Netzob Documentation

Release 0.4.1

Frédéric Guihéry, Georges Bossert

June 11, 2015

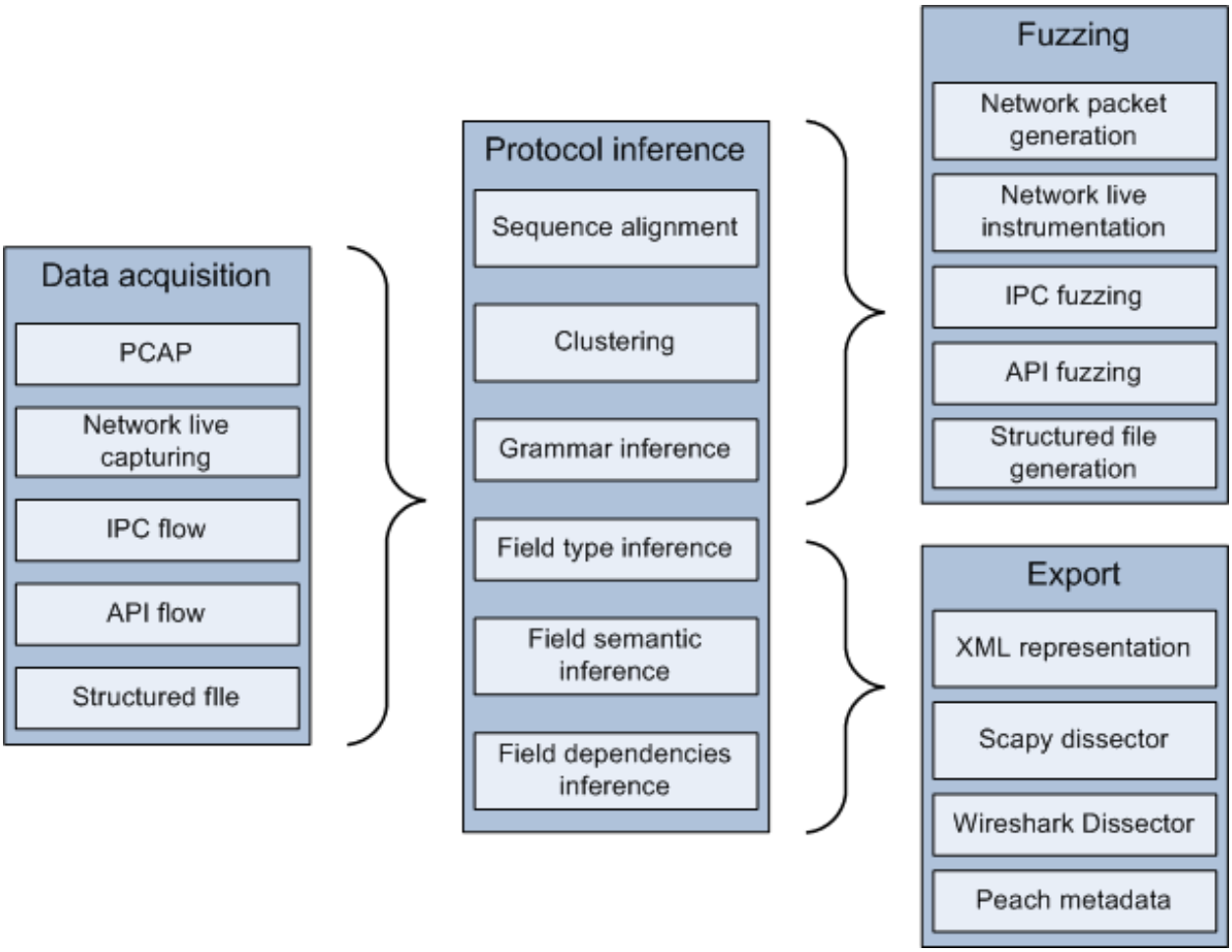
Contents

Netzob simplifies the work for security auditors by providing a complete framework for the reverse engineering of communication protocols. It handles different types of protocols : text protocols (like HTTP and IRC), fixed fields protocols (like IP and TCP) and variable fields protocols (like ASN.1 based formats). Netzob is therefore suitable for reversing network protocols, stuctured files and system and process flows (IPC and communication with drivers).

Netzob is provided with modules dedicated to capture data in multiple contexts (network, file, process and kernel data acquisition).

CHAPTER 1

The big picture

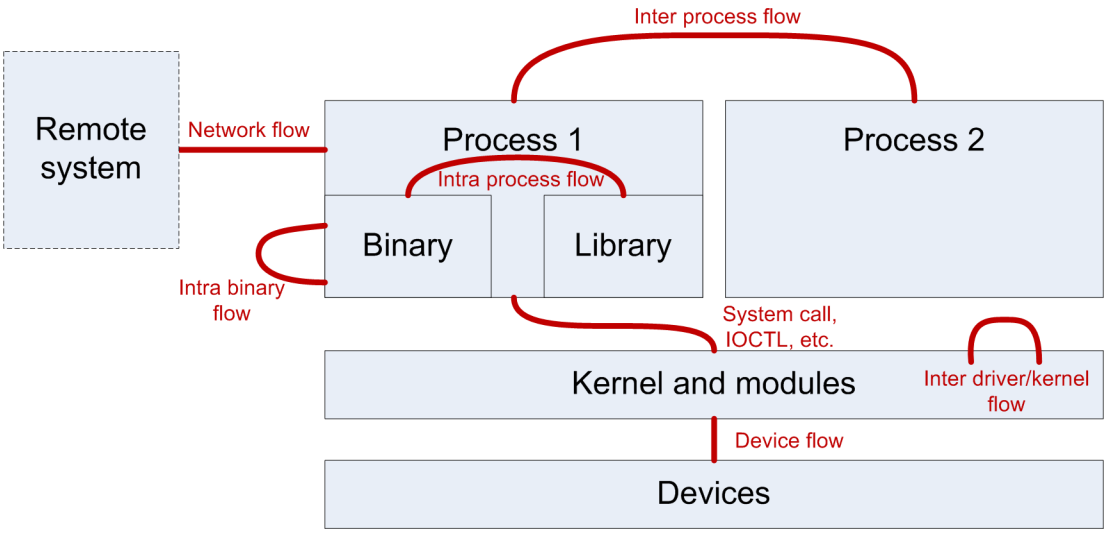


1.1 Table of contents

1.1.1 Introduction

1.1.2 Import

Communication protocols can be found is every parts of a system, as shown on the following picture:



Netzob can handle multiple kinds of input data. Hence, you can analyze network traffic, IPC communications, files structures, etc.

Import can either be done by using a dedicated captor or by providing already captured messages in a specific format.

Current accepted formats are:

- PCAP files
- Structured files
- Netzob XML files (used by Netzob for its internal representation of messages)

Current supported captors are:

- Network captor, based on the XXX library
- Intra Process communication captor (API calls), based on API hooking
- Inter Process Communication captor (pipes, shared memory and local sockets), based on system call hooking

Imported messages are manipulated by Netzob through specific Python objects which contains metadata that describes contextual parameters (timestamp or even IP source/destination for example). All the Python object that describe messages derived from an abstract object : AbstractMessage.

The next part of this section details the composition of each message object.

AbstractMessage

All the messages inherits from this definition and therefore has the following parameters :

- a unique ID
- a data field represented with an array of hex

NetworkMessage

A network message is defined with the following parameters :

- a timestamp
- the ip source

- the ip target
- the protocol (TCP/UDP/ICMP...)
- the layer 4 source port
- the layer 4 target port

Definition of a NetworkMessage :

FileMessage

A file message is defined with the following parameters :

- a filename
- the line number in the file
- the creation date of the file
- the last modification date of the file
- the owner of the file
- the size of the file

Definition of a NetworkMessage :

class netzob.Common.Models.FileMessage.**FileMessage**(*id*, *timestamp*, *data*, *filename*, *creationDate*, *modificationDate*, *owner*, *size*, *lineNumber*)

Definition of the factory for XML processing of a FileMessage :

class netzob.Common.Models.Factories.FileMessageFactory.**FileMessageFactory**

1.1.3 Modelization

Definition of a communication protocol

A communication protocol is as language. A language is defined through~:

- its vocabulary (the set of valid words or, in our context, the set of valid messages) ;
- its grammar (the set of valid sentences which, in our context, can be represented as a protocol state machine, like the TCP state machine).

A word of the vocabular is called a symbol. A symbol represents an abstract view of a set of similar messages. Similar messages refer to messages having the same semantic (for example, a TCP SYN message, a SMTP HELLO message, an ICMP ECHO REQUEST message, etc.).

A symbol is structured following a format, which specifies a sequence of fields (like the IP format). A field can be splitted into sub-fields. For example, a payload is a field of a TCP message. Therefore, by defining a layer as a kind of payload (which is a specific field), we can retrieve the so-called Ethernet, IP, TCP and HTTP layers from a raw packet ; each layer having its own vocabular and grammar.

Field's size can be fixed or variable. Field's content can be static of dynamic. Field's content can be basic (a 32 bits integer) or complex (an array). A field has four attributes~:

- the type defines its definition domain or set of valid values (16 bits integer, string, etc.) ;
- the data description defines the structuration of the field (ASN.1, TSN.1, EBML, etc.) ;

- the data encoding defines ... (ASCII, little endian, big endian, XML, EBML, DER, XER, PER, etc.) ;
- the semantic defines ... (IP address, port number, URL, email, checksum, etc.).

Field's content can be~:

- static ;
- dependant of another field (or a set of fields) of the same message (intra-message dependency) ;
- dependant of a field (or a set of fields) of a previous message in the grammar (inter-message dependency) ;
- dependant of the environment ;
- dependant of the application behaviour (which could depend on the user behaviour) ;
- random (the initial value of the TCP sequence number for example).

Modelization in Netzob

Netzob provides a framework for the semi-automated modelization (inference) of communication protocols, i.e. inferring its vocabular and grammar.

[INCLURE GRAPH]

- **Vocabular inference**
  - Message structure inference (based on sequence alignment)
  - Regoupment of similar message structures
  - Field type inference
  - Field dependencies from the same message and from the environment
  - Field semantic inference
- **Grammar inference**
  - Identification of the automata of the protocol
  - Fields dependencies with messages of previous states

All the functionalities of the framework are detailled in this chapter.

Vocabular inference

Structure inference

Regoupment of similar structures

Options during alignment process

- "read-only" process (do not require a participation in the communication).
- Identify the fixed and dynamic fields of all the messages.
- Regroups equivalent messages depending of their fields structures.
- **Clustering (Regroups equivalent messages using) :**
  - an UPGMA Algorithm to regroup similar messages

  - an openMP and MPI implementation
- **Sequencing, Alignment (Identification of fields in messages) :**
  - Needleman & Wunsch Implementation

Needleman and Wunsch algorithm

- Originaly a bio-informatic algorithm (sequencing DNA)
- Align two messages and identify common patterns and field structure
- Computes an alignment score representing the efficiency of the alignment

The following picture shows the sequence alignment of two messages.

```
01          1d6b815d385b4c6e13f72eb6c35f996c  00000000    c577b9d66edf5b73d29b5ae761e84463
01                                            00000000    326d078d656a9637a8ea7910d671374264
```

UPGMA algorithm

- Identify equivalent messages based on their alignment score.
- Build a hierarchical organization of the messages with the UPGMA algorithm (Unweighted Pair Group Method with Arithmetic Mean)

The following picture shows a regroupment of similar messages based on the result of the clustering process.

Name	Name	Name	Name	Name	Name	Name	Name
0500	.{,2}	0310000000	.{,2}	000000	.{,78}	0000	.{,743}
ascii, binary	ascii, binary	ascii, binary	binary	ascii, binary	binary	ascii, binary	binary
0500	00	0310000000	f4	000000	03000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	2000000030000000400000000000000005
0500	02	0310000000	f4	000000	03000000dc0000000000000060031323332320002000000000000000000000004c148c0300000	0000	20000000c0000000400000000000000005
0500	00	0310000000	f4	000000	08000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	2000000030000000400000000000000005
0500	02	0310000000	f4	000000	08000000dc0000000000000060031323332320002000000000000000000000004c148c0300000	0000	20000000c0000000400000000000000005
0500	00	0310000000	f4	000000	07000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	200000003000000040000000218f5a7505
0500	00	0310000000	f4	000000	06000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	200000003000000040000000218f5a7505
0500	00	0310000000	f4	000000	04000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	200000003000000040000000218f5a7505
0500	00	0310000000	f4	000000	05000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	200000003000000040000000218f5a7505
0500	02	0310000000	f4	000000	07000000dc0000000000000060031323332320002000000000000000000000004c148c0300000	0000	20000000c0000000400000000218f5a7505

Abstraction of a set of message The abstraction is the process of substituting the dynamic fields with their representation as a regex. An example of abstraction is shown on the follinw picture.

```
0007cb3e3d6b001cc07e38c30800
450000
(.{,2})
(.{,6})
00401100
(.{,2})
c0a8000ed41b28f0
(.{,4})
003500
(.{,2})
(.{,10})
0000010000000000000
(.{,46})
0000
(.{,2})
0001
```

Analyses after alignment process aaa

Message contextual menu aaa

Group contextual menu aaa

Refine regexes aaa

Slick regexes aaa

Concatenate aaa

Split column aaa

Merge columns aaa

Delete message aaa

Field type inference

Visualization options aaa

Type structure contextual menu aaa

Messages distribution This function shows a graphical representation of the distribution of bytes per offset for each message of the current group. This function helps to identify entropy variation of each fields. Entropy variation combined with byte distribution help the user to infer the field type.

[INCLUDE GRAPH]

Data typing

- **Primary types** [binary, ascii, num, base64...]
  - Definition domain, unique elements and intervals
  - Data carving (tar gz, png, jpg, ...)
  - Semantic data identification (emails, IP ...)

Domain of definition aaa

Change type representation aaa

Field dependencies from the same message and from the environment

Fields dependancies identification

- Length fields and associated payloads
- Encapsulated messages identifications

And from the environment...

Payload extraction The function "Find Size Fields", as its name suggests, is dedicated to find fields that contain any length value as well as the associated payload. It does this on each group. Netzob supports different encoding of the size field : big and little endian binary values are supported through size of 1, 2 and 4 bytes. The algorithm used to find the size fields and their associated payloads is desribed in the table XXX.

[INCLUDE ALGORITHM]

The following picture represents the application of the function on a trace example. It shows the automated extraction of the IP and UDP payloads from an Ethernet frame.

```
Name:          0007cb3e3d6b001cc07e38c30800
    Name:      450000                                                    Start of payload
    Name:      (.{,2}) / 43                                              Size field
    Name:      (.{,6}) / 7c4140
    Name:      00401100
    Name:      (.{,2}) / a7
    Name:      c0a8000ed41b28f0
        Name:  (.{,4}) / d22d                                            Start of payload
        Name:  003500
        Name:  (.{,2}) / 2f                                              Size field
        Name:  (.{,10}) / be02f7aa01
        Name:  0000010000000000
        Name:  (.{,46}) / 0b6164736572766572616d7306616474656368026465
        Name:  0000
        Name:  (.{,2}) / 1c
        Name:  0001
```

Field semantic inference

Data carving Data carving is the process of extracting semantic information from fields or messages. Netzob allows the extraction of the following semantic information :

- URL
- email
- IP address

[INCLUDE FIGURE]

Search aaa

Properties aaa

Grammar inference

Identification of the automata of the protocol

Fields dependencies with messages of previous states

1.1.4 Vocabular inference

Structure inference

Regoupment of similar structures

Options during alignment process

- "read-only" process (do not require a participation in the communication).
- Identify the fixed and dynamic fields of all the messages.
- Regroups equivalent messages depending of their fields structures.
- **Clustering (Regroups equivalent messages using) :**
  - an UPGMA Algorithm to regroup similar messages
  - an openMP and MPI implementation
- **Sequencing, Alignment (Identification of fields in messages) :**
  - Needleman & Wunsch Implementation

Needleman and Wunsch algorithm

- Originaly a bio-informatic algorithm (sequencing DNA)
- Align two messages and identify common patterns and field structure
- Computes an alignment score representing the efficiency of the alignment

The following picture shows the sequence alignment of two messages.

```
01          1d6b815d385b4c6e13f72eb6c35f996c  00000000    c577b9d66edf5b73d29b5ae761e84463
01                                            00000000    326d078d656a9637a8ea7910d671374264
```

UPGMA algorithm

- Identify equivalent messages based on their alignment score.
- Build a hierarchical organization of the messages with the UPGMA algorithm (Unweighted Pair Group Method with Arithmetic Mean)

The following picture shows a regroupment of similar messages based on the result of the clustering process.

Name	Name	Name	Name	Name	Name	Name	Name
0500	.{,2}	0310000000	.{,2}	000000	.{,78}	0000	.{,743}
ascii, binary	ascii, binary	ascii, binary	binary	ascii, binary	binary	ascii, binary	binary
0500	00	0310000000	f4	000000	03000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	20000000300000004000000000000005
0500	02	0310000000	f4	000000	03000000dc00000000000000600313233323200020000000000000000000004c148c0300000	0000	20000000c0000000400000000000005
0500	00	0310000000	f4	000000	08000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	20000000300000004000000000000005
0500	02	0310000000	f4	000000	08000000dc00000000000000600313233323200020000000000000000000004c148c0300000	0000	20000000c0000000400000000000005
0500	00	0310000000	f4	000000	07000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	20000000300000004000000218f5a7505
0500	00	0310000000	f4	000000	06000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	20000000300000004000000218f5a7505
0500	00	0310000000	f4	000000	04000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	20000000300000004000000218f5a7505
0500	00	0310000000	f4	000000	05000000dc00000000000200b4135be9e6e5a8035a9a67ce348a73e4c80000004c148c0300000	0000	20000000300000004000000218f5a7505
0500	02	0310000000	f4	000000	07000000dc00000000000000600313233323200020000000000000000000004c148c0300000	0000	20000000c0000000400000218f5a7505

Abstraction of a set of message

The abstraction is the process of substituting the dynamic fields with their representation as a regex. An example of abstraction is shown on the follinw picture.

```
0007cb3e3d6b001cc07e38c30800
450000
(.{,2})
(.{,6})
00401100
(.{,2})
c0a8000ed41b28f0
(.{,4})
003500
(.{,2})
(.{,10})
00000100000000000000
(.{,46})
0000
(.{,2})
0001
```

Analyses after alignment process

aaa

Message contextual menu

aaa

Group contextual menu

aaa

Refine regexes

aaa

Slick regexes

aaa

Concatenate

aaa

Split column

aaa

Merge columns

aaa

Delete message

aaa

Field type inference

Visualization options

aaa

Type structure contextual menu

aaa

Messages distribution

This function shows a graphical representation of the distribution of bytes per offset for each message of the current group. This function helps to identify entropy variation of each fields. Entropy variation combined with byte distribution help the user to infer the field type.

[INCLUDE GRAPH]

Data typing

- **Primary types** [binary, ascii, num, base64...]
  - Definition domain, unique elements and intervals
  - Data carving (tar gz, png, jpg, ...)
  - Semantic data identification (emails, IP ...)

Domain of definition

aaa

Change type representation

aaa

Field dependencies from the same message and from the environment

Fields dependancies identification

- Length fields and associated payloads
- Encapsulated messages identifications

And from the environment...

Payload extraction

The function "Find Size Fields", as its name suggests, is dedicated to find fields that contain any length value as well as the associated payload. It does this on each group. Netzob supports different encoding of the size field : big and little endian binary values are supported through size of 1, 2 and 4 bytes. The algorithm used to find the size fields and their associated payloads is desribed in the table XXX.

[INCLUDE ALGORITHM]

The following picture represents the application of the function on a trace example. It shows the automated extraction of the IP and UDP payloads from an Ethernet frame.

```
Name:          0007cb3e3d6b001cc07e38c30800
    Name:      450000                                                  Start of payload
    Name:      (.{,2}) / 43                                            Size field
    Name:      (.{,6}) / 7c4140
    Name:      00401100
    Name:      (.{,2}) / a7
    Name:      c0a8000ed41b28f0
        Name:  (.{,4}) / d22d                                          Start of payload
        Name:  003500
        Name:  (.{,2}) / 2f                                            Size field
        Name:  (.{,10}) / be02f7aa01
        Name:  0000010000000000000
        Name:  (.{,46}) / 0b6164736572766572616d7306616474656368026465
        Name:  0000
        Name:  (.{,2}) / 1c
        Name:  0001
```

Field semantic inference

Data carving

Data carving is the process of extracting semantic information from fields or messages. Netzob allows the extraction of the following semantic information :

- URL
- email
- IP address

[INCLUDE FIGURE]

Search

aaa

Properties

aaa

1.1.5 Grammar inference

Identification of the automata of the protocol

Fields dependencies with messages of previous states

1.1.6 Export

Netzob supports the export of the modelization of the protocol in the following formats: * XML meta representation * Dedicated Scapy Dissector * Dedicated Wireshark Dissector

Each export format in described in this chapter.

XML Export

Scapy Dissector

The Scapy documentation about building new dissectors is available at the following address:

http://www.secdev.org/projects/scapy/doc/build_dissect.html

Wireshark Dissector

The Wireshark documentation about building new dissectors is available at the following address:

http://www.wireshark.org/docs/wsdg_html_chunked/ChDissectAdd.html

1.1.7 Simulation

Todo

1.1.8 Fuzzing

- Live instrumentation through a dedicated proxy
- **Possibilities of variations :**
  - ...
  - ...

1.1.9 Annexes

GOT Poisoning

This idea has firstly been described by *Ryan O'Neill* in its article "Modern Day ELF Runtime infection via GOT poisoning".

Netzob has the following needs :

- inject between an application and its libraries;
- runtime injection,

- dump any value of the variables which are transfered from an application to a libs and “vice et versa”,
- modify the value of any variables which are transfered from an application to a libs and “vice et versa”,
- hidden approach,

Those needs are generated by the following functionalities :

- detection of all the running processes ;
- filter for dynamically linked processes ;
- display the available libs of a process ;
- display all the entry functions of a chosen lib ;
- inject a proxy between multiple functions of a lib and the application ;
- the injected proxy must be controllable (dump and modify variables).

As an example, Netzob must be able to :

- fetch all the sent and received message in an HTTPS connection ;
- modify for fuzzing experiments in an HTTPS connection.

Glossary

First a bit of vocabulary :

Term	Description
GOT	Global Offset Table
PLT	Procedure linking table

How it works

This approach is composed of two object :

- an *injector* which has the responsibility to inject the parasite into a running process;
- a *parasite* which aims are to dump and or to modify the values of the variable transfered between a process and its libs.

And the scenario of the hijacking algorithm is :

1. Locate binary of targeted process by parsing /proc/<pid>/maps
2. Parse PLT to get desired GOT address
3. Attach to the process
4. Find a place to inject the parasite loader shellcode
5. Inject new code and save original code we are overwriting
6. Modify EIP (save old EIP) to point to our code
7. Resume traced process so that it executes our parasite loader shellcode and load our parasite
8. Reset register, replace original code and allow process to resume
9. get base address of our parasite from %eax
10. find address of parasite function within shared lib by scanning for its code sequence

11. Retrieve and save value stored in desired GOT address
12. Modify the return address of the parasite function with the original function address
13. Overwrite desired GOT address with new value
14. Detach from process and enjoy

Step1 : Locate binary of targeted process by parsing /proc/<pid>/maps

INPUT :

- PID of the process

OUTPUT :

- the base address of the application
- the full path of the binary

OPERATIONS :

- Parse the file */proc/<pid>/maps*.

Step2 : Parse PLT to get desired GOT address

INPUT :

- the full path of the binary
- name of the function to hijack

OUTPUT :

- the GOT address / offset of the function to hijack

OPERATIONS :

- Parse the binary (like readelf -r does)

Step3 : Attach to the process

INPUT :

- PID of the process

OUTPUT :

OPERATIONS :

- Attach to the process using *PTRACE_ATTACH*

Step4 : Find a place to inject the parasite loader shellcode

INPUT :

- base address of the text segment of the process
- parasite loader shellcode

OUTPUT :

- the [start-end] offset in the text segment for the injection of shellcode

OPERATIONS :

- Computes the offset in the segment of the injected

Step5 : Inject new code and save original code we are overwriting

INPUT :

- start offset of the injection in the text
- parasite loader shellcode

OUTPUT :

- the [start-end] offset in the text segment for the injection of shellcode

OPERATIONS :

- backup the original code into memory using *ptrace(PTRACE_PEEKTEXT* ;
- load shellcode into text starting at offset using *ptrace(PTRACE_POKETEXT* ;

Step6 : Modify EIP (save old EIP) to point to our code

INPUT :

- start offset of the injection in the text

OUTPUT :

OPERATIONS :

- save the registre using *ptrace(PTRACE_GETREGS* ;
- modify reg.eip using *ptrace(PTRACE_GETREGS* to the start offset

Step7 : Resume traced process so that it executes our parasite loader shellcode and load our parasite

INPUT :

OUTPUT :

OPERATIONS :

- resume execution using *trace(PTRACE_CONT*

Step8 : Reset register, replace original code and allow process to resume

INPUT :

OUTPUT :

OPERATIONS :

- wait for the end of application using * wait(NULL);* ;
- retrieves the values of the registers and reset them to default ones
- detach from process using *ptrace(PTRACE_DETACH*

What is a workspace ?

A workspace contains all the resources files associated with the current analysis which includes : * the repository of automaton, * the logging configuration, * the list of available prototypes of functions which can be hijacked, * the repository of imported traces, * the global configuration file.

1.1.10 API

API description

class netzob.Common.Workspace.**Workspace**(*name*, *creationDate*, *path*, *pathOfTraces*, *pathOfLogging*, *pathOfPrototypes*, *lastProjectPath=None*, *importedTraces={}*)

Class definition of a Workspace

class netzob.Common.Project.**Project**(*id*, *name*, *creationDate*, *path*)

Class definition of a Project

The API section has a complete list of all the classes, methods, attributes and functions of the netzob module, together with short examples for many items.

A day Netzob will have a proper and efficient documentation. But this day, PYTHON will also have one ! :)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

n

F

N

P

W