Glastopf Documentation

Release 3.1.1

Glastopf Project

Nov 28, 2018

Contents

Contents:

CHAPTER 1

Development

Basics on how to develop Glastopf

1.1 Guidelines

1.1.1 Developers Guide

Indentation

- We are using 4 tab-spaces
- No one line conditionals

Style

- We obey to the PEP8

Copyright

- If you are adding a file/code which is produced only by you, feel free to add the license information and a notice who holds the copyrights.

Environment

- Eclipse with PyDev and Subclipse is a good combination.

Glastopf-runner for developers

It is recommended to use the *develop* functionality of distutils while hacking on glastopf. When using *develop* a egg link pointing to your repository directory will be places in site-packages - which saves you from doing **python setup.py install** over and over again. Example:

```
$ python setup.py develop
```

Checking if the egg was created as planned:

```
$ cat /Users/jkv/virtualenv/glastopf/lib/python2.7/site-packages/Glastopf.egg-link
  /Users/jkv/repos/glastopf
  ^ output from cat
```

After this, we can create a tmp workdir and start testing directly from the repo:

```
$ pwd
/Users/jkv/repos/glastopf
$ mkdir tmp
$ cd tmp/
$ python ../bin/glastopf-runner
2013-04-17 11:29:11,335 (glastopf.glastopf) Initializing Glastopf using "/Users/jkv/
↪repos/glastopf/tmp" as work directory.
2013-04-17 11:29:11,337 (glastopf.glastopf) Connecting to main database with: sqlite:/
↪//db/glastopf.db
```

1.2 Braindump

1.2.1 Attack classification

We can use the PHPIDS rules? https://dev.itratos.de/projects/php-ids/repository/entry/trunk/lib/IDS/default_filter.xml

1.2.2 SPDY

Do we want to support SPDY? http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1

1.2.3 PHP Interpreter

The REAL PHP interpreter would be awesome for RFI analysis and response generation. Maybe separated from the honeypot. I'm working on a modified version of Jose Nazario's PHP sandbox using funcall for PHP script analysis: http://monkey.org/~jose/software/rfi-sandbox/ I'll add the code to Glastopf later. We should think about if we want to provide this as a service for Glastopf instances.

1.2.4 SQL interpreter

Interpreter for SQL injections?

Jeremy: I guess detection of SQL input might be detected with the key Data Description and Manipulation Language keywords (CREATE, INSERT, etc). Wouldn't be very hard to discover the attacker's purpose. What's interesting to explore might be a probabilistic SQL module to the honeypot.

Jeremy's Dump after attending PyCon APAC

1.2.5 mod_wsgi

Possible integration option with the Apache webserver? Perhaps as a setup option as a complement to investigate attacks on an exposed/production server?

Lukas: Maybe setting up Apache as a proxy to Glastopf?

1.2.6 python curses

Cool terminal interface.

1.2.7 Modular Structure

A general purpose honeypot extensible by Python modules?

Lukas: You mean more general than a web server honeypot? I'm not sure if this is too much ;)

1.3 Emulators

1.3.1 Developing an attack emulator

Introduction

Glastopf's modular design allows for easy extension of the honeypot. This text will briefly demonstrate how to build a simple emulator which will emulate the very popular, and imaginary, php beerservice.

Creating a new handler from scratch involves two steps:

1. Adding a detection rule.
2. Writing an emulator to handle the request.

Detection pattern

A detection pattern is a regular expression which is tested against the url of incoming requests. The following pattern will match all requests which starts with `/beerservice.php`.

```
<request>
 <id>24</id>
 <patternDescription>beer service</patternDescription>
 <patternString><![CDATA[^/beerservice.php]]></patternString>
 <module>beerservice</module>
</request>
```

All request patterns can be found in the requests.xml file.

Adding a basic emulator

To create this emulator (handler), we need to create a module (file) with a filename that matches the <module> tag from the detection pattern. This module needs to be placed in the emulators directory. The python file for the beerservice module will be placed at:

```
glastopf/modules/handlers/emulators/beerservice.py
```

To create a basic handler we need to create a class with the following characteristics:

- Inherits from base_emulator.BaseEmulator.
- Override the handle(self, attack_event) method to provide the needed emulation.

To return http response back to the client you need to call the attack_event.http_request.set_response method which takes care of proper http header and other tedious stuff. If you need full control of the entire http response you can use attack_event.http_request.set_response instead. The following code shows a simple implementation of the beerservice emulator.

```
from glastopf.modules.handlers import base_emulator
import urlparse

class BeerManager(base_emulator.BaseEmulator):
    def __init__(self, data_dir):
        super(BeerManager, self).__init__(data_dir)

    def handle(self, attack_event):
        beer = attack_event.http_request.request_query['type][0]
        reponse = '{0} is a pretty lousy type of beer!'.format(beer)
        attack_event.http_request.set_response(reponse)
```

We can now start Glastopf and test our new emulator as follows.

```
$ curl http://localhost:8080/beerservice.php?type=Rauchbier
Rauchbier is a pretty lousy type of beer!
```

Adding files

If you need to add datafiles, you can add these to the data directory at:

```
glastopf/modules/handlers/emulators/data
```

All content of this directory will automatically be copied to the work directory of Glastopf, which allows for easy customization. You can get the path to the data directory by reading self.data_dir.

CHAPTER 2

Installation

Basics instruction on how to install Glastopf

2.1 Upgrade

Upgrading Glastopf might break backwards compatibility. If this is the case, we will add a note to the changelog and the release info.

Generally upgrading the required Python module is a good idea: pip install -U $module_name

CHAPTER 3

Background

General approach: Vulnerability type emulation instead of vulnerability emulation. Idea: Once 'perfectly' emulated we are able to handle unknown attacks from the same type. Implementation might be more complicated and delays the proper handling but once in place we are ahead of the attackers until the come up with a new method or flaw in on our side. Modular design to add new logging capabilities or attack type handler. Various database capabilities already in place. HPFeeds logging for centralized data collection. Popular attack type emulation already in place. Remote File Inclusion via a build-in PHP sandbox, Local file Inclusion providing files from a virtual file system and HTML injection via POST requests. The adversaries usually use search engines and special crafted search requests to find their victims. In order to attract them, Glastopf provide those keywords (aka dork) and extracts them also from request and extends it's attack surface automatically. So over time and with a growing number of attacks, the honeypot gets more and more attractive. In the feature we will make the SQL injection emulator pubic, provide IP profiling for crawler recognition and intelligent dork selection.

3.1 Feasible Solution for the Web Threat Jigsaw

Web sites are today's biggest and most vulnerable attack surface. A single compromised page can give you a lot of bang for your bugs.

Learning from expensive breaches was never feasible. We will explain how to use Honeypot, Sandbox and Botnet monitoring technology to gain information about current threats which ultimately helps us to find vulnerabilities before they get exploited, insight into the malware distribution network and the botnets used for mass exploitation.

Slides: `pdf` (408kb)

Video: Youtube (15 minutes)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search