A systematic analysis of the science of sandboxing

Michael Maass, Adam Sales, Benjamin Chung, Joshua Sunshine

Sandboxes are increasingly important building materials for secure software systems. In recognition of their potential to improve the security posture of many systems at various points in the development lifecycle, researchers have spent the last several decades developing, improving, and evaluating sandboxing techniques. What has been done in this space? Where are the barriers to advancement? What are the gaps in these efforts? We systematically analyze a decade of sandbox research from five top-tier security and systems conferences using qualitative content analysis, statistical clustering, and graph-based metrics to answer these questions and more. We find that the term "sandbox" currently has no widely accepted or acceptable definition. We use our broad scope to propose the first concise and comprehensive definition for "sandbox" that consistently encompasses research sandboxes. We learn that the sandboxing landscape covers a range of deployment options and policy enforcement techniques collectively capable of defending diverse sets of components while mitigating a wide range of vulnerabilities. Researchers consistently make security, performance, and applicability claims about their sandboxes and tend to narrowly define the claims to ensure they can be evaluated. Those claims are validated using multi-faceted strategies spanning proof, analytical analysis, benchmark suites, case studies, and argumentation. However, we find two cases for improvement: (1) the arguments researchers present are often *ad hoc* and (2) sandbox usability is mostly uncharted territory. We propose ways to structure arguments to ensure they fully support their corresponding claims and suggest lightweight means of evaluating sandbox usability.

A Systematic Analysis of the Science of Sandboxing

Michael Maass[1], Adam Sales[2], Benjamin Chung[1], Joshua Sunshine[1]

[1] Institute for Software Research, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

[2] Statistics Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Corresponding Author:

Michael Maass[1]

5000 Forbes Avenue, Wean Hall 4113, Pittsburgh, Pennsylvania, 15213, USA

Email address: mmaass@andrew.cmu.edu

Abstract

Sandboxes are increasingly important building materials for secure software systems. In recognition of their potential to improve the security posture of many systems at various points in the development lifecycle, researchers have spent the last several decades developing, improving, and evaluating sandboxing techniques. What has been done in this space? Where are the barriers to advancement? What are the gaps in these efforts? We systematically analyze a decade of sandbox research from five top-tier security and systems conferences using qualitative content analysis, statistical clustering, and graph-based metrics to answer these questions and more.

We find that the term "sandbox" currently has no widely accepted or acceptable definition. We use our broad scope to propose the first concise and comprehensive definition for "sandbox" that consistently encompasses research sandboxes. We learn that the sandboxing landscape covers a range of deployment options and policy enforcement techniques collectively capable of defending diverse sets of components while mitigating a wide range of vulnerabilities. Researchers consistently make security, performance, and applicability claims about their sandboxes and tend to narrowly define the claims to ensure they can be evaluated. Those claims are validated using multi-faceted strategies spanning proof, analytical analysis, benchmark suites, case studies, and argumentation. However, we find two cases for improvement: (1) the arguments researchers present are often *ad hoc* and (2) sandbox usability is mostly uncharted territory. We propose ways to structure arguments to ensure they fully support their corresponding claims and suggest lightweight means of evaluating sandbox usability.

A Systematic Analysis of the Science of Sandboxing

Michael Maass[1], Adam Sales[2], Benjamin Chung[1], and Joshua Sunshine[1]

[1]Institute for Software Research, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

[2]Statistics Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

1 Introduction

Sandboxes can be found where software components must be used but cannot currently be verified or trusted. Sandboxed components are often feared to be either malicious or vulnerable to attack. For example, popular browser engines (e.g. Google Chrome and Internet Explorer), productivity software (e.g. Microsoft Word and Adobe Reader), and operating system kernels (e.g. Windows 8) have all been sandboxed to varying degrees. Virtual machines, which run software on simulated hardware separate from the rest of the host system, are commonly used in malware analysis to contain malicious computations (e.g. Cuckoo) and sandboxes are used in mobile ecosystems (e.g. Android) to limit the havoc malicious applications can wreak on a device.

Sandboxes provide some salvation in a world full of complex components: Instead of vetting intractably enigmatic software systems, sandbox them and vet the relatively simple sandbox. What else do sandboxes in mainstream use have in common? They are largely dependent on relatively easily composed coarse-grained operating system features, they are essentially transparent to the users of sandboxed components, and they make little use of decades worth of research produced within the domain of software security sandboxing.

Researchers have spent the last several decades building sandboxes capable of containing computations ranging from fully featured desktop applications to subsets of nearly every kind of application in existence, from third party libraries in Java programs to ads on web sites. Sandboxes have been built to stop memory corruption exploits, ensure control- and data-flow integrity, enforce information flow constraints, introduce diversity where monocultures previously existed, and much more. What more can the community do to bring value?

In this paper, we use multidisciplinary techniques from software engineering, statistics, the social sciences, and graph analysis to systematically analyze the sandboxing landscape as it is reflected by five top-tier security and systems conferences. We aim to answer questions about what sandboxes can already do, how they do it, what it takes to use them, what claims sandbox inventors make about their creations, and how those claims are validated. We identify and resolve ambiguity in definitions for "sandbox", systematize ten years of sandbox research, and point out gaps in our current practices and propose ways forward in resolving them.

We contribute the following:

- A multi-disciplinary methodology for systematically analyzing the state of practice in a research domain (Section 3).

- The first concise definition for "sandbox" that consistently describes research sandboxes (Section 2).

- Systemization of the research sandboxing landscape (Section 4).

- Identification of and proposed solutions to (1) an over-reliance on *ad hoc* arguments for security validation and (2) the neglect of sandbox and policy usability (Section 5).

2   What is a sandbox?

In order to systematically analyze the "sandboxing" landscape we need to clarify the meaning of the term. We reviewed definitions used by practitioners and in papers within the field, both in the substance of the definitions and in their quality as definitions. This section reviews those definitions and establishes a definition for our use here, which we advance as an improved definition for the field.

A definition should be a concise statement of the exact meaning of a word and may be accompanied by narration of some properties implied by the definition. In this case, it should clearly distinguish between mechanisms that are and are not sandboxes. To gain widespread use, a new definition must include all mechanisms that are already widely considered to be sandboxes.

In software security contexts, the term "sandboxing" has grown ambiguous. In an early published use, it described an approach for achieving fault isolation (Wahbe et al., 1993). Discussions where practicing programmers are trying to understand what sandboxing is often fail to achieve a precise resolution and instead describe the term by listing products that are typically considered to be sandboxes or cases where sandboxes are often used.[1] However, we did find cases where attempts were made at a concise and general definition. A representative and accepted StackOverflow answer[2] started with, "In the context of IT security, 'sandboxing' means isolating some piece of software in such a way that whatever it does, it will not spread havoc elsewhere"—a definition that is not sufficiently precise to separate sandboxes from other defensive measures.

Even recently published surveys of sandbox literature have either acknowledged the ambiguity, then used overly-broad definitions that include mechanisms not traditionally considered to be sandboxes (Schreuders et al., 2013), or have relied entirely on the use of examples instead of a precise definition (Al Ameiri and Salah, 2011). Schreuders writes, "Although the terminology in use varies, in general a sandbox is separate from the access controls applied to all running programs. Typically sandboxes only apply to programs explicitly launched into or from within a sandbox. In most cases no security context changes take place when a new process is started, and all programs in a particular sandbox run with the same set of rights. Sandboxes can either be permanent where resource changes persist after the programs finish running, or ephemeral where changes are discarded after the sandbox is no longer in use. ..." This definition suffers from three problems. First, it is still overly reliant on examples and thus is unlikely to capture all security mechanisms that are uncontroversially called sandboxes. Along the same lines, characterizations prefaced with, "In most cases...", are not precise enough to reliably separate sandboxes from non-sandboxes. Finally, the comparison to access controls is not conclusive because it does not clarify which, if any, access control mechanisms applied to a subset of running programs are not sandboxes.

In this section we aim to resolve this ambiguity to lay the groundwork for our analysis's inclusion criteria. While this definition serves our purposes, we believe it can strengthen future attempts to communicate scientifically about sandboxes by adding additional precision. We derive a clear, concise definition for what a "sandbox" is using papers that appear in five top-tier security and operating system conferences, selected because their topics of interest are broad enough to include sandboxing papers most years. While we do not attempt to thoroughly validate our definition using commercial and open source sandboxes, it does encompass the tools with which we are most familiar.

We found 101 potential sandboxing papers. Out of these papers, 49 use the term "sandbox" at least once, and 14 provide either an explicit or implicit definition of the term that is clear enough to characterize. The remaining papers that use the term make no attempt at a definition or provide an ambiguous explanation, intertwined with other ideas, and spread over multiple sentences. Within the set of definitions we identify two themes: *sandboxing as encapsulation* and *sandboxing as policy enforcement*.

Sandboxing as encapsulation has a natural analogy: sandboxes on playgrounds provide a place for children to play with indisputably-defined bounds, making the children easier to watch, and where they are less likely to get hurt or hurt someone else. They also contain the sand, thus preventing it from getting strewn across neighboring surfaces. A similar analogy is used in an answer on the Security StackExchange to the question, "What is a sandbox?" Indeed, Wahbe was working to solve the problem of encapsulating software modules

---

[1] http://stackoverflow.com/questions/2126174/what-is-sandboxing
http://security.stackexchange.com/questions/16291/are-sandboxes-overrated
http://en.wikipedia.org/w/index.php?title=Sandbox_(computer_security)&oldid=596038515
[2] http://security.stackexchange.com/questions/5334/what-is-sandboxing

Table 1: Definitions that speak about "sandboxing" in terms of isolation.

Reference	Quote
(Zhang et al., 2013)	"SFI (Software(-based) Fault Isolation) uses instruction rewriting but provides isolation (sandboxing) rather than hardening, typically allowing jumps anywhere within a sandboxed code region."
(Zeng et al., 2013)	"It is a code-sandboxing technique that isolates untrusted modules from trusted environments. ... In SFI, checks are inserted before memory-access and control-flow instructions to ensure memory access and control flow stay in a sandbox. A carefully designed interface is the only pathway through which sandboxed modules interact with the rest of the system."
(Geneiatakis et al., 2012)	"Others works have also focused on shrinking the attack surface of applications by reducing the parts that are exposed to attack, and isolating the most vulnerable parts, using techniques like sandboxing and privilege separation."
(De Groef et al., 2012)	"Isolation or sandboxing based approaches develop techniques where scripts can be included in web pages without giving them (full) access to the surrounding page and the browser API."
(Cappos et al., 2010)	"Such sandboxes have gained widespread adoption with web browsers, within which they are used for untrusted code execution, to safely host plug-ins, and to control application behavior on closed platforms such as mobile phones. Despite the fact that program containment is their primary goal, flaws in these sandboxes represent a major risk to computer security."
(Reis et al., 2006)	"Wagner et al. use system call interposition in Janus to confine untrusted applications to a secure sandbox environment."
(Cox et al., 2006)	"Our work uses VMs to provide strong sandboxes for Web browser instances, but our contribution is much broader than the containment this provides."

107 (to keep a fault in a distrusted module from effecting other modules) when he popularized the term in this
108 domain.[3]

109   Table 1 lists the definitions we found that we characterize as falling within the theme of sandboxing as
110 isolation. Many of these definitions use the term "isolation," but we prefer the use of *encapsulation*. In
111 Object Oriented Programming, an object *encapsulates* related components and *selectively* restricts access to
112 some of those components. Isolation, on the other hand, sometimes refers to a stronger property in which
113 modules use entirely different resources and therefore cannot interfere with each other *at all*. Sandboxed
114 components often need to cooperate to be useful. Cooperation and the idea of disjoint resources are present
115 in Wahbe's original use of the term "sandbox": Wahbe was trying to reduce the communication overhead
116 present in hardware fault isolation by instead creating software domains that run in the same hardware
117 resources, but that do not interfere when faulty. One potential counterpoint to our use of "encapsulation"
118 is that the term typically is used to refer to cases where the inside (e.g. of an object) is protected from the
119 outside, but sandboxes often protect the external system from the contents of the sandbox. While this is
120 a fair point, this paper does discuss sandboxes that protect their contents from the outside and sandboxes
121 exist that simultaneously defend the inside from the outside and *vice versa* (Li et al., 2014). Furthermore,
122 one can consider that a sandbox encapsulates an external system that must be protected from a potentially
123 malicious component. Given these points, we maintain that encapsulation's recognition of cooperation is
124 important enough to use the term over isolation. Nevertheless, we retain the use of *isolation* when discussing
125 existing definitions.

---

[3]While it is clear from at least one publication that the term *sandbox* was used in computer security earlier than Wahbe's paper (Neumann, 1990), many early software protection papers cite Wahbe as the origin of the "sandbox" method (Zhong et al., 1997; Wallach et al., 1997; Schneider, 1997). At least one early commentator felt that this use of the term "sandbox" was merely renaming "trusted computing bases" (TCB) (McLean, 1997). We believe this section makes it clear that sandboxes meet common TCB definitions, but that not all TCBs are sandboxes.

Table 2: Definitions that speak about "sandboxing" in terms of policy enforcement.

Reference	Quote
(Xu et al., 2012)	"We automatically repackage arbitrary applications to attach user-level sandboxing and policy enforcement code, which closely watches the applications behavior for security and privacy violations such as attempts to retrieve a users sensitive information, send SMS covertly to premium numbers, or access malicious IP addresses."
(Chandra et al., 2011)	"The re-executed browser runs in a sandbox, and only has access to the clients HTTP cookie, ensuring that it gets no additional privileges despite running on the server."
(Politz et al., 2011)	"ADsafe, like all Web sandboxes, consists of two inter-dependent components: (1) a static verifier, called JSLint, which filters out widgets not in a safe subset of JavaScript, and (2) a runtime library, adsafe.js, which implements DOM wrappers and other runtime checks."
(Tang et al., 2010)	"Fundamentally, rule-based OS sandboxing is about restricting unused or overly permissive interfaces exposed by todays operating systems."
(Sun et al., 2008)	"Sandboxing is a commonly deployed proactive defense against untrusted (and hence potentially malicious) software. It restricts the set of resources (such as files) that can be written by an untrusted process, and also limits communication with other processes on the system."
(McCamant and Morrisett, 2006)	"Executing untrusted code while preserving security requires that the code be prevented from modifying memory or executing instructions except as explicitly allowed. Software-based fault isolation (SFI) or sandboxing enforces such a policy by rewriting the untrusted code at the instruction level."
(Provos, 2003)	"For an application executing in the sandbox, the system call gateway requests a policy decision from Systrace for every system call."

126   Table 2 presents seven quotes that discuss sandboxing in terms of restrictions or policy enforcement.
127 These definitions reflect different dimensions of the same idea: A *security policy* can state what is allowed,
128 verboten, or both. The "sandbox" is the subject that enforces the policy or "sandboxing" is the act of
129 enforcing a policy. In short, these quotes cast *sandboxing as policy enforcement*.
130   Careful inspection of our definition tables shows that the same technique, Sofware-based Fault Isolation
131 (SFI), appears in both tables. Zhang explicitly states that hardening is not used in SFI, but McCamant
132 very clearly refers to operations being "allowed" and the existence of a policy. While it could seem that the
133 *sandboxing as isolation* and *sandboxing as policy enforcement* camps are disjoint, we claim they are talking
134 about different dimensions of the same idea. Isolation refers to the *what*: An isolated environment where a
135 module cannot do harm or be harmed. Policy enforcement refers to the *how*: By clearly defining what is
136 or is not allowed. To use another childhood analogy, we often sandbox children when we place them in the
137 corner as a punishment. We isolate them by moving them away from everyone else and placing them in a
138 specific, bounded location, then we impose a security policy on them by making statements such as, "Do
139 not speak, look straight ahead, and think about what you did." We resolve ambiguity in the use of the term
140 "sandbox" by combining these themes:

141 **Sandbox** An encapsulation mechanism that is used to impose a security policy on software components.

142   This definition concisely and consistently describes the research sandboxes we identify in the remainder
143 of this paper.

144   3   Methodology

145 In this section, we discuss the steps we took in order to select and analyze sandboxing papers and the
146 sandboxes they describe. Our methodology is primarily based on the book "Qualitative Content Analysis in
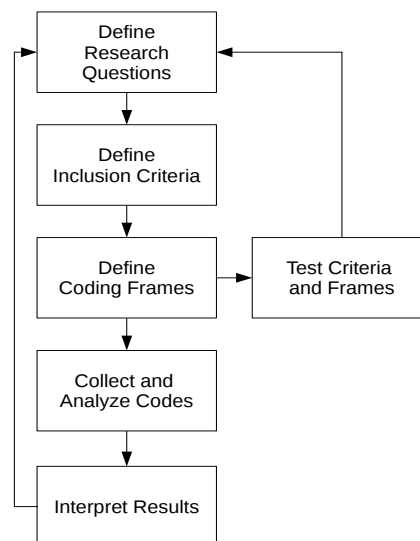
Figure 1: The iterative process used to define research questions, build a dataset, and interpret the set to answer the questions. This process is inspired by QCA (Schreier, 2012)

Practice" (QCA) (Schreier, 2012). Barnes (2013) provides a succinct summary of the methodology in Section 5.3 of his dissertation. This methodology originates in the social sciences (Berelson, 1952; Krippendorff, 2013; Denzin and Lincoln, 2011) and is intended to repeatably interpret qualitative data to answer a set of research questions. Figure 1 summarizes the iterative process we used to define our questions, pick and interpret papers (Sections 3.1 and 3.2), and develop our results (Section 3.3).

QCA goes well beyond a systematic literature review (Budgen and Brereton, 2006; Kitchenham et al., 2009). While both QCA and systematic reviews require the definition of research questions and repeatable processes for collecting source material, reviews stop short of detailed analysis. QCA carries on where reviews end. When performing QCA, researchers define coding frames to clearly and repeatably establish how the source material will be interpreted to answer the research questions. The frames contain codes that summarize blocks of data and definitions for each code. Furthermore, QCA methodologies dictate how the coding frames are to be applied, by segmenting the entirety of the data such that each segment can labeled with at most one code. This ensures that the data is coded without missing relevant data and while reducing the researcher's bias towards some bits of data. Finally, QCA requires researchers to test their full process before carrying out the analysis.[4] Together, these steps allow researchers to reliably and effectively interpret text to answer research questions that are not possible to answer using a purely quantitative analysis. For example, Schreier points out that a quantitative analysis can determine how many women appear in magazine advertisements relative to men, but a qualitative analysis (e.g. QCA) is required to determine whether or not women are more likely to be placed within trivial contexts than men in those ads (Schreier, 2012, p. 2).

The sandboxes we describe in this paper were selected from the proceedings of five conferences: IEEE Symposium on Security and Privacy (Oakland), Usenix Security, ACM Conference on Computer and Communications Security (CCS), ACM Symposium on Operating System Principles (SOSP), and Usenix Symposium on Operating System Design and Implementation (OSDI). We restricted our selection to particular conferences to improve reproducibility—because of this choice, the set of papers evaluated against our inclusion criteria is very well defined. To select these conferences, we collected all of the sandboxing papers we were aware of and the selected five venues contained far more sandboxing papers than any other venue.[5]

---

[4]We followed the QCA methodology specified by Schreier with one major deviation. We did not segment the text because the vast majority of the content in the papers is irrelevant to our needs. Most uses of QCA attempt to capture content of a text in its entirety. This was not our goal so we analyzed text more selectively.

[5]Based on earlier criticism of this paper, we reevaluated our data set by looking at the past four years of proceedings at unselected venues such as the USENIX Annual Technical Conference (ATC), Programming Language Design and Implemen-

173   The selected conferences are widely regarded as the top-tier conferences in software security and operating
174   systems.[6] Therefore, our data reflects the consensus of large communities.
175   Table 3 presents our twelve research questions, the areas each question attempts to illuminate, and a
176   comprehensive list of their answers as manifested by our paper corpus. We derived an initial set of questions
177   by considering which broad aspects of sandboxes are poorly understood and where better understanding may
178   change how the community performs research in this space. As a result, the questions are necessarily biased
179   by our own backgrounds and personal experiences. In particular, this lead to an emphasis on questions about
180   how mechanisms and policies are derived, applied, and evaluated. We added questions while we performed
181   the analysis when we found that we had the data to answer new and interesting questions. Overall, these
182   questions aim to capture a comprehensive snapshot of the current state of sandboxing research, with an
183   emphasis on where sandboxes fit into the process of securing software systems, what policies are enforced
184   and how they are defined and constructed, and what claims are made about sandboxes and how those claims
185   are validated.

3.1   Picking Papers

187   We selected papers from 10 years worth of proceedings at the five conferences mentioned above. We decided
188   whether a paper was included in our sample based on rigorous inclusion criteria so the process of includ-
189   ing/excluding papers is repeatable. The most important criterion is that the paper describes a sandbox that
190   meets the definition given in Section 2. The remaining criteria were added as we carried out the study to
191   exclude papers that are incapable of answering the research questions and to clarify relevant nuances in the
192   definition.
193   Papers were included if they met the following criteria:

194   • The paper documents the design of a novel tool or technique that falls under the *sandbox* definition

195   • The paper is a full conference paper

196   • The paper is about an instance of a sandbox (e.g. not a component for building new sandbox tools,
197   theoretical constructs for sandboxes, etc.)

198   • Techniques are applied using some form of automation (e.g. not through entirely manual re-architecting)

199   • A policy is imposed on an identifiable category of applications or application subsets

200   – The policy is imposed locally on an application (e.g. not on the principal the application executes
201   as, not on network packets in-transit, etc.)
202   – The category encompasses a reasonable number of real-world applications (e.g. doesn't require
203   the use of (1) a research programming language, (2) extensive annotations, or (3) non-standard
204   hardware)

205   We gathered papers by reading each title in the conference proceedings for a given year. We included a
206   paper in our initial dataset if the title gave any indication that the paper could meet the criteria. We refined
207   the criteria by reviewing papers in the initial dataset from Oakland before inspecting the proceedings from
208   other venues. We read the remaining papers' abstracts, introductions, and conclusions and excluded papers
209   as they were being interpreted if they did not meet the criteria. We maintained notes about why individual
210   papers were excluded from the final set.[7]

---

tation (PLDI), and Object-Oriented Programming, Systems, Languages and Applications (OOPSLA). These venues contained
fewer sandboxing papers than our selected venues, and those that appeared were not significantly different in form or content
from those in selected venues. In fact, with rare exceptions, the sandboxing papers at the unselected venues were written by
the same authors as one or more paper in our data set.

[6]http://www.core.edu.au/index.php/conference-rankings
https://personal.cis.strath.ac.uk/changyu.dong/ranking.html
http://faculty.cs.tamu.edu/guofei/sec_conf_stat.htm
http://webdocs.cs.ualberta.ca/~zaiane/htmldocs/ConfRanking.html
[7]Our full list of papers with exclusion notes is available in sandboxing_papers_2005-2014.xlsx in our supplemental materials
.

Table 3: Our research questions, the areas each question attempts to illuminate, and potential answers. The answers are codes in the content analysis process we apply. Answers are not necessarily mutually exclusive. Definitions for the terms in this table appear in our coding frames (see coding_frames.pdf in our supplemental material) with examples.

Question area	Question	Possible answers
Sandbox Lifecycle	Where in the architecture are policies enforced?	Component, Application, Host
	How and when are policies imposed?	Statically, Dynamically, Hybrid
Security outcomes	What resources do the sandboxes protect?	Memory, Code/Instructions, Files, User Data, Communications
	Which components do the sandboxes protect?	Component, Application, Application Class
	At what point will sandboxes catch exploits?	Pre-exploit, Post-exploit
Effort and applicability	What must be done to apply the sandboxes?	Nothing, Select Pre-made Policy, Write Policy, Run Tool, Install Tool
	What are the requirements on sandboxed components?	None, Source Code, Annotated Source Code, Special Compiler, Compiler-introduced Metadata, Sandbox Framework/Library Components
Policy provenance and manifestation	Who defines policies?	Sandbox Developer (Fixed), Sandbox User (User-defined), Application Developer (Application-defined)
	How are policies managed?	Central Policy Repository, No Management
	How are policies constructed?	Encoded in Sandbox Logic, Encoded in Application Logic, User Written
Research claims and validation	What claims are made about sandboxes?	Performance, Security, Applicability
	How are claims validated?	Proof, Analytical Analysis, Benchmark Suite, Case Studies, Argumentation, Using Public Data
	How are sandboxes released for review?	Source Code, Binaries, Not Available

3.2 Categorizing the Dataset

To interpret papers we developed coding frames[8] where a *category* is a research question and a *code* is a possible answer to the question. To ensure consistency in coding, our frames include detailed definitions and examples for each category and code. Our codes are not mutually exclusive: A question may have multiple answers. We developed the majority of our frames before performing a detailed analysis of the data, but with consideration for what we learned about sandboxing papers while testing the inclusion criteria above on our data from Oakland. We learned that evaluative questions were quite interesting while coding papers, thus frames concerning what claims were made about a sandbox and how those claims were validated became more fine-grained as the process progressed. Whenever we modified a frame, we updated the interpretations of all previously coded papers.

We tested the frames by having two coders interpret different subsets of the Oakland segment of the initial dataset. To interpret a paper, each category was assigned the appropriate code(s) and a quote justifying each code selection was highlighted and tagged in the paper's PDF.[9] While testing, the coders swapped quotes sans codes and independently re-assigned codes to ensure consistency, but we did not measure inter-rater reliability. Code definitions were revised where they were ambiguous. While there is still some risk that different coders would select different quotes or assign codes to the same quote, we believe our methodology sufficiently mitigated the risk without substantially burdening the process given the large scope of this effort.

After coding every paper, we organized the codes for each paper by category in a unified machine-readable file[10] (hereafter referred to as the summary of coded papers) for further processing.

3.3 Analyzing the Dataset

To summarize the differences and similarities between sandboxing papers, we attempted to identify clusters of similar sandboxing techniques. To do so, we first calculated a dissimilarity matrix for the sandboxes. For category k, let p_{ijk} be the number of codes that sandboxes i and j share, divided by the total number of codes in that category they *could* share. For categories in which each sandbox is interpreted with one and only one code, p_{ijk} is either 1 or 0; for other categories, it falls in the interval $[0, 1]$. Then the dissimilarity between i and j is $d_{ij} = \sum_k (1 - p_{ijk})$. We fed the resulting dissimilarity matrix into a hierarchical agglomerative clustering algorithm (Kaufman and Rousseeuw, 2009) (implemented in `R` with the `cluster` package (R Core Team, 2014; Maechler et al., 2014)). This algorithm begins by treating each sandbox as its own cluster, and then iteratively merges the clusters that are nearest to each other, where distance between two clusters is defined as the average dissimilarity between the clusters' members. The agglomerative clustering process is displayed in dendrograms. We stopped the agglomerative process at the point at which there were two clusters remaining, producing two lists of sandboxes, one list for each cluster. To interpret the resulting clusters, we produced bar charts displaying the code membership by cluster. We conducted this analysis three times: once using all of the categories to define dissimilarity, once using using all categories except those for claims, validation, and availability, and once using the validation categories. We do not present the plots from the analysis that ignored claims, validation, and availability because it did not produce results different from those generated using all categories.

We conducted correlational analyses to learn whether sandbox validation techniques have improved or worsened over time, or whether sandbox publications with better (or worse) validation received more citations. The validation codes were ordered in the following way: proof > analytical analysis > benchmarks > case study > argumentation > none. This ordering favors validation techniques that are less subjective. While it is possible for a highly ranked technique to be applied less effectively than a lower ranked technique (e.g. a proof that relies on unrealistic assumptions relative to a thorough case study) this ranking was devised after coding the papers and is motivated by the real world applications of each technique in our dataset. Each claim type (security, performance, and applicability), then, was an ordinal random variable, so rank-based methods were appropriate. When a sandbox paper belonged to two codes in a particular validation category, we used its highest-ordered code to define its rank, and lower-ordered codes to break ties. So, for instance, if paper A and paper B both included proofs, and paper A also included benchmarks, paper A

---

[8]Our full coding frames are available in coding_frames.pdf in our supplemental materials .

[9]A full list of quotes with code assignments is available in paper_quotes.pdf in our supplemental materials .

[10]The summarized version of our dataset is available as summary_of_coded_papers.xlsx in our supplemental materials . This spreadsheet was converted to a CSV to perform statistical and graph-based analyses.

would be ranked higher than paper B. To test if a claim type was improving over time, we estimated the Spearman correlation (Spearman, 1904) between its codes and the year of publication, and hence tested for a monotonic trend. Testing if papers with better validation, in a particular category, received more citations necessitated accounting for year of publication, since earlier papers typically have higher citation counts. To do so, we regressed paper citation rank against both publication year and category rank. (We used the rank of papers' citation counts as the dependent variable, as opposed to the citation counts themselves, due to the presence of an influential outlier—Terra (Garfinkel et al., 2003). Scatterplots show the relationship between citation ranks and publication year to be approximately linear, so a linear adjustment should suffice.) There was a "validation effect" if the coefficient on the validation measure was significantly different from zero. We conducted four separate regression analyses: one in which citation ranks were regressed on publication year and category ranks of all three validation criteria, and one in which citation ranks were regressed on publication year and security validation only, one in which citation ranks were regressed on publication year and performance validation only, and one in which citation ranks were regressed on publication year and applicability validation only.

We constructed a citation graph using the papers in our set as nodes and citations as edges as a final means of better understanding the sandboxing landscape. We clustered the nodes in this graph using the same clusters found statistically, using the process describe above, and using common topics of interest we observed. The topics of interest are typically based on the techniques the sandboxes apply (e.g. Control Flow Integrity (CFI), artificial diversity, etc.). We evaluate these clusters using the modularity metric, which enables us to compare the quality of the different categorizations. Modularity is the fraction of edges that lie within a partition, above the number that would be expected if edges were distributed randomly.

4 Results

We derived our results from the various statistical clusters of our summary of coded papers, trends explicit in this dataset, and observations made while reading the papers or analyzing our summarized data. As our dataset is public, we encourage readers to explore the data themselves. Note while interpreting the statistical clusters that they are not representative of how papers are related in terms of broad topics of interest. When we applied the statistical clusters to the citation graph of the papers in our set the modularity scores were -0.04 and 0.02 when papers were clustered based on all of the attributes we coded and just validation attributes respectively. These modularity scores mean that the statistical clusters are no better than randomly clustering papers when considering how they cite each other.

These poor modularity scores make sense because authors are much more likely to cite papers that use similar techniques or tackle similar problems than use similar validation strategies. We confirmed the latter observation by computing that the modularity for overlapping groups (Lázár et al., 2009) based on validation is -0.198, which confirms that partitions built from the validation techniques do not direct citation graph structure. Indeed, when we clustered papers in the citation graph based on topics of interest we observed while interpreting the set, the modularity score, 0.33, is significantly better than a random cluster. The citation graph with topic clusters is shown in Figure 2. While these clusters are potentially of sociotechnical interest to the community, we must look at lower-level attributes to understand how sandboxes are to be applied in practice and how they improve the security posture of real systems. The statistical clusters fill that role.

Figures 3 and 4 show the codes that are members of the fixed policy and user-defined policy clusters respectively when all categories are considered. The dendrogram for these clusters appears in Figure 5. Many of our results are interpretations of these charts. Table 4 succinctly describes our results per research question and references later sections where more details are found. The remainder of this section presents those details.

4.1 Sandboxes: Building Materials for Secure Systems

Sandboxes are flexible security layers ready to improve the security posture of nearly any type of application. While the deployment requirements and details vary from sandbox to sandbox, collectively they can be applied at many different points in a system's architecture and may be introduced at any phase in an

Table 4: Summary of our research questions and results.

Research Question	Results	Section
Where in a system's architecture are policies enforced?	There is an emphasis on enforcing policies in the operating system or transforming applications to enforce a policy over using application hosts (e.g. language-hosting virtual machines, browsers, etc.).	4.1
When are policies imposed?	Static, dynamic, and hybrid strategies are roughly equally favored in all domains but with a slight preference for strictly static or dynamic approaches.	4.1
What application resources are protected by sandboxes?	Sandboxes with fixed policies tend to prevent memory corruption or protect properties of application code (e.g. control flow). User-defined policies are correlated with policies that are more diverse and cover the gamut of application-managed resources.	4.1
What types of components are protected by sandboxes?	Sandboxes that use fixed policies tend to require the user to target specific components, while those with user-defined policies tend to allow for broader targeting.	4.1
At what point in the process of an attack will an exploit violate sandbox policies?	Sandboxes are primarily pro-active by disrupting exploits before a payload can be executed. Where users must define a policy, sandboxes tend to be pro-active in attempting to stop exploits, but also limit the range of possible behaviors a payload can exhibit.	4.1
What are the requirements of people applying sandboxes?	Sandboxes that have fewer requirements for people tend to have more requirements for the application. Similarly, having a fixed policy is correlated with more requirements of the application, while user-defined policies are correlated with more requirements of the user.	4.2
What are the requirements of components being sandboxed?	Sandboxes with fixed policies most-often require that applications be compiled using a special compiler.	4.2
Who defines sandbox policies?	Policies are most often defined by the sandbox developer at design time.	4.2
How are policies managed?	Policy management is largely ignored, even where users must write their own policies.	4.2
How are policies constructed?	Most policies are hardcoded in the sandbox.	4.2
What claims are made about sandboxes?	Applicability to new cases is often the impetus for improving existing techniques, but strong security and better performance are more often claimed.	4.3
How are claims validated?	Benchmarks and case studies are the most favored validation techniques for all types of claims. Where security claims are not validated using both benchmarks and case studies, ad-hoc arguments are heavily favored.	4.3
In what forms are sandboxes made available for review?	There is a recent slight increase in the release of sandbox source code, but generally no implementation artifacts are made available for review.	4.3
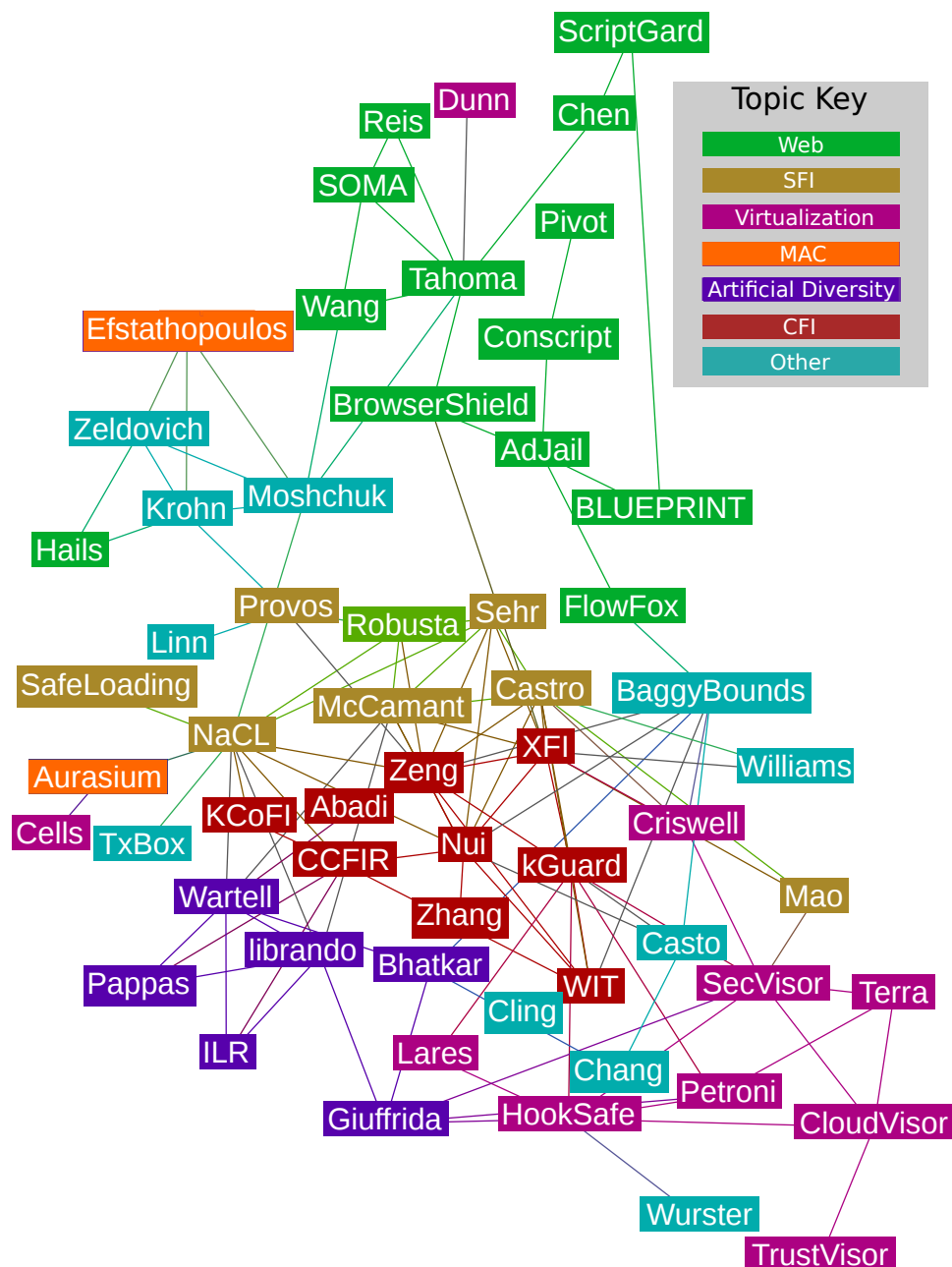
Figure 2: The citation graph for the papers in our set. The colors represent clusters based on topics of interest (modularity = 0.33). Papers cluster based on topics of interest, not necessarily their technical attributes or validation stratgies, thus we must look at lower level attributes to gain a broad understanding of the sandboxing landscape. Papers that were not linked to any of the other papers in the set are not shown. Categories bridging Mandatory Integrity and Access Control (MI/AC) were collapsed to simply Mandatory Access Control (MAC) for this graph. Our citation data can be found in sandboxbib.zip in our supplemental materials .
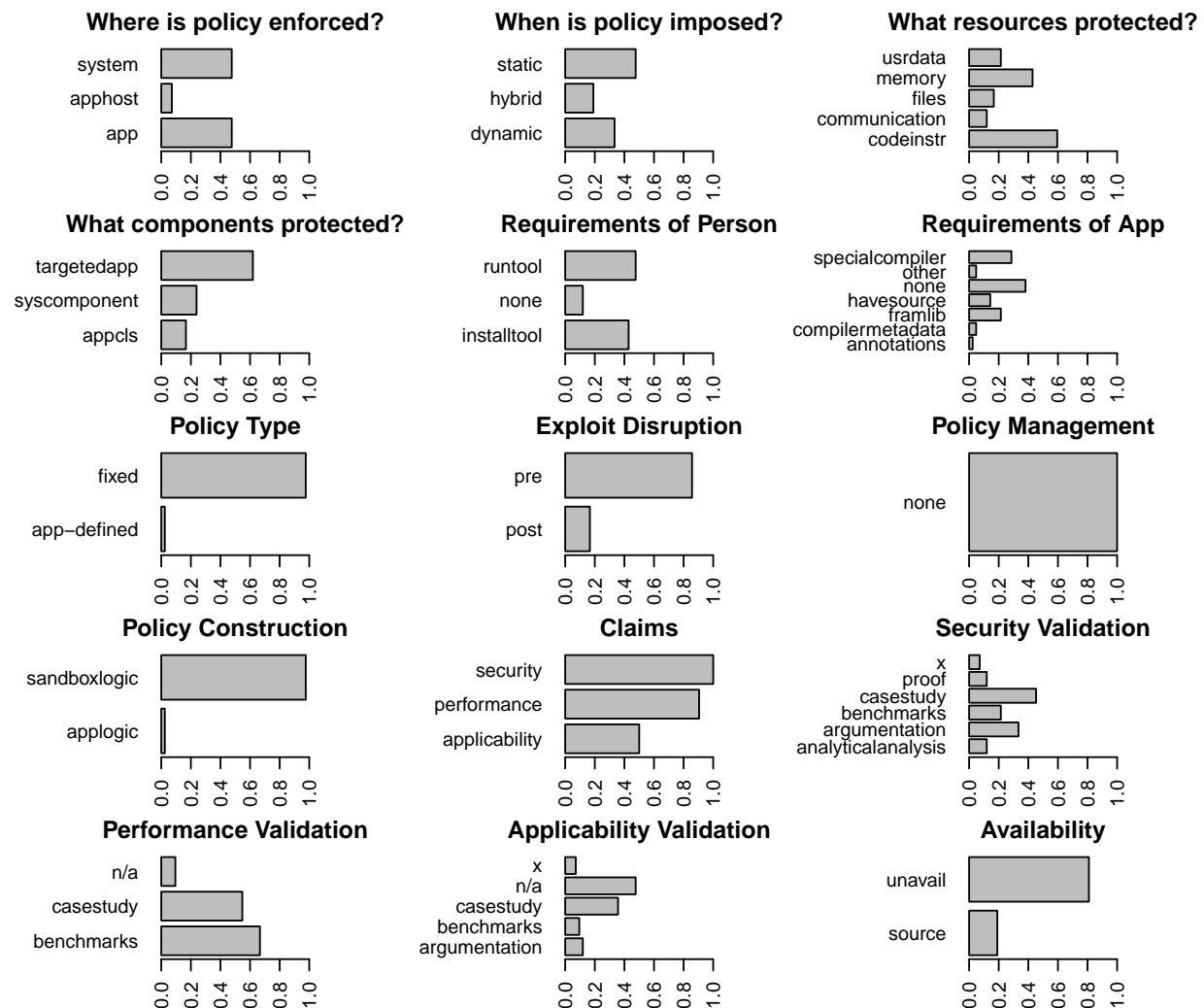
Figure 3: Breakdown of the representation of all codes for papers that emphasize fixed policies. Cases where a claim was made but not validated are labeled with an "x".
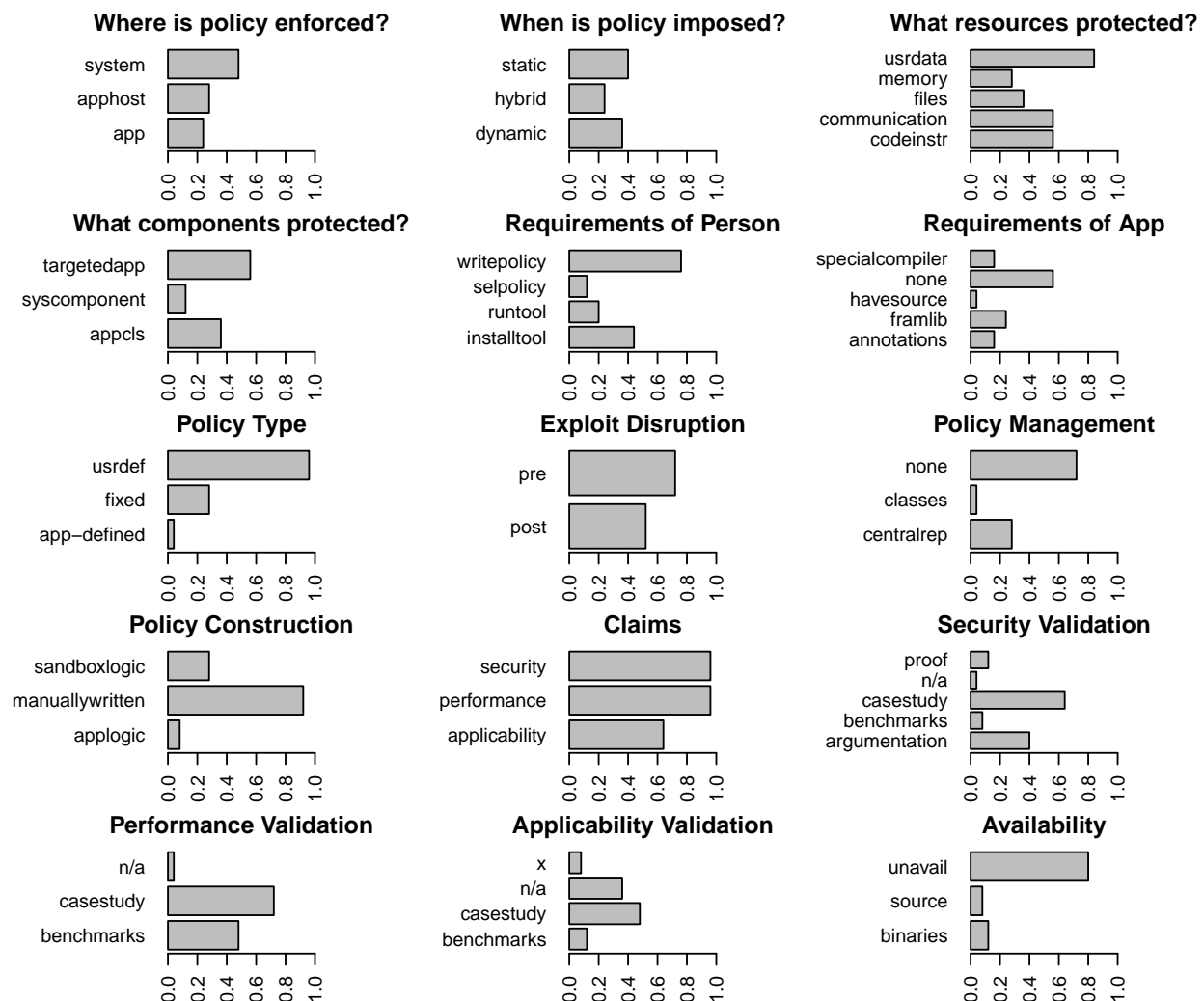
Figure 4: Breakdown of the representation of all codes for papers that emphasize user-defined policies. Some sandboxes support a fixed-policy with an optional user-defined policy (e.g. (Siefers et al., 2010)). Cases where a claim was made but not validated are labeled with an "x".

Dendrogram of agnes(x = distanceMatrix, diss = TRUE)

distanceMatrix
Agglomerative Coefficient = 0.57
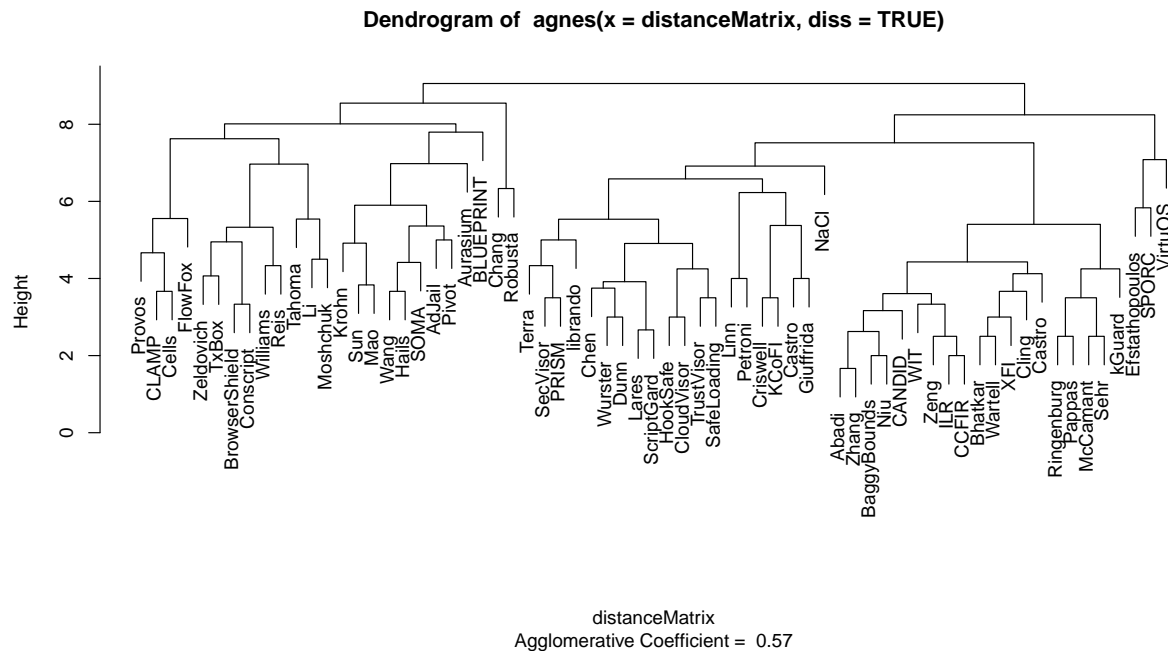
Figure 5: A dendrogram displaying the clusters for sandboxing papers taking into account all categories. At the top most level, where two clusters exist, the clusters respectively represent sandboxes that use fixed policies and those that use user-defined policies.

308 application's development lifecycle, starting with the initial implementation. In fact, sandboxes can even be
309 applied well after an application has been abandoned by its maintainer to secure legacy systems.
310 In our dataset, the policy enforcement mechanism for a sandbox is always deployed as a system com-
311 ponent, as a component of an application host, or by insertion directly into the component that is being
312 encapsulated. While application hosts are becoming more popular as many applications are moved into
313 web browsers and mobile environments, they are currently the least popular place to deploy policy enforce-
314 ment mechanisms for research sandboxes. Our set includes ten sandboxes where policies are enforced in the
315 application host, twenty-six in the component being encapsulated,[11] and thirty two in a system component.
316 We believe that application hosts are less represented because many existing hosts come with a sandbox
317 (e.g. the Java sandbox, Android's application sandbox, NaCl in Google Chrome, etc.). Indeed, all but one of
318 the sandboxes deployed in application hosts are for the web, where applications can gain substantial benefits
319 from further encapsulation and there is currently no *de facto* sandbox. The one exception is Robusta (Siefers
320 et al., 2010), which enhances the Java sandbox to encapsulate additional non-web computations.
321 System components are heavily represented because any sandbox that is to encapsulate a kernel, driver,
322 or other system component must necessarily enforce the policy in a system component. Fifteen of the
323 sandboxes fall into this category because they are encapsulating either a kernel or hypervisor. The remainder
324 could potentially enforce their policies from a less privileged position, but take advantage of the full access
325 to data and transparency to user-mode applications available to system components. This power is useful
326 when enforcing information flow across applications, when preventing memory corruption, or when otherwise
327 enforcing the same policy on every user-mode application.
328 Research sandboxes almost universally embed their enforcement mechanism in the application that is
329 being encapsulated when the application runs in user-mode. Application deployment is correlated with fixed
330 policies where modifying the application itself can lead to higher performance and where it makes sense to
331 ensure the enforcement mechanisms exists anywhere the application is, even if the application moves to a
332 different environment. Fixed-policies with embedded enforcement mechanisms are correlated with another

---

[11](Sehr et al., 2010) is counted twice because the enforcement mechanism is spread across the application and its host.

333 important deployment concern: statically imposed policies.

334 Imposing a policy statically, most often using a special compiler or program re-writer, is advantageous
335 because the policy and its enforcement mechanism can travel with the application and overhead can be
336 lower as enforcement is tailored to the targeted code. There are some cons to this approach. For example,
337 the process of imposing the policy cannot be dependent on information that is only available at run-time
338 and the policy is relatively unadaptable after it is set. Furthermore, because the policies are less adaptable,
339 sandboxes that statically impose security policies typically only encapsulate components that are targeted by
340 the person applying the sandbox. These are cases where dynamic mechanisms shine. Given these trade-offs,
341 it makes sense that papers in our set fall into one of two clusters when all codes are considered: Those that
342 are protecting memory and software code, which are relatively easy to encapsulate with a fixed policy, and
343 those managing behaviors manifested in external application communications or interactions with user-data
344 and files that are more easily encapsulated with an adaptable (typically user-defined) policy.

345 Generally hybrid deployments are used when the approach is necessarily dynamic but static pre-processing
346 lowers overhead. Sometimes, techniques begin as hybrid approaches and evolve to fully dynamic approaches
347 as they gain traction. For example, early papers that introduce diversity in binaries to make reliable exploits
348 harder to write (e.g. code randomization), tend to rely on compiler-introduced metadata, while later papers
349 did not need the extra help. This evolution broadens the applicability of the sandboxing technique. We
350 observed other techniques such as SFI and CFI evolve by reducing the number of requirements on the
351 application, the person applying the sandbox, or both.

4.2 Policy Flexibility as a Usability Bellwether

353 Requiring more work out of the user or more specific attributes of an application lowers the odds that a
354 sandbox will be applied, thus it is natural that research on specific techniques reduce these burdens over
355 time. We find that the nature of the policy has an influence on how burdensome a sandbox is. About
356 half of sandboxes with fixed policies require the application be compiled using a special compiler or uses
357 a sandbox-specific framework or library. Many fixed-policy sandboxes also require the user to run a tool,
358 often a program re-writer, or to install some sandbox component. In comparison, nearly all sandboxes with
359 flexible policies require the user to write a policy manually, but few have additional requirements for the
360 application. Given the burdens involved in manually writing a security policy, the message is clear—easy to
361 use sandboxes reduce the user-facing flexibility of the policies they impose.

362 Forty-eight sandboxes, more than two-thirds of our sample, use a fixed policy. In all of these cases the
363 policy itself exists within the logic of the sandbox. In the remaining cases, the policy is encoded in the logic
364 of the application twice (e.g. through the use of the sandbox as a framework), and the remaining seventeen
365 cases require the user to manually write a policy.

366 In cases where the user must manually write the policy, it would help the user if the sandbox supported
367 a mechanism for managing policies—to ensure policies do not have to be duplicated repeatedly for the same
368 application, to generate starter policies for specific cases, to ensure policies can apply to multiple applications,
369 etc. This type of management reduces the burden of having to manually write policies in potentially complex
370 custom policy languages. Support for the policy writer is also important because the policies themselves can
371 be a source of vulnerabilities (Rosenberg, 2012). Eight out of twenty-six cases where policy management
372 is appropriate offered some central mechanism for storing existing policies, where they could potentially be
373 shared among users. However, none of the papers in our sample list policy management as a contribution,
374 nor do any of the papers attempt to validate any management constructs that are present. However, it is
375 possible that there are papers outside of our target conferences that explicitly discuss management. For
376 example, programming languages and software engineering conferences are more focused on policy authoring
377 concerns and management may therefore be the focus of a paper that appears in one of those conferences.
378 However, in spite of the fact that two of the authors of this paper are active researchers in the Programming
379 Language community and three are active in the Software Engineering community, we are not aware of any
380 such paper.

4.3   The State of Practice in Sandbox Validation

There is little variation in the claims that are made about sandboxes. Most claim to either encapsulate a set of threats or to increase the difficulty of writing successful exploits for code-level vulnerabilities. All but four measure the performance overhead introduced by the sandbox. Thirty-seven papers, more than half, make claims about the types of components the sandbox applies to, typically because the paper applies an existing technique to a different domain or extends it to additional components.

While there is wide variety in how these claims are validated, we observe measurable patterns. In our data set, proof and analytical analysis were, by far, the least used techniques. The lack of analytical analysis is due to the fact that the technique is primarily useful when the security of the mechanism depends on randomness, which is true of few sandboxes in our set. However, proof appears in two cases: (1) to prove properties of data flows and (2) six papers prove the correctness of a mechanism enforcing a fixed policy. The rarity of proof in the sandboxing domain is not surprising given the difficulty involved. Proof is particularly difficult in cases where one would ideally prove that a policy enforcement mechanism is capable of enforcing all possible policies a user can define, which we did not see attempted. Instead, claims are often validated empirically or in ways that are *ad hoc* and qualitative.

In empirical evaluations, case studies are the most common technique for all claims, often because proof was not attempted and there is no existing benchmark suite that highlights the novel aspects of the sandbox. For example, papers for sandboxes with fixed policies often want to show a particular class of vulnerabilities can no longer be exploited in sandboxed code, thus examples of vulnerable applications and exploits for their vulnerabilities must be gathered or, very rarely, synthesized. When claims were empirically validated, the results were not comparable in fifteen out of sixty-two cases for performance, twenty-two out of forty-two cases for security, and twenty-four out of thirty-one cases for applicability because non-public data was used in the discussed experiments. Non-public data takes the form of unlabeled exploits, undisclosed changes to public applications, and unreleased custom example cases (e.g. applications built using a sandbox's framework where the examples were not released).

Security claims are notoriously difficult to formalize, hence the pervasive lack of proof. Many papers instead vet their security claims using multi-faceted strategies, often including both common empirical approaches: case studies and experiments using benchmark suites. However, Figures 6 and 7 illustrate an interesting finding: In twenty-nine papers where multi-faceted strategies are not used, authors pick one empirical tactic and argue that their claims are true. Argumentation in this space is problematic because all of the arguments are *ad hoc*, which makes evaluations that should be comparable difficult to compare at best but more often incomparable. Furthermore, we observed many cases where arguments essentially summarize as, "Our sandbox is secure because the design is secure," with details of the design occupying most of the paper in entirely qualitative form. Not only are these types of arguments difficult to compare in cases where sandboxes are otherwise quite similar, it is even harder to see if they are complete in the sense that every sub-claim is adequately addressed.

Our correlational analyses show no significant trends in security or applicability analyses, however performance validation has improved over time. Table 5 summarizes the Spearman correlations and their p-values per validation category. Spearman correlations fall in the range [-1,1], where a value of 0 is interpreted as no correlation, positive values show a positive correlation, and negative values a negative correlation. The magnitude of the coefficient grows towards 1 as time and the validation rank become closer to perfect monotonic functions (i.e. when a positive and perfect monotonic relationship exists, the Spearman correlation is 1).

Performance validation is positively, and statistically significantly, correlated with the passage of time. We observe that performance validation has advanced from a heavy reliance on benchmark suites to the use multi-faceted strategies that include the use of benchmark suites and case studies (typically to perform micro-benchmarks) that make use of public data—which ensures the results are comparable with future sandboxes. While the applicability validation correlation is not statistically significant, we observe that argumentation was abandoned early on in favor of case studies, with some emphasis on including benchmark suites in later years. There is no apparent change in security validation over time.

We fit linear models to each validation category separately and together relative to ranked citation counts to see if validation practices are predictive of future citations. All of the models achieved an R-squared value of 0.54 which suggests that passage of time and validation practices jointly explain about half of the variance
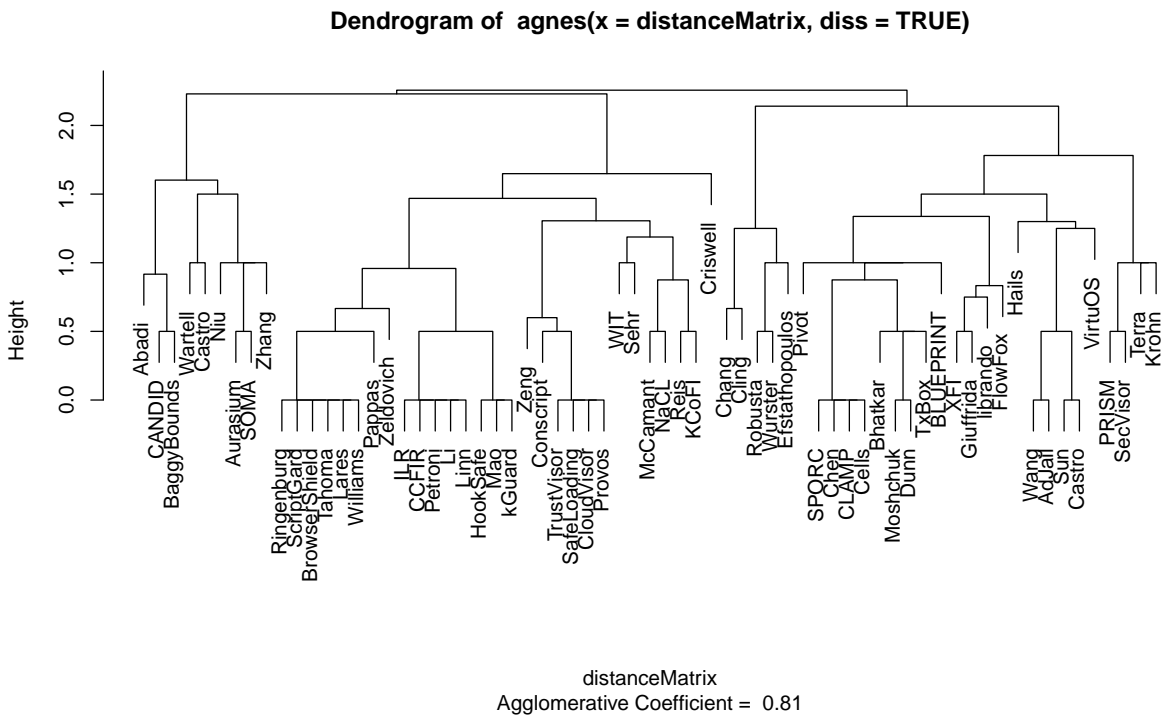
Figure 6: A dendrogram displaying the clusters for sandboxing papers taking into account validation categories. At the top most level, where two clusters exist, the clusters respectively represent sandboxes that emphasize multi-faceted empirical security validation and those that do not.

Table 5: The Spearman correlations and their statistical significances per validation category. Data with correlation coefficients closer to 1 have stronger correlations.

	Correlation (ρ)	p-value
Security Validation	-0.02	0.894
Performance Validation	0.30	0.014
Applicability Validation	0.20	0.105
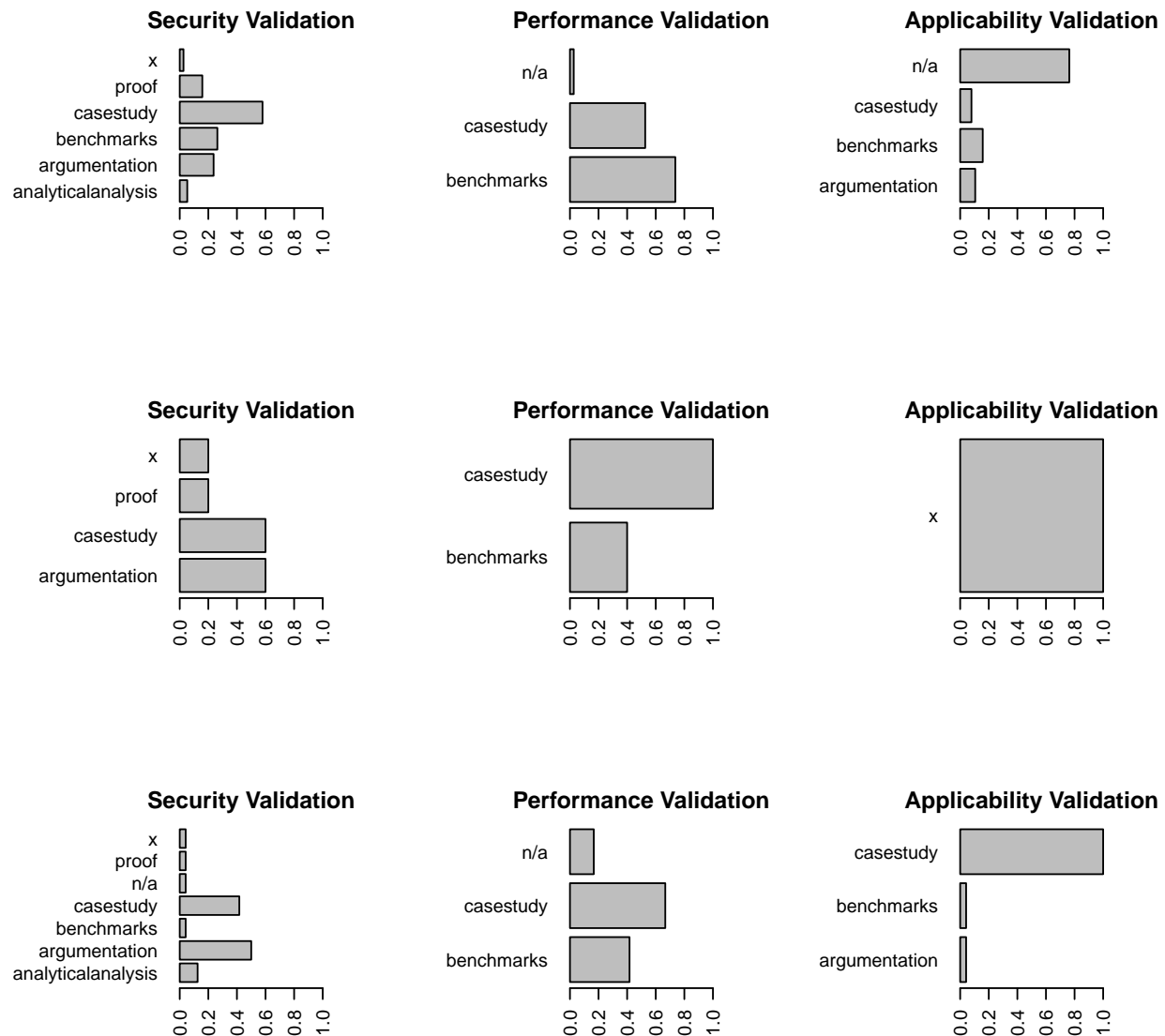
Figure 7: Breakdown of the representation of validation codes per claim type for the three validation clusters found in our dataset. Each row contains the data for one cluster. The bottom two clusters include papers that do not emphasize multi-faceted security validation strategies, instead relying on case studies and arguments that security claims are true. Cases where a claim was made but not validated are labeled with an "x".

in citation count ranks. Validation practices on their own are not predictive of how highly cited a paper will become.

5   Strengthening Sandboxing Results

The existing body of knowledge within the sandboxing community provides a strong basis for securing current and future software systems. However, the results in Section 4 highlight several gaps. In this section we discuss how structured arguments can solve the problems presented by incomparable and incomplete *ad hoc* arguments (Section 5.1) and possible ways to enhance sandbox and policy usability (Section 5.2).

5.1   Structured Arguments

Sandboxes are often evaluated against coarse criteria such as the ability to stop exploits against certain classes of vulnerabilities, to encapsulate certain categories of operations, or to function in new environments. However, these coarse criteria typically require the sandbox to address a number of sub-criteria. For example, Zhang and Sekar (2013) provide CFI without requiring compiler support or *a priori* metadata, unlike earlier implementations. To ensure the technique is secure, they must be sure that independently transformed program modules maintain CFI when composed. Details that clarify how an individual criterion is fulfilled can easily be lost when *ad hoc* arguments are used in an effort to persuade readers that the criterion has been met; particularly in sandboxes with non-trivial design and implementation details. This can leave the reader unable to compare similar sandboxes or confused about whether or not contributions were validated.

Since many of the security criteria are repeated across most papers, the cost of developing substructure can be amortized across lots of communal use. There are many possible ways to structure arguments in support to security claims:

- Assurance cases (Weinstock et al., 2007; Kelly, 1999) provide graphical structures that explicitly tie claims together in trees that show how claims are narrowed. Knight (2015) provides a concise introduction to the topic. These structures also explicitly link leaf claims to the evidence that supports the claim. Assurance cases were created in response to several fatal accidents resulting from failures to systematically and thoroughly understand safety concerns in physical systems. Their use has spread to security and safety critical systems of nearly every variety in recent decades with case studies from aerospace (Graydon et al., 2007) and a sandbox called S^3 (Rodes et al., 2015) that was not analyzed as part of this study (Nguyen-Tuong et al., 2014). Sandboxing papers can use assurance cases to decompose claims to their most simple components, then link those components to relevant evidence in the paper (e.g. a summary of specific results, a specific section reference, etc.).

- Maass et al. (2014) use a qualitative framework to compare sandboxes based on what happens when a sandbox fails, is bypassed, or holds. Authors could structure their arguments by using the framework to describe their specific sandbox without performing explicit comparisons.

- Structured abstracts (Hartley, 2004; R. Brian Haynes et al., 1990) are used in many medical journals to summarize key results and how those results were produced. These abstracts have the benefit of being quick to read while increasing the retention of information, largely thanks to the use of structure to guide authors in precisely summarizing their work.

- Papers could provide a table summarizing their contributions and the important design or implementation details that reflect the contribution.

All of these approaches provide the reader with data missing in *ad hoc* arguments: A specific map from the claims made about a sandbox to evidence that justifies the claim has been met. They are also necessarily qualitative, but as we saw earlier, arguments are often used where more rigorous approaches are currently intractable. We believe that adding structure to these arguments is a reasonable advancement of the state of practice in sandbox validation.

Table 6: Claims made about sandboxes (●: Security, ⚐: Performance, and ✎: Applicability) and their valida-
tion strategies (✿: Proof, ✄: Analytical Analysis, ⟳: Benchmarks, ⌂: Case Studies, and ⚇: Argumentation).
Grayed out icons mean a claim was not made or a strategy was not used. Icons made by Freepik from
www.flaticon.com.

Category	Citation	Conference	Claims	● Val.	⚐ Val.	✎ Val.
Other (Syscall)	Provos (2003)	Usenix				
Virtualization	Garfinkel et al. (2003)	SOSP				
Diversity	Bhatkar et al. (2005)	Usenix				
Other (Syscall)	Linn et al. (2005)	Usenix				
CFI	Abadi et al. (2005)	CCS				
Other (Memory)	Ringenburg and Grossman (2005)	CCS				
MAC	Efstathopoulos et al. (2005)	SOSP				
Web	Cox et al. (2006)	Oakland				
SFI	McCamant and Morrisett (2006)	Usenix				
CFI, SFI	Erlingsson et al. (2006)	OSDI				
Other (DFI)	Castro et al. (2006)	OSDI				
Web	Reis et al. (2006)	OSDI				
Other (InfoFlow)	Zeldovich et al. (2006)	OSDI				
MI/AC	Li et al. (2007)	Oakland				
Web	Bandhakavi et al. (2007)	CCS				
Web	Chen et al. (2007)	CCS				
Virtualization	Petroni and Hicks (2007)	CCS				
Virtualization	Seshadri et al. (2007)	SOSP				
Virtualization	Criswell et al. (2007)	SOSP				
Web	Wang et al. (2007)	SOSP				
Other (InfoFlow)	Krohn et al. (2007)	SOSP				
CFI	Akritidis et al. (2008)	Oakland				
Virtualization	Payne et al. (2008)	Oakland				
MI/AC	Sun et al. (2008)	Oakland				
Other (TaintTrack)	Chang et al. (2008)	CCS				
Web	Oda et al. (2008)	CCS				
Other (OS)	Williams et al. (2008)	OSDI				
SFI	Yee et al. (2009)	Oakland				
Web	Louw and Venkatakrishnan (2009)	Oakland				
Web	Parno et al. (2009)	Oakland				
Other (Memory)	Akritidis et al. (2009)	Usenix				
Virtualization	Wang et al. (2009)	CCS				
SFI	Castro et al. (2009)	SOSP				
Virtualization	McCune et al. (2010)	Oakland				
Web	Meyerovich and Livshits (2010)	Oakland				
Other (Memory)	Akritidis (2010)	Usenix				
SFI	Sehr et al. (2010)	Usenix				
Web	Louw et al. (2010)	Usenix				
Other (OS)	Wurster and van Oorschot (2010)	CCS				
SFI, Other (UserPolicy)	Siefers et al. (2010)	CCS				
Web	Feldman et al. (2010)	OSDI				
MI/AC	Owen et al. (2011)	Oakland				
Other (Transactions)	Jana et al. (2011)	Oakland				
CFI	Zeng et al. (2011)	CCS				
Web	Saxena et al. (2011)	CCS				
Web	Chen et al. (2011)	CCS				
Virtualization	Zhang et al. (2011)	SOSP				
SFI	Mao et al. (2011)	SOSP				
Virtualization	Andrus et al. (2011)	SOSP				
Diversity	Pappas et al. (2012)	Oakland				
Diversity	Hiser et al. (2012)	Oakland				
SFI	Payer et al. (2012)	Oakland				
CFI	Kemerlis et al. (2012)	Usenix				
Diversity	Giuffrida et al. (2012)	Usenix				
MI/AC	Xu et al. (2012)	Usenix				
Diversity	Wartell et al. (2012)	CCS				
Web, Other (InfoFlow)	De Groef et al. (2012)	CCS				
Virtualization	Dunn et al. (2012)	OSDI				
Web (MI/AC)	Giffin et al. (2012)	OSDI				
CFI	Zhang et al. (2013)	Oakland				
CFI	Zhang and Sekar (2013)	Usenix				
CFI, SFI	Niu and Tan (2013)	CCS				
Diversity	Homescu et al. (2013)	CCS				
Other (OS)	Moshchuk et al. (2013)	CCS				
Virtualization	Nikolaev and Back (2013)	SOSP				
CFI	Criswell et al. (2014)	Oakland				
Web	Mickens (2014)	Oakland				

5.2 Sandbox and Policy Usability

Sandbox and policy usability are concerns of interest to the following stakeholders: *Practitioners* that must correctly use sandboxes to improve the security postures of their systems and *users* that must work with sandboxed applications. Some security researchers do attempt to make their sandboxes more usable by providing policy management or reducing requirements on the user, but usability is definitely not a focus of any of the papers in our sample.

Our data shows that, with very few exceptions, sandbox researchers thoroughly evaluate the performance of their sandboxes. Why is there focus on this practical concern but not on usability? We observe that a focus on performance evaluation is partially motivated by the fact that overhead is relatively easy to quantify, but we also saw many cases where researchers were explicitly concerned with whether or not a sandbox was too resource intensive for adoption. The latter is a reasonable concern; Szekeres et al. (2013) pointed out that many mitigations for memory corruption vulnerabilities are not adopted because performance concerns outweigh protection merits.

While the idea that performance is an important adoption concern is compelling and likely reflects reality, we cannot correlate performance with the adoption of the sandboxes in our set. We cannot find a correlation because the sandboxes and their techniques in our set remain almost entirely unadopted. We only found four cases where sandboxes in our set were either directly adopted or where the techniques they evaluate are clearly implemented in a different but adopted sandbox. A lack of adoption is present even for techniques where performance and applicability have been improved over multiple decades (e.g. SFI). Three of the adopted sandboxes were created by the industry itself or by entities very closely tied to it: Google NaCl was designed with the intention of adopting it in Google Chrome in the short term (Yee et al., 2009; Sehr et al., 2010) and the paper on `systrace` was published with functioning open source implementations for most Unix-like operating systems (Provos, 2003). While the case for adoption is weaker, Cells (Andrus et al., 2011) is a more advanced design than one VMware developed in parallel (Berlind, 2012), although the sandboxes both aim to partition phones into isolated compartments using virtualization (e.g. one for work and one for personal use). More recently, Microsoft has stated that Visual Studio 2015 will ship with an exploit mitigation that we believe is equivalent to what the research community calls CFI (Hogg, 2015). A third party analysis supports this belief, however the uncovered implementation details differ from the techniques implemented in published research (Tang, 2015).

We argue that the need to evaluate the usability of our sandboxes is evidenced by the observation that performance and security evaluation are not sufficient to drive adoption. Usability is of particular concern in cases where the sandbox requires developers without security expertise (1) to re-architect applications to apply the sandbox and/or (2) to develop a security policy. In practice, it is quite common for developers without a security focus to apply sandboxes, particularly Java's. In fact, usability issues have factored into widely publicized vulnerabilities in how sandboxes were applied to Google Chrome and Adobe Reader as well as the many vulnerable applications of the Java sandbox (Coker et al., 2015). In all of these cases applying the sandbox is a relatively manual process where it is difficult for the applier to be sure he is fully imposing the desired policy and without missing relevant attack surfaces. These usability issues have caused vulnerabilities that have been widely exploited to bypass the sandboxes. We call on the community to evaluate the following usability aspects of their sandboxes where appropriate:

- The intended users are capable of writing policies for the component(s) to be sandboxed that are neither over- or under-privileged.

- Policy enforcement mechanisms can be applied without missing attack surfaces that compromise the sandbox in the targeted component(s).

- Source code transformations (e.g. code re-writing or annotations) do not substantially burden future development or maintenance.

- The sandbox, when applied to a component, does not substantially alter a typical user's interactions with the sandboxed component.

Ideally many of these points would be evaluated during user studies with actual stakeholders. However, we believe that we can make progress on all of these points without the overhead of a full user study,

528 particularly because we are starting from a state where no usability evaluations are performed. For example,
529 authors can describe correct ways to determine what privileges in their policy language a component needs or
530 even provide tools to generate policies to mitigate the risks presented by under- and over-privileged policies.
531 Similarly, tooling can be provided to help users install policy enforcement mechanisms or check that manual
532 applications of a mechanism are correct. Sandbox developers can transform or annotate representative open
533 source applications and use repository mining[12] to determine how sandbox alternations are affected by code
534 evolution present in the repository (Kagdi et al., 2007; Yan et al., 2014; Mauczka et al., 2010; Stuckman
535 and Purtilo, 2014). Finally, a summary of how the sandbox qualitatively changes a user's experience with a
536 sandboxed component would provide a gauge for how much the sandbox burdens end-users.

6   Enabling Meta-Analysis

538 We believe a key contribution of this work is the use of multi-disciplinary and systematic methodologies for
539 drawing conclusions about a large body of security techniques. In this section, we discuss the generalizability
540 of our methodology and suggest other areas to which it can be applied. Then, we discuss some challenges
541 that we faced when doing this research and suggest changes that would address these challenges.

6.1   Generalizability of Methodology

543 The methodology employed in this paper is based on two research approaches: Qualitative Content Analysis
544 and Systematic Literature Reviews. Qualitative Content Analysis is primarily used in the humanities and
545 social sciences. Systematic Literature Reviews were first applied to medical studies and are used primarily in
546 empirical fields. The differences between sandboxing papers are bigger than the differences between studies
547 of a particular cancer treatment. In addition, sandboxing papers do not fit into the "native" domains of
548 either approach—their primary contributions are designs, techniques, and implementations.
549   The result of these differences, is that most literature reviews and systemizations in computing are done
550 in an ad hoc manner. Our computing research is worthy of a more rigorous approach and we think the
551 methodology applied in this paper can and should be applied to other topics. In fact, any topic of active
552 research where the primary contributions is an engineered artifact, but without a clear and precise definition,
553 would be amenable to our approach. These topics span computing research from software engineering
554 (e.g. service oriented architecture, concurrent computation models) to systems (e.g. green computing, no
555 instruction set computing) to human-computer interaction (e.g. GUI toolkits, warning science).

6.2   Meta-analysis Challenges and Suggested Solutions

557 In our experience, the biggest roadblock standing in the way of applying the same techniques to other
558 segments of the research community lies in the difficulty involved in collecting analyzable metadata about
559 papers. We experienced several fixable issues:

560 - The major publishers in computer science—IEEE, ACM, and Usenix—do not provide publicly available
561   mechanisms to collect metadata and either rate limit or outright ban scraping.[13] In our case, the
562   painstaking process of collecting and curating analyzable metadata across several sources limited our
563   ability to explore hypotheses about our dataset's papers and their relationships to publications not in
564   the set.

565 - The metadata is limited and contains little semantic content—typically the metadata includes the
566   authors, title, data, and DOI, but little else. If abstracts and keywords were easier to harvest we could
567   have more systematically derived topics of interest within the sandboxing community.

568 - Links to papers on publisher websites use internal identifiers (e.g. `http://dl.acm.org/citation.cfm?`
569   `id=2498101`) instead of DOI. This makes it difficult to reference papers across publisher repositories.

570 - Conference websites have inconsistent layouts, which increases the difficulty of data collection.

---

[12]`http://msrconf.org`
[13]In at least one case ACM provided a copy of their digital library for scraping (Bergmark et al., 2001)

We believe easier access to this data would have allowed us to draw more conclusions about how sandboxing papers are related and how the sandboxing landscape has evolved over time. For example, we explored the idea of using a more developed citation graph than Figure 2 to trace the lineage of sandboxing techniques, but found the required resource expenditures were outside of our means. This data may provide support for explanations regarding the lack of advancement in security validation practices (e.g. by showing an emphasis on a different but important dimension of advancement). These points are important to understand how we got to the current state of practice, thus improving our ability to recognize and advance means for enhancing our results.

On another data collection point, we averaged about 45 minutes per paper to code the data necessary to answer our research questions. While we do not claim that our research questions are of universal interest to the sandboxing community, we did observe that papers that answer all or most of the questions in the abstract are often clearly written throughout and easy to interpret. A small minority of sandboxing papers have far less specific abstracts. In these cases, the papers often took double the average time to comprehend and interpret. It may be useful to strive to clearly answer questions like ours in future papers to show practitioners the value sandbox researchers bring to the table.

7   Threats to Validity

Due to the complexity of the text and concepts we are interpreting, there is some risk that other coders would assign quotes to different codes. Different codes will change the results, but we believe this risk is mitigated through our tests of the coding frame and by our efforts to select clear quotes. Furthermore, the correlative nature of our results ensures that a few code divergences will not dramatically change the analysis's outcomes.

The primary risk is that we are missing relevant quotes that add codes to our dataset. This is typically mitigated in QCA by fully segmenting the text, but we decided against that strategy because of the very large data set we studied and irrelevance of most of the text to our goals. We did search PDFs for relevant keywords we observed were commonly linked to specific codes throughout the process (e.g. "proof", "available" to find the availability of sandbox artifacts for evaluation, "experiment" to signal a case study or benchmark, etc.) to decrease the odds of missing a code. While this does mitigate the risk, it is still likely that our results under-approximate the state of the sandboxing landscape.

8   Conclusion

We systematically analyzed the sandboxing landscape as it is represented by five top-tier security and systems conferences. Our analysis followed a multidisciplinary strategy that allowed us to draw conclusions backed by rigorous interpretations of qualitative data, statistics, and graph analysis. Based on our results, we conclude that the sandbox research community will benefit from the use of structured arguments in support of security claims and the validation of sandbox and policy usability. We suggested lightweight ways to move forward in achieving these goals. Our data also shows that there is a dearth of science regarding the management of security policies for sandboxes, although we did not discuss this gap in depth.

References

Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-flow Integrity Principles, Implementations, and Applications. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS '05)*. ACM, New York, NY, USA, 340–353. `DOI:http://dx.doi.org/10.1145/1102120.1102165`

Periklis Akritidis. 2010. Cling: A Memory Allocator to Mitigate Dangling Pointers. In *USENIX Security (USENIX Security'10)*. USENIX Association, Berkeley, CA, USA, 12–12. `http://dl.acm.org/citation.cfm?id=1929820.1929836`

Periklis Akritidis, Cristian Cadar, Costin Raiciu, Manuel Costa, and Miguel Castro. 2008. Preventing Memory Error Exploits with WIT. In *IEEE Symposium on Security and Privacy (SP '08)*. IEEE Computer Society, Washington, DC, USA, 263–277. `DOI:http://dx.doi.org/10.1109/SP.2008.30`

Periklis Akritidis, Manuel Costa, Miguel Castro, and Steven Hand. 2009. Baggy Bounds Checking: An Efficient and Backwards-compatible Defense Against Out-of-bounds Errors. In *USENIX Security (SSYM'09)*. USENIX Association, Berkeley, CA, USA, 51–66. `http://dl.acm.org/citation.cfm?id=1855768.` `1855772`

F. Al Ameiri and K. Salah. 2011. Evaluation of Popular Application Sandboxing. In *International Conference for Internet Technology and Secured Transactions (ICITST), 2011*. IEEE, Washington, DC, USA, 358 – 362.

Jeremy Andrus, Christoffer Dall, Alexander Van't Hof, Oren Laadan, and Jason Nieh. 2011. Cells: A Virtual Mobile Smartphone Architecture. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '11)*. ACM, New York, NY, USA, 173–187. `DOI:http://dx.doi.org/10.1145/2043556.2043574`

Sruthi Bandhakavi, Prithvi Bisht, P. Madhusudan, and V. N. Venkatakrishnan. 2007. CANDID: Preventing Sql Injection Attacks Using Dynamic Candidate Evaluations. In *ACM Conference on Computer and Communications Security (CCS) (CCS '07)*. ACM, New York, NY, USA, 12–24. `DOI:` `http://dx.doi.org/10.1145/1315245.1315249`

Jeffrey M. Barnes. 2013. *Software Architecture Evolution*. Ph.D. Dissertation. Carnegie Mellon University.

Bernard Berelson. 1952. *Content Analysis in Communication Research*. Free Press, Glencoe, IL, USA.

Donna Bergmark, Paradee Phempoonpanich, and Shumin Zhao. 2001. Scraping the ACM Digital Library. *SIGIR Forum* 35 (Sept. 2001), 1–7. `DOI:http://dx.doi.org/10.1145/511144.511146`

David Berlind. 2012. VMware shows Android based virtual machines. `http://www.informationweek.` `com/mobile/mobile-devices/ces-2012-vmware-shows-android-based-virtual-machines/d/d-id/` `1102135?`. (2012). Accessed: 2015-04-30.

Sandeep Bhatkar, R. Sekar, and Daniel C. DuVarney. 2005. Efficient Techniques for Comprehensive Protection from Memory Error Exploits. In *USENIX Security (SSYM'05)*. USENIX Association, Berkeley, CA, USA, 17–17. `http://dl.acm.org/citation.cfm?id=1251398.1251415`

David Budgen and Pearl Brereton. 2006. Performing Systematic Literature Reviews in Software Engineering. In *International Conference on Software Engineering (ICSE '06)*. ACM, New York, NY, USA, 1051–1052. `DOI:http://dx.doi.org/10.1145/1134285.1134500`

Justin Cappos, Armon Dadgar, Jeff Rasley, Justin Samuel, Ivan Beschastnikh, Cosmin Barsan, Arvind Krishnamurthy, and Thomas Anderson. 2010. Retaining Sandbox Containment Despite Bugs in Privileged Memory-safe Code. In *ACM Conference on Computer and Communications Security (CCS) (CCS '10)*. ACM, New York, NY, USA, 212–223. `DOI:http://dx.doi.org/10.1145/1866307.1866332`

Miguel Castro, Manuel Costa, and Tim Harris. 2006. Securing Software by Enforcing Data-flow Integrity. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI '06)*. USENIX Association, Berkeley, CA, USA, 147–160. `http://dl.acm.org/citation.cfm?id=1298455.1298470`

Miguel Castro, Manuel Costa, Jean-Philippe Martin, Marcus Peinado, Periklis Akritidis, Austin Donnelly, Paul Barham, and Richard Black. 2009. Fast Byte-granularity Software Fault Isolation. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '09)*. ACM, New York, NY, USA, 45–58. `DOI:` `http://dx.doi.org/10.1145/1629575.1629581`

Ramesh Chandra, Taesoo Kim, Meelap Shah, Neha Narula, and Nickolai Zeldovich. 2011. Intrusion Recovery for Database-backed Web Applications. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '11)*. ACM, New York, NY, USA, 101–114. `DOI:http://dx.doi.org/10.1145/2043556.2043567`

Walter Chang, Brandon Streiff, and Calvin Lin. 2008. Efficient and Extensible Security Enforcement Using Dynamic Data Flow Analysis. In *ACM Conference on Computer and Communications Security (CCS) (CCS '08)*. ACM, New York, NY, USA, 39–50. `DOI:http://dx.doi.org/10.1145/1455770.1455778`

Eric Yawei Chen, Jason Bau, Charles Reis, Adam Barth, and Collin Jackson. 2011. App Isolation: Get the Security of Multiple Browsers with Just One. In *ACM Conference on Computer and Communications Security (CCS) (CCS '11)*. ACM, New York, NY, USA, 227–238. `DOI:http://dx.doi.org/10.1145/2046707.2046734`

Shuo Chen, David Ross, and Yi-Min Wang. 2007. An Analysis of Browser Domain-isolation Bugs and a Light-weight Transparent Defense Mechanism. In *ACM Conference on Computer and Communications Security (CCS) (CCS '07)*. ACM, New York, NY, USA, 2–11. `DOI:http://dx.doi.org/10.1145/1315245.1315248`

Zack Coker, Michael Maass, Tianyuan Ding, Claire Le Goues, and Joshua Sunshine. 2015. Evaluating the Flexibility of the Java Sandbox. In *Annual Computer Security Applications Conference (ACSAC 2015)*. ACM, New York, NY, USA, 1–10. `DOI:http://dx.doi.org/10.1145/2818000.2818003`

Richard S. Cox, Steven D. Gribble, Henry M. Levy, and Jacob Gorm Hansen. 2006. A Safety-Oriented Platform for Web Applications. In *IEEE Symposium on Security and Privacy (SP '06)*. IEEE Computer Society, Washington, DC, USA, 350–364. `DOI:http://dx.doi.org/10.1109/SP.2006.4`

J. Criswell, N. Dautenhahn, and V. Adve. 2014. KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. In *IEEE Symposium on Security and Privacy*. IEEE, Washington, DC, USA, 292–307. `DOI:http://dx.doi.org/10.1109/SP.2014.26`

John Criswell, Andrew Lenharth, Dinakar Dhurjati, and Vikram Adve. 2007. Secure Virtual Architecture: A Safe Execution Environment for Commodity Operating Systems. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '07)*. ACM, New York, NY, USA, 351–366. `DOI:http://dx.doi.org/10.1145/1294261.1294295`

Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. 2012. FlowFox: A Web Browser with Flexible and Precise Information Flow Control. In *ACM Conference on Computer and Communications Security (CCS) (CCS '12)*. ACM, New York, NY, USA, 748–759. `DOI:http://dx.doi.org/10.1145/2382196.2382275`

Norman K. Denzin and Yvonna S. Lincoln. 2011. Introduction: The Discipline and Practice of Qualitative Research. In *The Sage Handbook of Qualitative Research* (4th ed.). Sage, Thousand Oaks, CA, USA.

Alan M. Dunn, Michael Z. Lee, Suman Jana, Sangman Kim, Mark Silberstein, Yuanzhong Xu, Vitaly Shmatikov, and Emmett Witchel. 2012. Eternal Sunshine of the Spotless Machine: Protecting Privacy with Ephemeral Channels. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI'12)*. USENIX Association, Berkeley, CA, USA, 61–75. `http://dl.acm.org/citation.cfm?id=2387880.2387887`

Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. 2005. Labels and Event Processes in the Asbestos Operating System. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '05)*. ACM, New York, NY, USA, 17–30. `DOI:http://dx.doi.org/10.1145/1095810.1095813`

Úlfar Erlingsson, Martín Abadi, Michael Vrable, Mihai Budiu, and George C. Necula. 2006. XFI: Software Guards for System Address Spaces. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI '06)*. USENIX Association, Berkeley, CA, USA, 75–88. `http://dl.acm.org/citation.cfm?id=1298455.1298463`

Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. 2010. SPORC: Group Collaboration Using Untrusted Cloud Resources. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 1–. `http://dl.acm.org/citation.cfm?id=1924943.1924967`

706 Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. 2003. Terra: A Virtual Machine-
707   based Platform for Trusted Computing. In *ACM Symposium on Operating Systems Principles (SOSP)*
708   *(SOSP '03)*. ACM, New York, NY, USA, 193–206. `DOI:http://dx.doi.org/10.1145/945445.945464`

709 Dimitris Geneiatakis, Georgios Portokalidis, Vasileios P. Kemerlis, and Angelos D. Keromytis. 2012. Adaptive
710   Defenses for Commodity Software Through Virtual Application Partitioning. In *ACM Conference on*
711   *Computer and Communications Security (CCS) (CCS '12)*. ACM, New York, NY, USA, 133–144. `DOI:`
712   `http://dx.doi.org/10.1145/2382196.2382214`

713 Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, John C. Mitchell, and Alejandro
714   Russo. 2012. Hails: Protecting Data Privacy in Untrusted Web Applications. In *USENIX Symposium on*
715   *Operating Systems Design and Implementation (OSDI) (OSDI'12)*. USENIX Association, Berkeley, CA,
716   USA, 47–60. `http://dl.acm.org/citation.cfm?id=2387880.2387886`

717 Cristiano Giuffrida, Anton Kuijsten, and Andrew S. Tanenbaum. 2012. Enhanced Operating System Se-
718   curity Through Efficient and Fine-grained Address Space Randomization. In *USENIX Security (Se-*
719   *curity'12)*. USENIX Association, Berkeley, CA, USA, 40–40. `http://dl.acm.org/citation.cfm?id=`
720   `2362793.2362833`

721 P.J. Graydon, J.C. Knight, and E.A. Strunk. 2007. Assurance Based Development of Critical Systems. In
722   *Dependable Systems and Networks*. IEEE, Washington, DC, USA, 347–357. `DOI:http://dx.doi.org/`
723   `10.1109/DSN.2007.17`

724 James Hartley. 2004. Current findings from research on structured abstracts. *Journal of the Medical Library*
725   *Association* 92, 3 (2004), 368–371.

726 Jason Hiser, Anh Nguyen-Tuong, Michele Co, Matthew Hall, and Jack W. Davidson. 2012. ILR: Where'D My
727   Gadgets Go?. In *IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer Society, Washington,
728   DC, USA, 571–585. `DOI:http://dx.doi.org/10.1109/SP.2012.39`

729 Jim Hogg. 2015. Control Flow Guard. `http://blogs.msdn.com/b/vcblog/archive/2014/12/08/`
730   `visual-studio-2015-preview-work-in-progress-security-feature.aspx`. (2015). Accessed: 2015-
731   04-30.

732 Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. 2013. Librando: Transparent code
733   randomization for just-in-time compilers. In *ACM Conference on Computer and Communications Security*
734   *(CCS) (CCS '13)*. ACM, New York, NY, USA, 993–1004. `DOI:http://dx.doi.org/10.1145/2508859.`
735   `2516675`

736 S. Jana, D.E. Porter, and V. Shmatikov. 2011. TxBox: Building Secure, Efficient Sandboxes with System
737   Transactions. In *IEEE Symposium on Security and Privacy*. IEEE, Washington, DC, USA, 329–344. `DOI:`
738   `http://dx.doi.org/10.1109/SP.2011.33`

739 Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. 2007. A survey and taxonomy of approaches for
740   mining software repositories in the context of software evolution. *Journal of Software Maintenance and*
741   *Evolution: Research and Practice* 19, 2 (2007), 77–131.

742 Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: An introduction to cluster analysis.*
743   Vol. 344. John Wiley & Sons, New York, NY, USA.

744 T. Kelly. 1999. *Arguing Safety - A Systematic Approach to Safety Case Management.* Ph.D. Dissertation.
745   Department of Computer Science, University of York.

746 Vasileios P. Kemerlis, Georgios Portokalidis, and Angelos D. Keromytis. 2012. kGuard: Lightweight Ker-
747   nel Protection Against Return-to-user Attacks. In *USENIX Security (Security'12)*. USENIX Association,
748   Berkeley, CA, USA, 39–39. `http://dl.acm.org/citation.cfm?id=2362793.2362832`

Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Information and Software Technology* 51, 1 (Jan. 2009), 7–15. `DOI:http://dx.doi.org/10.1016/j.infsof.2008.09.009`

J. Knight. 2015. The Importance of Security Cases: Proof Is Good, But Not Enough. *IEEE Security Privacy Magazine* 13, 4 (July 2015), 73–75. `DOI:http://dx.doi.org/10.1109/MSP.2015.68`

Klaus H. Krippendorff. 2013. *Content Analysis: An Introduction to Its Methodology.* Sage, Thousand Oaks, CA, USA.

Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information Flow Control for Standard OS Abstractions. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '07)*. ACM, New York, NY, USA, 321–334. `DOI:http://dx.doi.org/10.1145/1294261.1294293`

Anna Lázár, Dániel Ábel, and Tamás Vicsek. 2009. Modularity measure of networks with overlapping communities. (2009).

Ninghui Li, Ziqing Mao, and Hong Chen. 2007. Usable Mandatory Integrity Protection for Operating Systems. In *IEEE Symposium on Security and Privacy (SP '07)*. IEEE Computer Society, Washington, DC, USA, 164–178. `DOI:http://dx.doi.org/10.1109/SP.2007.37`

Yanlin Li, Jonathan McCune, James Newsome, Adrian Perrig, Brandon Baker, and Will Drewry. 2014. Mini-Box: A Two-Way Sandbox for x86 Native Code. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association, Philadelphia, PA, 409–420. `https://www.usenix.org/conference/atc14/technical-sessions/presentation/li_yanlin`

C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, and J. H. Hartman. 2005. Protecting Against Unexpected System Calls. In *USENIX Security (SSYM'05)*. USENIX Association, Berkeley, CA, USA, 16–16. `http://dl.acm.org/citation.cfm?id=1251398.1251414`

Mike Ter Louw, Karthik Thotta Ganesh, and V. N. Venkatakrishnan. 2010. AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements. In *USENIX Security (USENIX Security'10)*. USENIX Association, Berkeley, CA, USA, 24–24. `http://dl.acm.org/citation.cfm?id=1929820.1929852`

Mike Ter Louw and V. N. Venkatakrishnan. 2009. Blueprint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers. In *IEEE Symposium on Security and Privacy (SP '09)*. IEEE Computer Society, Washington, DC, USA, 331–346. `DOI:http://dx.doi.org/10.1109/SP.2009.33`

Michael Maass, William L. Scherlis, and Jonathan Aldrich. 2014. In-nimbo Sandboxing. In *Symposium and Bootcamp on the Science of Security (HotSoS '14)*. ACM, New York, NY, USA, Article 1, 12 pages. `DOI:http://dx.doi.org/10.1145/2600176.2600177`

Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. 2014. *cluster: Cluster Analysis Basics and Extensions.* R Foundation for Statistical Computing, Vienna, Austria. R package version 1.15.2 — For new features, see the 'Changelog' file (in the package source).

Yandong Mao, Haogang Chen, Dong Zhou, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. 2011. Software Fault Isolation with API Integrity and Multi-principal Modules. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '11)*. ACM, New York, NY, USA, 115–128. `DOI:http://dx.doi.org/10.1145/2043556.2043568`

A. Mauczka, C. Schanes, F. Fankhauser, M. Bernhart, and T. Grechenig. 2010. Mining security changes in FreeBSD. In *Mining Software Repositories (MSR)*. IEEE, Washington, DC, USA, 90–93. `DOI:http://dx.doi.org/10.1109/MSR.2010.5463289`

Stephen McCamant and Greg Morrisett. 2006. Evaluating SFI for a CISC Architecture. In *USENIX Security (USENIX-SS'06)*. USENIX Association, Berkeley, CA, USA, Article 15, 16 pages. `http://dl.acm.org/citation.cfm?id=1267336.1267351`

Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. 2010. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Symposium on Security and Privacy (SP '10)*. IEEE Computer Society, Washington, DC, USA, 143–158. `DOI:http://dx.doi.org/10.1109/SP.2010.17`

J. McLean. 1997. Is the Trusted Computing Base Concept Fundamentally Flawed?. In *IEEE Symposium on Security and Privacy (SP '97)*. IEEE Computer Society, Washington, DC, USA, 2–. `http://dl.acm.org/citation.cfm?id=882493.884390`

Leo A. Meyerovich and Benjamin Livshits. 2010. ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser. In *IEEE Symposium on Security and Privacy (SP '10)*. IEEE Computer Society, Washington, DC, USA, 481–496. `DOI:http://dx.doi.org/10.1109/SP.2010.36`

James Mickens. 2014. Pivot: Fast, Synchronous Mashup Isolation Using Generator Chains. In *IEEE Symposium on Security and Privacy (SP '14)*. IEEE Computer Society, Washington, DC, USA, 261–275. `DOI:http://dx.doi.org/10.1109/SP.2014.24`

Alexander Moshchuk, Helen J. Wang, and Yunxin Liu. 2013. Content-based Isolation: Rethinking Isolation Policy Design on Client Systems. In *ACM Conference on Computer and Communications Security (CCS) (CCS '13)*. ACM, New York, NY, USA, 1167–1180. `DOI:http://dx.doi.org/10.1145/2508859.2516722`

Peter G. Neumann. 1990. Inside Risks: Insecurity About Security? *Commun. ACM* 33 (Aug. 1990), 170–170. `DOI:http://dx.doi.org/10.1145/79173.79113`

Anh Nguyen-Tuong, J.D. Hiser, M. Co, J.W. Davidson, J.C. Knight, N. Kennedy, D. Melski, W. Ella, and D. Hyde. 2014. To B or not to B: Blessing OS Commands with Software DNA Shotgun Sequencing. In *Dependable Computing Conference*. IEEE, Washington, DC, USA, 238–249. `DOI:http://dx.doi.org/10.1109/EDCC.2014.13`

Ruslan Nikolaev and Godmar Back. 2013. VirtuOS: An Operating System with Kernel Virtualization. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '13)*. ACM, New York, NY, USA, 116–132. `DOI:http://dx.doi.org/10.1145/2517349.2522719`

Ben Niu and Gang Tan. 2013. Monitor Integrity Protection with Space Efficiency and Separate Compilation. In *ACM Conference on Computer and Communications Security (CCS) (CCS '13)*. ACM, New York, NY, USA, 199–210. `DOI:http://dx.doi.org/10.1145/2508859.2516649`

Terri Oda, Glenn Wurster, P. C. van Oorschot, and Anil Somayaji. 2008. SOMA: Mutual Approval for Included Content in Web Pages. In *ACM Conference on Computer and Communications Security (CCS) (CCS '08)*. ACM, New York, NY, USA, 89–98. `DOI:http://dx.doi.org/10.1145/1455770.1455783`

C. Owen, D. Grove, T. Newby, A. Murray, C. North, and M. Pope. 2011. PRISM: Program Replication and Integration for Seamless MILS. In *IEEE Symposium on Security and Privacy*. IEEE, Washington, DC, USA, 281–296. `DOI:http://dx.doi.org/10.1109/SP.2011.15`

Vasilis Pappas, Michalis Polychronakis, and Angelos D. Keromytis. 2012. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization. In *IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer Society, Washington, DC, USA, 601–615. `DOI:http://dx.doi.org/10.1109/SP.2012.41`

Bryan Parno, Jonathan M. McCune, Dan Wendlandt, David G. Andersen, and Adrian Perrig. 2009. CLAMP: Practical Prevention of Large-Scale Data Leaks. In *IEEE Symposium on Security and Privacy (SP '09)*. IEEE Computer Society, Washington, DC, USA, 154–169. `DOI:http://dx.doi.org/10.1109/SP.2009.21`

839 Mathias Payer, Tobias Hartmann, and Thomas R. Gross. 2012. Safe Loading - A Foundation for Secure
840   Execution of Untrusted Programs. In *IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer
841   Society, Washington, DC, USA, 18–32. `DOI:http://dx.doi.org/10.1109/SP.2012.11`

842 Bryan D. Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. 2008. Lares: An Architecture for Secure
843   Active Monitoring Using Virtualization. In *IEEE Symposium on Security and Privacy (SP '08)*. IEEE
844   Computer Society, Washington, DC, USA, 233–247. `DOI:http://dx.doi.org/10.1109/SP.2008.24`

845 Nick L. Petroni, Jr. and Michael Hicks. 2007. Automated Detection of Persistent Kernel Control-flow Attacks.
846   In *ACM Conference on Computer and Communications Security (CCS) (CCS '07)*. ACM, New York, NY,
847   USA, 103–115. `DOI:http://dx.doi.org/10.1145/1315245.1315260`

848 Joe Gibbs Politz, Spiridon Aristides Eliopoulos, Arjun Guha, and Shriram Krishnamurthi. 2011. ADsafety:
849   Type-based Verification of JavaScript Sandboxing. In *USENIX Security (SEC'11)*. USENIX Association,
850   Berkeley, CA, USA, 12–12. `http://dl.acm.org/citation.cfm?id=2028067.2028079`

851 Niels Provos. 2003. Improving Host Security with System Call Policies. In *USENIX Security (SSYM'03)*.
852   USENIX Association, Berkeley, CA, USA, 18–18. `http://dl.acm.org/citation.cfm?id=1251353.`
853   `1251371`

854 Cynthia D. Mulrow R. Brian Haynes, Douglas G. Altman Edward J. Huth, and Martin J. Gardner. 1990.
855   More Informative Abstracts Revisited. *Annals of Internal Medicine* 113, 1 (1990), 69–76. `DOI:http:`
856   `//dx.doi.org/10.7326/0003-4819-113-1-69`

857 R Core Team. 2014. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical
858   Computing, Vienna, Austria. `http://www.R-project.org/`

859 Charles Reis, John Dunagan, Helen J. Wang, Opher Dubrovsky, and Saher Esmeir. 2006. BrowserShield:
860   Vulnerability-driven Filtering of Dynamic HTML. In *USENIX Symposium on Operating Systems Design*
861   *and Implementation (OSDI) (OSDI '06)*. USENIX Association, Berkeley, CA, USA, 61–74. `http://dl.`
862   `acm.org/citation.cfm?id=1298455.1298462`

863 Michael F. Ringenburg and Dan Grossman. 2005. Preventing Format-string Attacks via Automatic and
864   Efficient Dynamic Checking. In *ACM Conference on Computer and Communications Security (CCS)*
865   *(CCS '05)*. ACM, New York, NY, USA, 354–363. `DOI:http://dx.doi.org/10.1145/1102120.1102166`

866 Benjamin D. Rodes, John C. Knight, Anh Nguyen-Tuong, Jason D. Hiser, Michele Co, and Jack W. Davidson.
867   2015. A Case Study of Security Case Development. In *Engineering Systems for Safety*. Safety-Critical
868   Systems Club, Newcastle upon Tyne, UK, 187–204.

869 Dan Rosenberg. 2012. Poking holes in AppArmor Profiles. `http://blog.azimuthsecurity.com/2012/09/`
870   `poking-holes-in-apparmor-profiles.html`. (2012). Accessed: 2015-04-30.

871 Prateek Saxena, David Molnar, and Benjamin Livshits. 2011. SCRIPTGARD: Automatic Context-sensitive
872   Sanitization for Large-scale Legacy Web Applications. In *ACM Conference on Computer and Communi-*
873   *cations Security (CCS) (CCS '11)*. ACM, New York, NY, USA, 601–614. `DOI:http://dx.doi.org/10.`
874   `1145/2046707.2046776`

875 Fred B. Schneider. 1997. Towards Fault-Tolerant and Secure Agentry. In *International Workshop on*
876   *Distributed Algorithms (WDAG '97)*. Springer-Verlag, London, UK, UK, 1–14. `http://dl.acm.org/`
877   `citation.cfm?id=645954.675785`

878 Margrit Schreier. 2012. *Qualitative Content Analysis in Practice* (1st ed.). SAGE Publications Ltd, Thousand
879   Oaks, CA, USA.

880 Z. Cliffe Schreuders, Tanya McGill, and Christian Payne. 2013. The state of the art of application restrictions
881   and sandboxes: A survey of application-oriented access controls and their shortfalls. *Computers & Security*
882   32 (Feb. 2013), 219–241. `DOI:http://dx.doi.org/10.1016/j.cose.2012.09.007`

883 David Sehr, Robert Muth, Cliff Biffle, Victor Khimenko, Egor Pasko, Karl Schimpf, Bennet Yee, and Brad
884 Chen. 2010. Adapting Software Fault Isolation to Contemporary CPU Architectures. In *USENIX Security*
885 *(USENIX Security'10)*. USENIX Association, Berkeley, CA, USA, 1–1. `http://dl.acm.org/citation.`
886 `cfm?id=1929820.1929822`

887 Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. 2007. SecVisor: A Tiny Hypervisor to Provide
888 Lifetime Kernel Code Integrity for Commodity OSes. In *ACM Symposium on Operating Systems Principles*
889 *(SOSP) (SOSP '07)*. ACM, New York, NY, USA, 335–350. `DOI:http://dx.doi.org/10.1145/1294261.`
890 `1294294`

891 Joseph Siefers, Gang Tan, and Greg Morrisett. 2010. Robusta: Taming the Native Beast of the JVM. In
892 *ACM Conference on Computer and Communications Security (CCS) (CCS '10)*. ACM, New York, NY,
893 USA, 201–211. `DOI:http://dx.doi.org/10.1145/1866307.1866331`

894 C. Spearman. 1904. The Proof and Measurement of Association Between Two Things. *American Journal*
895 *of Psychology* 15 (1904), 88–103.

896 J. Stuckman and J. Purtilo. 2014. Mining Security Vulnerabilities from Linux Distribution Metadata. In
897 *Software Reliability Engineering Workshops (ISSREW)*. IEEE, Washington, DC, USA, 323–328.

898 Weiqing Sun, R. Sekar, Gaurav Poothia, and Tejas Karandikar. 2008. Practical Proactive Integrity Preserva-
899 tion: A Basis for Malware Defense. In *IEEE Symposium on Security and Privacy (SP '08)*. IEEE Computer
900 Society, Washington, DC, USA, 248–262. `DOI:http://dx.doi.org/10.1109/SP.2008.35`

901 Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. Eternal War in Memory. In *IEEE Symposium*
902 *on Security and Privacy (SP '13)*. IEEE Computer Society, Washington, DC, USA, 48–62. `DOI:http:`
903 `//dx.doi.org/10.1109/SP.2013.13`

904 Jack Tang. 2015. Exploring Control Flow Guard in Windows 10. `http://blog.trendmicro.com/`
905 `trendlabs-security-intelligence/exploring-control-flow-guard-in-windows-10/`. (2015). Ac-
906 cessed: 2015-09-10.

907 Shuo Tang, Haohui Mai, and Samuel T. King. 2010. Trust and Protection in the Illinois Browser Op-
908 erating System. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*
909 *(OSDI'10)*. USENIX Association, Berkeley, CA, USA, 1–8. `http://dl.acm.org/citation.cfm?id=`
910 `1924943.1924945`

911 Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. 1993. Efficient software-based
912 fault isolation. *SIGOPS Oper. Syst. Rev.* 27, 5 (Dec. 1993), 203–216. `DOI:http://dx.doi.org/10.1145/`
913 `173668.168635`

914 Dan S. Wallach, Dirk Balfanz, Drew Dean, and Edward W. Felten. 1997. Extensible Security Architectures
915 for Java. In *Symposium on Operating Systems Principles (SOSP '97)*. ACM, New York, NY, USA, 116–128.
916 `DOI:http://dx.doi.org/10.1145/268998.266668`

917 Helen J. Wang, Xiaofeng Fan, Jon Howell, and Collin Jackson. 2007. Protection and Communication Ab-
918 stractions for Web Browsers in MashupOS. In *ACM Symposium on Operating Systems Principles (SOSP)*
919 *(SOSP '07)*. ACM, New York, NY, USA, 1–16. `DOI:http://dx.doi.org/10.1145/1294261.1294263`

920 Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning. 2009. Countering Kernel Rootkits with Lightweight
921 Hook Protection. In *ACM Conference on Computer and Communications Security (CCS) (CCS '09)*.
922 ACM, New York, NY, USA, 545–554. `DOI:http://dx.doi.org/10.1145/1653662.1653728`

923 Richard Wartell, Vishwath Mohan, Kevin W. Hamlen, and Zhiqiang Lin. 2012. Binary Stirring: Self-
924 randomizing Instruction Addresses of Legacy x86 Binary Code. In *ACM Conference on Computer and*
925 *Communications Security (CCS) (CCS '12)*. ACM, New York, NY, USA, 157–168. `DOI:http://dx.doi.`
926 `org/10.1145/2382196.2382216`

Charles B. Weinstock, Howard F. Lipson, and John Goodenough. 2007. *Arguing Security: Creating Security Assurance Cases*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

Dan Williams, Patrick Reynolds, Kevin Walsh, Emin Gün Sirer, and Fred B. Schneider. 2008. Device Driver Safety Through a Reference Validation Mechanism. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI'08)*. USENIX Association, Berkeley, CA, USA, 241–254. `http://dl.acm.org/citation.cfm?id=1855741.1855758`

Glenn Wurster and Paul C. van Oorschot. 2010. A Control Point for Reducing Root Abuse of File-system Privileges. In *ACM Conference on Computer and Communications Security (CCS) (CCS '10)*. ACM, New York, NY, USA, 224–236. `DOI:http://dx.doi.org/10.1145/1866307.1866333`

Rubin Xu, Hassen Saïdi, and Ross Anderson. 2012. Aurasium: Practical Policy Enforcement for Android Applications. In *USENIX Security (Security'12)*. USENIX Association, Berkeley, CA, USA, 27–27. `http://dl.acm.org/citation.cfm?id=2362793.2362820`

Yan Yan, Massimilliano Menarini, and William Griswold. 2014. Mining Software Contracts for Software Evolution. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Washington, DC, USA, 471–475.

B. Yee, D. Sehr, G. Dardyk, J.B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. 2009. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *IEEE Symposium on Security and Privacy*. IEEE, Washington, DC, USA, 79–93. `DOI:http://dx.doi.org/10.1109/SP.2009.25`

Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazieres. 2006. Making Information Flow Explicit in HiStar. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI) (OSDI '06)*. USENIX Association, Berkeley, CA, USA, 263–278. `http://dl.acm.org/citation.cfm?id=1298455.1298481`

Bin Zeng, Gang Tan, and Úlfar Erlingsson. 2013. Strato: A Retargetable Framework for Low-level Inlined-reference Monitors. In *USENIX Security (SEC'13)*. USENIX Association, Berkeley, CA, USA, 369–382. `http://dl.acm.org/citation.cfm?id=2534766.2534798`

Bin Zeng, Gang Tan, and Greg Morrisett. 2011. Combining Control-flow Integrity and Static Analysis for Efficient and Validated Data Sandboxing. In *ACM Conference on Computer and Communications Security (CCS) (CCS '11)*. ACM, New York, NY, USA, 29–40. `DOI:http://dx.doi.org/10.1145/2046707.2046713`

Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, L. Szekeres, S. McCamant, D. Song, and Wei Zou. 2013. Practical Control Flow Integrity and Randomization for Binary Executables. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, Washington, DC, USA, 559–573. `DOI:http://dx.doi.org/10.1109/SP.2013.44`

Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. 2011. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *ACM Symposium on Operating Systems Principles (SOSP) (SOSP '11)*. ACM, New York, NY, USA, 203–216. `DOI:http://dx.doi.org/10.1145/2043556.2043576`

Mingwei Zhang and R. Sekar. 2013. Control Flow Integrity for COTS Binaries. In *USENIX Security (SEC'13)*. USENIX Association, Berkeley, CA, USA, 337–352. `http://dl.acm.org/citation.cfm?id=2534766.2534796`

Qun Zhong, Nigel Edwards, and Owen Rees. 1997. *Operating System Support for the Sandbox Method and Its Application on Mobile Code Security*. Technical Report HPL-97-153. HP Laboratories Bristol. `http://www.hpl.hp.com/techreports/97/HPL-97-153.pdf`