Recovering High Level Constructs from C++ Binaries

Rukayat Ayomide Erinfolami(rerinfo1@binghamton.edu) and Aravind Prakash(aprakash@binghamton.edu)

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Why recover constructs from Binaries?

●Useful for enforcing CFI policies on binaries.
●Source code based defenses reveal design information. May not be attractive to vendors. Consumers have to rely on binary level solutions for security.
●Source code may be unavailable to implement source code base CFI defenses.
●Useful for program understanding.

On Design Inference from Binaries Compiled using Modern C++ Defenses

●All modern C++ defenses either explicitly embed design information into the binary,
e.g FCFI & Shrinkwrap, SafeDispatch, TypeSan (TS), CaVer (CV) and Hextype.
●Or implicitly embed,
e.g OVT, VTrust, VIP.
●These defenses reveal a significant aspect of design with vendors do not necessarily want to make public.

Results

Program	FCFI	OVT	TS	CV
Spidermonkey	0.33	0.35	-	0.38
Xalanc	0.02	0.23	0.15	0.43
Soplex	0.05	0.07	0.13	0.18
Povray	0.1	0.33	0.11	0.45
Omnetpp	0	0.21	0.1	0.32
DealII	0.12	-	0.16	0.86
Namd	0	1	0	0
CplusplusThread	0	0.09	0	0.2

Table 1: Graph edit distance (GED) of class hierarchy graph recovered from binaries compiled with different compiler based defenses. GED close to 0 = high similarity, close to 1 = low similarity

DeClassifier: Class-Inheritance Inference Engine for Optimized C++ Binaries

Existing binary level class hierarchy recovery tools such as Marx, VCI, SmartDec e.t.c fail in one or more of the following:

- Poor recovery rate in the presence of optimization
- Ignore direction of inheritance
- Inability to differentiate inheritance from composition

To address these problems, we present DeClassifier.

Techniques

●VTable accumulation and grouping:Primary and secondary VTables of a class are grouped together.
●Destructor-Constructor analysis: We combine constructor (ctor) and destructor (dtor) analysis for optimal recovery. Destructors tend to be retained in the binary, even in the face of optimization.
●Object Layout Analysis (OLA): When neither dtor nor ctor is present for a given class, we do OLA to identify characteristics of an object useful for reasoning about direction of inheritance
●Overwrite Analysis: The vptr of a base class gets overwritten by that of its derived class. We identify these relationships and assign direction of inheritance using results from OLA.

Results

Program	#Classes		#Edges		Ctor only		Ctor + Dtor		Ctor + Dtor + OLA	
	GT	Binary	GT	Used	P(%)	R(%)	P(%)	R(%)	P(%)	R(%)
libebml	27	26	22	22	100.0	54.6	100.0	86.4	100.0	86.4
libflac	18	18	10	10	100.0	30.0	100.0	100.0	100.0	100.0
libzmq	76	64	76	53	100.0	60.4	97.8	75.5	100.0	79.3
libwx_baseu	285	262	264	198	100.0	14.1	100.0	43.9	100.0	47.5
libwx_baseu_net	44	43	19	17	100.0	35.3	92.9	76.5	100.0	82.4
libwx_gtk2u_adv	266	229	118	83	100.0	18.1	88.2	18.1	91.4	38.6
libwx_gtk2u_aui	62	59	11	11	50.0	11.1	50.0	11.1	80.0	44.4
libwx_gtk2_core	683	621	481	293	95.1	13.3	94.7	30.4	93.8	61.4
libwx_gtk2u_html	138	123	74	36	100.0	13.9	88.9	44.4	89.5	47.2
libwx_gtk2u_xrc	122	102	12	-	0.0	0.0	0.0	0.0	0.0	0.0
Average					**84.5**	**25.1**	**81.3**	**48.6**	**85.4**	**58.4**
Doxygen	974	870	866	469	100.0	3.0	68.2	57.6	94.7	80.2
Xalanc	975	875	710	577	100.0	45.4	78.7	65.3	98.3	79.4
DealII	874	687	854	678	98.4	18.6	99.1	80.1	98.4	81.9
Omnetpp	112	105	102	97	100.0	22.7	100.0	58.8	98.7	78.4
Soplex	29	25	22	12	100.0	8.3	66.7	16.7	100.0	50.0
Povray	32	24	21	12	100.0	23.1	100.0	58.3	100.0	58.3
Average					**99.7**	**20.2**	**85.5**	**56.1**	**98.4**	**71.4**

Table 2: Precision and recall of class hierarchy recovered by DeClassifier from binaries compiled with O2 optimization

Devil is Virtual: Reversing Virtual Inheritance in C++ Binaries

Prior solutions have focused on recovering single and multiple inheritance and on harnessing them to enforce CFI policies, while ignoring virtual inheritance (VH). However, our study shows that VH is not uncommon. We found 11% of libraries (including libstdc++) and 2% of executables to contain VH. We also identified security implications of ignoring VH.

Security Implication

The presence of virtual inheritance introduces additional structures, one of which is the construction VTable (CV). Defenses such as Marx do not differentiate between a CV and a regular VTable. However, using a CV instead of a regular VTable will result in object out of bounds access. The equation below gives the number of CV in a binary at depth n of virtual inheritance:

$$\sum(n+1) - 1 = \frac{(n+1)(n+2)}{2} - 1 \implies O(n^2) \qquad (1)$$

Our Solution

We built VirtAnalyzer to recover virtual inheritance from binaries with high precision and accuracy. Figure 1 shows an overview of VirtAnalyzer. Our evaluation shows that VirtAnalyzer can recover up to 100% of virtual bases and 95.5% of intermediate bases.

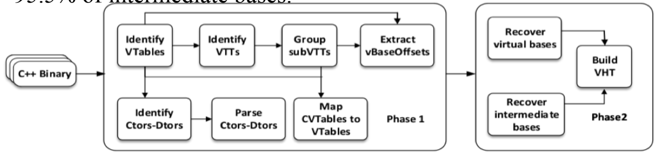

Figure 1: Overview of VirtAnalyzer

References

1.On Design Inference from Binaries Compiled using Modern C++ Defenses. RAID'19
2.DeClassifier: Class-Inheritance Inference Engine for Optimized C++ Binaries. AsiaCCS'19