Name	Description
Verbosity	**Value:** `1` *(type=**int**)*

E Module pmproxy

E.1 Variables

Name	Description
loaderror	**Value:** `False` *(type=`bool`)*

E.2 Class pmproxy

__builtin__.object
└ **pmproxy**

Known Subclasses: webscarab

Base class for proxies

These are the functions that have to be implemented for a new proxy to function with the framework

E.2.1 Methods

get(*self*, *session*, *tinfo*)

Get a specific transaction

Return Value
dict

get_next(*self*, *session*)

Get the next transaction

Return Value
dict

session_info(*self*, *where*, *name*)

Get information on the specified session

Return Value
dictionary

sessions(*self*, *where*)

Get a list of all known sessions

Parameters

`where:` location to look (path, database name, etc), None to use default

Return Value

list of sessions

Inherited from object: __init__, __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

E.2.2 Class Variables

Name	Description
proxy_name	**Value:** `'Unknown (set proxy_name)'` *(type=`str`)*