FRODO: Format Reverser Of Data Objects
by Anton Dorfman
HITB 2014, Amsterdam
About me

- Fan of & Fun with Assembly language
- Researcher
- Scientist
- Teach Reverse Engineering since 2001
- Candidate of technical science
- Lecturer at Samara State Technical University and Samara State Aerospace University
Breaking the rules

- There is the rule RTFM (Read The F**king Manual)
- Nobody likes it
- I’m not exception

- First of all I start my research, and second – try to find related works and analyze them
- After this I generalize ideas from them
- Now I do my best to put all these ideas into the project
Samples
What is it?
What is it? Hint 2

<table>
<thead>
<tr>
<th>MZ</th>
<th>bb...</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-----------</td>
<td>@.....</td>
</tr>
<tr>
<td>X-----------</td>
<td></td>
</tr>
<tr>
<td>.----------</td>
<td>.MT...</td>
</tr>
</tbody>
</table>

is program cannot be run in DOS

mode...$........

Richrèz

[......PE...d......]

Ei[J.........P...]

p55

[........P......]

[1F.........]

[........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]

[.........]
What is it?
What is it?
What is it? Hint 1

GET /manma_mia/p

GET /python/users/HTT

P/1.1..TE: deflate
tegzip;q=0.3..C

connection: TE, c
lose..Host: 10.0
.
User-
.br.8080..User-

Agent: libwww-pc

rl/6.03....
What is it? Hint 2

| ..'23..!U&v`.. |
| E..:O: | O.?Ms.... |
| E.... |
| yE........:R:$. |

GET /manna_mia/p
| python/users/ HTT |
| /P/1.1..TE: defla |
| te, gzip: q=0.3..C |
| connection: TE, c |
| HHost: 10.0 |
| 2.0:000..User- |
| Agent: libwww-pe |
| ri/6.03.... |
What is Data Format?

- File formats – multimedia formats, database formats, internal formats for exchanging between program components and etc.
- Protocols – network protocols, hardware device interaction protocols, protocols of interaction between driver and user space application and etc.
- Structures in memory – OS structures, application structures and etc.
Why RE Data Formats is important?

- Reverse engineering any program
- Reverse engineering undocumented/proprietary file formats, network protocols, structures in memory
- Fuzzing
- Memory forensics
- Examination of protocol implementation
- Vulnerability discovery
- Exploit generation
- Kernel rootkit detection
- Malware classification
- OS kernel fingerprinting
- Replay network interaction
- Zero-day vulnerability signature generation
Standard way

- Hex editor
- Researcher
- Brain (equally important as a hex editor)
- Basic knowledge how data can be organized (in brain)
- Analysis of the executable file that manipulates with data format

- But this way is a hard and challenging task and existing manual approaches tend to be time-consuming, tedious, boring and error-prone. As an example, after numerous trials and errors, it took 12 years for the open-source Samba project to reverse engineer the Microsoft SMB protocol!
Related works analysis
Related works

- NoName – “Extracting Output Formats from Executables” by J. Lim, T. Reps, B. Liblit, 2006
- REWARDS – “Automatic Reverse Engineering of Data Structures from Binary Execution” by Zhiqiang Lin, Xiangyu Zhang, Dongyan Xu, 2010
3 ways of researching

- Static RE analysis – analyze in static, just using binary file how applications parse and handle data format: NoName
- Dynamic RE analysis – analyze in dynamic how applications parse and handle data format to understand it often uses dynamic taint analysis and dynamic binary instrumentation: Tupni, AutoFormat, REWARDS, Howard
- Statistic analysis – try to extract header, structures, fields and try to find relationship between data based on some amount of samples and use statistics of changes and ranges of values: Protocol Informatics Project, Discoverer, Laika
Protocol Informatics Project

- Uses global and local sequence alignment – Needleman Wunsch and Smith Waterman algorithms – with sources
A tool for automatically reverse engineering the protocol message formats of an application from its network trace

A key property of Discoverer is that it operates in a protocol-independent fashion by inferring protocol idioms commonly seen in message formats of many application-level protocols.

Tested on 1 text protocol – HTTP and 2 binary protocols RPC and CIFS/SMB
Laika

- Detects structures in memory using Bayesian unsupervised learning
- For fixed size structures only
- 2 key features: identifying the positions and sizes of objects, and determining which objects are similar based on their byte values.
- Laika identify object positions and sizes by using potential pointers in the image to estimate object positions and sizes.
- The basic block types are address (points into heap/stack), zero, string, and data (everything else)
Laika

Memory Image

Block Types

Classification Results

Class 1
- Address: 650028
- Array?: Yes
- Blocks: A0AD

Class 2
- Address: 650048
- Array?: Yes
- Blocks: A0AD

Composition

Class 1
- Pointer on Class 1
- Pointer on Class 1
- Pointer on Class 2
- Integer

Class 2
- String
Tupni

- Based on taint tracking engine
- Tested on WMF, BMP, JPG, PNG, TIF, DNS, RPC, TFTP, HTTP, FTP
AutoFormat

- Application field: 2 text-based HTTP SIP, 3 binary-based DHCP, RIP, OSPF, hybrid (mixed) CIFS/SMB and unknown used by malware structure of the protocol format by revealing possible relations (e.g., sequential, parallel, and hierarchical) among the message fields
- By monitoring the program execution, it collect the execution context information for every message byte (annotated with its offset in the entire message) and cluster them to derive the protocol format
AutoFormat

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>804B50A</td>
<td>mov esi,[ebp+0C]</td>
</tr>
<tr>
<td>804B50D</td>
<td>mov [ebp+0C],esi</td>
</tr>
</tbody>
</table>

Application Binary

Protocol Payload

Instruction Address

804B50A mov esi,[ebp+0C]

Data Reference

Cell-Stack
Input Payload and Their Propagation

Context-Aware Execution Monitor

Log with Call-Stack and Instruction Address

Finest-grained-field Identification
Parallel-field Identification
Hierarchical-field Identification
Sequential-field Identification

Protocol Field Identifier

Protocol Format
Main Idea of FRODO

- All programs use data formats
- Data formats are developed by human (not pets or aliens)
- Sometimes looks like that no human works on it...
- Data formats are abstractions of implementation details
- Format developers use common data organization concepts and similar thoughts when creating new data formats
- If we find regularities in data format organization rules we can automate searching of them
Definitions

- Data – information for representing which Data Format is developed
- MetaData – some structure for describing Data Format
- Field – some value used to describe Data Format
- Structure – way for organizing various fields
- Header – most common type of MetaData – structure before data, may contain substructures with fields
FRODO Tasks

- Data Format analysis – generate specification
- Format specification checking – the difference between specification and realisation
- Memory dump reconstruction – find various data format structures in memory and links between them
Tasks to solve

- Extract header, separate it from data
- Find field boundaries
- Find value ranges of fields
- Find structures and substructures
- Find types of fields
- Detect bit and byte ordering
- Determine semantics of fields
Levels of abstraction

- Bit Order
- Byte Order
- Fields Size
- Field Basic Type
- Field Type
- Structure
- Field Semantics
Types of fields

- Service fields – for describing Data Format (size of structure and etc.)
- Common fields – “fields from life” (time, date and etc.)
- Specific fields – we can find range for that type (bit flags and etc.)
Field Size and Levels of field interpretation

- Commonly field is a byte sequence
- Sometimes field is a bit sequence

- Fixed size in bytes (1, 2, 3, 4, ...)
- Fixed size in bits (1, 2, 3, 4, ...)
- Variable size in bytes
- Variable size in bits
Basic field types

- Hex value – by default
- Decimal value
- Character value (up to 4 symbols)
- String (ASCII or Unicode)
- Float value
ASCII String

- Usually fixed size, remaining space after text filled with 0
- Variable sized text ended with 0, or there are size of this text field

- Usually text string contain their meanings
Service fields

- ID
- Offset
- Size
- Quantity
- Flag
- Counter
Common:
Identifier of data format, also known as «magic number»

Properties:
Every copy of data format structure contains the same ID value
Field size – can be n bytes
Usually ID of data format – first n bytes of instance
Often Consist of char symbols – “PE”
Often Looks like “magic” in hex – BE BA FE CA

Subtypes:
ID of data format – exist in all instances of that data format
ID of structure – exist in all instances of that data structure
Offset

- Properties:
  - Offset pointed inside instance of data format
  - Offset pointed inside concerned block
  - Depends on it can be absolute or relative
  - Field size – depends on architecture – 2, 4, 8 and etc bytes

- Subtypes:
  - Offset to data
  - Offset to another field
  - Offset to another structure
  - Offset to the instance of the same structure (next or previous in linked list)
Size

- Common:
  - Size of metadata or data in data format

- Properties:
  - Can’t be more then concerned block
  - Measured in bytes

- Subtypes:
  - Size of data
  - Size of metadata
  - Size of structure
  - Size of field
Size

- Notepad.exe
- Frodo.exe
Quantity

- Properties:
  - Quantity of structures of some type in data format
  - Quantity can be concerned as size of same type elements array
  - Elements size – more then 1 byte – word (2 byte), double word, paragraph (16 byte) and etc.
  - Quantity multiplied by the size can’t be more then concerned block

- Subtypes:
  - Quantity of same type structures
  - Quantity of same type fields
Flag

- Properties:
  - Usually this field – combination of bit values
  - Data range of this field have limited values

- Subtypes:
  - Bit Flag
  - Enum value flag
  - Bool value flag
Flag

- Notepad.exe
- Frodo.exe
Counter

- Properties:
  - Sequence number of the packet in the protocol communication, or sequence number of the frame in multimedia format
  - Usually counter is incremented by 1

- Subtypes:
  - Incremented counter
  - Decremented counter
  - Starts with 0
  - Begins with another value
  - Changes by 1
  - Changes by another value
Common fields

- Time
- Date
- Protect
- Etc.
Time and Date

- Storing format
- Resolution
- Moment of beginning
- Range
Protect

- CRC value of whole block
- CRC value of data
- CRC value of metadata
- CRC value of structure
- Various Hash functions and etc.
Protect

- Notepad.exe
- Frodo.exe
Methods of FRODO

- Range checking
- Value subtraction
- Hamming distance
- Entropy of blocks checking
- Some heuristics
FRODO Summary

- Written in Assembler x86
- Executable file size – 35840 bytes – many internal
- Fast and furious!
- Testing is on its active phase now
Thank you! Questions? ;-)  
  
- dorfmananton@gmail.com