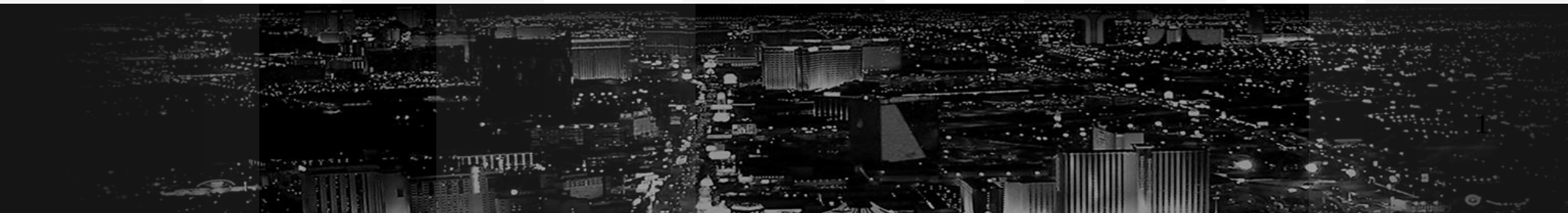
DISTRIBUTING THE RECONSTRUCTION OF HIGH-LEVEL INTERMEDIATE REPRESENTATION FOR LARGE SCALE MALWARE ANALYSIS

Alexander Matrosov (@matrosov)
Eugene Rodionov (@vxradius) [1]
Gabriel Negreira Barbosa (@gabrielnb)
Rodrigo Rubira Branco (@BSDaemon)

{alexander.matrosov || gabriel.negreira.barbosa || rodrigo.branco}
noSPAM intel.com
[1] rodionov *noSPAM* eset.com



Disclaimer

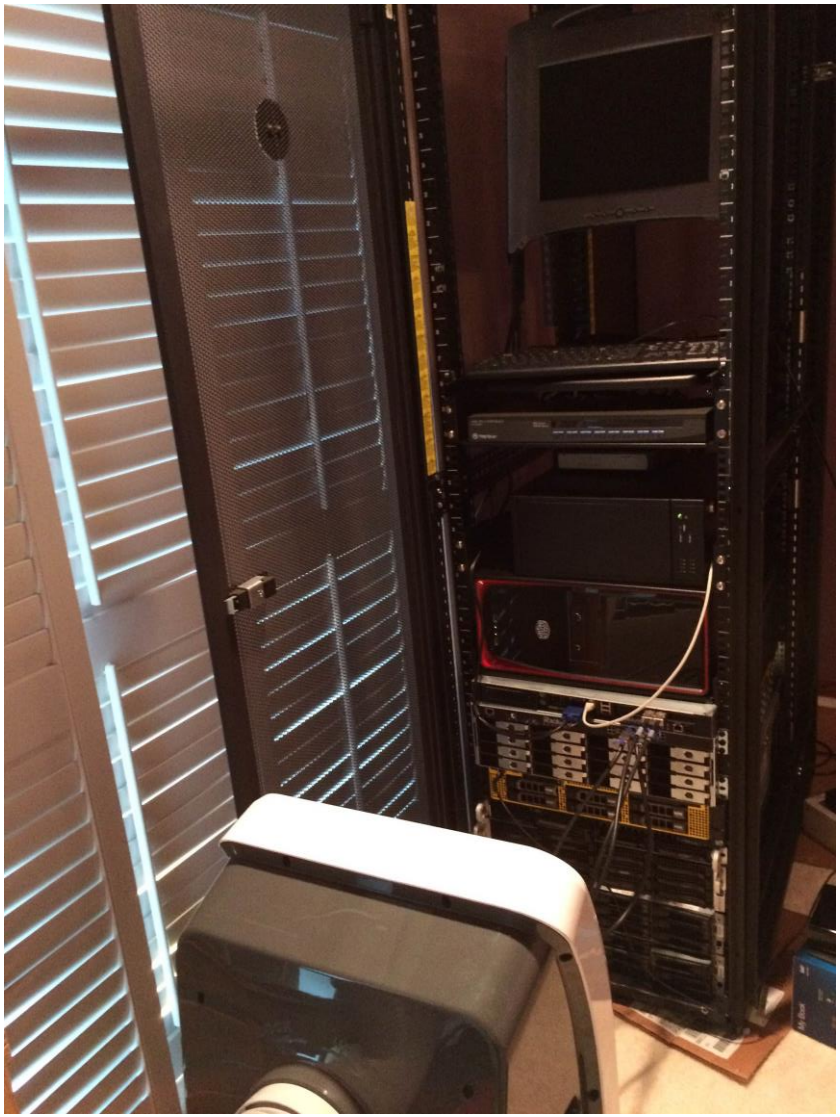
We don’t speak for our employer. All the opinions and information here are of our responsibility (actually no one ever saw this talk before).

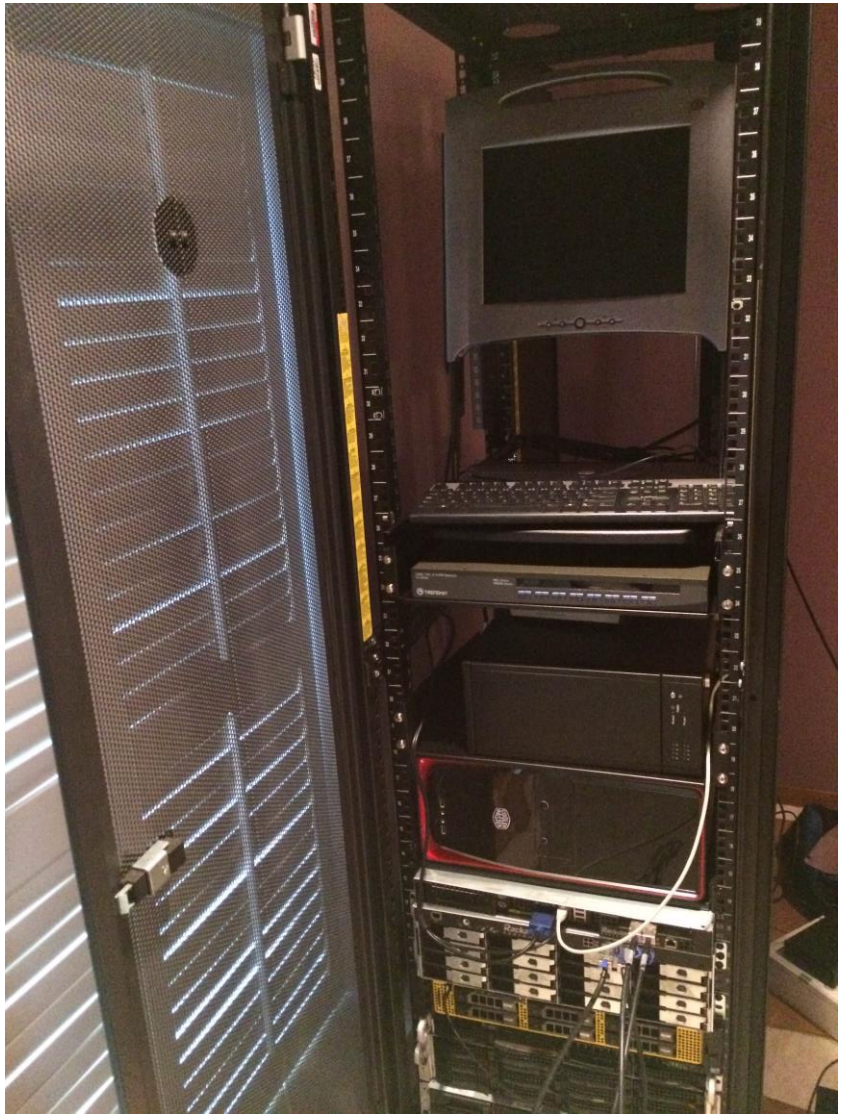
So, mistakes and bad jokes are all

OUR responsibilities

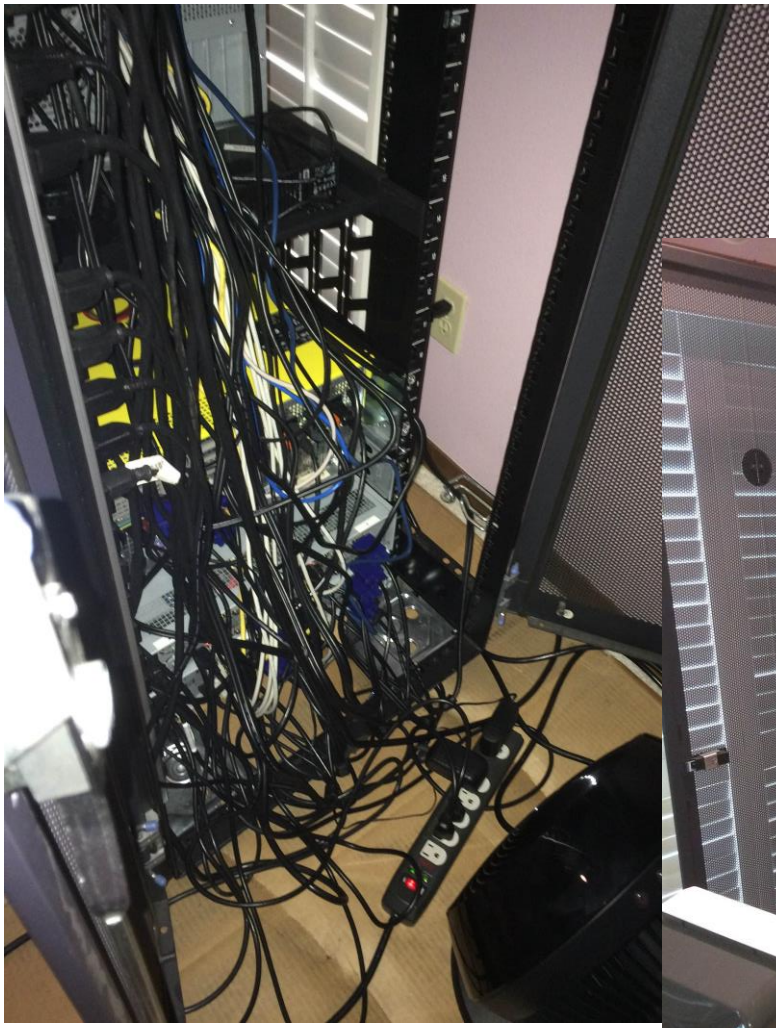
Previous years…

We learned! (Pictures from the back intentionally omitted) …





… Or Not! (Thanks to the smoke and fire detection mechanism)



Agenda

- Introduction / Motivation
- Objectives
- Methodology
- Distributing IDA Pro (with Decompiler)
- Results
- Validating the Methodology and Toolset
- Resources
- Conclusions
- Acknowledgments

Introduction / Motivation

- Number of new malware samples grows at an absurd pace
- We still see words such as ‘many’ instead of the actual number of analyzed samples
- Assumptions without concrete data supporting them
- **INDUSTRY-RELATED RESEARCH NEEDS RESULTS, THUS NOT PROMISING POINTS ARE NOT LOOKED AFTER**

Objectives

- **Demonstrate** the possibility of in-depth large-scale malware analysis

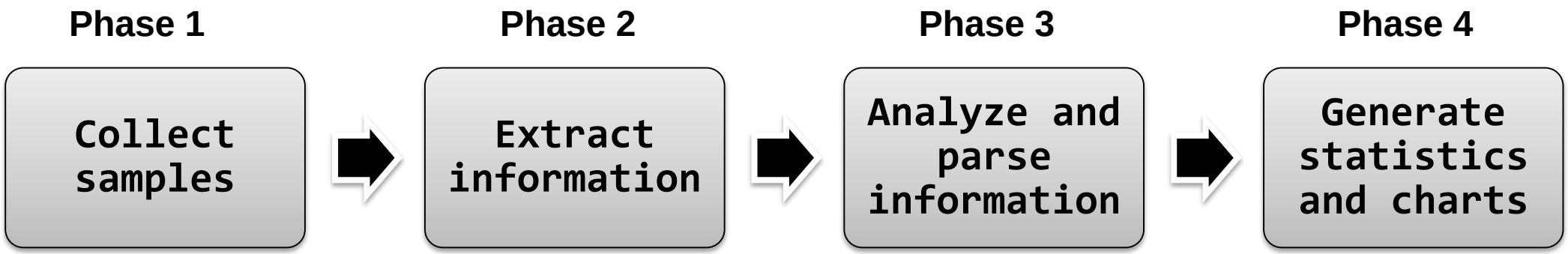
- **Distribute and scale** IDA Pro (with Decompiler) to leverage its functionalities for automated malware analysis

- **Share with the community** the obtained results:
  - ✓ IDA Pro IDBs, plugins and scripts
  - ✓ Intermediate representation
  - ✓ MS Visual C++ reconstructed types
  - ✓ And more...

Methodology: Highlights

- **Analyzed 32-bit and x86-64-bit PE not-packed samples from public sources**
- **No malware size limitations at all**
- **Preference on MS Visual C++ samples because of HexRaysCodeXplorer OO types reconstruction feature**
- **Details on the infrastructure already discussed in Black Hat Las Vegas 2012 presentation**

Methodology: Overview of the process



Pre-process samples and collect millions of 32-bit and x86-64-bit not-packed PE malware samples

Run different malware analysis algorithms on the collected samples and store results on the filesystem.

Parse and structure the results.

Generate statistics and charts based on structured information.

Methodology: Only static analysis

- We only used static analysis
- Not detectable by malware… unless it exploits the analysis environment!
- Prone to anti-disassembly tricks
- Has some limitations… but powerful tools and techniques are available
- IDA Pro rocks!! ☺

Methodology: Malware analysis algorithms

- **HexRaysCodeXplorer (by @REhints) used for:**
  - ✓ Ctrees* for some IDA-recognized functions
  - ✓ MS Visual C++ object-oriented types REconstruction
- **Ctrees depth analysis**
  - ✓ Highly-modified version of pathfinder by @devttyS0
- **AES-NI and GETSEC detection**
- **OO “this” usage study**
- **Crypto usage detection based on IdaScope by @push_pnx**

** - ctrees is the intermediate representation in Hex-Rays decompiler*

Constraints and Limitations:
Dumping Ctrees

Enumerate routines

- Iterate through recognized routines in idb
- Process first 60 routines of size larger than 0x160 bytes
- Process first 30 crypto (using AES-NI) routines
- Process first 40 other functions bigger than 0x60 bytes

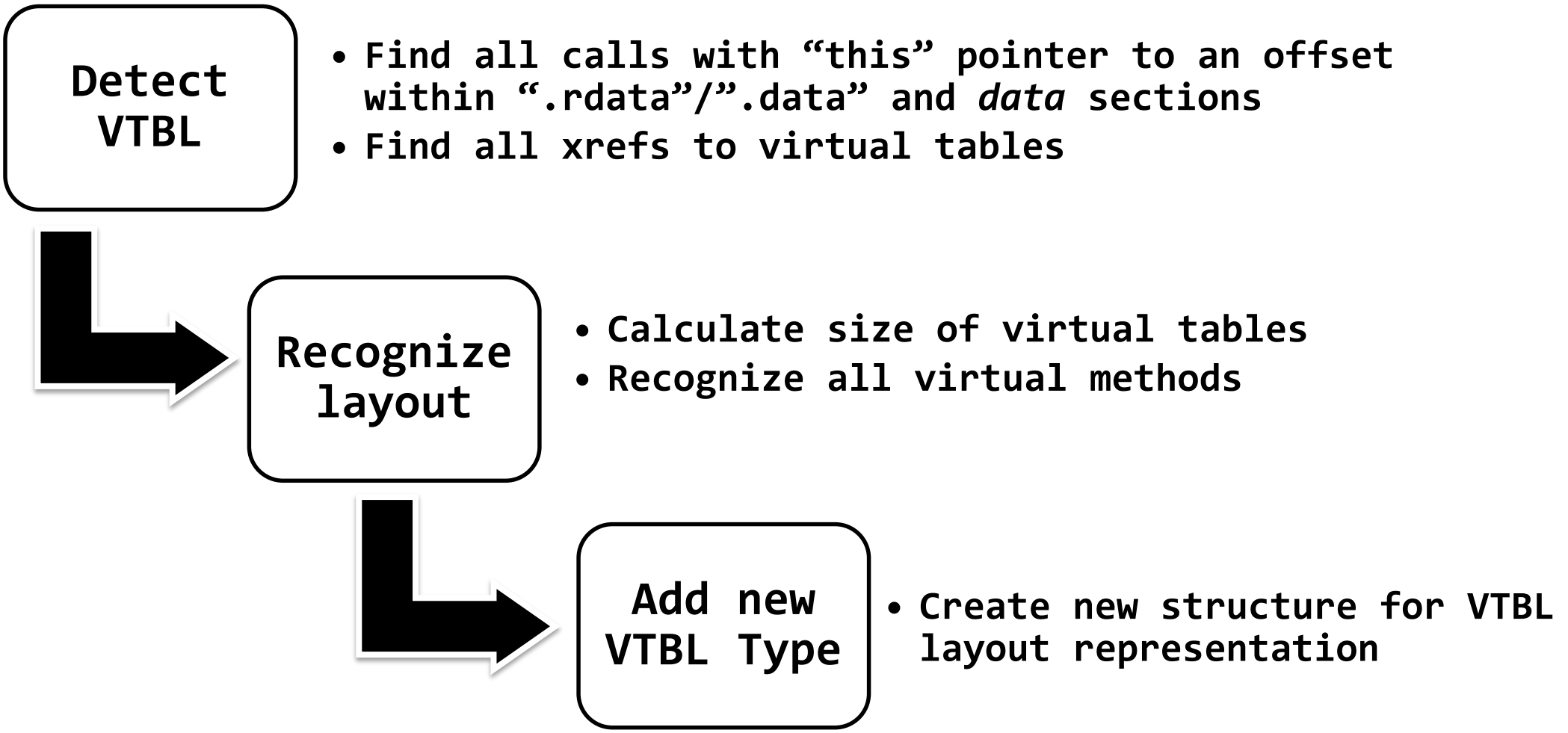
Obtain IR

- Decompile routine to get ctree (IR)
- Serialize ctree to string

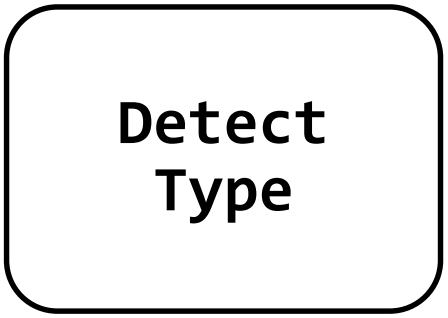
Ctree normalization

- See implementation of *ctree_dumper_t::filter_citem()*
- Use normalized ctree for comparison

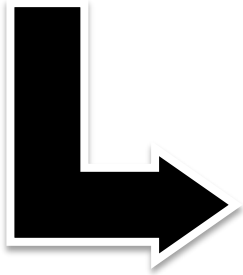
Constraints and Limitations:
VTBL reconstruction algorithm

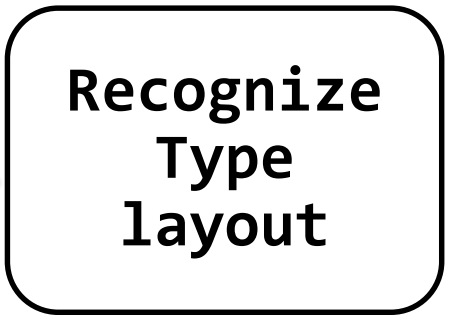


Constraints and Limitations :
Complex types REconstruction algorithm

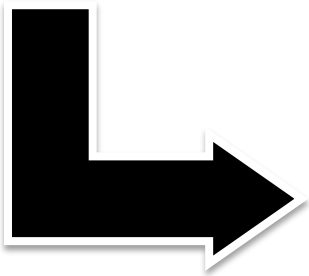


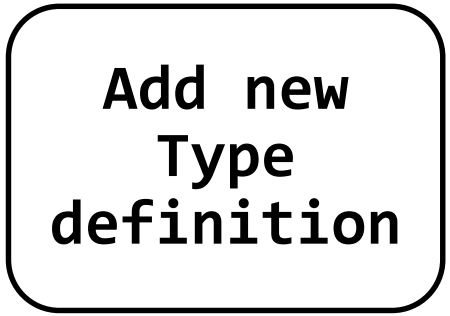
- Find pointers to possible type instances
- Find initialization routine entry point





- Find all references to possible type address space
- Find all xrefs to the attributes of the identified type
- Reconstruct data flow for the identified type



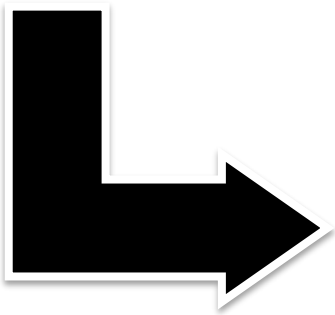


- Create new local type if it has more than 3 attributes

Constraints and Limitations:
Ctrees Depth Analysis

Enumerate code xrefs to the routine

- Use breadth-first search algorithm
- Limit: 100 nodes



Get statistics

- Distance from entry point
- depth counter
- number of xrefs

Constraints and Limitations:
AES-NI and GETSEC Detection

Analyze code sections

- Entry point section is always analyzed

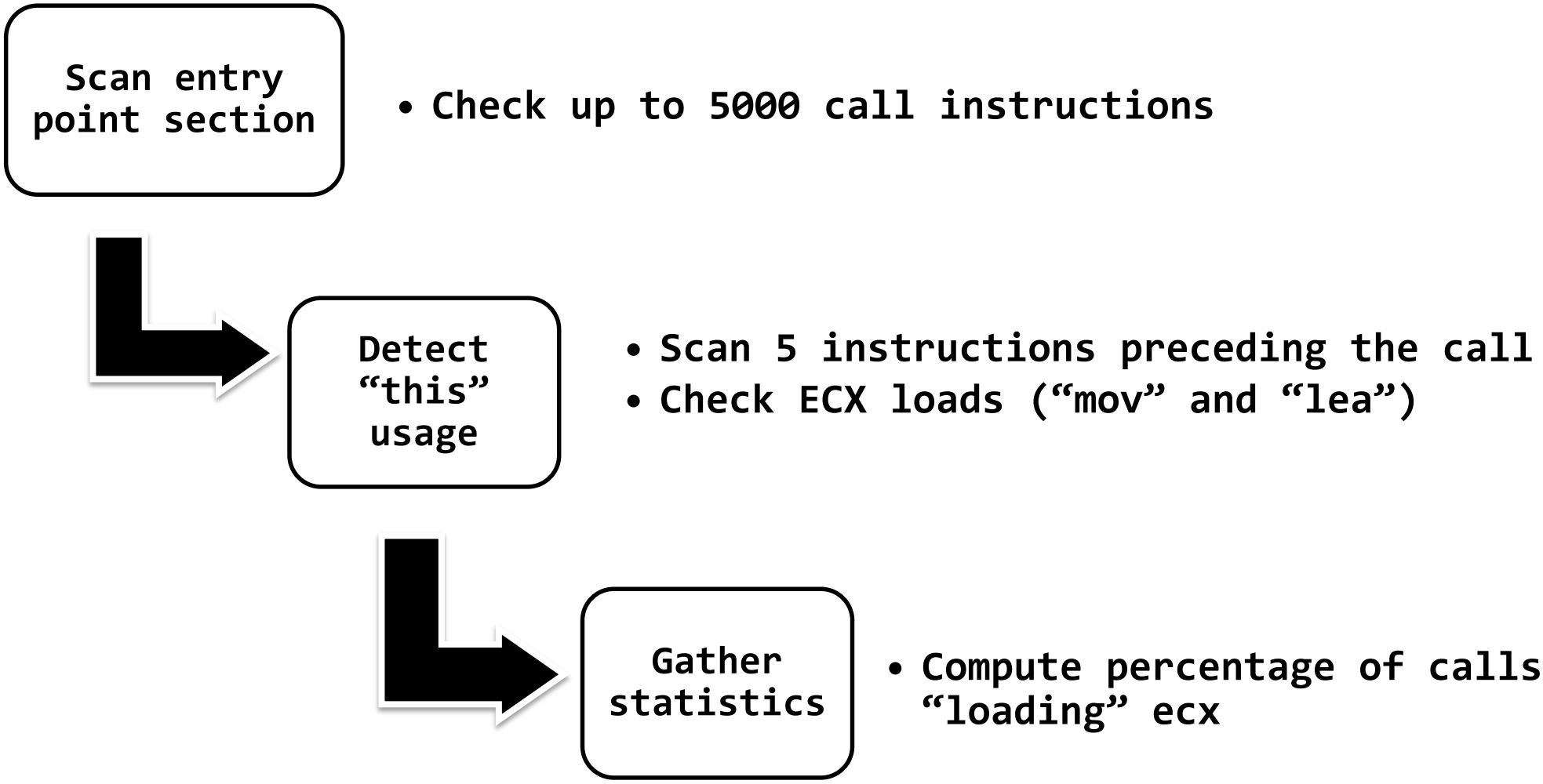
Scan first 512 Kb of sections

- Disassemble with linear sweep
- Reject if disassembly > 20 Mb

Detect instructions

- Check for GETSEC and all AES-NI instructions
- Reject match if a “bad” is present in 15 surrounding instructions

Constraints and Limitations:
C++ “this” usage study



Distributing IDA Pro: Highlights

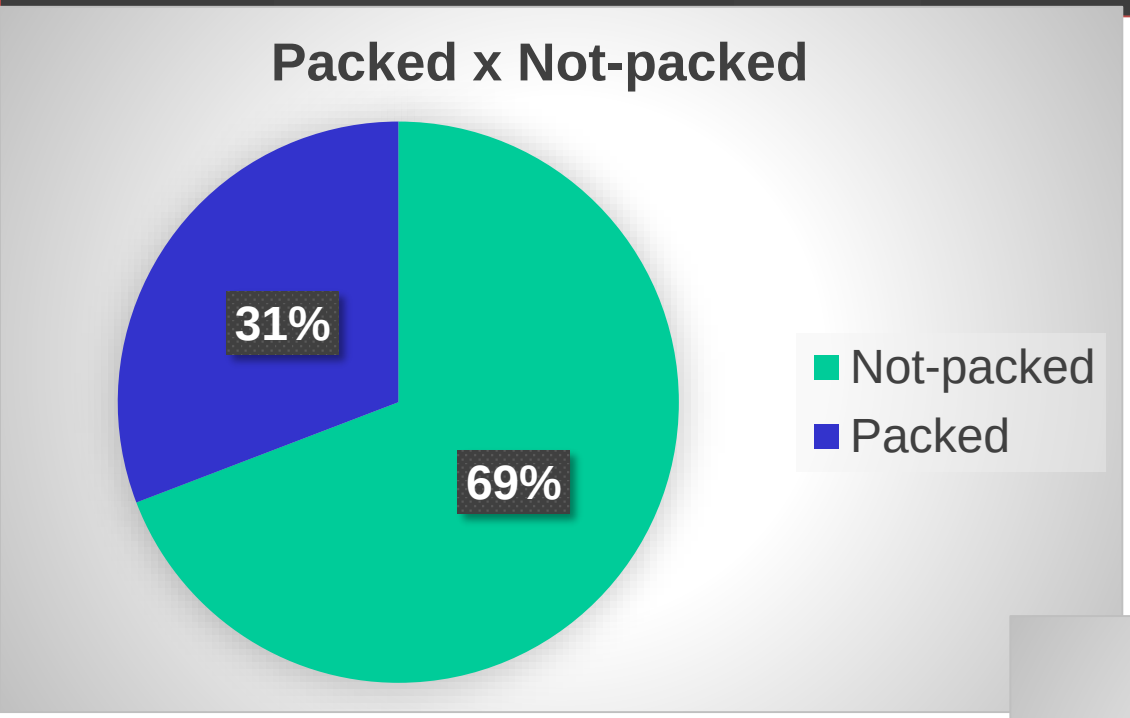
- **Unexpected performance benefits on IDA because the information is structured**
  - ✓ But we also came across some disadvantages: SDK is complex, function signatures change from version to version and is not fully documented
- **Good performance in commodity hardware**
- **C-based plugins are usually not compatible with Linux/Mac**
  - ✓ Portability efforts are required

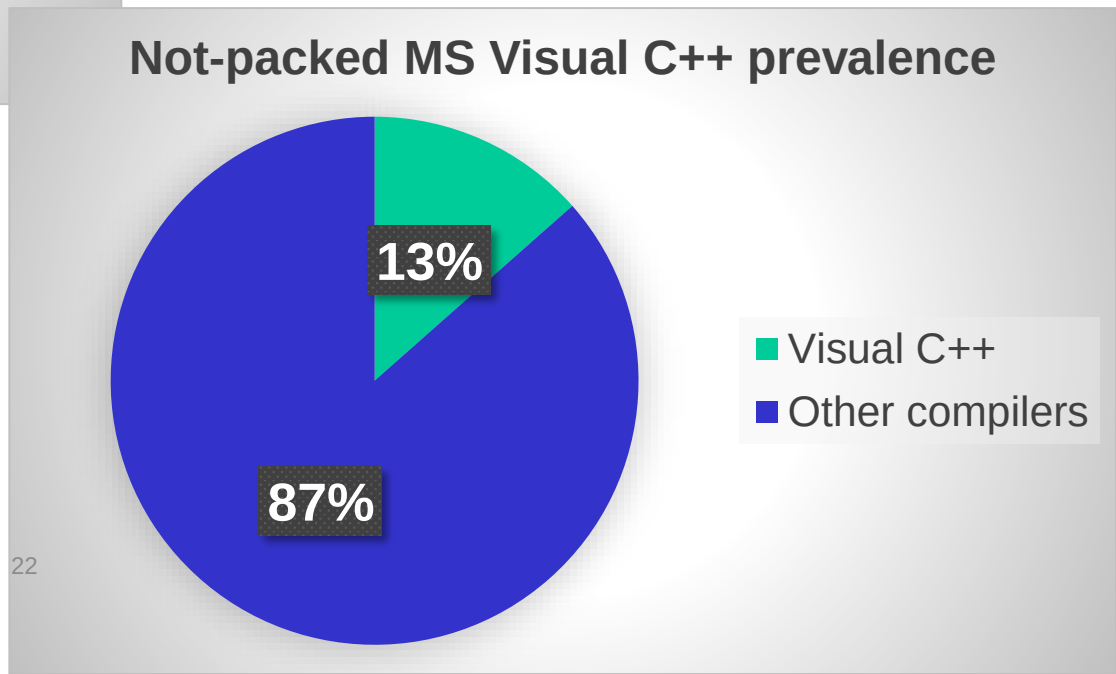
Distributing IDA Pro: Highlights

- **IDA plugins are usually not made to scale**
  - **Target single-sample analysis**
  - **Focus on users interacting with IDA Pro interface**
- **Automated malware analysis exercises much more the internal plugin flows than manual analysis**
  - ✓ **As a result, corner cases and bugs were identified in many plugins including HexRaysCodeXplorer**

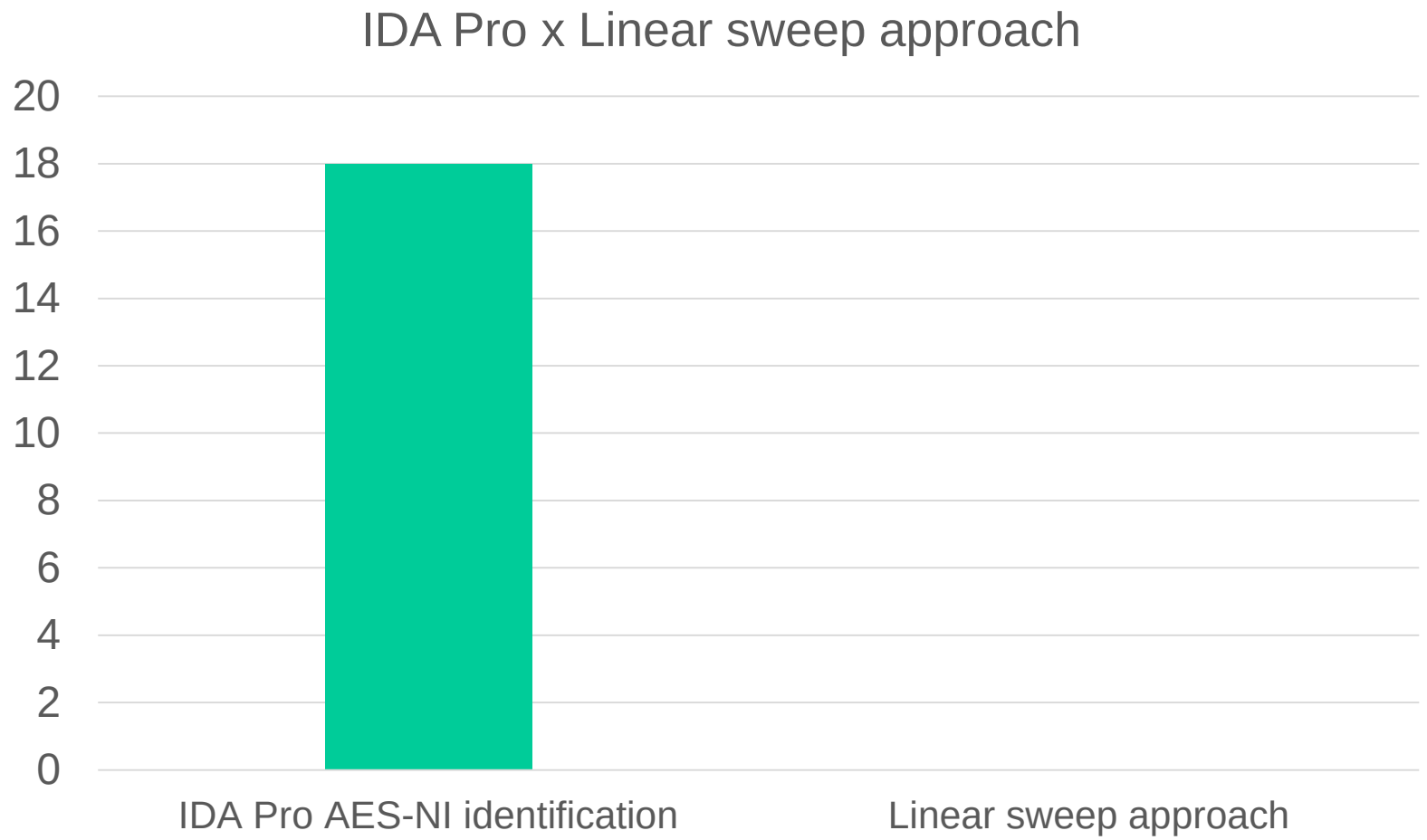
Results

Pre-processing – Total: **7,829,441**

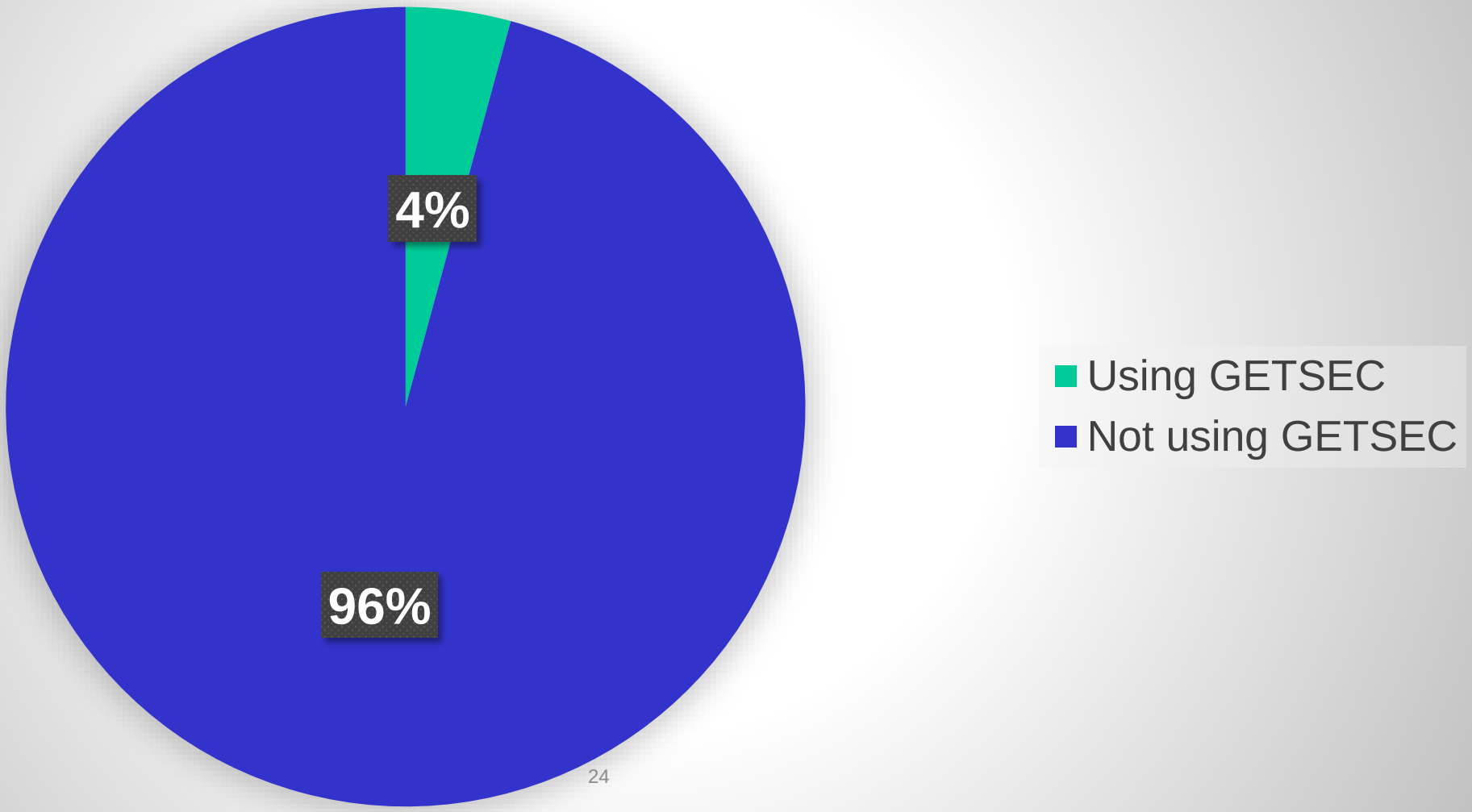




AES-NI Usage (IDA Pro x standalone)



GETSEC Usage



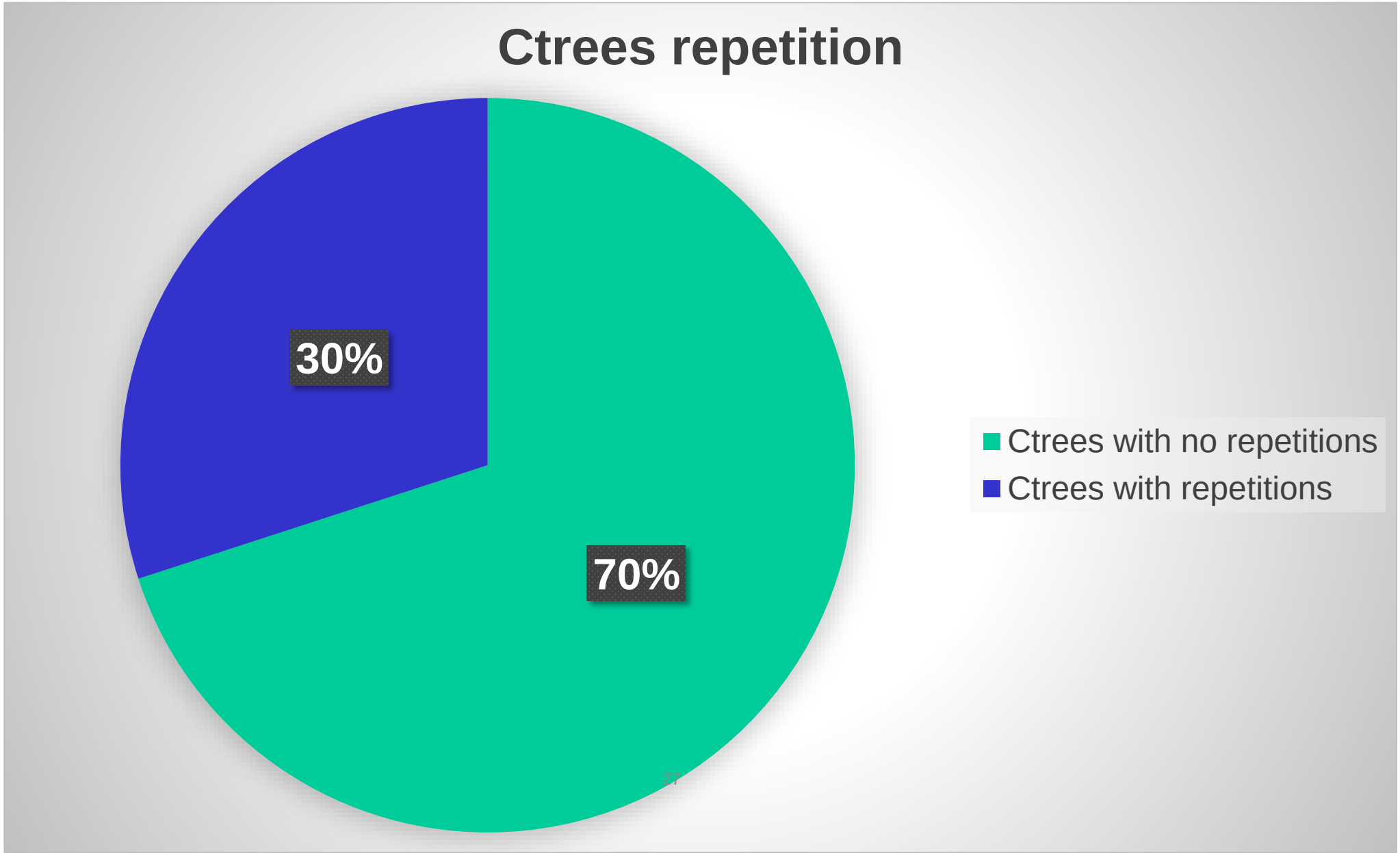
C++ “this” Usage Study – Top 10 Percentages

% calls loading ECX	Prevalence (%)
4	7.420991619
18	5.845574961
30	5.810101164
10	5.247588099
16	4.788962581
5	4.468431488
3	4.348707424
19	3.988901769
20	3.905284962
46	3.193908642

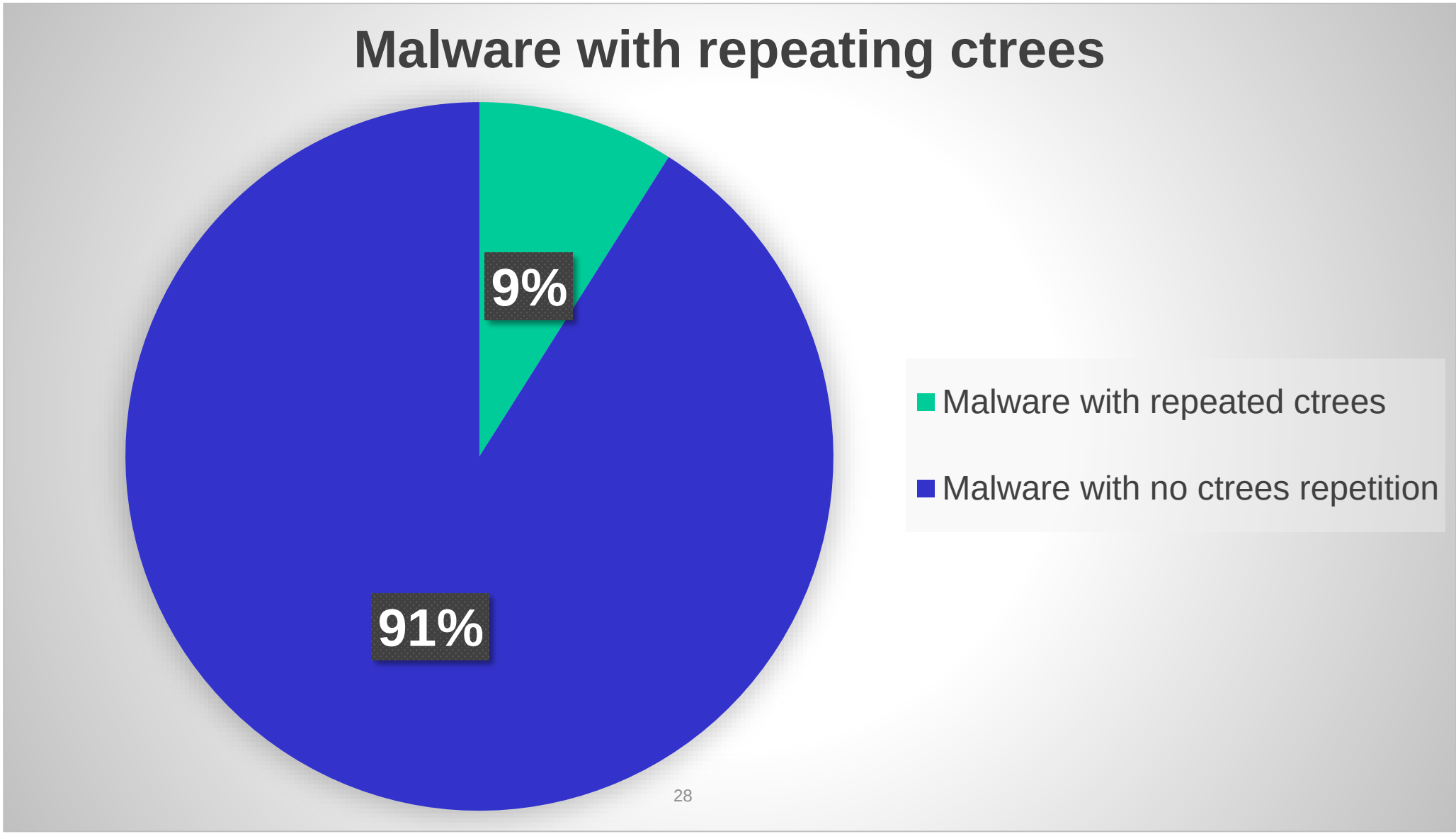
Ctrees: Top 10 repeated ctrees (with repetition number). Total: 8,422,576

	Number of repetitions	Percentage
	40606	0.482109036
	38800	0.460666665
	34718	0.412201683
	20190	0.239712886
	19999	0.237445171
	17635	0.209377749
	17060	0.202550859
	14959	0.177605996
	14439	0.171432113
	14072	0.167074776
Total	232478	2.760176934

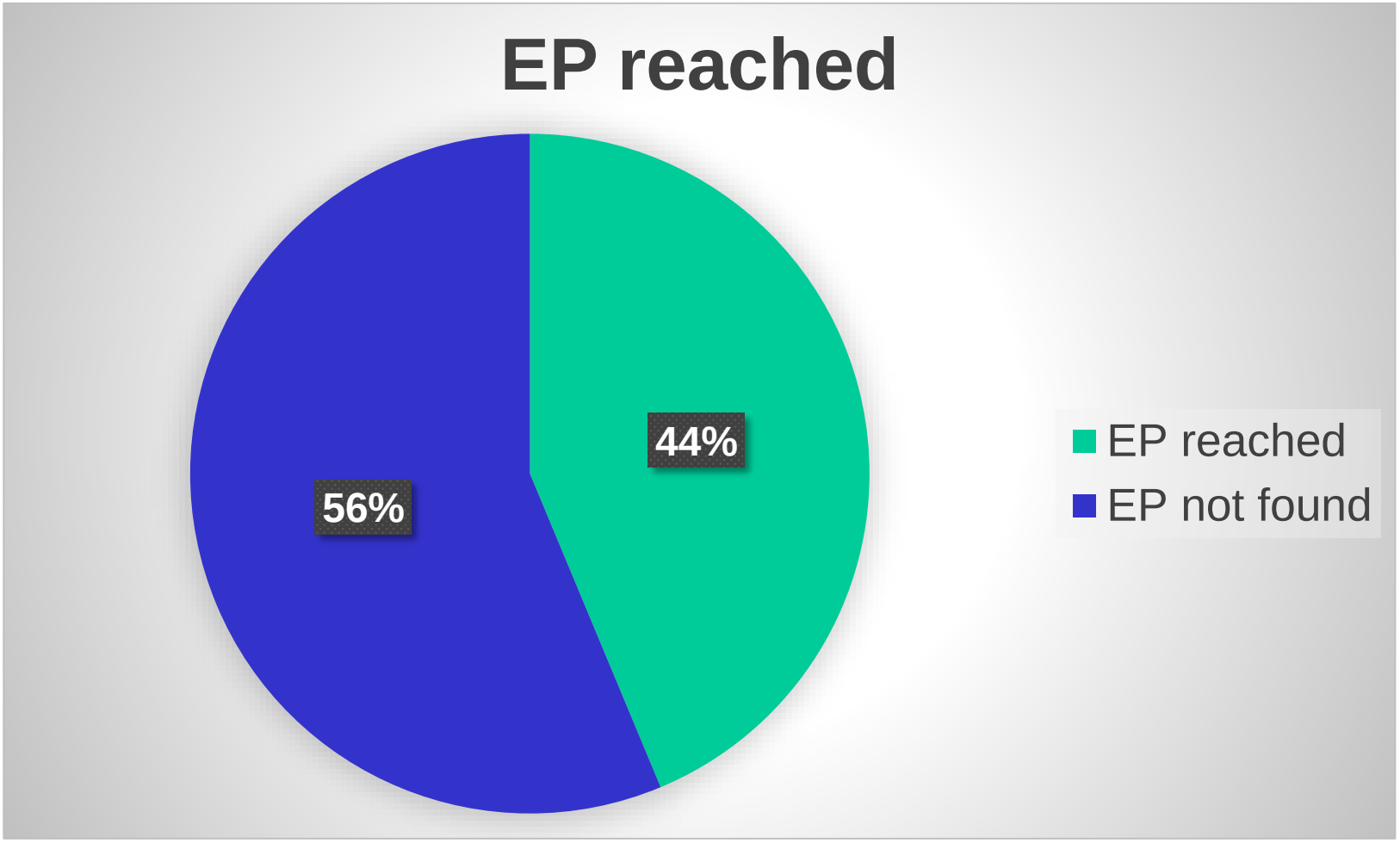
Unique Ctrees: Repeated x Not-Repeated



Ctrees: samples with repeated x non-repeated ctrees



Ctrees reaching EP + avg + std of their depth



EP reached → Average depth: 5.1940 (standard deviation: 2.3588)
82,646 or 0.98% of ctrees are directly under the EP

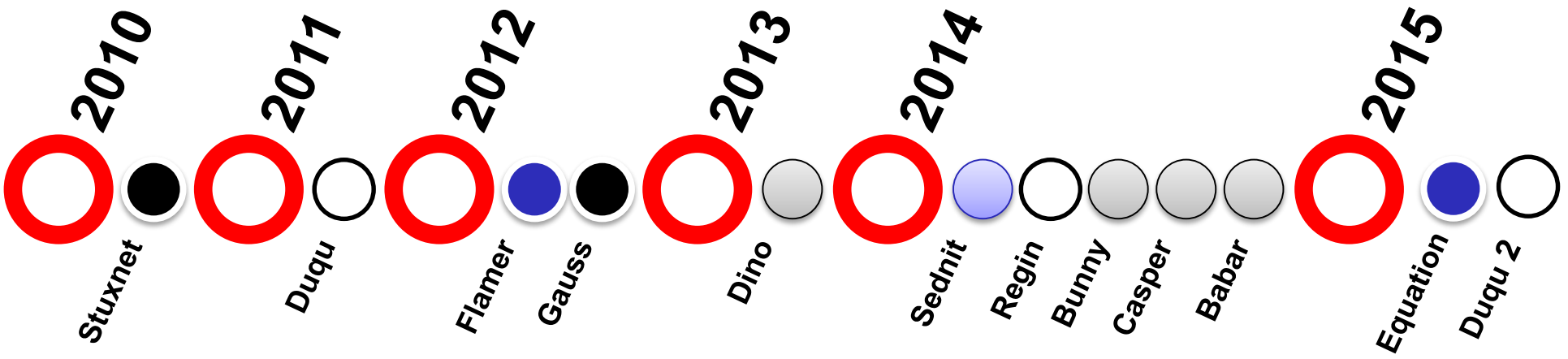
Ctrees max parents (code xref) – Top 10

Top 10 - Max number of parents	
Number of parents	**Occurrences**
11126	1
10989	3
9463	1
9023	1
8907	1
8837	2
8794	1
8226	1
7536	1
6917	5

VALIDATING THE METHODOLOGY AND TOOLSET

ANALYSIS OF C++ TARGETED MALWARE

Modern C++ Malware in Targeted Attacks



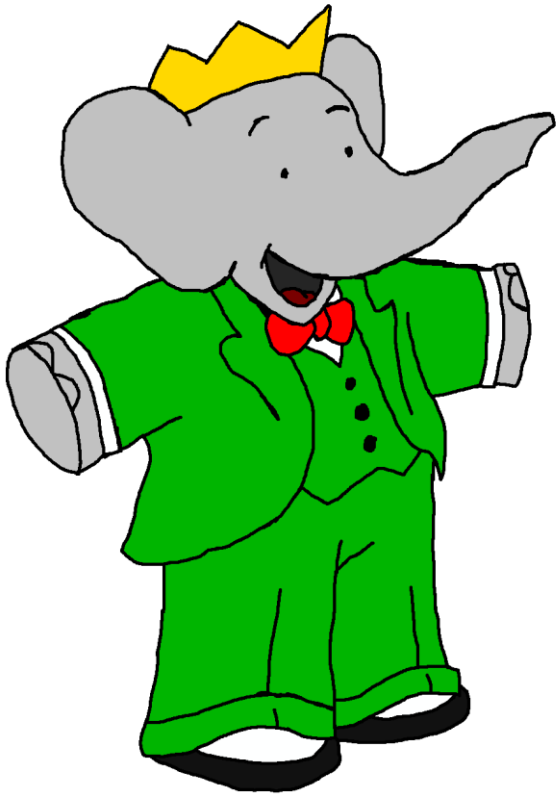
-- Stuxnet relations

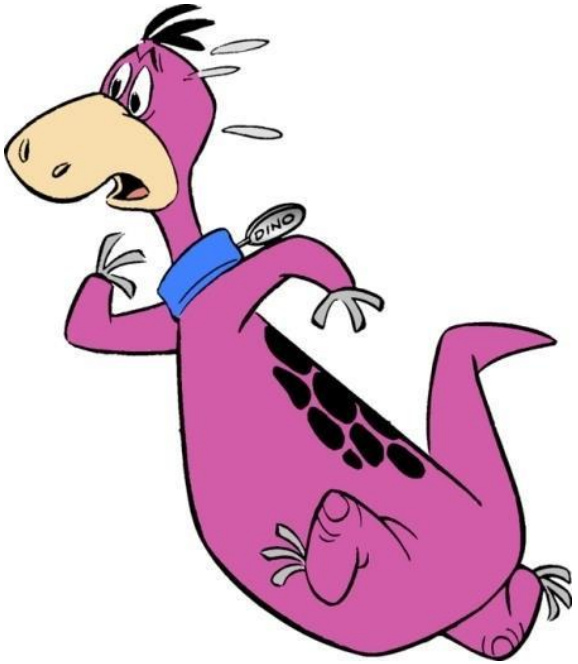
-- Duqu relations

-- Equation relations

-- Animal Farm family

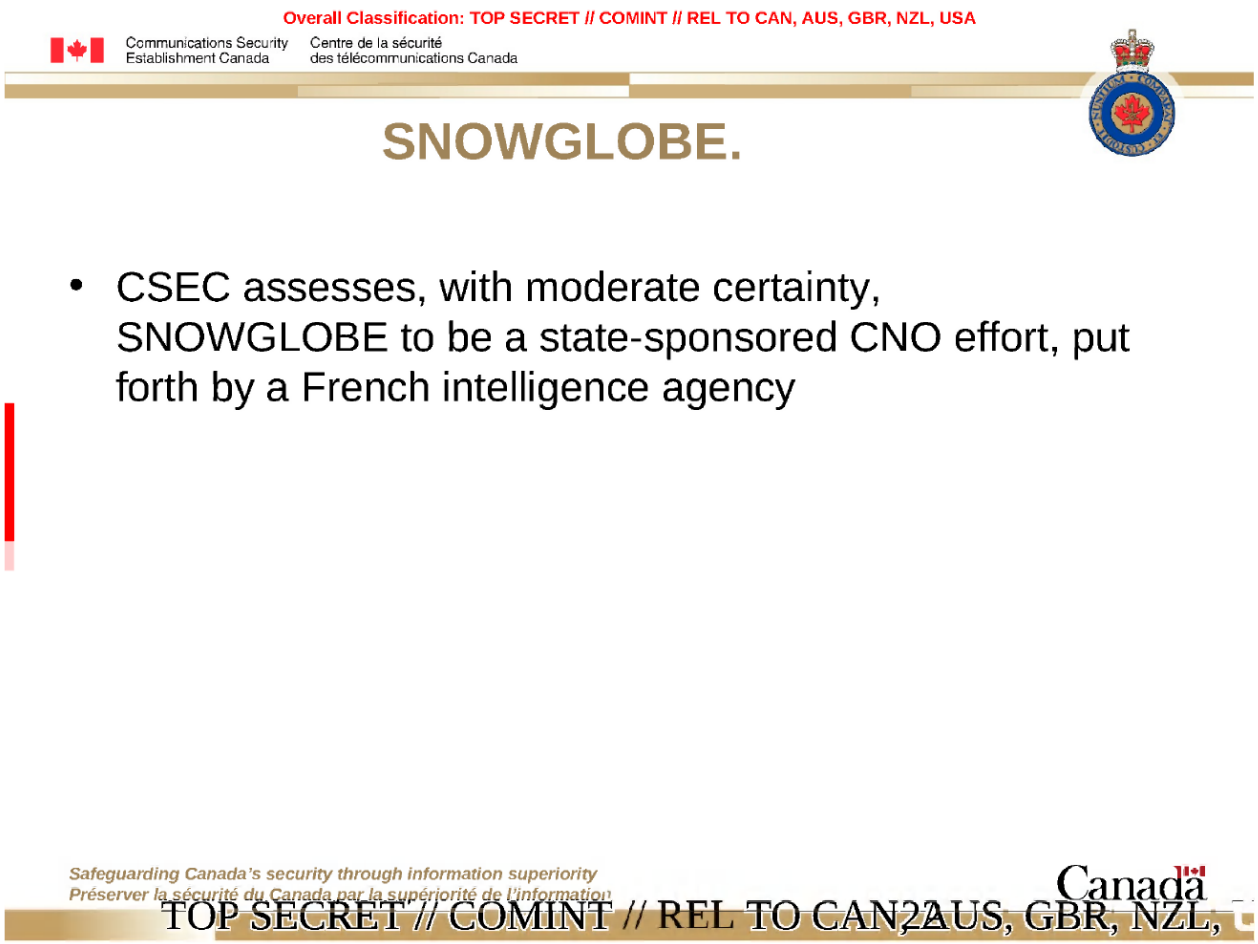
Animal Farm Case Study





Animal Farm* Case Study

- Discovered by CSEC as operation SNOWGLOBE
- Samples: NBOT, Dino, Babar, Bunny, Casper
- Written in MS Visual C++



** - "Totally Spies", Joan Calvet, Marion Marschalek, Paul Rascagnères, http://recon.cx/2015/slides/recon2015-01-joan-calvet-marion-marschalek-paul-rascagneres-Totally-Spies.pdf*

Casper vs. Dino in HexRaysCodeXplorer

Casper's virtual function tables:

```
0x417240 - 0x417250:  VTABLE_CLASS1  methods count: 4
0x417250 - 0x41725c:  VTABLE_AUTODEL_INTERFACE  methods count: 3
0x41725c - 0x417274:  VTABLE_CLASS3  methods count: 6
0x417274 - 0x41728c:  VTABLE_SCHTASKS  methods count: 6
0x41748c - 0x41749c:  VTABLE_RUNKEY_REG  methods count: 4
0x41749c - 0x4174ac:  VTABLE_RUNKEY_BAT  methods count: 4
0x4174ac - 0x4174bc:  VTABLE_RUNKEY_API  methods count: 4
0x4174bc - 0x4174cc:  VTABLE_RUNKEY_WMI  methods count: 4
0x4175fc - 0x41760c:  VTABLE_RUNKEY_DEFAULT  methods count: 4
0x41760c - 0x417618:  VTABLE_AUTODEL_DEL  methods count: 3
0x41774c - 0x417758:  VTABLE_Autodel_API  methods count: 3
0x417758 - 0x417764:  VTABLE_Autodel_WMI  methods count: 3
0x417944 - 0x417950:  VTABLE_AUTODEL_DEFAULT_DEL  methods count: 3
0x417a9c - 0x417ab0:  VTABLE_CLASS14  methods count: 5
0x417ab0 - 0x417ac4:  VTABLE_CLASS15  methods count: 5
0x417af4 - 0x417b08:  VTABLE_CLASS16  methods count: 5
0x417b30 - 0x417b38:  VTABLE_RUNKEY_PARENT  methods count: 2
0x418afc - 0x418b08:  VTABLE_CLASS18  methods count: 3
0x418b08 - 0x418b14:  VTABLE_PROCESSINJ  methods count: 3
0x418b14 - 0x418b20:  VTABLE_CLASS20  methods count: 3
0x418b20 - 0x418b2c:  VTABLE_CLASS21  methods count: 3
0x418b50 - 0x418b58:  VTABLE_CLASS22  methods count: 2
0x418d68 - 0x418d70:  const std::bad_alloc::`vftable'  methods count: 2
0x418d84 - 0x418d8c:  const std::exception::`vftable'  methods count: 2
```

Dino's virtual function tables:

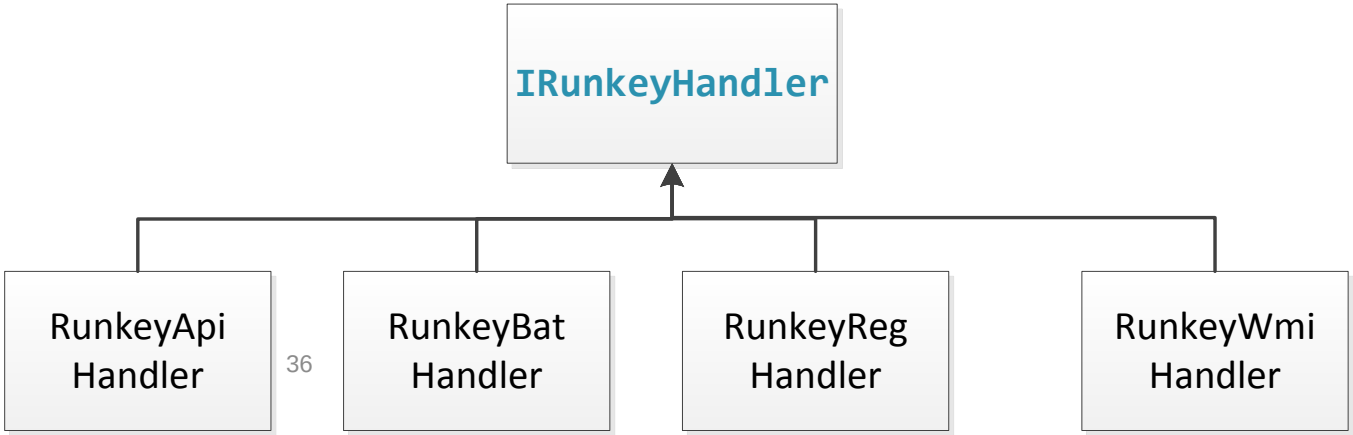
```
0x1061368 - 0x1061370:  const std::bad_alloc::`vftable'  methods count: 2
0x1061b30 - 0x1061b38:  const std::exception::`vftable'  methods count: 2
0x106aac0 - 0x106aad0:  VTABLE_RUNKEY_INTERFACE  methods count: 4
0x106aad0 - 0x106aae0:  VTABLE_RUNKEY_API  methods count: 4
0x106aae0 - 0x106aaf0:  VTABLE_RUNKEY_REG  methods count: 4
0x106aaf0 - 0x106ab00:  VTABLE_RUNKEY_BAT  methods count: 4
0x106ab00 - 0x106ab10:  off_106AB00  methods count: 4
0x106ab10 - 0x106ab2c:  VTABLE_SERVICE_INTERFACE  methods count: 7
0x106ab2c - 0x106ab48:  VTABLE_SERVICE_API  methods count: 7
0x106ab48 - 0x106ab64:  VTABLE_SERVICE_SC  methods count: 7
0x106ab64 - 0x106ab80:  off_106AB64  methods count: 7
0x106ab80 - 0x106ab88:  VTABLE_AUTODEL_INTERFACE  methods count: 2
0x106ab88 - 0x106ab90:  VTABLE_AUTODEL_API  methods count: 2
0x106ab90 - 0x106ab98:  VTABLE_AUTODEL_DEL  methods count: 2
0x106ab98 - 0x106aba0:  off_106AB98  methods count: 2
0x106aba4 - 0x106abc4:  off_106ABA4  methods count: 8
0x106abc4 - 0x106ac00:  VT_CTFC_AbstractSocket  methods count: 15
0x106ac00 - 0x106ac3c:  VT_CTFC_StandardSocket  methods count: 15
0x106ac3c - 0x106ac48:  VT_CTFC_HTTP_Request  methods count: 3
0x106ac48 - 0x106ac5c:  VT_CTFC_HTTP_Forms  methods count: 5
0x106ac5c - 0x106ac70:  VT_CTFC_HTTP_Form  methods count: 5
0x106ac70 - 0x106ac84:  VT_CTFC_HTTP_Form_Multipart  methods count: 5
0x106ac84 - 0x106aca4:  VT_COM_SERV  methods count: 8
0x106aca4 - 0x106acb4:  VT_FS  methods count: 4
0x106acb4 - 0x106acc4:  VT_RAMFS  methods count: 4
0x106acc4 - 0x106acd8:  off_106ACC4  methods count: 5
0x106acd8 - 0x106acec:  VT_CLASS2  methods count: 5
0x106acec - 0x106ad00:  VT_STORAGE_FILE  methods count: 5
0x106ad00 - 0x106ad14:  VT_STORAGE_REG  methods count: 5
0x106b0b0 - 0x106b0b8:  const std::bad_exception::`vftable'  methods count: 2
```

Casper vs. Dino: RUNKEY

Defines how the dropper interacts with the Windows Registry:

- ✓ **API – call Windows Registry APIs directly**
- ✓ **BAT – modify Windows registry in a batch file using "reg" commands**
- ✓ **REG – modify Windows registry by using "reg" command in a command prompt**
- ✓ **WMI – modify Windows registry by using StdRegProv class**

```
struct IRunkeyHandler
{
    LPVOID addKey;
    LPVOID deleteKey;
    LPVOID queryKey;
    LPVOID destructor;
};
```

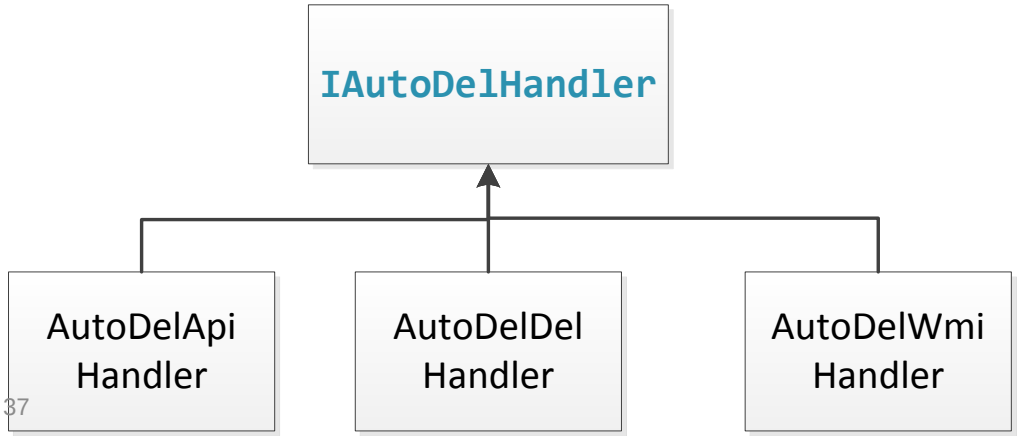


Casper vs. Dino: AUTODEL

Defines how dropper removes itself from machine after its execution

- ✓ **DEL – remove itself by using command prompt**
- ✓ **API – remove itself by calling *MoveFileEx***
- ✓ **WMI – remove itself by using command prompt created through create method of the *Win32_Process* WMI class**

```
struct IAutoDelHandler {
    LPVOID delete;
    LPVOID deleteAscii;
    LPVOID destructor;
};
```



Object Instantiation: Constructors

Casper's RUNKEY constructor:

```
  FN_BuildAVStrat();
  if ( AV_STRATEGY_RUNKEY_API )
  {
    v2 = operator new(0xCu);
    v3 = v2;
    if ( v2 )
    {
      FN_GetRunKey(v2);
      *v3 = &VTABLE_RUNKEY_API;
LABEL_18:
      *v1 = v3;
      return v1;
    }
    goto LABEL_17;
  }
  if ( AV_STRATEGY_RUNKEY_REG )
  {
    v4 = operator new(0xCu);
    v3 = v4;
    if ( v4 )
    {
      FN_GetRunKey(v4);
      *v3 = &VTABLE_RUNKEY_REG;
      goto LABEL_18;
    }
LABEL_17:
    v3 = 0;
    goto LABEL_18;
  }
```

Dino's RUNKEY constructor:

```
sub_1005F80(&a1->vftbl_1_0106AAD0);
a1->vftbl_1_0106AAD0 = &VTABLE_RUNKEY_API;
v16 = 0;
sub_1005F80(&a1->vftbl_2_0106AAE0);
a1->vftbl_2_0106AAE0 = &VTABLE_RUNKEY_REG;
sub_1005F80(&a1->vftbl_3_0106AAF0);
a1->vftbl_3_0106AAF0 = &VTABLE_RUNKEY_BAT;
sub_1005F80(&a1->vftbl_4_0106AB00);
a1->vftbl_4_0106AB00 = &VTABLE_RUNKEY_DEFAULT;
LOBYTE(v16) = 3;
v1 = FN_GetAPIModules_();
v11 = *v1;
v12 = *(v1 + 4);
v13 = *(v1 + 8);
v2 = 0;
v14 = *(v1 + 12);
v3 = *(v1 + 16);
v7 = &a1->vftbl_4_0106AB00;
v8 = &a1->vftbl_1_0106AAD0;
v9 = &a1->vftbl_2_0106AAE0;
v10 = &a1->vftbl_3_0106AAF0;
v15 = v3;
if ( v13 )
{
  v11 = *v1;
  v12 = *(v1 + 4);
  v13 = *(v1 + 8);
  v4 = *(v1 + 12);
  v15 = *(v1 + 16);
  v14 = v4;
  do
    v5 = v13 >> v2++;
  while ( !(v5 & 1) && v2 < 8u );
}
result = a1;
a1->field_0 = (&v7)[v2];
```

Object Instantiation: Type REconstruction

Casper's RUNKEY constructor:

```
FN_BuildAVStrat();
if ( AV_STRATEGY_RUNKEY_API )
{
  v2 = operator new(0xCu);
```

Please enter text

Please edit the type declaration

```
struct struct_name_RUNKEY
{
  struct_name_RUNKEY_VTABLE_0_004175FC *vftbl_0_004175FC;
  int field0;
  int field1;
};
```

OK Cancel Help

```
    FN_GetRunKey(v4);
    *v3 = &VTABLE_RUNKEY_REG;
    goto LABEL_18;
  }
LABEL_17:
  v3 = 0;
  goto LABEL_18;
```

!=

Dino's RUNKEY constructor:

```
sub_1005F80(&a1->vftbl_1_0106AAD0);
a1->vftbl_1_0106AAD0 = &VTABLE_RUNKEY_API;
v16 = 0;
sub_1005F80(&a1->vftbl_2_0106AAE0);
```

Please enter text

Please edit the type declaration

```
struct struct_name_RUNKEY
{
  int field_0;
  struct_name_RUNKEY_VTABLE_4_0106AAD0 *vftbl_1_0106AAD0;
  _BYTE gap8[8];
  struct_name_RUNKEY_VTABLE_16_0106AAE0 *vftbl_2_0106AAE0;
  _BYTE gap14[8];
  struct_name_RUNKEY_VTABLE_28_0106AAF0 *vftbl_3_0106AAF0;
  _BYTE gap20[8];
  struct_name_RUNKEY_VTABLE_40_0106AB00 *vftbl_4_0106AB00;
};
```

OK Cancel Help

```
  v13 = *(v1 + 8);
  v4 = *(v1 + 12);
  v15 = *(v1 + 16);
  v14 = v4;
  do
    v5 = v13 >> v2++;
  while ( !(v5 & 1) && v2 < 8u );
}
result = a1;
a1->field_0 = (&v7)[v2];
```

Dino vs. NBOT in HexRaysCodeXplorer

Dino's virtual function tables:

```
0x106aac0 - 0x106aad0:  VTABLE_RUNKEY_INTERFACE  methods count: 4
0x106aad0 - 0x106aae0:  VTABLE_RUNKEY_API  methods count: 4
0x106aae0 - 0x106aaf0:  VTABLE_RUNKEY_REG  methods count: 4
0x106aaf0 - 0x106ab00:  VTABLE_RUNKEY_BAT  methods count: 4
0x106ab00 - 0x106ab10:  off_106AB00  methods count: 4
0x106ab10 - 0x106ab2c:  VTABLE_SERVICE_INTERFACE  methods count: 7
0x106ab2c - 0x106ab48:  VTABLE_SERVICE_API  methods count: 7
0x106ab48 - 0x106ab64:  VTABLE_SERVICE_SC  methods count: 7
0x106ab64 - 0x106ab80:  off_106AB64  methods count: 7
0x106ab80 - 0x106ab88:  VTABLE_AUTODEL_INTERFACE  methods count: 2
0x106ab88 - 0x106ab90:  VTABLE_AUTODEL_API  methods count: 2
0x106ab90 - 0x106ab98:  VTABLE_AUTODEL_DEL  methods count: 2
0x106ab98 - 0x106aba0:  off_106AB98  methods count: 2
0x106aba4 - 0x106abc4:  off_106ABA4  methods count: 8
0x106abc4 - 0x106ac00:  VT_CTFC_AbstractSocket  methods count: 15
0x106ac00 - 0x106ac3c:  VT_CTFC_StandardSocket  methods count: 15
0x106ac3c - 0x106ac48:  VT_CTFC_HTTP_Request  methods count: 3
0x106ac48 - 0x106ac5c:  VT_CTFC_HTTP_Forms  methods count: 5
0x106ac5c - 0x106ac70:  VT_CTFC_HTTP_Form  methods count: 5
0x106ac70 - 0x106ac84:  VT_CTFC_HTTP_Form_Multipart  methods count: 5
0x106ac84 - 0x106aca4:  VT_CUM_SERV  methods count: 8
0x106aca4 - 0x106acb4:  VT_FS  methods count: 4
0x106acb4 - 0x106acc4:  VT_RAMFS  methods count: 4
0x106acc4 - 0x106acd8:  off_106ACC4  methods count: 5
0x106acd8 - 0x106acec:  VT_CLASS2  methods count: 5
0x106acec - 0x106ad00:  VT_STORAGE_FILE  methods count: 5
0x106ad00 - 0x106ad14:  VT_STORAGE_REG  methods count: 5
0x106b0b0 - 0x106b0b8:  const std::bad_exception::`vftable'  methods count: 2
```

NBOT's virtual function tables:

```
0x43d95c - 0x43d964:  const wmiException::`vftable'  methods count: 2
0x43d980 - 0x43d988:  const NBOT_Handler::`vftable'  methods count: 2
0x43d98c - 0x43d9c8:  const CTFC_AbstractSocket::`vftable'  methods count: 15
0x43d9cc - 0x43da08:  const CTFC_StandardSocket::`vftable'  methods count: 15
0x43da0c - 0x43da18:  const CTFC_HTTP_Request::`vftable'  methods count: 3
0x43da1c - 0x43da30:  const CTFC_HTTP_Forms::`vftable'  methods count: 5
0x43da34 - 0x43da48:  const CTFC_HTTP_Form::`vftable'  methods count: 5
0x43da4c - 0x43da60:  const CTFC_HTTP_Form_Multipart::`vftable'  methods count: 5
0x43da64 - 0x43da6c:  const NBOT_CUM::`vftable'  methods count: 2
0x43da70 - 0x43da78:  const NBOT_ATCLEAR::`vftable'  methods count: 2
0x43da7c - 0x43da84:  const NBOT_AT::`vftable'  methods count: 2
0x43da88 - 0x43da90:  const NBOT_PING::`vftable'  methods count: 2
0x43da94 - 0x43da9c:  const NBOT_EXEC::`vftable'  methods count: 2
0x43daa0 - 0x43daa8:  const NBOT_HTTP_FLOOD::`vftable'  methods count: 2
0x43daac - 0x43dab4:  const NBOT_ASP_FLOOD::`vftable'  methods count: 2
0x43dab8 - 0x43dac4:  const NBOT_TCP_FLOOD::`vftable'  methods count: 3
0x43dac8 - 0x43dad4:  const NBOT_WEB_FLOOD::`vftable'  methods count: 3
0x43dad8 - 0x43dae4:  const NBOT_WEB_POST_FLOOD::`vftable'  methods count: 3
0x43dae8 - 0x43daf0:  const NBOT_STATISTICS::`vftable'  methods count: 2
0x43daf4 - 0x43dafc:  const NBOT_KILLER::`vftable'  methods count: 2
0x43db00 - 0x43db08:  const NBOT_CONFIG::`vftable'  methods count: 2
0x43db0c - 0x43db14:  const NBOT_UPLOAD::`vftable'  methods count: 2
0x43db18 - 0x43db20:  const NBOT_UPDATE::`vftable'  methods count: 2
```

Exploring NBOT's RTTI

Vftable	Methods	Flags	Type	Hierarchy
0043D98C	15		CTFC_AbstractSocket	CTFC_AbstractSocket:
0043D968	1		CTFC_Anti_AV_Mailslot	CTFC_Anti_AV_Mailslot: CTFC_Anti_AV_Interface;
0043D970	1		CTFC_Anti_AV_NULL	CTFC_Anti_AV_NULL: CTFC_Anti_AV_Interface;
0043DA34	5		CTFC_HTTP_Form	CTFC_HTTP_Form: CTFC_HTTP_Forms, CTFC_HTTP_Request;
0043DA4C	5		CTFC_HTTP_Form_Multipart	CTFC_HTTP_Form_Multipart: CTFC_HTTP_Forms, CTFC_HTTP_Request;
0043DA1C	5		CTFC_HTTP_Forms	CTFC_HTTP_Forms: CTFC_HTTP_Request;
0043DA0C	3		CTFC_HTTP_Request	CTFC_HTTP_Request:
0043D9CC	15	M	CTFC_StandardSocket	CTFC_StandardSocket: CTFC_AbstractSocket, CNoImport;
0043DAAC	2		NBOT_ASP_FLOOD	NBOT_ASP_FLOOD: NBOT_Handler;
0043DA7C	2		NBOT_AT	NBOT_AT: NBOT_Handler;
0043DA70	2		NBOT_ATCLEAR	NBOT_ATCLEAR: NBOT_Handler;
0043DA64	2		NBOT_COM	NBOT_COM: NBOT_Handler;
0043DB00	2		NBOT_CONFIG	NBOT_CONFIG: NBOT_Handler;
0043DA94	2		NBOT_EXEC	NBOT_EXEC: NBOT_Handler;
0043DAA0	2		NBOT_HTTP_FLOOD	NBOT_HTTP_FLOOD: NBOT_Handler;
0043D980	2		NBOT_Handler	NBOT_Handler:
0043DAF4	2		NBOT_KILLER	NBOT_KILLER: NBOT_Handler;
0043DA88	2		NBOT_PING	NBOT_PING: NBOT_Handler;
0043DAE8	2		NBOT_STATISTICS	NBOT_STATISTICS: NBOT_Handler;
0043DAB8	3		NBOT_TCP_FLOOD	NBOT_TCP_FLOOD: NBOT_Handler;
0043DB18	2		NBOT_UPDATE	NBOT_UPDATE: NBOT_Handler;
0043DB0C	2		NBOT_UPLOAD	NBOT_UPLOAD: NBOT_Handler;
0043DAC8	3		NBOT_WEB_FLOOD	NBOT_WEB_FLOOD: NBOT_TCP_FLOOD, NBOT_Handler;
0043DAD8	3		NBOT_WEB_POST_FLOOD	NBOT_WEB_POST_FLOOD: NBOT_TCP_FLOOD, NBOT_Handler;

Type REconstruction: CTFC_HTTP_Form_Multipart

Dino

```
struct_name_2 *__thiscall FN_BuildCTFC_HTTP_Form_Multipart(int this, char *a2)
{
  int v2; // edi@1
  unsigned int v3; // ST10_4@5
  unsigned __int16 v4; // ax@5
  int v5; // eax@5
  char v7; // [sp+18h] [bp-110h]@5
  int v8; // [sp+124h] [bp-4h]@1

  v2 = this;
  sub_1018370(this, this);
  v8 = 0;
  *(_DWORD *)v2 = VT_CTFC_HTTP_Form_Multipart;
  *(_DWORD *)(v2 + 288) = CreateMutexW(0, 0, 0);
  *(_DWORD *)(v2 + 284) = 0;
  *(_DWORD *)(v2 + 280) = 0;
  *(_DWORD *)(v2 + 300) = 0;
  *(_DWORD *)(v2 + 296) = 4096;
  *(_DWORD *)(v2 + 292) = malloc(0x1000u);
  LOBYTE(v8) = 1;
  if ( a2 )
  {
    if ( *(_DWORD *)(v2 + 8) )
    {
      free(*(void **)(v2 + 8));
      *(_DWORD *)(v2 + 8) = 0;
    }
    *(_DWORD *)(v2 + 8) = _strdup(a2);
  }
  *(_WORD *)(v2 + 4) = 1;
  v3 = (unsigned int)(unsigned __int16)GetTickCount() >> 2;
  v4 = GetTickCount();
  FN_printf_1("--------------------------%4x%4x", v4, v3);
  FN_printf_1("multipart/form-data; boundary=%s", v2 + 24);
  v5 = *(_DWORD *)(v2 + 12);
  sub_10090E0("Content-Type", &v7);
  *(_DWORD *)(v2 + 16) = 0;
  *(_DWORD *)(v2 + 20) = 0;
  return (struct_name_2 *)v2;
}
```

NBOT

```
struct_name_2 *__thiscall FN_BuildCTFC_HTTP_Form_Multipart(int this, char *a2)
{
  int v2; // edi@1
  __int16 v3; // ax@5
  unsigned __int16 v4; // ax@5
  int v5; // ecx@5
  int v6; // eax@5
  char v8; // [sp+18h] [bp-110h]@5
  int v9; // [sp+124h] [bp-4h]@1

  v2 = this;
  sub_40CAF0(this);
  v9 = 0;
  *(_DWORD *)v2 = CTFC_HTTP_Form_Multipart::`vftable';
  *(_DWORD *)(v2 + 288) = CreateMutexW(0, 0, 0);
  *(_DWORD *)(v2 + 284) = 0;
  *(_DWORD *)(v2 + 280) = 0;
  *(_DWORD *)(v2 + 300) = 0;
  *(_DWORD *)(v2 + 296) = 4096;
  *(_DWORD *)(v2 + 292) = malloc(0x1000u);
  LOBYTE(v9) = 1;
  if ( a2 )
  {
    if ( *(_DWORD *)(v2 + 8) )
    {
      free(*(void **)(v2 + 8));
      *(_DWORD *)(v2 + 8) = 0;
    }
    *(_DWORD *)(v2 + 8) = _strdup(a2);
  }
  *(_WORD *)(v2 + 4) = 1;
  v3 = GetTickCount();
  v4 = GetTickCount();
  sub_415000(v4, v2 + 24);
  sub_415000(v5, &v8);
  v6 = *(_DWORD *)(v2 + 12);
  sub_408880("Content-Type", &v8);
  *(_DWORD *)(v2 + 16) = 0;
  *(_DWORD *)(v2 + 20) = 0;
  return (struct_name_2 *)v2;
}
```

Type REconstruction: CTFC_HTTP_Form_Multipart

Dino

```
struct_name_2 *__thiscall FN_BuildCTFC_HTTP_Form_Multipart(int this, char *a2)
{
  int v2; // edi@1
  unsigned int v3; // ST10_4@5
  unsigned __int16 v4; // ax@5
  int v5; // eax@5
  char v7; // [sp+18h] [bp-110h]@5
  int v8; // [sp+124h] [bp-4h]@1
```

Please enter text

Please edit the type declaration

```
struct struct_name_CTFC_HTTP_Form_Multipart
{
  struct_name_2_VTABLE_0_0106AC70 *vftbl_0_0106AC70;
  __int16 field_4;
  int field_8;
  int field_12;
  int field_16;
  int field_20;
  int field_24;
  _BYTE gap1C[252];
  int field_280;
  int field_284;
  int field_288;
  int field_292;
  int field_296;
  int field_300;
};
```

OK Cancel Help

```
  sub_10090E0("Content-Type", &v7);
  *(_DWORD *)(v2 + 16) = 0;
  *(_DWORD *)(v2 + 20) = 0;
  return (struct_name_2 *)v2;
}
```

NBOT

```
struct_name_2 *__thiscall FN_BuildCTFC_HTTP_Form_Multipart(int this, char *a2)
{
  int v2; // edi@1
  __int16 v3; // ax@5
  unsigned __int16 v4; // ax@5
  int v5; // ecx@5
  int v6; // eax@5
  char v8; // [sp+18h] [bp-110h]@5
```

Please enter text

Please edit the type declaration

```
struct struct_name_CTFC_HTTP_Form_Multipart
{
  struct_name_2_VTABLE_0_0043DA4C *vftbl_0_0043DA4C;
  __int16 field_4;
  int field_8;
  int field_12;
  int field_16;
  int field_20;
  int field_24;
  _BYTE gap1C[252];
  int field_280;
  int field_284;
  int field_288;
  int field_292;
  int field_296;
  int field_300;
};
```

OK Cancel Help

```
  sub_408880("Content-Type", &v8);
  *(_DWORD *)(v2 + 16) = 0;
  *(_DWORD *)(v2 + 20) = 0;
  return (struct_name_2 *)v2;
}
```

Animal Farm: Shared C++ Types

	NBOT	Casper	Bunny	Babar	Dino
wmiException	X		X	X	
basic_AvWmiManager	X		X	X	
basic_WmiManager	X		X	X	
CTFC_HTTP_Form	X	X			X
CTFC_HTTP_Forms	X	X			X
CTFC_HTTP_Form_Multipart	X	X			X
CTFC_HTTP_Request	X	X			X
CTFC_AbstractSocket	X	X			X
CTFC_StandardSocket	X	X			X
RunKeyApi		X			X
RunKeyBat		X			X
RunKeyReg		X			X
RunKeyWmi		X			X
RunKeyDefault		X			X
AutoDelApi		X			X
AutoDelDel		X			X
AutoDelWmi		X			X
AutoDelDefault		X			X

Animal Farm: Shared C++ Types

	NBOT	Casper	Bunny	Babar	Dino
NBOT		6 shared custom types	3 shared custom types	3 shared custom types	6 shared custom types
Casper					15 shared custom types
Bunny				3 shared custom types	
Babar					
Dino					

Conclusions

- We demonstrated that IDA Pro scale really well and all its powerful features can be used in automated malware analysis systems
  - ✓ CALL TO ACTION: IDA Pro plugin developers to start adding batch mode switches and optimize the algorithms

- Want to run your IDA plugin on millions of malwares? Let us know! ☺

Resources

Presentation, code and instructions on how to download samples, IDBs and outputs will be available at:

https://github.com/REhints/blackhat2015

CodeXplorer v2.0 [BH Edition]

- Finally plugin support Linux/Mac/Windows
- Options for analysis in IDA batch mode
- Multiple bug fixes and code review
- Improvements for Types and VTBL's reconstruction
- New Features:
  - ✓ dump Ctrees information for additional analysis
  - ✓ dump all reconstructed types information

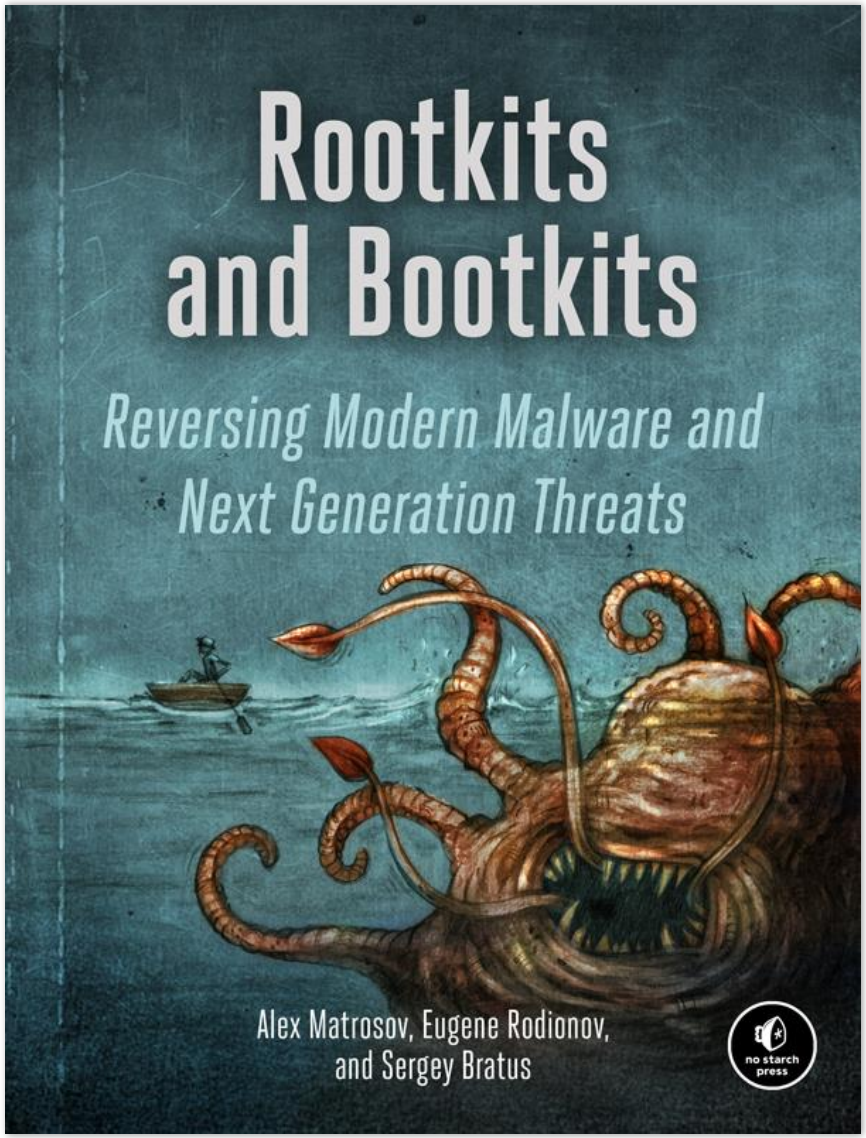
https://github.com/REhints/HexRaysCodeXplorer

Acknowledgements

Personally to **Ilfak Guilfanov (@ilfak)** and **Hex-Rays team** for supporting this research

All the researchers releasing malware-related techniques!!!

The new RE book is coming soon!



Https://www.nostarch.com/rootkits

THE END ! Really !?

Alexander Matrosov (@matrosov)
Eugene Rodionov (@vxradius) [1]
Gabriel Negreira Barbosa (@gabrielnb)
Rodrigo Rubira Branco (@BSDaemon)

{alexander.matrosov || gabriel.negreira.barbosa || rodrigo.branco}
noSPAM intel.com
[1] rodionov *noSPAM* eset.com