BurpSmartBuster

A smart way to find hidden treasures

Patrick Mathieu
patrick@hackfest.ca
@pathetiq
https://github.com/pathetiq/burpsmartbuster

DEF CON - DemoLabs
August 6th 2016

Description

Bruteforcing non-indexed data is often used to discover hidden files and directories which can lead to information disclosure or even a system compromise when a backup file is found. This brute force technique is still useful today, but the tools are lacking the application context and aren't using any smart behaviour to reduce the brute force scanning time or even be stealthier. BurpSmartBuster, a Burp Suite Plugin offers to use the application context and add the smart into the Buster!

This presentation will reveal this new open-source plugin and will show practical case of how you can use this new tool to accelerate your Web pentest to find hidden treasures! The following will be covered:

- How to **add context** to a web bruteforce tool
- How we can **be stealthier**
- How to **limit the number of requests**: Focus only on what is the most critical
- Show how **simple** the **code** is and how you can help to make it even better

Objectives

- Brute force static files based with the application context
- Test existing files, extensions and directories
- Test done using useful developers and sysadmin extensions, files and directories
- Test based on dynamic content of the website
- Offer a stealthier option : limit number of test

Why build this new tool?

- Files and directory test are not based on robots.txt from top 1000 website. Less random and 404 items
- Look for what matters and have a good probability to exist
- Test files are not just static, but dynamic and based on the website domain, filename, extension and directories.
- Few test are based on the technologies
  - Do not look for .phps extension if we aren’t on a php website (experimental, need more work)

How is it done?

- Burp instance
- processHttpMessage() intercept all request and response of Burp Suite
- smartRequest()
  - getSmartData() : robots.txt, spidering words, sitemap.xml and dynamic content (domain name, files, directories, etc)
  - Use Requestor() object to execute all requests threaded
    - List everything that is found in sitemap + logs

Logic

- Application context : What current exist
    - Environment
        - Domain name
        - Existing files, extensions and directories name
- Application technology (todo, and limited)
    - (TODO: based on few file extension, need work on headers and more files)

Smart : Focus/Stealth or not

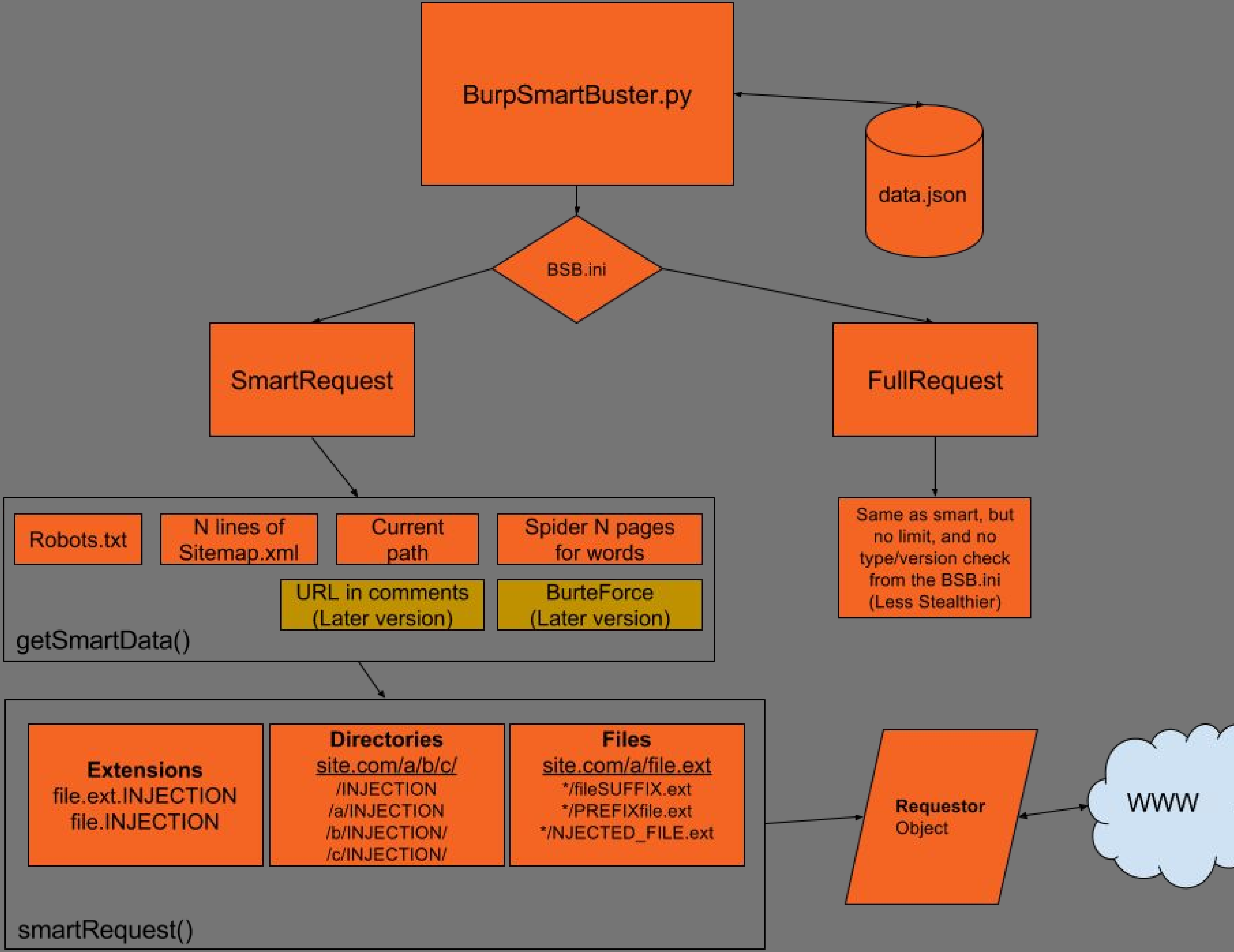
- Configurable
  - Limit number of request (or not) by category (files, extensions, directories)
  - Limit the spidering
  - Chose smartRequest or fullRequest

Simple code

Python 2.7 (I know!) and Straightforward

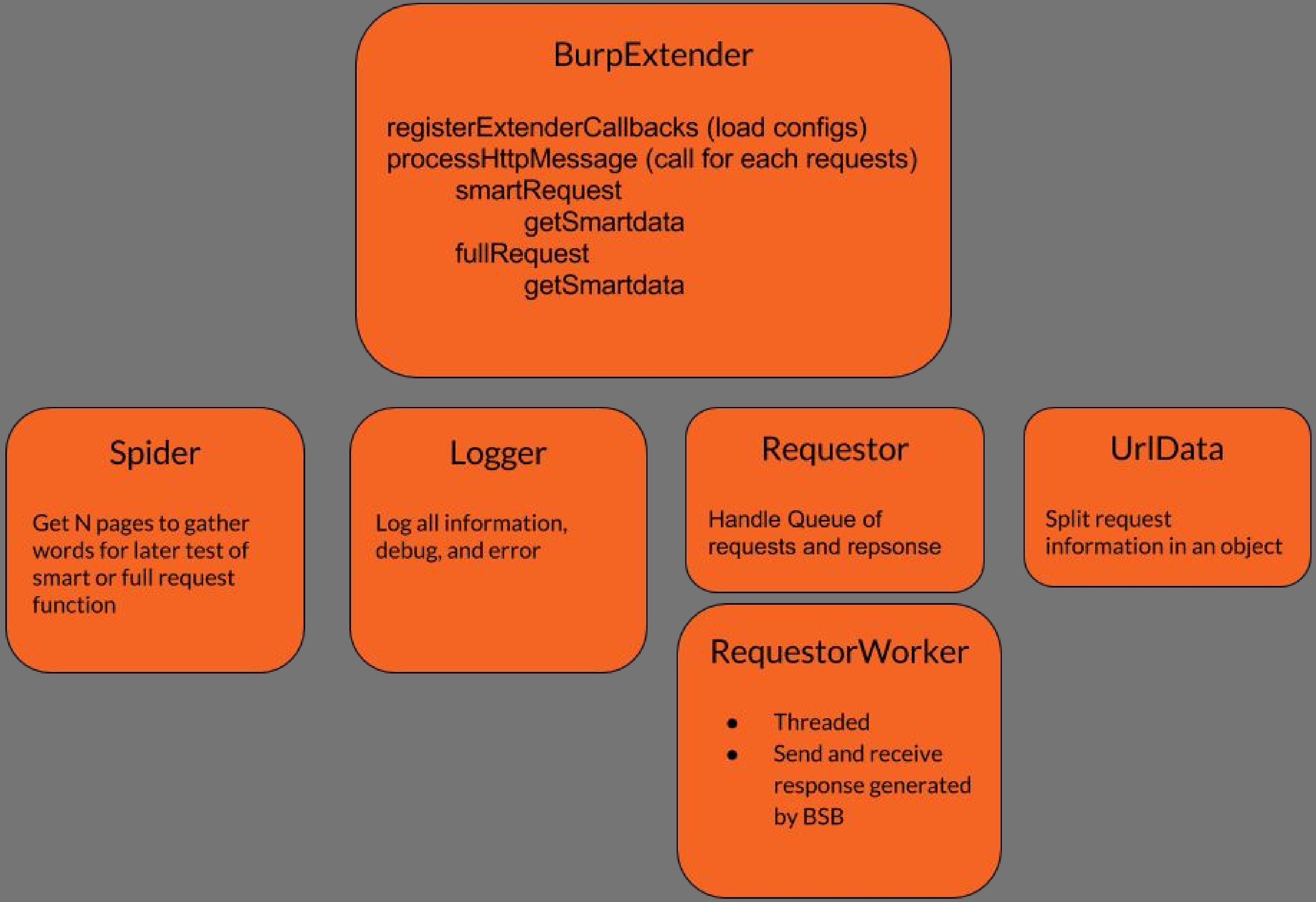
- BurpExtender
  - smartRequest()
    - getSmartData()
- Spider object (spider N pages for words)
- Requestor object (threaded web requests)
  - Queue.queue and deque
  - Add to sitemap, list if a file is found or not
- Logger
  - Add details to log file
- UrlData
  - Splitted the data from request and response of Burp Proxy intercepted by **processHttpMessage()**

DIAGRAM



BurpSmartBuster.py
data.json
BSB.ini
SmartRequest
FullRequest
Robots.txt
N lines of Sitemap.xml
Current path
Spider N pages for words
URL in comments (Later version)
BurteForce (Later version)
getSmartData()
Same as smart, but no limit, and no type/version check from the BSB.ini (Less Stealthier)
Extensions
file.ext.INJECTION
file.INJECTION
Directories
site.com/a/b/c/
/INJECTION
/a/INJECTION
/b/INJECTION/
/c/INJECTION/
Files
site.com/a/file.ext
*/fileSUFFIX.ext
*/PREFIXfile.ext
*/NJECTED_FILE.ext
smartRequest()
Requestor Object
WWW

DIAGRAM



BurpExtender

```
registerExtenderCallbacks (load configs)
processHttpMessage (call for each requests)
        smartRequest
                getSmartdata
        fullRequest
                getSmartdata
```

Spider

Get N pages to gather words for later test of smart or full request function

Logger

Log all information, debug, and error

Requestor

Handle Queue of requests and repsonse

UrlData

Split request information in an object

RequestorWorker

- Threaded
- Send and receive response generated by BSB

_BSB.ini

```
[NumberOfTests]
Paths: 5
Files: 5
Extensions: 5
Directories: 5

[Spider]
RecursiveDirs: 3
NumberOfPages: 5

[Smart]
Local: off
Smart: on
File: off
Spider: off

[InScope]
ScopeOnly: on

[Ignore]
FileType: gif,jpg,png,css,js,ico

[Technical]
TrailingSlash: on
```

_

BSB.ini

[NumberOfTests]
Paths: 5 → limit the number of path to look in
Files: 5 → define the number of files to test
Extensions: 5 → define the number of extensions to test
Directories: 5 → define the number of directories to test

[Spider]
RecursiveDirs: 3 → define the recursive level for testing when we find a directory (todo)
NumberOfPages: 5 → define the number of page to spider to gather words and number from it

[Smart]
Local: off → Only use data.json (todo)
Smart: on → Use smart information (robots.txt, sitemap.xml, spidering words, current path and files)
File: off → user file (todo)
Spider: off → spider only (todo)

[InScope]
ScopeOnly: on → execute our test only on request which are defined in scope

[Ignore]
FileType: gif,jpg,png,css,js,ico → ignore those filetype (todo, basically, done by Burp now)

[Technical]
TrailingSlash: on → For a / at the end of a directory that we test.

_DATA.json

Extensions

```
{"name":".zip", "description":"compress", "type":"default"},
{"name":".bak", "description":"backup", "type":"default"},
{"name":".swp", "description":"autosave", "type":"default"},
{"name":".old", "description":"old", "type":"default"},
{"name":".phps", "description":"development", "type":"default"}
```

Fileprefix

```
{"name":"~", "description":"backup", "type":"default"},
{"name":".", "description":"backup", "type":"default"},
{"name":"Old_", "description":"old", "type":"default"},
{"name":"old_", "description":"old", "type":"default"},
{"name":"Copy%20of%20", "description":"copy", "type":"default"}
```

Filesuffix

```
{"name":"%20-%20Copy", "description":"copy", "type":"default"},
{"name":"(1)", "description":"copy", "type":"default"},
{"name":"%20-%20Copy", "description":"copy", "type":"default"},
{"name":"%20copy", "description":"copy", "type":"default"}
```

Files

```
{"name":"web.config", "description":"config", "type":"default"},
{"name":"wp-config.php", "description":"config", "type":"default"},
{"name":".git/HEAD", "description":"repository", "type":"git"}
```

Directories

```
{"name":"config", "description":"config", "type":"default"},
{"name":"dump", "description":"privacy", "type":"default"},
{"name":"private", "description":"privacy", "type":"default"}
```

_DATA.json

Extensions

Extensions are used on all files that are intercept by BurpSuite.

- We test file by adding extension to them (File.ext → File.ext.ourExt)
- We test file by changing the extension to them (File.ext → File.ourExt)

Fileprefix

Each request that is a file being intercept we will add our own prefix to the file and test if it exists.
File.ext → PrefixFile.ext

Example: Copy%20of%20File.ext → Copy of File.ext

Filesuffix

Each request that is a file being intercept we will add our own suffix to the file and test if it exists.
File.ext → FileSuffix.ext

Example: File%20-%20Copy.ext → File - Copy.ext

Files

All directories being intercept will be test with our files.
http://site.com/a/b/c/d/File.ext, we will test : /files*, /a/files* , /b/files*, /c/files*, /d/files*

Directories

All directories being intercept will be test with our directories.
http://site.com/a/b/c/d/File.ext, we will test: /directories* /a/directories* , /b/directories*, /c/directories*, /d/fildirectorieses*

?

HACKFEST.ca

Call For Papers

https://hackfest.ca/en/cfp

220+ persons on-site CTF

https://hackfest.ca/en/ctf

Patrick Mathieu

Senior security consultant
SecurityCompass.com

Cofounder
Hackfest.ca

- ➔ **Email**
  patrick@hackfest.ca
- ➔ **Twitter**
  @PathetiQ
- ➔ **Hackfest**
  http://Hackfest.ca
- ➔ GitHub
- ➔ https://github.com/pathetiq/