Analyzing Control Flow Integrity with LLVM-CFI

Paul Muntean, Matthias Neumayer, Zhiqiang Lin,

Gang Tan, Jens Grossklags, and Claudia Eckert

Technical University of Munich, Germany

The Ohio State University, USA

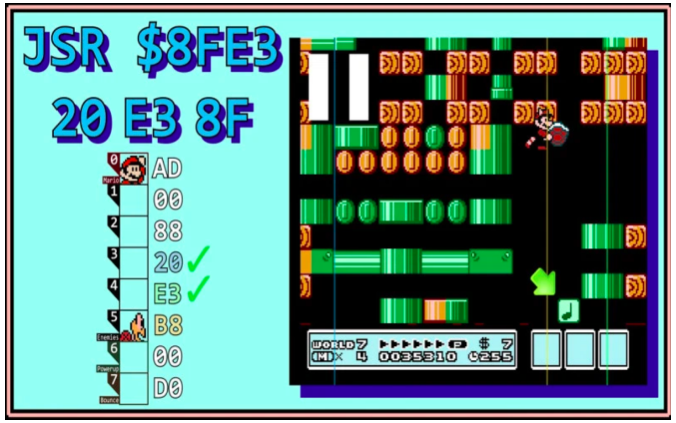
Penn State University, USA

12.12.2019

Annual Computer Security Applications Conference (ACSAC)

Background

Background



The Hacker News
@TheHackersNews

➡️ Arbitrary Code Execution (CVE-2019-7094) affects #Adobe Photoshop CC (Versions 19.1.7 and earlier / 20.0.2 and earlier) for Windows and #macOS,

➡️Arbitrary Code Execution (CVE-2019-7095) affects Adobe Digital Editions (Versions 4.5.10.185749 and below) for #Windows.

Costin Raiu
@craiu

A few days ago our technologies caught a new Chrome 0day exploit used in the wild and we reported it to Google. Just released-Chrome 78 patches it, credits to my colleagues @antonivanovm and Alexey Kulaev for finding the bug. chromereleases.googleblog.com/2019/10/stable…

Stable Channel Update for Desktop
The stable channel has been updated to 78.0.3904.87 for Windows, Mac, and Linux, which will roll out over the coming days/weeks.
chromereleases.googleblog.com

337 2:33 PM - Nov 1, 2019

164 people are talking about this

Most commercial software is vulnerable to arbitrary code exec. attacks

Background: „Arbitrary Code Execution“ Stats

Common Vulnerabilities and Exposures

CVE List | CNAs | WGs | Board | About | News & Blog

NVD
Go to for:
CVSS Scores
CPE Info
Advanced Search

Search CVE List | Download CVE | Data Feeds | Request CVE IDs | Update a CVE Entry

TOTAL CVE Entries: **126873**

HOME > CVE > SEARCH RESULTS

Search Results

There are **20948** CVE entries that match your search.

Name	Description
CVE-2019-9957	Stored XSS within Quadbase EspressReport ES (ERES) v7.0 update 7 allows remote attackers to execute malicious JavaScript and inject arbitrary source code into the target pages. The XSS payload is stored by creating a new user account, and setting the username to an XSS payload. The stored payload can then be triggered by accessing the "Set Security Levels" or "View User/Group Relationships" page. If the attacker does not currently have permission to create a new user, another vulnerability such as CSRF must be exploited first.
CVE-2019-9949	Western Digital My Cloud Cloud, Mirror Gen2, EX2 Ultra, EX2100, EX4100, DL2100, DL4100, PR2100 and PR4100 before firmware 2.31.183 are affected by a code execution (as root, starting from a low-privilege user session) vulnerability. The cgi-bin/webfile_mgr.cgi file allows arbitrary file write by abusing symlinks. Specifically, this occurs by uploading a tar archive that contains a symbolic link, then uploading another archive that writes a file to the link using the "cgi_untar" command. Other commands might also be susceptible. Code can be executed because the "name" parameter passed to the cgi_unzip command is not sanitized.
CVE-2019-9875	Deserialization of Untrusted Data in the anti CSRF module in Sitecore through 9.1 allows an authenticated attacker to execute arbitrary code by sending a serialized .NET object in an HTTP POST parameter.
CVE-2019-9874	Deserialization of Untrusted Data in the Sitecore.Security.AntiCSRF (aka anti CSRF) module in Sitecore CMS 7.0 to 7.2 and Sitecore XP 7.5 to 8.2 allows an unauthenticated attacker to execute arbitrary code by sending a serialized .NET object in the HTTP POST parameter __CSRFTOKEN.
CVE-2019-9865	When RPC is enabled in Wind River VxWorks 6.9 prior to 6.9.1, a specially crafted RPC request can trigger an integer overflow leading to an out-of-bounds memory copy. It may allow remote attackers to cause a denial of service (crash) or possibly execute arbitrary code.
CVE-2019-9845	madskristensen Miniblog.Core through 2019-01-16 allows remote attackers to execute arbitrary ASPX code via an IMG element with a data: URL, because SaveFilesToDisk in Controllers/BlogController.cs writes a decoded base64 string to a file without validating the extension.
CVE-2019-9842	madskristensen MiniBlog through 2018-05-18 allows remote attackers to execute arbitrary ASPX code via an IMG element with a data: URL, because SaveFilesToDisk in app_code/handlers/PostHandler.cs writes a decoded base64 string to a file without validating the extension.
CVE-2019-9829	Maccms 10 allows remote attackers to execute arbitrary PHP code by entering this code in a template/default_pc/html/art Edit action. This occurs because template rendering uses an include operation on a cache file, which bypasses the prohibition of .php files as templates.

Background: Protection Approaches

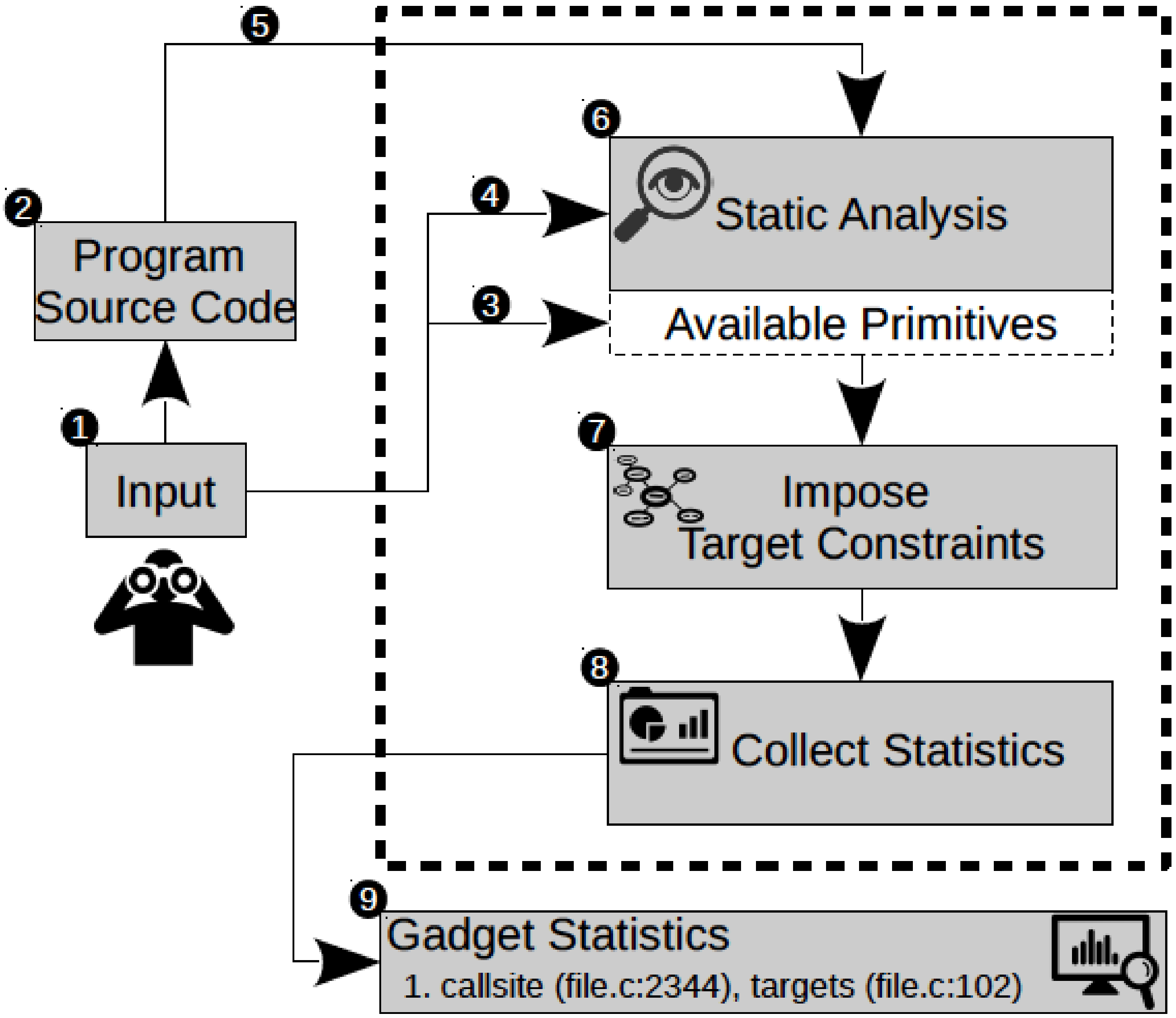
- Static Control Flow Integrity
- Dynamic Control Flow Integrity
- DEP
- ASLR
- Re-randomization
- Information Hiding
- XOM
- CPI/CPS
- Windows RFG
- Intel CET
- HW-based Approaches
- Etc.

→ **This work:**

Analyzing and ranking of static CFI policies

Methodology

Methodology and Approach Overview



Workflow of

our approach

Symbols Used for Analysis

Symbol	Description
P	the analyzed program
Cs	set of all indirect callsites of P
Cs_{virt}	set of P virtual callsites
V	all virtual func. contained in a virtual table hierarchy
V_{sub}	a virtual table sub-hierarchy
v_t	a virtual table
v_e	a virtual table entry (virtual function)
vc_s	a virtual callsite
nv_f	a non-virtual function
v_f	a virtual function (virtual table entry)
C	a class hierarchy contained in P
C_{sub}	a class sub-hierarchy contained in P
c_s	an indirect callsite
nt_{pcs}	callsite's number and type of parameters
nt_{pct}	calltarget's number and type of parameters
F	set of all virtual and non-virtual functions in P
F_{virt}	set of all virtual functions in P
S	set of function signatures
M	calltarget matching set based on the policy rules

Symbol

Descriptions

Methodology: Example of Policy Modeling

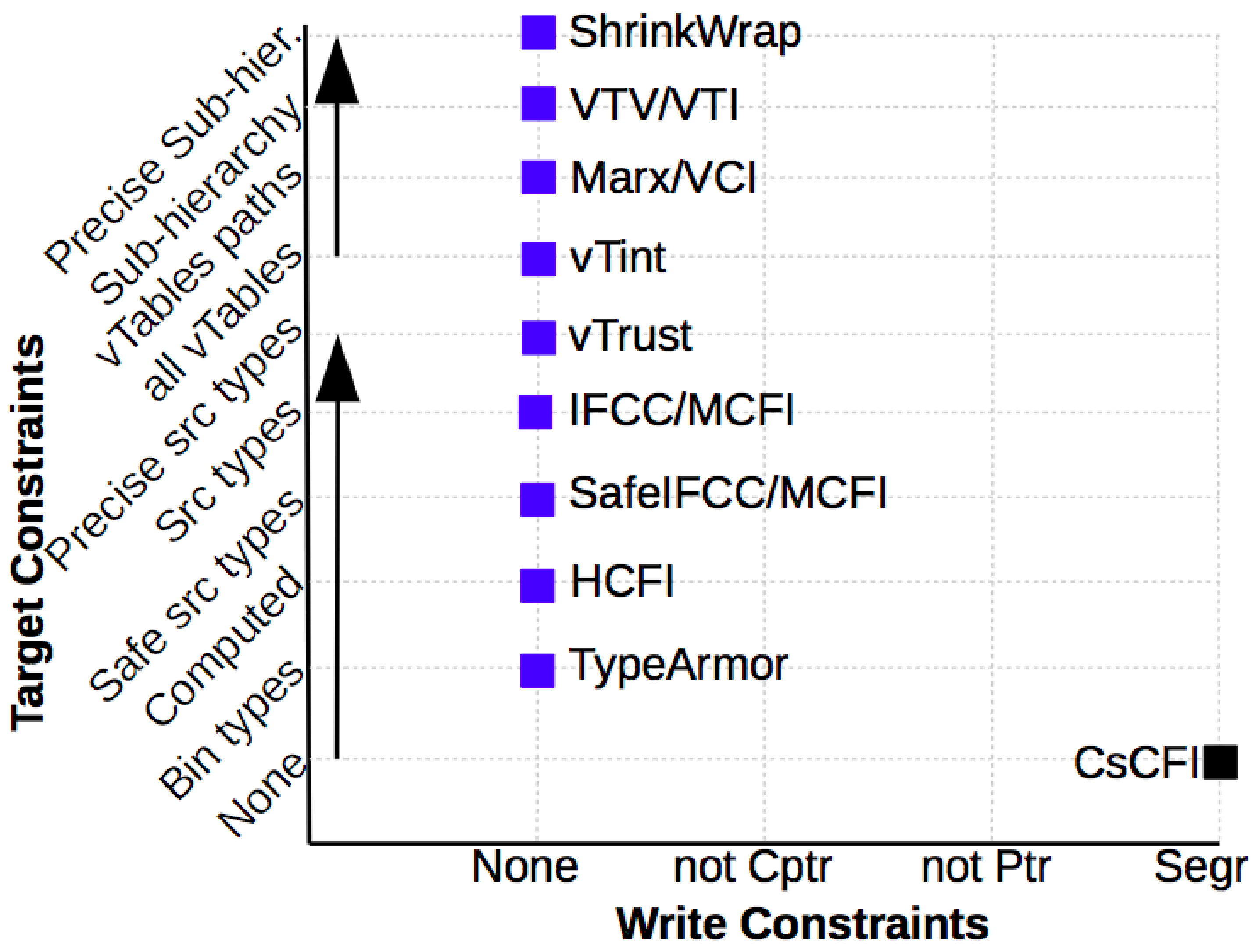
One of the eight CFI policies modeled with LLVM-CFI

Src types. (IFCC/MCFI) [34] We formalize this CFI policy Ψ as the tuple $\langle Cs, F, F_{virt} S, M \rangle$ where the relations hold: (1) $V \subseteq F$, (2) $v_f \in F_{virt}$, (3) $nv_f \in F$, (4) $nt_{pcs} \in S$, (5) $nt_{pct} \in S$, (6) $f_{rt} \in S$, (7) $c_s \in Cs$, and (8) $M \subseteq Cs \times F \times S$.

LLVM-CFI's Analysis. For each indirect callsite c_s count the number of virtual functions and non-virtual functions located in the program F for which the number and type of parameters required at the calltarget nt_{pct} matches the number and type of arguments provided by the callsite nt_{pcs}. The return type of the matching function is ignored. Compared to *Safe src types*, this policy distinguishes between different pointer types. This means that these are not interchangeable and that the function signatures are more strict. Neither the return value of the matching function nor the name of the function are taken into consideration.

Example of a modeled CFI policy

Methodology: Characteristics of Analyzed Policies



Target constraints
vs. write constraints
vs. analyzed CFI policies

Results

RQ1: What Metrics Can be Used within LLVM-CFI?

$\boxed{\textit{Definition 5.1 (\textbf{CTR}).}}$ Let ics_i be a particular indirect callsite in a program P, ctr_i is the total number of legitimate calltargets for an ics_i after hardening a program with a certain CFI policy.

$$CTR = \sum_{i=1}^{n} ctr_i$$

*Definition 5.2 (**RTR**).* Let irs_i be a particular indirect return site in the program P, then rtr_i is the total number of available return targets for each irs_i after hardening the backward edge of a program with a CFI policy.

$$RTR = \sum_{i=1}^{n} rtr_i$$

*Definition 5.3 (**fCGA**).* Let cgf_i be the total number of legitimate calltargets that are allowed and which contain gadgets according to a gadget finding tool. Then, the forward code reuse gadget availability $fCGA$ metric is: $fCGA = \sum_{i=1}^{n} cgf_i$.

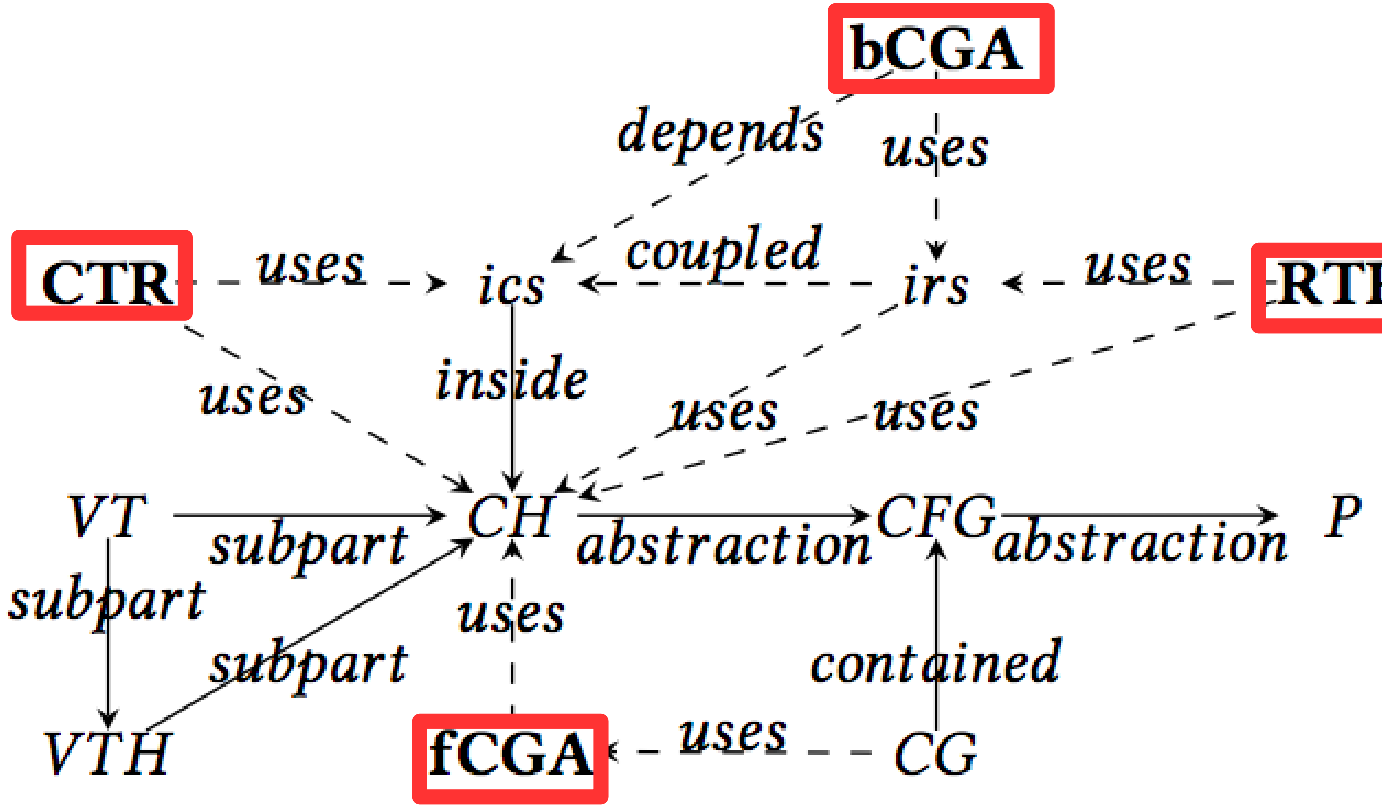
*Definition 5.4 (**bCGA**).* Let cgr_i be the total number of legitimate callee return addresses which contain code gadgets according to a gadget finding tool. Then, the backward code reuse gadget availability $bCGA$ metric is: $bCGA = \sum_{i=1}^{n} cgr_i$.

RQ1: Which Metrics can LLVM-CFI Model?

Symbol	Description
ics	indirect call site (*i.e.*, x86 `call` instruction)
irs	indirect return site (*i.e.*, x86 `ret` instruction)
P	program
VT	virtual table
VTI	virtual table inheritance
CH	class hierarchy
CFG	control flow graph
CG	code reuse gadget
CTR	indirect calltarget reduction
RTR	indirect return target reduction
$fCGA$	forward-edge based CG availability
$bCGA$	backward return-edge based CG availability

Metrics shortnames and main used metadata

RQ1: How are these Metrics Interrelated?



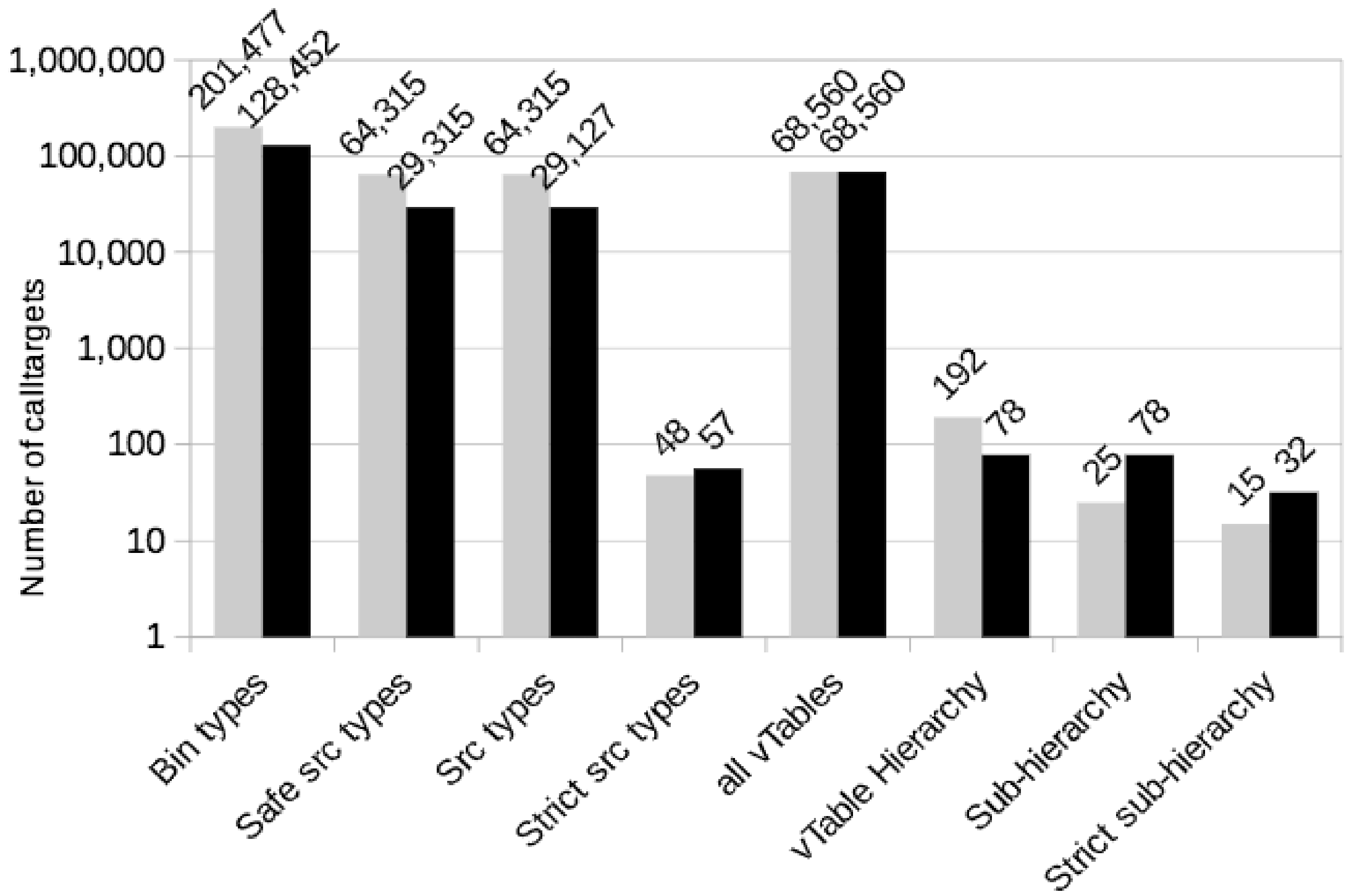
Our 4 CFI metrics and interdependencies with program metadata

RQ2: What is the Residual Attack Surface for the Policies?

P	Targets Median					Targets Distribution					
						NodeJs			MKSnapshot		
	NodeJS	*MKSnaphot*	*Total*	*Min*	*Max*	*Min*	*90p*	*Max*	*Min*	*90p*	*Max*
(1)	21,950	15,817	15,817	15,817	21,950	12,545	30,179	32,478	8,714	21,785	23,376
	(21,950)	(15,817)	(20,253)	(15,817)	(21,950)	(885)	(30,179)	(32,478)	(244)	(21,785)	(23,376)
(2)	2,885	2,273	2,273	2,273	2,885	0	5,751	5,751	1	4,436	4,436
	(88)	(495)	(139)	(88)	(21,950)	(0)	(5,751)	(5,751)	(0)	(4,436)	(4,436)
(3)	1,511	1,232	1,232	1,232	1,511	0	5,751	5,751	1	4,436	4,436
	(56)	(355)	(139)	(56)	(355)	(0)	(5,751)	(5,751)	(0)	(4,436)	(4,436)
(4)	3	2	3	2	3	0	499	730	0	507	756
(5)	6,128	2,903	6,128	2,903	6,128	6,128	6,128	6,128	2,903	2,903	2,903
(6)	2	1	2	1	2	0	54	243	0	16	108
(7)	2	1	1	1	2	0	7	243	0	11	108
(8)	2	1	1	1	2	0	6	243	0	9	108

NodeJS remaining residual attack surface for the 8 CFI policies

RQ3: What are the Scores of the Eight CFI Policies?



Scores obtained by each analyzed CFI defense for Chrome

RQ4: How can LLVM-CFI be Used to Rank Policies?

		Bin types			Safe src types			Src types		
P	B	*Avg*	*SD*	*90p*	*Avg*	*SD*	*90p*	*Avg*	*SD*	*90p*
a	32,478	64.0	20.43	92.92	3.82	5.83	17.71	3.38	5.64	17.71
b	6,201	54.03	18.76	87.89	13.54	9.27	21.21	13.46	9.36	21.21
c	232,593	56.83	19.84	86.62	11.71	12.11	27.65	11.64	12.16	27.65
d	1,949	52.18	26.5	92.0	2.7	3.01	8.21	2.46	3.01	8.21
e	594	65.25	27.81	97.98	2.94	3.18	7.41	2.93	3.19	7.41
f	225	69.75	7.11	68.89	1.0	0.97	0.89	1.0	0.97	0.89
g	1,270	54.91	24.85	92.28	6.38	4.56	11.73	6.36	4.57	11.73
h	2,880	65.19	16.51	84.62	1.25	2.52	1.88	1.2	2.52	1.88
Avg	34,773	60.3	34.39	87.9	5.4	5.18	12.09	5.3	5.17	12.08

Normalized results obtained by using all indirect callsites

RQ5: What are the General Results?

Results for virtual and pointer based callsites

P	Value	Callsite write cons.	Baseline all func.	Targets (Non-) & virt. func. (1)	(2)	(3)
	Min			885	0	0
	90p			30,179	5,751	5,751
a	Max	none	32,478	32,478	5,751	5,751
	Med			21,950	88	56
	Avg			20,787	1,242	1,099
	Min			357	0	0
	90p			5,450	1,315	1,315
b	Max	none	6,201	6,201	1,315	1,315
	Med			2,608	1,315	1,315
	Avg			3,350	840	835
	Min			3,612	0	0
	90p			201,477	64,315	64,315
c	Max	none	232,593	232,593	64,315	64,315
	Med			97,041	8,672	7,394
	Avg			132,182	27,238	27,074
	Min			99	0	0
	90p			1,793	160	160
d	Max	none	1,949	1,915	160	160
	Med			1,070	18	16
	Avg			1,017	53	48
	Min			37	0	0
	90p			582	44	44
e	Max	none	594	582	44	44
	Med			395	6	6
	Avg			388	17	17
	Min			92	0	0
	90p			155	2	2
f	Max	none	225	221	17	17
	Med			155	2	2
	Avg			157	2	2
	Min			422	1	1
	90p			1,172	149	149
g	Max	none	1,270	1,259	149	149
	Med			719	75	75
	Avg			697	81	81
	Min			1,266	1	1
	90p			2,437	54	54
h	Max	none	2,880	2,635	391	391
	Med			1,994	16	14
	Avg			1,877	36	35

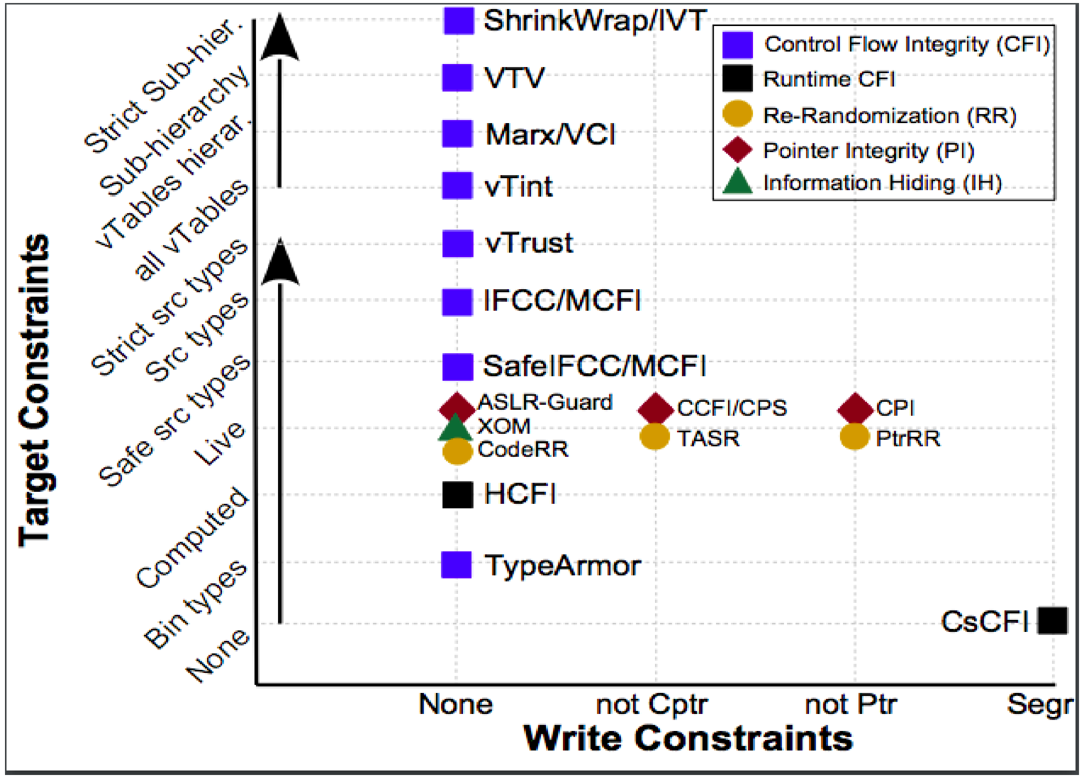
RQ6: How can LLVM-CFI Pave the Way to CRA?

CS	#	Base only vFunc	Base all func	Eight Target Policies (1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
a	5	6,300	32,478	31,305	4	4	1	6,128	1	1	1
b	2	6,300	32,478	21,950	719	719	49	6,128	57	53	49
c	3	6,300	32,478	27,823	136	136	1	6,128	1	1	1
d	1	6,300	32,478	12,545	810	810	1	6,128	72	12	12
e	1	6,300	32,478	1,956	810	810	1	6,128	72	13	13
f	1	6,300	32,478	1,956	810	810	6	6,128	20	19	19
g	3	6,300	32,478	1,956	810	810	6	6,128	20	19	19
h	2	6,300	32,478	3,106	35	35	8	6,128	48	13	5
i	2	6,300	32,478	3,106	2,984	2,984	49	6,128	53	53	49
j	2	6,300	32,478	3,106	719	719	49	6,128	53	53	19

10 controllable callsites and their legitimate targets under the Sub-hierarchy CFI policy

Discussion

Discussion



The analyzed CFI policies and other CFI policies

Summary

Summary

- LLVM-CFI is a control-flow integrity defense analysis framework, and the first tool which allows an analyst to thoroughly compare conceptual/deployed static CFI policies.
-
- LLVM-CFI paves the way towards automated control-flow hijacking attack construction.
-
- An analyst can drastically cut down the time needed to search for gadgets which are compatible with state-of-the-art CFI defenses contained in many real-world programs.
-
- Many CFI defenses can be easily bypassed using LLVM-CFI when analyzing a vulnerable program.

Questions?