ID, CONF 20XX -

Where is the policy enforced? --

When is the policy imposed? --

What is protected by the policy? (fine grained) --

What is protected by the policy? (coarse grained) --

Requirements of the person applying the sandbox --

Requirements of the application --

Security Policy Type --

Policy enforcements place in kill chain --

Policy Management --

Policy Construction --

Validation Claim --

Validation --

KCoFI, Oakland 2014 -

Where is the policy enforced? -- System:
"KCoFI protects commodity operating systems from classical control- flow hijack attacks, return-to-user attacks, and code segment modification attacks."

When is the policy imposed? -- Hybrid:
"KCoFI has several unique requirements. First, it must instrument commodity OS kernel code; existing CFI enforcement mechanisms use either compiler or binary instrumentation [4], [10], [18]. Second, KCoFI must understand how and when OS kernel code interacts with the hardware. For example, it must understand when the OS is modifying hardware page tables in order to prevent errors like writeable

and executable memory. Third, KCoFI must be able to control modification of interrupted program state in order to prevent ret2usr attacks.”

What is protected by the policy? (fine grained) -- Code instructions:
See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- System Level Component:
See Where is the policy enforced?

Requirements of the person applying the sandbox -- Install a tool:
“The SVA-OS instructions described later in this section are implemented as a run-time library that is linked into the kernel.”

Requirements of the application -- Use special compiler:
“All software, including the operating system and/or hypervisor, is compiled to the virtual instruction set that SVA provides.”

Requirements of the application -- Use sandbox as framework/library:
“Because the operating system must interface with the hardware via the SVA-OS instructions, it must be ported to the SVA virtual instruction set. This is similar to porting the operating system to a new architecture, but a relatively simple virtual architecture, and only requires modifying the lowest-level parts of the kernel. No reorganization of the kernel or modifications to drivers are needed.”

Security Policy Type -- Fixed policy:
See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit:
See Where is the policy imposed?

Policy Management -- No management:
See Where is the policy enforced?

Policy Construction --Encoded in the logic of the sandbox:
See Where is the policy enforced?

Validation -- Case studies (Performance)
“Since network applications make heavy use of operating system services, we measured the performance of the thttpd web server and the remote secure login sshd server.”

Validation -- Benchmark suite (Performance)

“To measure file system performance, we used the Postmark benchmark [27]. We used the LMBench microbenchmarks [28] to measure the performance of individual system calls.”

Validation -- Benchmark suite (Security)
“We evaluate the security of our system for the FreeBSD 9.0 kernel on the x86-64 architecture. We find that all the Return Oriented Programming (ROP) gadgets found by the ROPGadget tool [14] become unusable as branch targets. We also find that our system reduces the average number of possible indirect branch targets by 98.18%”

Validation -- Proof (Security):
“To verify that our design correctly enforces control-flow integrity, we have built a formal model of key features of our system (including the new protections for OS operations) using small-step semantics and provided a partial proof that our design enforces control-flow integrity. The proofs are encoded in the Coq proof system and are mechanically verified by Coq.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

NOTE: There are some recent FreeBSD mailing list posts from the author discussing releasing the source code to FreeBSD, but this does not happened at the time of this writing.

Pivot, Oakland 2014 -

Where is the policy enforced? -- Application:
“Pivot is a new JavaScript isolation framework for web applications.”

When is the policy imposed? -- Hybrid:
“Pivot rewrites each domain’s JavaScript code, combining all of the code in a frame into a single generator function [9]. In contrast to a normal function, which uses the return statement to return a single value per function execution, and which loses its activation record upon returning, a generator uses the yield statement to remember its state across invocations. A generator can yield different values after each invocation. Figure 2 provides a simple example of a generator function that returns the factorial sequence. A rewritten Pivot frame is a generator function that

starts execution when invoked by the local Pivot library, and yields to that library upon invoking an RPC."

What is protected by the policy? (fine grained) -- Data:
"Browsers give each iframe a separate JavaScript runtime; each runtime has distinct global variables, heap objects, visual display areas, and so on. Iframes from different origins cannot directly manipulate each other's state—instead, they must communicate using the asynchronous, pass-by-value postMessage() call."

What is protected by the policy? (fine grained) -- Code/Instructions:
See Data above.

What is protected by the policy? (coarse grained) -- Targeted Application:
See What is protected by the policy?

Requirements of the person applying the sandbox -- Write a security policy (optional to weaken isolation):
"Using that library, a satellite can register one or more public RPC interfaces with the Pivot master frame."

Requirements of the application -- Use special compiler:
"By rewriting JavaScript call sites, Pivot can detect RPC invocations."

Requirements of the application -- Use sandbox as framework/library:
See Requirements of the person applying the sandbox -- write a security policy

Security Policy Type -- User defined policy
See Requirements of the person applying the sandbox -- write a security policy

Policy enforcements place in kill chain -- pre-exploit
See What is protected by the policy? (fine grained) -- Data

Policy Management -- No management
No quote.

Policy Construction -- Manually written policies
See Requirements of the person applying the sandbox -- write a security policy

Validation Claim -- Security
"Furthermore, since Pivot uses iframes as isolation containers, it allows the safe composition of rewritten code with unrewritten code."

“Since each satellite frame contains an isolated JavaScript runtime, Pivot allows untrusted satellite code to use the full JavaScript language, including powerful functions like eval().”

Validation Claim -- Applicability
“Since each satellite frame contains an isolated JavaScript runtime, Pivot allows untrusted satellite code to use the full JavaScript language, including powerful functions like eval().”

Validation -- Argumentation (Security):
“If two frames belong to different origins, cross-frame interactions are restricted to the postMessage() API, which asynchronously transfers immutable strings.”
“Pivot relies on the browser to enforce memory isolation between the trusted master frame and the untrusted satellite frames. However, nothing prevents a satellite from trying to subvert the Pivot infrastructure within its own frame. For example, a satellite can directly generate RPC requests by crafting its own postMessage() calls. A satellite can also try to attack Pivot’s virtualized event framework, e.g., by looking for baroque JavaScript aliases to the underlying non-virtualized functions [19]. ”

Validation -- Case Studies (Applicability):
“Using an empirical analysis of JavaScript call graphs in real web applications, we demonstrate that the vast majority of function calls do not cross library boundaries. Thus, Pivot’s trusted master/untrusted satellite decomposition is a natural one [...]
To determine how often web pages make cross-library function calls, we visited the top 20 websites in the United States as determined by Alexa.”

Validation -- Case Studies (Performance)
“To test the end-to-end latencies of cross-domain RPCs, we built a mashup application that integrated three untrusted JavaScript libraries. We picked these particular libraries because they were used to evaluate Jigsaw in the original Jigsaw paper.”

Validation -- Public Data (Applicability):

See other validation quotes.

Availability -- Not Available:

No mention in paper

CCFIR, Oakland 2013 -

Where is the policy enforced? -- Application
"We build CCFIR as a purely binary transformation."

When is the policy imposed? -- Hybrid
"We identify the valid targets in binary modules and rewrite them so that the valid targets can be distinguished from invalid ones efficiently. Then we insert checks before each indirect control transfer instruction to make this distinction."

What is protected by the policy? (fine grained) -- Code instructions
"CCFIR enforces a policy on indirect control transfers that prevents jumps to any but a white-list of locations; it also distinguishes between calls and returns, and prevents unauthorized returns into sensitive functions."

What is protected by the policy? (coarse grained) -- Targeted Application
See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a tool
See When is the policy imposed?

Requirements of the application -- Have compiler introduced metadata:
"[CCFIR] analyzes binary executables based on relocation tables"

Security Policy Type -- Fixed policy
See What is protected by the policy (fine grained)

Policy enforcements place in kill chain -- Pre-exploit
See What is protected by the policy? (fine grained)

Policy Management -- No management
See What is protected by the policy? (fine grained)

Policy Construction --Encoded in the logic of the sandbox
See What is protected by the policy? (fine grained)

Validation claims -- Security:
"CCFIR-hardened versions of IE6, Firefox 3.6 and other applications are protected effectively."

Validation -- Benchmark suite (Performance):

“The execution time overhead of this checking is low, 3.6%/8.6% (average/max) over SPECint2000”

Validation -- Case studies (Security):
“We also chose 10 publicly available exploits from Metasploit [53] against FF3, IE6 and 5 other applications. These experiments are performed in a virtual machine running Windows XP SP3 within a separate experiment network. Table III shows the 10 vulnerabilities attacked by exploits we used.
Taking CVE-2011-0065 as an example, this vulnerability exists in Firefox 3.x before 3.6.17. It is a use-after-free vulnerability which can cause arbitrary code execution, when exploited by techniques such as heap spray [54].
After hardening the vulnerable module xul.dll with CCFIR, we drive Firefox to access the attack URL again, and the error handler added by CCFIR is triggered. The remaining 9 exploits, which target IE6 and other 5 applications, are also prevented by CCFIR in a similar manner.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

SafeLoading, Oakland 2012 -

Where is the policy enforced? -- System:
“The approach replaces the standard loader with a security-aware trusted loader.”

When is the policy imposed? -- Dynamic:
“No static modifications or static analysis are needed to execute an application in the sandbox.”

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:
See Table 5.

What is protected by the policy? (coarse grained) -- Class of Applications:
“The Trusted Runtime Environment (TRuE) places the loader in the trusted computing base and runs all application code in a sandbox.”

Requirements of the person applying the sandbox -- Install a tool:

See What is protected by the policy? (coarse grained)

NOTE: This paper extends libdetox which is first written about in “Fine-grained user-space security through virtualization”. From that paper, “flexible per-process user-defined policy-based system call interposition in user-space without the need for context switches to validate specific calls and without additional privileged code in the kernel”. The user can optionally write a policy, but we do not code that for this paper because the syscall policy enforcement is not a novel contribution of this paper (i.e. SafeLoading get it for free by using libdetox).

Requirements of the application -- Other:
“Self-modifying code in the application is not supported, i.e., the application is not allowed to generate new code at runtime. Code can only be added to the runtime image of an application through the secure loader API.”

Security Policy Type -- Fixed:
“The combination of the secure loader and the user-space sandbox enables the safe execution of untrusted code in user- space. Code injection attacks are stopped before any unintended code is executed. Furthermore, additional information provided by the loader can be used to support additional security properties, e.g., inlining of Procedure Linkage Table calls reduces the number of indirect control flow transfers and therefore limits jump-oriented attacks.”

Policy enforcements place in kill chain -- Pre:
“The user-space sandbox builds on the secure loader and subsequently dynamically checks for malicious code and ensures that all control flow instructions of the application adhere to an execution model.”

Policy Management -- None
Fixed policy.

Policy Construction -- Encoded in the logic of the sandbox:
See Security Policy Type.

Validation -- Argumentation (Security):
“Discussion of TRuE’s security features”

Validation -- Benchmark Suite (Performance):
“We use the SPEC CPU2006 benchmarks version 1.0.1 to evaluate the performance and feasibility of our prototype implementation.”

Validation -- Case study (Performance):
“We measured OpenOffice startup as a stress test and worst-performance metric, 145 DSOs are loaded, relocated, and executed with very low code reuse.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Source Code:

“The source code of the prototype implementation of TRuE is available as open-source at http://nebelwelt.net/projects/TRuE.”

ILR, Oakland 2012 -

Where is the policy enforced? -- Application:
“ILR randomizes the location of every instruction in a program, thwarting an attacker’s ability to re-use program functionality (e.g., arc-injection attacks and return-oriented programming attacks).”

When is the policy imposed? -- Static:
“ILR adopts an execution model where each instruction has an explicitly specified successor. Thus, each instruction’s successor is independent of its location. This model of execution allows instructions to be randomly scattered throughout the memory space. Hiding the explicit successor information prevents an attacker from predicting the location of an instruction based on the location of another instruction.”
*Note: The security policy is imposed entirely statically. However, ILR requires a runtime component (PVM) for the program to execute.

What is protected by the policy? (fine grained) -- Code/Instructions:
See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Target Application:
See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a tool:
“The compiler and the linker collaborate to produce an executable file where instructions are laid out so they can be loaded into memory when the program is executed.”

Requirements of the application -- No additional requirements:
"ILR has an offline analysis phase to relocate instructions in the binary and generate a set of rewriting rules that describe how and where the newly located instructions are to be executed, and how control should flow between them, (shown as the fallthrough map in Figure 1). The randomized program is executed on the native hardware by a PVM that uses the fallthrough map to guide execution."

Security Policy Type -- Fixed policy:
See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit:
See Where is the policy enforced?

Policy Management -- No management
Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:
See Where is the policy enforced?

Validation -- Benchmark suite (Performance):
"We evaluated the effectiveness and performance of the ILR prototype using the SPEC CPU2006 benchmark suite."

Validation -- Case studies (Security):
"To verify that our technique stops attacks that are successful against ASLR and W⊕X protected systems, we performed a number of tests on vulnerable programs."

Validation -- Public Data (Performance):

See other validation quotes. NOTE: They did a security experiment with Adobe Reader, but the results of this experiment are not reasonably comparable.

Availability -- Not Available:

No mention in paper

Pappas, Oakland 2012 -

Where is the policy enforced? -- Application:

“Our approach is based on narrow-scope modifications in the code segments of executables using an array of code transformation techniques, to which we collectively refer as in-place code randomization. These transformations are applied statically, in a conservative manner, and modify only the code that can be safely extracted from compiled binaries, without relying on symbolic debugging information.”

When is the policy imposed? -- Static:
See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/instructions:
“We present a novel code randomization method that can harden third- party applications against return-oriented programming.”

What is protected by the policy? (coarse grained) -- Target application:
See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a tool:
See Where is the policy enforced?

Requirements of the application -- None:
See Where is the policy enforced?

Security Policy Type -- Fixed:
See What is protected by the policy (fine grained)?

Policy enforcements place in kill chain -- Pre-exploit
See Where is the policy enforced?

Policy Management --
Fixed policy

Policy Construction -- Encoded in the logic of the Sandbox:
Where is the policy enforced? -- Application

Validation -- Analytical Analysis (Security):
“We provide a detailed analysis of how in-place code randomization affects available gadgets using a large set of 5,235 PE files. On average, the applied transformations effectively eliminate about 10%, and probabilistically break about 80% of the gadgets in the tested files.”

Validation -- Case Studies (Security):

“We evaluated the effectiveness of in-place code randomization using publicly available ROP exploits against vulnerable Windows applications [53], [62], [63], as well as generic ROP payloads based on commonly used DLLs [64], [65] ... From these gadgets, in-place code randomization can alter six of them: one gadget is completely eliminated, while the other five broken gadgets have 2, 2, 3, 4, and 6 possible states, respectively, resulting to a total of 287 randomized states (*in addition* to the always eliminated gadget, which also alone breaks the ROP code).”

Validation -- Case Studies (Performance):
“We took advantage of the extensive and diverse code execution coverage of this experiment to also evaluate the impact of in-place code randomization to the runtime performance of the modified code. Among the different code transformations, instruction reordering is the only one that could potentially introduce some non-negligible overhead, given that sometimes the chosen ordering may be sub-optimal. We measured the overall CPU user time for the completion of all tests by taking the average time across multiple runs, using both the original and the randomized versions of the DLLs. In all cases, there was no observable difference in the two times, within measurement error.”

Validation -- Public Data (Security, Performance):

See other validation quotes. Uses Wine tests for performance

Availability -- Source Code:

“Our prototype implementation is publicly available at http://nsl.cs.columbia.edu/projects/orp”

TxBox, Oakland 2011 -

Where is the policy enforced? -- System:

“TxBox consists of a relatively simple, policy-agnostic security monitor running in the OS kernel and a user-level policy manager.”

When is the policy imposted? -- Dynamic:

“Our prototype system, TxBox, uses transactions for (1) speculative execution of untrusted applications, (2) uncircumventable enforcement of system-call policies, and (3) automatic recovery from the effects of malicious execution.”

What is protected by the policy? (Fine Grained) -- Files, Communication, User Data:

“All changes made by the violating program to the file system effectively disappear, child processes are stopped, and buffered local inter-process communication is canceled, leaving concurrent updates made by other programs undisturbed.”
“TxBox supports two types of system objects in policies: inodes (of directories or files) and sockets.”

What is protected by the policy? (Coarse Grained) -- Targeted Application or Classes of Applications (those in a specific path) depending on configuration:

“The administrator can specify either the sandboxed process when installing a policy, or a path and a list of events. In the latter case, the policy manager will automatically associate the policy with any program residing on that path.”

Requirements of the person applying the sandbox -- Write a Security Policy:

See Security Policy Type.

Requirements of the application -- No Additional Requirements:

No requirements stated.

Security Policy Type -- User-Defined Policy:

“The system administrator uses the policy manager to define the policy as a set of regular expressions over system call names and arguments and system objects such as inodes and socket descriptors.”

Policy enforcements place in kill chain -- Post-exploit:

“When TxBox detects a policy violation, the transaction is aborted and the system automatically reverts to a good local state (except for the effects of previously allowed external I/O in certain enforcement regimes—see Section V-D).”

Policy Management -- No Management:

No policy management mechanism as defined by this category is mentioned. It’s clear that TxBox does allow policies to be shared manually as policies are written down and can be moved from system to system.

Policy Construction: Manually Written Policy

See Security Policy Type.

Validation Claim -- Security

“TxBox cannot be circumvented by a sandboxed process. Its kernel-based enforcement mechanism prevents exploitation of incorrect mirroring of the kernel state, TOCTTOU races, and/or other semantic gaps between the security monitor and the OS [21, 58].”
“If the monitor determines later that the program has violated a security policy, it aborts the transaction and the system is automatically rolled back to a benign state.”

Validation Claim -- Performance

“On realistic workloads, the performance overhead of TxBox is less than 20% including the cost of supporting transactions and less than 5% over untrusted execution in a transactional OS.”

Validation Claim -- Applicability (Policies are expressive)/Validation -- Case Study:

“TxBox can enforce a rich class of practical security policies.”
“We downloaded the source code of the vim editor and configured it to install in /usr/local and compile using make. Next, we ran “make install” in a sandbox with the BLACKLIST WREGEX *I:164564* policy where 164564 is the inode number of the directory /usr/local/bin.”

NOTE: There are four other case studies with similar quotes.

Validation -- Case Study (Performance):

Scalability: “To evaluate the scalability of TxBox, we built a simple application which opens 100 existing files and measured how its runtime varies with the increase in the size of the policy (the number of inodes included in the policy).”
“To measure the performance implications of multiple invocations of the policy decision engine on network-I/O-intensive workloads, we sandbox wget in TxBox and use it to download different large files from the Internet.”
“Micro-benchmarks. Table II shows the overhead of TxBox or individual system calls—including read, write, and fork/exec—compared to the base Linux kernel with and without Dazuko.”

"For gzip, which does not involve many file-system operations, the overhead of TxBox is negligible (1.007×). For make, which involves more file-system operations than gzip, the overhead is 1.18×. On the other hand, PostMark benchmark involves a large number of file-system operations and represents the worst-case scenario for TxBox because it requires a large number of shadow objects to be created."
"When system transactions are used for sandboxing, the overhead for trusted, non-sandboxed applications is modest. The average non-transactional overhead is 44% at the scale of a single system call on TxBox, using the same microbenchmark described in [48]."

Validation -- Case Study (Security):

"To simulate the effect of a malicious multimedia converter trying to write to unrelated files in a user's home directory, we configured ffmpeg, a popular open-source codec, to create output files in the /home/user1/ directory."

Validation -- Argumentation (Security):

"By design, TxBox is immune to TOCTTOU attacks. Sandboxed processes run inside separate transactions, so changes made by one of them to their shared state will not be visible to the other until the transaction commits."
"Policy enforcement in TxBox is performed by inspecting objects in the workset and thus cannot be evaded by splitting updates between the parent and the child."
"Rather than construct an error-prone mapping of system-call arguments or hardware-level events to OS state changes, TxBox directly inspects pending changes to kernel state made by the transaction wrapping the sandboxed process and its children."
Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

PRISM, Oakland 2011 -

Where is the policy enforced? -- System:
See Figure 1.

When is the policy imposed? -- Dynamic:
"The untrusted COTS programs in the SLS partitions send at-level file edits as *diff* transactions to the TCB. The TCB *verifies* that BLP semantics will be observed and then *patches* these transactions into its canonical representation of the file."

What is protected by the policy? (fine grained) -- Files:
See When is the policy imposed?

What is protected by the policy? (coarse grained) -- Targeted Application:
"We describe how to combine a minimal Trusted Computing Base (TCB) with polyinstantiated and slightly augmented Commercial Off The Shelf (COTS) software programs in separate Single Level Secure (SLS) partitions to create Multi Level Secure (MLS) applications."

Requirements of the person applying the sandbox -- Install a tool:
"The untrusted COTS applications in the separate system-high partitions are augmented with untrusted PRISM add-in modules."

Requirements of the application -- Other:
"The application-specific add-in fulfills two functions. It translates incoming at-level canonical MLSDoc files into an application readable format; and translates at-level edits into outgoing MLSDiff patches."

Security Policy Type -- Fixed:
"These MLS applications can coordinate fine grained (intra-document) Bell LaPadula (BLP) [6] separation between information at multiple security levels."

Policy enforcements place in kill chain -- Pre:
See When is the Policy Imposed?

Policy Management -- None:
Fixed policy.

Policy Construction -- Encoded in Logic of the Sandbox
See When is the Policy Imposed?

Validation -- Case Studies (Applicability):
"We demonstrate the utility of this approach using Microsoft Word and DokuWiki."

Validation -- Security (Argumentation):
"We finalise our discussion on security properties by arguing the high tractability of certifying products based on our architecture."

Validation -- Security (Analytical Analysis):
"The potential capacity of such a channel can be approximated by modelling it as a discrete noiseless channel driven by a Markov process."

Validation -- Public Data (Applicability):

See other validation quotes.

Availability -- Not Available:

No mention in paper

~~IBEX, Oakland 2011 -~~

~~**Where is the policy enforced? --**~~

~~**When is the policy imposed? -- Static:**~~
~~"We develop a methodology based on refinement typing (proven sound) to verify that extensions written in Fine [30], a dependently typed ML dialect, satisfies our safety condition."~~

~~**What is protected by the policy? (fine grained) --**~~

~~**What is protected by the policy? (coarse grained) --**~~

~~**Requirements of the person applying the sandbox --**~~

~~**Requirements of the application -- Use sandbox as framework/library**~~
~~"We provide developers with an API that exposes core browser functionality to extensions. We expect programmers to write extensions in high- level, type-safe languages that are amenable to formal analysis."~~

~~**Security Policy Type -- User-defined policy:**~~
~~"Our language, based on Datalog, allows the specification of fine-grained authorization and data flow policies on web content and browser state accessible by extensions."~~

~~**Policy enforcements place in kill chain --**~~

~~**Policy Management --**~~

~~Policy Construction --~~

~~Validation Claim --~~

~~Validation --~~

Note: IBEX excluded because this is a framework for creating secure-by-construction browser extensions, not for sandboxing extensions.

ConScript, Oakland 2010 -

Where is the policy enforced? -- Application host:
“CONSCRIPT, a browser-based aspect system for security proposed in this paper, focuses on empowering the hosting page to carefully constrain the code it executes.”
Note: The application host is a the web application hosting third-party code.

When is the policy imposed? -- Dynamic:
“We modified the execution of a user-defined function to first check whether advice was registered and enabled. If so, execution proceeds by running the advice function.”

What is protected by the policy? (fine grained) -- Communication, Code/Instructions, User Data:
“We present 17 wide-ranging security and reliability policies. We show how to concisely express these policies in CONSCRIPT, often with only several lines of JavaScript code. These policies fall into the broad categories of con- trolling script introduction, imposing communication restrictions, limiting dangerous DOM interactions, and restricting API use.”

What is protected by the policy? (coarse grained) -- Class of Applications
See Where is the policy enforced?
*Note: all code loaded inside the host application is sandboxed

Requirements of the person applying the sandbox -- Select a pre-made security policy:
See Where is the policy enforced?
*Note: also possible to write a security policy, but this is not the expected mode of use

Requirements of the application -- None:

“Next in this “bootup sequence”, advice registration is performed. An appropriate analogy here is that advice is “kernel-level”, trusted code. Advice can be registered by the hosting page, which may subsequently proceed to load third- party, potentially untrusted JavaScript. However, the subse- quent script’s execution will be restricted through advice registered by the hosting page.”

Security Policy Type -- User defined policy
See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit:
See When is the policy imposed?

Policy Management -- No management:
“In CONSCRIPT, this kind of behavior augmentation is done via the script include tag to provide a policy as follows:
<SCRIPT SRC="script.js" POLICY="function () {...}">”
*Note: paper includes the full definition of 17 widely applicable policies. In some sense the paper is a “central policy repository.”

Policy Construction --Manually written policies:
See Where is the policy enforced?

Validation -- Case studies (performance):
“Our primary focus is on the runtime overhead introduced with CONSCRIPT instru- mentation compared to alternative techniques. Section VII-A talks about our experimental setup. Section VII-B evaluates micro-benchmarks and Section VII-C focuses on applying CONSCRIPT advice to large AJAX sites and applications such as MSN, GMail, and Live Desktop.”

Validation -- Argumentation (security):
“In this section, we consider attacks against CONSCRIPT advice policies. Auditing policies published by other researchers, we found that they are quite tricky to get right. This is true even for policies consisting of only a few lines of JavaScript [2, 3]. While the idea of aspects is by no means new [15], in an adversarial environment, aspects are subject to a host of difficult issues.
In our attack model, we distinguish between kernel code (code loaded before an untrusted library) and user code (un- trusted libraries that may execute after the loading sequence). It is our intention to protect against advice tampering, i.e. user code that attempts to interfere with the way advice is applied and followed at runtime by tampering with code or data.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

TrustVisor, Oakland 2010 -

Where is the policy enforced? -- System:
“We develop a special-purpose hypervisor, called TrustVisor, designed to provide a measured, isolated execution environment for security-sensitive code modules without trusting the OS or the application that invokes the code module.”

When is the policy imposed? -- Dynamic:
“A DRTM-like mechanism provides the valuable security properties of a known-good initial state, memory protection from DMA accesses, and integrity measurement of the launched code before it executes.”

What is protected by the policy? (fine grained) -- code, data:
“Our goal is to provide data secrecy and integrity, as well as execution integrity for security-sensitive portions of an application, executing the code in isolation from the OS, untrusted application code, and system devices.”

What is protected by the policy? (coarse grained) -- Targeted Application:
See What is protected by the policy? (fine grained)

Requirements of the person applying the sandbox -- Install a tool:
See Where is the policy enforced?

Requirements of the application -- Annotated source code:
“A PAL is identified to Trust- Visor via a registration process that employs an application- level hypercall interface, with the PAL execution environment initialized by TrustVisor to a well-known, secure configuration.”

Security Policy Type -- Fixed policy:
“In secure guest mode, a PAL executes in isolation from the legacy OS and its applications.”

Policy enforcements place in kill chain -- Pre-exploit:
See When is the policy imposed?

Policy Management -- No management:
Fixed policy

Policy Construction -- Encoded in the logic of the sandbox:
See When is the policy imposed?

Validation -- Benchmark (Performance):
“We execute both compute-bound and I/O-bound applications with TrustVisor. For compute- bound applications, we use the SPECint 2006 suite. For I/O- bound applications, we select a range of benchmarks, including building the Linux kernel, Bonnie,6 Postmark [17], net-perf,7 and unmodified Apache web server performance.”

Validation -- Case studies (Performance):
“We evaluate the overhead when TrustVisor receives control in 5 cases (Tables 2 and 3): (a) when an application registers a PAL, (b) when any function inside the PAL is called, (c) when a function inside the PAL finishes execution and returns to the application, (d) when an application unregisters a PAL, and (e) when a PAL calls any μTPM function. We use microbenchmarks to measure the overhead of the TrustVisor framework in cases (a) – (d), and the overhead of μTPM operations provided by TrustVisor in case (e). We also evaluate the performance of real applications to illustrate the overall performance in a practical environment.”

Validation -- Argumentation (Security):
“The total size of TrustVisor implementation is 7889 lines of C and assembly code (the sum of the debug, initialization, and runtime code). The runtime TCB is about 6481 lines, which includes 3919 lines of RSA and other libraries. This is the full extent of the software TCB for TrustVisor, which places it within the reach of formal verification and manual audit techniques.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

~~HyperSafe, Oakland 2010 –~~

Excluded. Hypersafe provides control flow intergrity to Hypervisors by taking advantage of two techniques: non-bypassable memory lockdown and restricted

pointer indexing. The first of these techniques is more fundamental and their approach relies on manual editing the hypervisor code to make use of the technique:
"Non-bypassable memory lockdown [...] serves as the cornerstone for the entire scheme and enables the unique hypervisor self-protection."
"The first key technique is implemented by directly modifying the hypervisor source code while the second key technique is implemented as a compiler extension to the re-targetable."

NaCl, Oakland 2009 -

Where is the policy enforced? -- Application host:
"Native Client provides sandboxed execution of native code and portability across operating systems, delivering native code performance for the browser."

When is the policy imposed? -- Dynamic:
"Native Client is organized in two parts: a constrained execution environment for native code to prevent unintended side effects, and a runtime for hosting these native code extensions through which allowable side effects may occur safely."

What is protected by the policy? (fine grained) -- Memory, Code, User Data:
"To eliminate side effects the validator must address four subproblems: Data integrity: no loads or stores outside of data sandbox, reliable disassembly, no unsafe instructions, control flow integrity.

What is protected by the policy? (fine grained) -- Files:
"The service runtime also provides the common POSIX file I/O interface, used for operations on communications channels as well as web-based read-only content. As the name space of the local file system is not accessible to these interfaces, local side effects are not possible."

What is protected by the policy? (fine grained) -- Communication:
"To prevent unintended network access, network system calls such as connect() and accept() are simply omitted. NaCl modules can access the network via JavaScript in the browser. This access is subject to the same constraints that apply to other JavaScript access, with no net effect on network security."

What is protected by the policy? (coarse grained) -- Targeted Application:
See "Where is the policy enforced?"

Requirements of the person applying the sandbox -- Install a tool:

“Prior to running the [NaCl-based] photo application, the user has installed Native Client as a browser plugin.”

Requirements of the application -- Use special compiler:
“We have modified the standard GNU tool chain, using version 4.2.2 of the gcc collec- tion of compilers [22], [29] and version 2.18 of binutils [23] to generate NaCl-compliant binaries.”

Requirements of the application -- Use sandbox as library:
“The sandboxes prevent unwanted side effects, but some side effects are often necessary to make a native module useful. For interprocess communications, Native Client provides a reliable datagram abstraction, the “Inter-Module Communications” service or IMC.”

Security Policy Type -- Fixed policy:
See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Pre-exploit
See What is protected by the policy? (fine grained)

Policy Management -- No policy management:
Fixed policy

Policy Construction -- Encoded in the logic of the sandbox:
See What is protected by the policy? (fine grained)

Validation -- Benchmark (Performance):
“We first consider the overhead of making native code side effect free. To isolate the impact of the NaCl binary constraints (Table 1), we built the SPEC2000 CPU benchmarks using the NaCl compiler, and linked to run as a standard Linux binary. The worst case for NaCl overhead is CPU bound applications, as they have the highest density of alignment and sandboxing overhead. Figure 4 and Table 4 show the overhead of NaCl compilation for a set of benchmarks from SPEC2000. The worst case performance overhead is crafty at about 12%, with other benchmarks averaging about 5% overall.”

Validation -- Proof (Security):
“Theorem: S contains all addresses that can be reached from an instruction with address in S.”

Validation -- Argumentation (Security):

“These mechanisms will allow us to incorporate layers of protection based on our confidence in the robustness of the various components and our understanding of how to achieve the best balance between performance, flexibility and security.
In the next section we hope to demonstrate that secure implementations of these facilities are possible and that the specific choices made in our own implementation work are sound.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Source Code:

“By describing Native Client here and making it available as open source, we hope to encourage community scrutiny and contributions.”

CLAMP, Oakland 2009 -

Where is the policy enforced? -- System:

“We developed a prototype using platform virtualization (based on the Xen hypervisor [1]) to isolate system components on the web server.”

When is the policy imposed? -- Dynamically:

“A trusted User Authentication module verifies user identities and instantiates a new virtual web server instance for each user. The database queries issued by a particular virtual web server are constrained by a trusted Query Restrictor to access only the data for the user assigned to that web server.”

What is protected by the policy? (fine grained) -- User Data:

“CLAMP prevents web server compromises from leaking sensitive user data by (1) ensuring that a user’s sensitive data can only be accessed by code running on behalf of that user, and (2) isolating code running on behalf of different users.”

What is protected by the policy? (coarse grained) -- Targeted Application:

“Our experience adapting three real-world LAMP applications to use CLAMP demonstrates the benefits of an architecture designed for compatibility with web application stacks.”

Requirements of the person applying the sandbox -- Install a Tool, Write a Policy:

"First, CLAMP provides a straightforward way to express and audit access control policies using a single policy file instead of using checks scattered throughout the code."
See Where is the policy enforced? and When is the policy imposed?

Requirements of the application -- None:

See What is protected by the policy? NOTE: Minor changes to application are counted as installation.

Security Policy Type -- Used-defined policy:

See Requirements of the person applying the sandbox.

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- Centralized Repository:

See Requirements of the person applying the sandbox. NOTE: Because the policy is in a central file, it's easy to see how to share the policy with others.

Policy Construction -- Manually written policy:

See Requirements of the person applying the sandbox.

Validation Claim -- Security:

" CLAMP protects sensitive data by enforcing strong access control on user data and by isolating code running on behalf of different users."

Validation Claim -- Applicability:

"By focusing on minimizing developer effort, we arrive at an architecture that allows developers to use familiar operating systems, servers, and scripting languages, while making relatively few changes to application code – less than 50 lines in our applications."

Validation Claim -- Performance:

"Finally, our unoptimized prototype suggests that the user-perceived slowdown due to CLAMP's use of virtualization is not prohibitive (typical request latency for osCommerce is 5-10 ms slower than native)."

Validation -- Argumentation (Security):

"CLAMP relies on trusted system components to enforce its security properties. These CLAMP components have three primary sources of attack robustness: a reduced trusted computing base (TCB), a minimized interface for each component of the TCB, and defense-in-depth."

Validation -- Case Study (Applicability):

"We have developed a proof-of-concept implementation of the CLAMP architecture and applied it to osCommerce [19], MyPhpMoney [3], and HotCRP [10]."

Validation -- Case Study (Performance):

"For these benchmarks, clients retrieve osCommerce pages from either a "native" server running directly on hardware or a CLAMP server as described in Section 6."

Availability -- Not Available:

No mention in paper

BLUEPRINT, Oakland 2009 -

Where is the policy enforced? -- Application host:
"Therefore, in BLUEPRINT, we enable a web application to effectively take control of parsing decisions. By systematically reasoning about the flow of untrusted HTML in a browser, we develop an approach that provides facilities for a web application to automatically create a structural representation — a "blueprint" — of untrusted web content that is free of XSS attacks."

When is the policy imposed? -- Dyamic:
See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions, User Data:
See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Targeted Application:
See Where is the policy enforced?

Requirements of the person applying the sandbox -- Write a security policy:
"To use BLUEPRINT, statements in a web application that output untrusted data need to be identified and instrumented with calls to our server-side module."

Requirements of the application --Use sandbox as framework/library:
See Requirements of the person applying the sandbox.

Security Policy Type -- User-defined policy:
See Requirements of the person applying the sandbox.

Policy enforcements place in kill chain -- pre-exploit:
See Where is the policy enforced?

Policy Management -- No management:
Not mentioned.

Policy Construction -- Encoded in the logic of the application:
See Where is the policy enforced?

Validation -- Applicability (Case Studies):
Section heading: "Compatibility and expressiveness" ->
"We integrated BLUEPRINT with two popular web applications that produce HTML output based on untrusted user input: MediaWiki, the code base used for the web site Wikipedia, and WordPress, a popular blog application. Both applications directly allow HTML as input from untrusted users."

Validation -- Security (Benchmark suite):
"The primary goal of our defense is to be effective against a wide variety of XSS attacks. For our testing we chose XSS Cheat Sheet [4], which includes complex examples of XSS attack strings. Many of these examples are noteworthy for undermining sophisticated, real-world regular expression based defenses. Further- more, the cheat sheet contains attacks that combine exploits of several browser parse quirks to achieve execution of arbitrary script commands."

Validation -- Performance (Case Studies):

“We stove to stress our implementation and evaluate the performance of BLUEPRINT-enabled WordPress and MediaWiki under the worst possible conditions for varying amounts of embedded untrusted content.”

Validation -- Public Data (Security):

See other validation quotes.

Availability -- Source Code:

“A prototype implementation of B LUEPRINT is available for experimentation at the project website: http://sisl.rites.uic.edu/blueprint”

Lares, Oakland 2009 -

Where is the policy enforced? -- System:
“We show design techniques that allow installation of protected hooks into an untrusted VM. These hooks will trap execution in the untrusted VM and transfer control to software in the protected VM.”

When is the policy imposed? -- Dynamic:
See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code instructions:
“We do not consider new malware detection or prevention techniques as these areas are orthogonal to the research presented here. Any system that uses active monitoring, including any future advances in the field, can benefit from the added protections that our work provides. The primary research contribution of this work is an architecture to perform secure, active monitoring in a virtualized environment.”

What is protected by the policy? (coarse grained) -- Class of applications:
“This architecture is generally applicable to any system that requires secure and active monitoring, and it builds on prior work that described techniques for passively monitoring memory and file system data [22].”

Requirements of the person applying the sandbox -- Install a tool:
See Requirements of the application -- Use sandbox as framework/library

Requirements of the application -- Use sandbox as framework/library:

“The security VM contains the back-end components of our architecture. These include the security application, the security driver, and an introspection API. The security application is where the decision-making functionality of the monitoring solution is implemented. It can be any software component that makes use of Lares, like an anti-virus tool or a host-based IDS. The security driver is the communications agent responsible for relaying events between the trampoline and the security application. These include hook notifications transmitted by the trampoline in the guest VM and relayed by the hypervisor, and decisions sent by the security application to the hypervisor. The introspection API provides the necessary introspection functionality to the security application, allowing it to collect additional information about the event that was trapped.”

Security Policy Type -- Fixed policy:
See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit
See Where is the policy enforced?

Policy Management -- No management:
Fixed policy

Policy Construction -- Encoded in the logic of the sandbox:
See Where is the policy enforced?

Validation -- Security (Case studies) :
“Our test system consisted of the guest VM running with the trampoline and hooks initialized. To ensure the syn- thetic attack works properly, we then ran it without any memory protections enabled. During this test, the syn- thetic attack worked as expected, hijacking the hook and effectively preventing any execution of the trampoline code. Next, we repeated the test with the memory protections enabled. This time the synthetic attack failed to complete its installation because it was unable to change the write- protected entry in the SSDT and the trampoline code con- tinued to execute normally.”

Validation -- Performance (Case studies):
“Hook processing is the key operation where the Lares architecture will differ in performance from a traditional architecture. Therefore, our benchmark measurements look at the time required to process a single hook in the Lares architecture and compare that with a traditional architecture. To measure the hook processing time with the Lares architecture, we instrumented the trampoline code. We retrieved the value of the processor’s performance counter before and after the VMCALL instruction. The processor’s performance counter was obtained using a function provided by Windows, KeQueryPerformanceCounter. The difference

between these two measurements represents the time needed for inter-VM communication and hook processing within the security VM. However, this measurement is noisy. It can be influenced by cache effects, VM scheduling, physical interrupts, CPU frequency scaling, and other loads on the system. We took several steps to minimize the influence of this noise in our measurements. First, we pinned each VM to its own CPU core. Next, we disabled unnecessary services in the security VM. Then we disabled CPU frequency scaling in the BIOS.

Our test system consisted of the guest VM running with the trampoline and hooks initialized. To ensure the syn- thetic attack works properly, we then ran it without any memory protections enabled. During this test, the syn- thetic attack worked as expected, hijacking the hook and effectively preventing any execution of the trampoline code. Next, we repeated the test with the memory protections enabled. This time the synthetic attack failed to complete its installation because it was unable to change the write- protected entry in the SSDT and the trampoline code con- tinued to execute normally."

Availability -- Not Available:

No mention in paper

Sun, Oakland 2008 -

Where is the policy enforced? -- System:

"Our enforcement framework, described in Section 5, consists of a small, security-critical enforcement component that resides in the OS kernel, and a user-level component that incorporates more complex features that enhance functionality without impacting security."

When is the policy imposed? -- Static:

See Requirements of the person applying the sandbox (NOTE: Analysis automates labeling and policy generation)

What is protected by the policy? (fine grained) -- Files, User-data:

"In contrast, we develop an approach in this paper that aims to provide positive assurance about overall system integrity. Our method, called PPI (Practical Proactive Integrity protection), identifies a subset of objects (which are typically files) as integrity-critical and a set of untrusted objects. We assume that system integrity is preserved as long as untrusted objects are prevented from influencing the contents of integrity-critical objects either directly (e.g., by copying of an untrusted object over an integrity-critical object) or indirectly through intermediate files."

What is protected by the policy? (coarse grained) -- System:

"Provide positive assurances about system integrity on a contemporary Workstation, e.g., a Linux CentOS/Ubuntu desktop consisting of hundreds of benign applications and tens of untrusted applications. Integrity should be preserved even if untrusted programs run with root privileges."

Requirements of the person applying the sandbox -- Write a Policy, Run a Tool:

"The large number of objects and subjects in a modern OS distribution motivates automated policy development. We envision policy development to be undertaken by a security expert — for instance, a member of a Linux distribution development team. The goal of our analysis is to minimize the effort needed on the part of this expert."
NOTE: The analysis is sound and writes a (mostly complete) start policy.

Requirements of the application -- None:

"In contrast, we have been able to develop a practical information-flow based integrity protection for desktop Linux systems by focusing on (a) automating the development of integrity labels and policies, (b) providing a degree of assurance that these labels and policies actually protect system integrity, and (c) developing a flexible framework that can support contemporary applications while minimizing usability problems as well as the need to designate applications as "trusted.""

Security Policy Type -- User-defined policy:

See Requirements of the person applying the sandbox

Policy enforcements place in kill chain -- Pre-exploit:

"Most malware, including rootkits and spyware, should be detected when they attempt to install themselves, and removed automatically and cleanly."

Policy Management -- No management:

None specified

Policy Construction -- Manually written policy:

See Requirements of the person applying the sandbox

Validation Claim -- Security:

"Evaluation of our prototype implementation on a Linux desktop distribution shows that it does not break or inconvenience the use of most applications, while stopping a variety of sophisticated malware attacks."

Validation Claim -- Applicability:

See Requirements of the application

Validation Claim -- Performance:

"We observed that PPI did not introduce noticeable overhead for most system calls except for open (and other similar system calls such as stat)."

Validation -- Case Studies (Applicability):

"The set of initial integrity-critical file objects include files within /boot, /etc/init.d/, /dev/sda and /dev/kmem. We identified 26 untrusted application packages, which include:"

Validation -- Case Studies (Security):

"In this experiment, we downloaded around 10 up-to-date rootkits from [1]."
"To test the effectiveness of PPI under this threat, we crafted a "malicious" rpm package."
"In this attack, we created another piece of malware that first created an executable with an enticing name and waited for users on the system to run it."
"Malformed data input. Similar to the above example, a malformed jpeg file was downloaded from some unknown source, so PPI marked it as low-integrity."

Validation -- Benchmark Suite (Performance):

"For microbenchmark evaluation, we used LMbench version 3 [23] to check the performance degradation of popular system calls."

Validation -- Case Studies (Performance):

"For macrobenchmark, we measured 3 typical applications running within PPI during runtime."

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

WIT, Oakland 2008 -

Where is the policy enforced? -- Application:
"WIT uses the points-to analysis to assign a color to each object and to each write instruction such that all objects that can be written by an instruction have the same color. It instruments the code to record object colors at runtime and to check that instructions write to the right color. The color of memory locations is recorded in a color table that is updated when objects are allocated and deallocated. Write checks look up the color of the memory location being written in the table and check if it is equal to the color of the write instruction. This ensures write integrity."

When is the policy imposed? -- Hybrid:
See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory:
See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Targeted Application:
See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a tool:
See Where is the policy enforced?

Requirements of the application -- Use a special compiler:
"We implemented the points-to and the write safety analysis using the Phoenix compiler framework [30]. These anal- ysis operate on Phoenix's medium level intermediate representation (MIR), which enables them to be applied to different languages and target architectures."

Security Policy Type -- Fixed policy:
See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit:
See Where is the policy enforced?

Policy Management -- None:
Fixed policy.

Policy Construction -- Embedded in the logic of the sandbox:
See Where is the policy enforced?

Validation -- Performance (Benchmark suite) :
"In our first experiment, we measured the overhead added by WIT to 9 programs from the SPEC CPU 2000 benchmark suite [40] (gzip, vpr, mcf, crafty, parser, gap, vortex, bzip2 and twolf) 3, and to 9 programs from the Olden [12] benchmark suite (bh, bisort, em3d, health, mst, perimeter, power, treeadd, and tsp). We chose these programs to facilitate comparison with other techniques that have been eval-uated using the same benchmark suites."

Validation -- Security (Benchmark suite) :
"We ran experiments to evaluate the precision of the points-to analysis and its impact on security. We used WIT to compile nine programs from the SPEC CPU 2000 bench-mark suite [40] (gzip, vpr, mcf, crafty, parser, gap, vortex, bzip2 and twolf)."

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Li, Oakland 2007 -

Where is the policy enforced? -- System:

"We also discuss our implementation of the UMIP model for Linux using the Linux Security Modules framework, and show that it is simple to configure, has low overhead, and effectively defends against a number of network-based attacks."

When is the policy imposed? -- Hybrid:

See Security Policy Type and What is protected by the policy? (fine grained)

What is protected by the policy? (fine grained) -- Files, Communication, User Data:

“When a process performs an operation that makes it potentially contaminated, it drops its integrity. Such operations include communicating with the network, receiving data from a low-integrity process through an interprocess communication channel, and reading or executing a file that is potentially contaminated. A low-integrity process by default cannot perform sensitive operations.”

What is protected by the policy? (coarse grained) -- Class of Applications (all that run on this system):

“Basic UMIP Model: Each process has one bit that denotes its integrity level. When a process is created, it inherits the integrity level of the parent process. When a process performs an operation that makes it potentially contaminated, it drops its integrity. A low-integrity process by default cannot perform sensitive operations.”

Requirements of the person applying the sandbox -- Install a Tool, Write a Policy:

See Where is the policy enforced? and Security Policy Type

Requirements of the application -- None:

“Second, existing applications and common usage practices can still be used under UMIP.”

Security Policy Type -- Fixed Policy, User-defined policy (for exceptions):

“One novel feature of UMIP is that, unlike previous MAC systems, UMIP uses existing DAC information to identify which files are to be protected. In UMIP, a file is write-protected if its DAC permission is not world-writable, and a file is read-protected if it is owned by a system account (e.g., root, bin, etc.) and is not world-readable. A low-integrity process (even if running as root) by default is forbidden from writing any write-protected file, reading any read-protected file, or changing the DAC permission of any (read- or write-) protected file.”

“For another example, exception policies can be specified for some programs so that even when they are running in low-integrity processes, they can access some protected resources.”

Policy enforcements place in kill chain -- Post-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- Central Repository:

“Any exception to the above default policy must be specified in a policy file, which is loaded when the module starts.”

Policy Construction -- Encoded in the logic of the sandbox, Manually written policy (for exception):

See Security Policy Type

Validation Claim -- Security, Performance:

See Where is the policy enforced? NOTE: They aren’t claiming their MIP system is applicable to more applications but that it is easier to use.

Validation -- Case Studies (Security):

“In our experiments, we use the NetCat tool to offer an interactive root shell to the attacker in the experiment. We execute NetCat in “listen” mode on the test machine as root. When the attacker connects to the listening port, NetCat spawns a shell process, which takes input from the attacker and also directs output to him. From the root shell, we perform the following three attacks and compare what happens without our protection system with what happens when our protection system is enabled.”

Validation -- Benchmark Suite (Performance):

“Our performance evaluation uses the Lmbench 3 benchmark and the Unixbench 4.1 benchmark suites.”

Validation -- Public Data (Performance):

See other validation quotes. Security experiment is no reasonably comparable

Availability -- Not Available:

No mention in paper

Tahoma, Oakland 2006 -

Where is the policy enforced? -- System:

“This paper describes the architecture and implementation of the Tahoma Web browsing system. Key to Tahoma is the browser operating system (BOS), a new trusted software layer on which Web browsers execute.”
“We have implemented a prototype of Tahoma using Linux and the Xen virtual machine monitor.”

When is the policy imposed? -- Hybrid:

“First, the BOS runs the client-side component of each Web application (e.g., on-line banking, Web mail) in its own virtual machine. This provides strong isolation between Web services and the user’s local resources. Second, Tahoma lets Web publishers limit the scope of their Web applications by specifying which URLs and other resources their browsers are allowed to access. This limits the harm that can be caused by a compromised browser. Third, Tahoma treats Web applications as first-class objects that users explicitly install and manage, giving them explicit knowledge about and control over downloaded content and code.”
See Requirements of the application

What is protected by the policy? (fine grained) -- Memory, Communication, Code/Instructions, User Data:

See When is the policy imposed?

What is protected by the policy? (coarse grained) -- Class of Applications:

See When is the policy imposed? NOTE: Web applications

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- Annotated source code:

“Tahoma Web applications are first-class objects and are explicitly defined and managed. The Web service specifies the characteristics of its application in a manifest, which the BOS retrieves when it first accesses the service.”

Security Policy Type -- Fixed Policy, Application-defined policy:

See When is the policy imposed? and Requirements of the application

Policy enforcements place in kill chain -- Pre/post-exploit:

See When is the policy imposed?

Policy Management -- Central policy repository:

See Requirements of the application NOTE: policies are stored with webapps and distributed to anyone that needs them

Policy Construction -- Encoded in the logic the sandbox, Encoded in the logic of the application:

See When is the policy imposed? and Requirements of the application

Validation Claim -- Security:

“Our security evaluation shows that Tahoma can prevent or contain 87% of the vulnerabilities that have been identified in the widely used Mozilla browser.”

Validation Claim -- Performance:

“In addition, our measurements of latency, throughput, and responsiveness demonstrate that users need not sacrifice performance for the benefits of stronger isolation and safety.”

Validation -- Case Studies (Security):

“We examined each of the 109 Mozilla vulnerabilities to determine whether Tahoma successfully contains or eliminates the threat within the affected browser instance, or whether the attacker can use the vulnerability to harm external resources or Web applications.”

Validation -- Case Studies (Performance):

“Figure 7 shows the cost of forking a new Tahoma browser instance in a virtual machine compared to the cost of starting a new browser in native Linux. The top half of the table shows two different Tahoma cases.”
“To measure the Web-object fetch latency, we started several concurrent browser instances, each scripted to fetch a Web object repeatedly. We measured the average latency to fully retrieve the object from a dedicated server on the local”
“To measure the performance of the Tahoma window manager, we ran a variable number of virtual machines, each containing an MPlayer browser instance, which we consider a “worst case” test.”
“To measure Tahoma’s input performance, we recorded the delay between the time a user presses a key and the time the corresponding character is rendered by a Konqueror browser instance.”

Validation -- Public Data (Security):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Bugiel, Usenix 2013 -

Excluded: Not a sandbox. Framework for developing other sandboxes.
“We show how our security framework can instantiate selected use-cases. The first one is an attack-specific related work, the well-known application centric security solution Saint [39]. The second one is a privacy protecting solution that uses fine-grained and user-defined access control to personal data. We also mention other useful security models that can be instantiated with FlaskDroid.”

Zhang, Usenix 2013 -

Where is the policy enforced? -- Application

“Each shared library and executable is instrumented independently to enforce CFI.”

When is the policy imposed? -- Statically

“Our implementation utilizes objdump to perform linear disassembly.”

"After rewriting, the instrumented assembly file is processed using the system assembler (in our case, the GNU assembler gas) to produce an object file. We extract the code from this object file and then use the objcopy tool to inject it into the original ELF file."

What is protected by the policy? (fine grained) -- Code/Instructions

See Security Policy Type and Policy enforcements place in kill chain.

What is protected by the policy? (coarse grained) -- Targeted Application

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool

See When is the policy imposed?

Requirements of the application -- No Additional Requirements

"Compiler independence and support for hand-coded assembly: Our approach does not make strong assumptions regarding the compiler used to generate a binary, such as the the conventions for generating jump tables."

Security Policy Type -- Fixed Policy

"An ideal CFI implementation will restrict program execution to exactly the set of program paths that can be taken. In practice, due to the fact that targets of indirect control-flow (ICF) transfers are difficult to predict, CFI implementations enforce a conservative approximation of ideal CFI."

Policy enforcements place in kill chain -- Pre-exploit

"Its enforcement can defeat most injected and existing code attacks, in-
cluding those based on Return-Oriented Programming (ROP)."

Policy Management -- No management

A fixed policy is used thus no policy management is required.

See Security Policy Type.

Policy Construction -- Encoded in the logic of the sandbox

"We present the first practical approach for CFI enforcement that scales to large binaries as well as shared libraries without requiring symbol, debug, or relocation information. We have developed techniques that cope with the challenges presented by static analysis and transformation of large programs, including those of Firefox, Adobe Acrobat 9, GIMP-2.6 and glibc."

Validation claims -- Security

"Our technique ensures that when an executable is loaded and run, CFI property is enforced globally across the executable and all the shared libraries used by it."
"This provides evidence that our approach achieves compatibility with COTS binaries without incurring a major reduction in its quality of protection."
"Moreover, on the SPEC CPU 2006 benchmark, our technique also eliminated about 93% of ROP gadgets that were found by the popular ROP gadget discovery tool ROPGadget [35]."
"Our results show that bin-CFI defeats the vast majority of control-flow hijack attacks from the RIPE benchmark [45]."

Validation claims -- Applicability

"Changes to saved return address would cause these uses to break, thus leading to application failure. For this reason, our instrumentation has been designed to provide full transparency."

See Policy Construction.

Validation claims -- Performance/Validation -- Benchmark Suite

"We describe several optimization techniques in Section 6 that have reduced the overhead to about 8.54% across the SPEC CPU benchmark suite."

Validation -- Benchmark Suite (Applicability)

"We tested the SPEC CPU2006 programs (Figure 8). This benchmark comes with scripts to verify outputs, thus simplifying functionality testing."

Validation -- Benchmark Suite (Security)

"Figure 8 compares the AIR metric for bin-CFI with strict-CFI, reloc-CFI, bundle-CFI and instr-CFI. To calculate AIR of reloc-CFI, we recompiled SPEC2006 programs using "-g" and a linker option "-Wl,-emit-relocs"to retain all the relocations in executables."
"To evaluate control flow hijack defense, we used the RIPE [45] test suite."

Validation -- Public Data (Security, Performance, Applicability):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Aurasium, Usenix 2012 -

Where is the policy enforced? -- Application:

“We develop a novel solution called Aurasium that bypasses the need to modify the Android OS while providing much of the security and privacy that users desire. We automatically repackage arbitrary applications to attach user-level sandboxing and policy enforcement code, which closely watches the application’s behavior for security and privacy violations such as attempts to rerieve a user’s sensitive information, send SMS covertly to premium numbers, or access malicious IP addresses.”

When is the policy imposed? -- Statically:

“To insert Aurasium’s Java code into an existing application, we have to take the original classes.dex, disassemble it back to a collection of individual classes, add Aurasium’s classes, and then re-assemble everything back to create the new classes.dex.”

What is protected by the policy? (Fine Grained) -- Files, Communication, User Data:

“We are interested primarily in enforcing some security policy that protects the device from untrusted applications. This includes not only attempts by the application to access sensitive information, leaking to the outside world or modifying it, but also attempts by the application to escalate privilege and to gain root access on the device by running suspicious system calls and loading native libraries.”

What is protected by the policy? (Coarse Grained) -- Targeted Application:

“We have a web interface that allows users to upload arbitrary applications and download the Aurasium repackaged and hardened version.”

Requirements of the person applying the sandbox -- Select a pre-made security policy, Run a tool

“Aurasium displays a warning message and prompts the user to either accept the requested access or deny it. The user can also make Aurasium store that user’s answer to the request so that the same request never prompts the user for approval again and the cached answer is used instead.”

See What is protected by the policy?

Requirements of the application -- No Additional Requirements:

No requirements stated.

Security Policy Type -- User-defined Policy

“Depending on the enforced polices at repackaging time, an application queries the
ASM for a policy decision via IPC mechanisms with intents describing the sensitive operation it is about to perform, and the ASM either prompts the user for consent, uses a remembered user decision recorded earlier, or automatically makes a decision without user interaction by enforcing a predefined policy embedded at repackaging time.”

Policy enforcements place in kill chain -- Pre-exploit:

“Aurasium displays the destination number as well as the SMS’s content, so users can make informed decision on whether to allow the operation or not.”

See Security Policy Type.

Policy Management -- Classes of Policies or No Management depending on configuration:

“Aurasium-wrapped applications are self-contained in the sense that the policy logic and the relevant user interface are included in the repackaged application bundle, and so are remembered user decisions stored locally in the application’s data directory. Alternatively, Aurasium Security Manager (ASM) can also be installed, enabling central handling of policy decisions of all repackaged application on the device.”

Policy Construction -- Manually written policies:

See Requirements of the person applying the sandbox. NOTE: The checks to throw prompts are encoded in the sandbox, but the policy that is acted on is defined by the user by answering prompts.

Validation claims -- Security:

"We provide a way of protecting users from malicious applications"

Validation claims -- Applicability:

"...without making any changes to the underlying Android architecture. This makes Aurasium a technology that can be widely deployed."

Validation claim -- Performance:

"It has low memory and runtime overhead..."

Validation -- Case Study (Security):

"Figure 5 illustrates how Aurasium intercepts SMS messages sent to a premium number, which is initiated by the malicious application AndroidOS.FakePlayer [2] found in the wild. ... We also observed malware NickySpy [3] leaking device IMEI via SMS in another test run."

Validation -- Argumentation (Security):

"Because fundamentally Aurasium code runs in the same process context as the application's code, there is no strong barrier between the application and Aurasium. Hence, it is non-trivial to argue that Aurasium can reliably sandbox arbitrary Android applications. We describe possible ways that a malicious application can break out of Aurasium's policy enforcement mechanism and discuss possible mitigation against them."

Validation -- Benchmark Suite (Applicability):

"We applied Aurasium to 3491 applications crawled from a third-party application store7 and 1260 known malicious applications [39]. Table 1 shows the success rate of repackaging for each category of applications."
"Out of 3476 successfully repackaged application, we performed tests on 3189 standalone runnable applications8 on the device. We were able to start all of the

applications in the sense that Aurasium successfully reported the interception of the first API invocation for all of them."

Validation -- Benchmark Suite (Performance):

"We take two Android benchmark applications from the official market and apply Aurasium to them in order to check if Aurasium introduces significant performance overhead to a real-world application."

Validation -- Public Data (Performance, Applicability):

See other validation quotes.

Availability -- Binaries:

"We have a web interface 6 that allows users to upload arbitrary applications and download the Aurasium repackaged and hardened version."

Giuffrida, Usenix 2012 -

Where is the policy enforced? -- System:

"Finally, we present the first comprehensive live rerandomization strategy, which we found to be particularly important inside the OS."

See What is protected by the policy?

When is the policy imposed? -- Hybrid:

"In our design, all the OS processes (and the microkernel) are randomized using a link-time transformation implemented with the LLVM compiler framework [42]."

See Where is the policy enforced (live rerandomization).

What is protected by the policy? (fine grained) -- Memory:

"ASR is a well-established defense mechanism to protect user programs against memory error exploits [12, 39, 14, 72, 73]; all the major operating systems include some support for it at the application level [1, 68]."

What is protected by the policy? (coarse grained) -- System Level Component:

“In this paper, we explore the benefits of address space randomization (ASR) inside the operating system and present the first comprehensive design to defend against classic and emerging OS-level attacks.”

Requirements of the person applying the sandbox -- Run a Tool:

See When is the policy imposed?

Requirements of the application -- Use Special Compiler:

“Our approach is to transform the bit-code with another LLVM link-time pass, which embeds metadata information into the binary and makes run-time state introspection and automated migration possible.”

See When is the policy imposed?

Security Policy Type -- Fixed Policy:

See When is the policy imposed? and Requirements of the application?

Policy enforcements place in kill chain -- Pre-Exploit:

“The goal of address space randomization is to ensure that code and data locations are unpredictable in memory, thus preventing attackers from making precise assumptions on the memory layout.”

Policy Management -- No Management:

A fixed policy is used thus no policy management is required.

See Security Policy Type.

Policy Construction -- Encoded in the logic the sandbox:

See When is the policy imposed? and Requirements of the application?

Validation claims -- Security, Performance:

“Our approach addresses all the challenges considered and improves existing ASR solutions in terms of both performance and security, especially in light of emerging ROP-based attacks.”

Validation claims -- Applicability/Validation -- Case Study:

“In addition, we consider the application of our design to component-based OS architectures, presenting a fully fledged prototype system and discussing real-world applications of our ASR technique.”

Validation -- Benchmark Suite (Performance):

“To evaluate the performance of our ASR technique, we ported the C programs in the SPEC CPU 2006 benchmark suite to our prototype system.”

Validation -- Case Study (Performance):

“We also put together a devtools macrobenchmark, which emulates a typical syscall-intensive workload with the following operations performed on the OS source tree: compilation, find, grep, copying, and deleting.”

Validation -- Analytical Analysis (Security):

“Their entropy analysis applies also to other second-generation ASR techniques, and, similarly, to our technique, which, however, provides additional entropy thanks to internal layout randomization and live rerandomization.”
“Assuming the same pi(t) in every round for simplicity, it can be shown that the expected time before the attacker can complete the probing phase in a single rerandomization window (and thus the attack) is: ...”

Validation -- Argumentation (Security):

“Second-generation techniques, in turn, allow the attacker to corrupt the right memory location by learning the relative distance/alignment between two memory objects. In this respect, our internal layout randomization provides better protection, forcing the attacker to learn the relative distance/alignment between two memory elements in the general case.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

kGuard, Usenix 2012 -

Where is the policy enforced? -- System:

“We present kGuard, a compiler plugin that augments the kernel with compact inline guards, which prevent ret2usr with low performance and space overhead.”

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions:

See Where is the policy enforced? and Security Policy Type

What is protected by the policy? (coarse grained) -- System Level Component:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

See Where is the policy enforced?

Requirements of the application -- Have source code, Use special compiler:

“We present the design and implementation of kGuard, a compiler plugin that protects the kernel from ret2usr attacks by injecting fine-grained inline guards during compilation. Our approach does not require modifications to the kernel or additional software, such as a VMM.”

Security Policy Type -- Fixed Policy:

“Similar to CFI, we rely on inline checks injected before every unsafe control-flow transfer.”

Policy enforcements place in kill chain -- Pre-exploit:

See When is the policy imposed? and Security Policy Type

Policy Management -- No management:

See When is the policy imposed? and Security Policy Type

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed? and Security Policy Type

Validation claims -- Security:

“Our evaluation demonstrates that Linux kernels compiled with kGuard become impervious to a variety of control-flow hijacking exploits.”

Validation claims -- Performance:

”kGuard exhibits lower overhead than previous work, imposing on average an overhead of 11.4% on system call and I/O latency on x86 OSs, and 10.3% on x86-64. The size of a kGuard-protected kernel grows between 3.5% and 5.6%, due to the inserted checks, while the impact on real-life applications is minimal (≤1%).”

Validation -- Case Study (Security):

“Table 1 summarizes our test suite, which consisted of a collection of 8 exploits that cover a broad spectrum of different flaws, including direct NULL pointer dereferences, control hijacking via tampered data structures (data pointer corruption), function and data pointer overwrite, arbitrary kernel-memory nullification, and ret2usr via kernel stack-smashing.”

Validation -- Case Study (Performance):

“We begin with the evaluation of kGuard using a set of real-life applications that represent different workloads. In particular, we used a kernel build and two popular server applications. The Apache web server, which performs mainly I/O, and the MySQL RDBMS that is both I/O driven and CPU intensive.”

Validation -- Benchmarks (Performance):

“Since the injected CFAs are distributed throughout many kernel subsystems, such as the essential net/ and fs/, we used the LMbench [50] microbenchmark suite to measure the impact of kGuard on the performance of core kernel system calls and facilities.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Source code:

“The prototype implementation of kGuard is freely available at: http://www.cs.columbia.edu/~vpk/research/kguard/”

Sehr, Usenix 2010 -

Where is the policy enforced? -- Application, Application Host:

“The original NaCl x86-32 system relies on a set of rules for code generation that we briefly summarize here:”
“All rules are checked by a verifier before a program is executed.”

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:

“We present software fault isolation schemes for ARM and x86-64 that provide control-flow and memory integrity…”

What is protected by the policy? (coarse grained) -- Targeted Application:

“This work extends Google Native Client [30].”

Requirements of the person applying the sandbox -- Run a Tool:

See What is protected by the policy?

Requirements of the application -- Have source code, Use special compiler (NaCl build tools), Use sandbox as a framework/library (Pepper API):

“We have modified LLVM 2.6 [13] to implement our ARM SFI design.”
“Our x86-64 SFI implementation is based on GCC 4.4.3, requiring a patch of about 2000 lines to the compiler, linker and assembler source.”

See What is protected by the policy?

Security Policy Type -- Fixed-Policy:

See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-Exploit:

“Ensuring that untrusted code cannot execute any forbidden instructions (e.g. undefined encodings, raw system calls).
Ensuring that untrusted code cannot store to memory locations above 1GB.
Ensuring that untrusted code cannot jump to memory locations above 1GB (e.g. into the service runtime implementation).”

See Where is the policy enforced?

Policy Management -- No Management:

See Security Policy Type

Policy Construction -- Encoded in the logic of the sandbox:

See Where is the policy enforced?

Validation claims -- Performance:

“...with average performance overhead of under 5% on ARM and 7% on x86-64.”
“Our experience suggests that these SFI implementations benefit from instruction-level parallelism, and have particularly small impact for workloads that are data memory-bound, both properties that tend to reduce the impact of our SFI systems for future CPU implementations.”

Validation claims -- Security:

“We achieve these goals by adapting to ARM the approach described by Wahbe et al. [28].” In reference to the goals listed in the quote in Policy enforcements place in kill chain

Validation -- Benchmarks (Performance):

“In this section we evaluate the performance of our ARM and x86-64 SFI schemes by comparing against the relevant non-SFI baselines, using C and benchmarks from
SPEC2000 INT CPU [12].”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Source Code:

“Source code for Google Native Client can be found at:
http://code.google.com/p/nativeclient/.”

AdJail, Usenix 2010 -

Where is the policy enforced? -- Application Host:

“To enforce the publisher’s policy, we leverage browser enforcement of the same-origin policy (SOP) [50], an access control mechanism available in all major JavaScript-enabled browsers.”

When is the policy imposed? -- Hybrid:

“All messages sent between the real and shadow pages are mediated by our policy enforcement mechanism.”
“The architecture of our implementation requires changes to the original web page (real page) and creation of a corresponding shadow page as described in §3.1.”
“The first modification is to remove the ad script (Figure 4a). Second, we add the tunnel script (Figure 4b) to the end of the page. The third modification to the original page is annotation of HTML elements with policies, which we discussed at length in §4.1.”

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions, User Data:

“These policies are specified by a website to restrict the capabilities of third-party scripts, specifically with reference to access and modification of first-party (site owned) content, as well as control over the screen.”
“For instance, in our web-mail example, an integrity policy can be enforced such that email message content cannot be tampered with, but can still be read (for contextual targeting of ads). Publishers may also choose to restrict where ads can appear on the page.”

What is protected by the policy? (coarse grained) -- Target Application:

See When is the policy imposed?

Requirements of the person applying the sandbox -- Write a security policy, Run a Tool:

"By default, ad script is given no access to any part of the real page unless granted by policies (i.e., default-deny)."
"The publisher can annotate any HTML element of the real page with a policy attribute."

See When is the policy imposed?

Requirements of the application -- Annotated source code, Use sandbox as a library or framework:

See When is the policy imposed? and Requirements of the person applying the sandbox

Security Policy Type -- Fixed Policy (SOP), User Defined Policy:

See When is the policy imposed? and Requirements of the person applying the sandbox

Policy enforcements place in kill chain -- Pre-Exploit:

See Requirements of the application

Policy Management -- No Management:

No Mangement specified and seems unlikely given the nature of program annotations.

Policy Construction -- Encoded in the logic the sandbox (SOP), Manually written policies:

See Where is the policy enforced? and Requirements of the person applying the sandbox

Validation claims -- Applicability:

"We find that AdJail provides excellent compatibility for most ads."

Validation claims -- Security:

“We also demonstrate the strong protection offered by AdJail from many significant threats posed by online ads.”

Validation claims -- Performance:

“In our experiments, the currently unoptimized AdJail prototype encountered at most a 1.69× slowdown in rendering ads.”

Validation -- Case Studies (Applicability):

“To evaluate how well ADJAIL works with existing adscripts, we tested it on six popular ad networks: Yahoo! Network, Google AdSense, Microsoft Media Network, Federated Media Publishing, AdBrite and Clicksor.”

Validation -- Case Studies (Security):

“To evaluate the security provided by AdJail we installed the RoundCube webmail v0.3.1 software on our web server. We integrated two ad network scripts on the main webmail interface: one ad script was included directly on the page, and the other was embedded using AdJail. A single trial consisted of replacing each of the two ad scripts with a malicious script designed to perform one specific attack or policy violation. We then observed if the malicious script functioned correctly in the non-sandboxed location, and whether the attack was prevented in the sandboxed location. Several trials were conducted to assess different attack vectors, and to determine the least restrictive policy required to defend each vector.

Validation -- Benchmark Suite (Performance):

“To measure ad rendering latencies incurred by our policy enforcement mechanism, we placed each ad script on a typical blog page instrumented with benchmarking code. There were a total of 12 instances of the blog page: for each of the six ad networks evaluated in §5.1, one version of the blog page used the original ad, and a second version used ADJAIL to enforce the policies in Table 3.”

Availability -- Not Available:

No mention in paper

Cling, Usenix 2010 -

Where is the policy enforced? -- Application:

"The Cling memory allocator is a drop-in replacement for malloc designed to satisfy three requirements: ..."

When is the policy imposed? -- Hybrid (Compile to use it or LD_PRELOAD):

"Cling comes as a shared library providing implementations for the malloc and the C++ operator new allocation interfaces. It can be preloaded with platform specific mechanisms (e.g. the LD PRELOAD environment variable on most Unix-based systems) to override the system's memory allocation routines at program load time."

What is protected by the policy? (fine grained) -- Memory:

"Cling utilizes more address space, a plentiful resource on modern machines, to prevent type-unsafe address space reuse among objects of different types."

What is protected by the policy? (coarse grained) -- Targeted Application:

See When is the policy imposed?

Requirements of the person applying the sandbox -- Run a Tool:

See When is the policy imposed?

Requirements of the application -- No additional requirements:

See LD_PRELOAD remark in When is the policy imposed?

Security Policy Type -- Fixed Policy:

See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Pre-Exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No Management (Fixed Policy):

See What is protected by the policy? (fine grained)

Policy Construction -- Encoded in the logic of the sandbox:

See What is protected by the policy? (fine grained)

Validation claims -- Security, Performance:

"Cling disrupts a large class of attacks against use-after-free vulnerabilities, notably including those hijacking the C++ virtual function dispatch mecha- nism, with low CPU and physical memory overhead even for allocation intensive applications."

Validation claims -- Applicability:

"Similar in spirit, Cling is a pragmatic memory allocator modification for defending against use-after-free vulnerabilities that is readily applicable to real programs and has low overhead."

Validation -- Benchmark Suite (Performance):

"We used benchmarks from the SPEC CPU 2000 and (when not already included in CPU 2000) 2006 benchmark suites [22]."

Validation -- Case Studies (Performance):

"In the final set of experiments, we ran Cling with Firefox. Since, due to the size of the program, this is the most interesting experiment, we provide a detailed plot of memory usage as a function of time (measured in allocated Megabytes of memory), and we also compare against the naive solution of Section 2.2."

Validation -- Argumentation (Security):

"These cases cover generic exploitation techniques and attacks observed in the wild. The remaining attacks are less practical but may be exploitable in some cases, depending on the application and its use of data. Some constraints may still be useful; for example, attacks that hijack data pointers are constrained to only access memory in the corresponding field of another object of the same type. In some cases, this may prevent dangerous corruption or data leakage."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

BaggyBounds, Usenix 2009 -

Where is the policy enforced? -- Application:

"Our system shares the overall architecture of backwards compatible bounds checking systems for C/C++ (Figure 2). It converts source code to an intermediate representation (IR), finds potentially unsafe pointer arithmetic operations, and inserts checks to ensure their results are within bounds. Then, it links the generated code with our runtime library and binary libraries—compiled with or without checks—to create a hardened executable."

When is the policy imposed? -- Statically:

"Similar to previous work, we provide bounds checking wrappers for Standard C Library functions such as strcpy and memcpy that operate on pointers. We replace during instrumentation calls to these functions with calls to their wrappers."

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory:

See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

See Where is the policy enforced?

Requirements of the application -- Use a special compiler:

See Where is the policy enforced?

Security Policy Type -- Fixed Policy:

See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-Exploit:

See Where is the policy enforced?

Policy Management -- No Management:

No management stated.

Policy Construction -- Encoded in the logic of the sandbox

See Where is the policy enforced?

Validation claims -- Applicability, Performance:

“In this paper we present a backwards compatible bounds checking technique that substantially reduces performance overhead.”
“Our technique has low overhead in practice—only 8% throughput decrease for Apache—and is more than two times faster than the fastest previous technique and about five times faster—using less memory—than recording object bounds using a splay tree.”

Validation claims -- Security:

“Bounds checking C and C++ code protects against a wide range of common vulnerabilities.”

Validation -- Argumentation (Applicability):

“Baggy bounds checking works even when instrumented code is linked against libraries that are not instrumented. The library code works without change because it performs no checks but it is necessary to ensure that instrumented code works when accessing memory allocated in an uninstrumented library. This form of interoperability is important because some libraries are distributed in binary form.”

Validation -- Benchmark Suite (Performance):

“In Section 4 we evaluate the performance of our system using the Olden benchmark (to enable a direct comparison with Dhurjati and Adve [15]) and SPECINT 2000.”

Validation -- Benchmark Suite (Security):

“We also verify the efficacy of our system in preventing attacks using the test suite described in [34], ...

Validation -- Case Studies (Applicability):

“... and run a number of security critical COTS components to confirm its applicability.”

Validation -- Public Data (Security, Performance, Applicability):

See other validation quotes.

Availability -- Not Available:

No mention in paper

McCamant, Usenix 2008 -

Where is the policy enforced? -- Application:

“This effect is achieved by rewriting the machine instructions of code after compilation to directly enforce limits on memory writes and control flow.”

When is the policy imposed? -- Statically:

“The rewriting phase of PittSFIeld is implemented as a text processing tool, of about 720 lines of code, operating on input to the GNU assembler gas.”

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:

“The basic task for any SFI implementation is to prevent certain potentially unsafe instructions (such as memory writes) from being executed with improper arguments (such as an effective address outside an allowed data area). The key challenges are to perform these checks efficiently, and in such a way that they cannot be bypassed by maliciously designed jumps in the input code.”

What is protected by the policy? (coarse grained) -- Targeted Application:

See where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

See When is the policy imposed?

Requirements of the application -- Have source code:

"Because it operates on assembly code, our prototype rewriting tool is intended to be used by a code producer. A system that instead operates on off-the-shelf binaries without the code producer's cooperation is often described as a goal of SFI research, but has rarely been achieved in practice."

Security Policy Type -- Fixed Policy:

See where is the policy enforced?

Policy enforcements place in kill chain -- Pre-Exploit:

See Where the policy enforced?

Policy Management -- No Management

See Security Policy Type (fixed policy)

Policy Construction -- Encoded in the logic of the sandbox:

See Where the policy enforced?

Validation claims -- Security, Performance:

"We describe an implementation which provides a robust security guarantee and has low runtime overheads (an average of 21% on the SPECint2000 benchmarks)."

Validation -- Benchmark Suite (Performance):

"For better comparison with other work, we here concentrate on a standard set of compute-intensive programs, the integer benchmarks from the SPEC CPU2000 suite."

Validation -- Proof (Security):

"Specifically, we have constructed a completely formal and machine-checked proof of the fact that our technique ensures the security policy it claims to."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Source Code:

"Our implementation is publicly available (the version described here is 0.4), as are the formal model and lemmas used in the machine-checked proof. They can be downloaded from the project web site at http://pag.csail.mit.edu/ ~~s~~mcc/projects/pittsfield/ ."

Bhatkar, Usenix 2005 -

Where is the policy enforced? -- Application:

"Our implementation uses a source-to-source transformation on C programs. Note that a particular randomization isn't hard-coded into the transformed code. Instead, the transformation produces a self-randomizing program: a program that randomizes itself each time it is run, or continuously during runtime. This means that the use of our approach doesn't, in any way, change the software distribution model that is prevalent today."

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory:

"Our approach makes the memory locations of program objects (including code as well as data objects) unpredictable. This is achieved by randomizing the absolute locations of all objects, as well as the relative distance between any two objects."

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

"The main component of our implementation is a source code transformer which uses CIL [22] as the front-end, and Objective Caml as the implementation language."

See Where is the policy enforced?

Requirements of the application -- Have source code:

See Where is the policy enforced?

Security Policy Type -- Fixed Policy:

See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-Exploit

See What is protected by the policy?

Policy Management -- No Management:

Fixed Policy

Policy Construction -- Encoded in the logic the sandbox:

See Where is the policy enforced?

Validation claims -- Performance:

“We have collected data on the performance impact of the randomizing transformations.”

Validation claims -- Applicability:

“Our approach is implemented as an automatic, source-to-source transformation, and is fully compatible with legacy C code. It can interoperate with preexisting (untransformed) libraries.”

Validation claims -- Security:

“Hence the approach presented in this paper can address the full range of attacks that exploit memory errors.”

Validation -- Case Studies (Performance, Applicability):

“Figure 1 shows the test programs and their workloads.”

Validation -- Case Studies (Security):

"We have not carried out a detailed experimental evaluation of effectiveness because today's attacks are quite limited, and do not exercise our transformation at all. In particular, they are all based on a detailed knowledge of program memory layout."

Validation -- Analytical Analysis (Security):

"For this reason, we rely primarily on an analytical approach in this section. We first analyze memory error exploits in general, and then discuss attacks that are specifically targeted at randomization."

Availability -- Not Available:

No mention in paper

Linn, Usenix 2005 -

Where is the policy enforced? -- System:

"Our binary rewriting tools analyze binaries and add system call location information to them, without requiring the source code. This information is contained in a new section of an ELF binary file. Our modified OS kernel checks system call addresses only if an executable contains this additional section."

When is the policy imposed? -- Static:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions:

"This paper proposes a comprehensive set of techniques which limit the scope of remote code injection attacks. These techniques prevent any injected code from making system calls and thus restrict the capabilities of an attacker."

What is protected by the policy? (coarse grained) -- Targeted Application:

"if an executable does not contain this section, the intrusion detection mechanism is not invoked."

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

“We use post-link-time binary rewriting to identify the address of each system call instruction in the executable (our implementation currently uses the PLTO binary rewriting system for Intel x86 ELF executables [31]). This information is then added to the ELF executable as a new section, the Interrupt Address Table (IAT); the associated headers in the ELF file modified appropriately; and the file written back out.”

Requirements of the application -- No additional requirements:

See Requirements of the person applying the sandbox

Security Policy Type -- Fixed Policy:

See Requirements of the person applying the sandbox

Policy enforcements place in kill chain -- Post-Exploit:

What is protected by the policy? (fine grained)

Policy Management -- No Management

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Requirements of the person applying the sandbox

Validation claims -- Security:

“In defending against the traditional ways of harming a system these techniques significantly raise the bar for compromising the host system forcing the attack code to take extraordinary steps that may be impractical in the context of a remote code injection attack.”

Validation claims -- Performance:

“Our experiments indicate that the technique is effective and incurs only small runtime overheads.”

Validation -- Case Studies (Security):

"We decided, instead, to use a set of carefully constructed synthetic attacks, whose design we describe here."

Validation -- Benchmark Suite (Performance):

"To evaluate the effect of IAT checks on realistic benchmarks, we used ten benchmarks from the SPECint-2000 benchmark suite."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Provos, Usenix 2003 -

Where is the policy enforced? -- System:

"The user space policy daemon uses the kernel interface to start monitoring processes and to get information about pending policy decisions or state changes."

When is the policy imposed? -- Dynamic:

"We create policies automatically by running an application and recording the system calls that it executes."

What is protected by the policy? (fine grained) -- Files, Communication, User Data:

"We observe that the only way to make persistent changes to the system is through system calls. They are the gateway to privileged kernel operations. By monitoring and restricting system calls, an application may be prevented from causing harm."

What is protected by the policy? (coarse grained) -- Targeted Application:

See When is the policy imposed?

Requirements of the person applying the sandbox -- Write a policy, Run a Tool:

“She then either improves the current policy by appending a policy statement that covers the current system call, terminates the application, or decides to allow or deny the current system call invocation.”

See Where is the policy enforced? and When is the policy imposted?

Requirements of the application -- No additional requirements:

NOTE: Works as syscall level in the kernel, no specific properties of application required.

Security Policy Type -- User Defined Policy:

See Requirements of the person applying the sandbox

Policy enforcements place in kill chain -- Post-Exploit:

“The policies describe the desired behavior of services or user applications on a system call level and are enforced to prevent operations that are not explicitly permitted.”

See What is protected by the policy? (fine grained)

Policy Management -- Central policy repository:

“This event can install a new policy from the policy database.”

Policy Construction -- Manually Written Policies (mostly tool generated, but user is asked to write some policies at runtime):

See Requirements of the person applying the sandbox

Validation claims -- Security:

“We introduce a system that eliminates the need to run programs in privileged process contexts.”

Validation claims -- Performance:

“We show that Systrace is efficient and does not impose significant performance penalties.”

Validation -- Argumentation (Security):

“To enforce security policies effectively by system call interposition, we need to resolve the following challenges: incorrectly replicating OS semantics, resource aliasing, lack of atomicity, and side effects of denying system calls [18, 34, 37]. We briefly explain their nature and discuss how we address them.”

Validation -- Case Studies (Performance):

“We conduct the microbenchmarks of a single system call by repeating the system call several hundred thousand times and measuring the real, system, and user time. The execution time of the system call is the time average for a single iteration.
As a baseline, we measure the time for a single geteuid system call without monitoring the application. We compare the result with execution times obtained by running the application under Systrace with two different policies.”

Validation -- Benchmark Suite (Performance):

“To assess the performance penalty for applications that frequently access the filesystem, we created a benchmark similar to the Andrew benchmark [22].”

Availability -- Source Code:

“Systrace is currently available for Linux, Mac OS X, NetBSD, and OpenBSD; we concentrate on the OpenBSD implementation.”

Niu, CCS 2013 -

Where is the policy enforced? -- Application:

“Low-level inlined reference monitors weave monitor code into a program for security. To ensure that monitor code cannot be by-passed by branching instructions, some form of control-flow integrity must be guaranteed. Past approaches to protecting monitor code either have high space overhead or do not support separate compilation. We present Monitor Integrity Protection (MIP), a form of coarse-grained control-flow integrity.”

When is the policy imposed? -- Statically:

NOTE: This sandbox adds NaCl dependencies as a runtime dependency. We considered this to be equivalent to requiring a specific compiler runtime (e.g. the MS VC++ runtime) when classifying this paper and did not consider it dynamic policy imposition.

“We have built a MIP toolchain for rewriting and running x86 Linux applications. It operates at assembly level and takes advantage of symbolic information in assembly code for rewriting and for building the chunk table. As a result, it is compatible with any compiler such as GCC.
Fig. 2 visualizes the work flow of MIP’s toolchain. Application source code is compiled into assembly code by a compiler such as GCC. The compiler produces meta information including labels and assembly directives that are embedded in assembly files. Afterwards, MIPRewriter performs assembly-level rewriting by extending the assembly code streamer of LLVM-MC, LLVM’s assembling and disassembling component. MIPRewriter first transforms the assembly file’s content into a stream of instructions, assembly labels and strings.”

What is protected by the policy? (fine grained) -- Code/Instructions:

See Where is the policy enforced

What is protected by the policy? (coarse grained) -- Targeted Application:

See When is the policy imposed?

Requirements of the person applying the sandbox -- Run a Tool:

See When is the policy imposed?

Requirements of the application -- Use a special compiler:

See When is the policy imposed?

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Pre-exploit:

See Where is the policy enforced

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation Claim -- Security, Performance, Applicability:

“We show that this simple idea is effective in protecting monitor code integrity, enjoys low space and execution-time overhead, supports separate compilation, and is largely compatible with an existing compiler toolchain.”

Validation -- Case Study (Applicability, Security):

“We next present a case study in which we implement Software-based Fault Isolation (SFI) on top of MIP. The SFI policy is to restrict control flow within a code region and restrict memory access within a data region.”

Validation -- Benchmark Suite (Security, Performance, Applicability):

“We evaluated the space and time efficiency, and the support for separate compilation of MIP and MIP-based SFI. The experiments were conducted on both x86-32 and x86-64 platforms. In SPEC-CPU2006, only C benchmarks were included, with nine integer performance testing programs and three floating-point performance testing programs.”

Validation -- Public Data (Security, Performance, Applicability):

See other validation quotes.

Availability -- Not Available:

No mention in paper

librando, CCS 2013 -

Where is the policy enforced? -- System:

“We implement this approach as a system-wide service that can simultaneously harden multiple running JITs. It hooks into the memory protections of the target OS and randomizes newly generated code on the fly when marked as executable.”

When is the policy imposed? -- Dynamically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory:

See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Targeted Application:

See Requirements of the person applying the sandbox

Requirements of the person applying the sandbox -- Install a Tool or None:

None in white box mode. Install a tool in black box mode:

“The library diversifies dynamically-generated code under one of the following models (illustrated in Figure 5):

Black box diversification with no assistance from the compiler (the compiler is a black box and the library has no knowledge of compiler internals). The library attaches to the compiler and intercepts all branches into and out of dynamically-generated code, without requiring any changes to compiler internals.

White box diversification with some assistance from the compiler (the library has some knowledge of compiler internals). The code emitter notifies librando through an API when it starts running undiversified code. The library provides the diversified code addresses to the compiler, and the compiler executes diversified code directly. We change all compiler branches into emitted code to use the addresses returned by librando. This approach is intended as a middle ground between the previous model and a manual implementation of randomization for each compiler, and it requires that compiler source code is available.”

Requirements of the application -- None or have source code:

See Requirements of the person applying the sandbox

Security Policy Type -- Fixed Policy:

“Our solution implements two popular defensive techniques: NOP insertion and constant blinding.”

Policy enforcements place in kill chain -- Pre-exploit:

See Security Policy Type

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Security Policy Type

Validation Claim -- Applicability:

“In order to provide “black box” JIT hardening, librando needs to be extremely conservative. For example, it completely preserves the contents of the calling stack, presenting each JIT with the illusion that it is executing its own generated code.”

Validation Claim -- Performance:

“Yet in spite of the heavy lifting that librando performs behind the scenes, the performance impact is surprisingly low. For Java (HotSpot), we measured slowdowns by a factor of 1.15×, and for compute-intensive JavaScript (V8) benchmarks, a slowdown of 3.5×. For many applications, this overhead is low enough to be practical for general use today.”

Validation Claim -- Security:

“Randomization techniques such as constant blinding raise the cost to the attacker, but they significantly add to the burden of implementing a JIT. There are a great many JITs in use today, but not even all of the most commonly used ones randomize their outputs. We present librando, the first comprehensive technique to harden JIT compilers in a completely generic manner by randomizing their output transparently ex post facto. We implement this approach”

Validation -- Case Study (Applicability):

“We demonstrate applicability of black box diversification on two pervasive industrial-strength JIT compilers: Oracle’s HotSpot (used in the Java Virtual Machine) and Google’s V8 (used in the Chrome web browser).”

Validation -- Benchmark Suite (Performance):

“First, we benchmarked V8 using the benchmark suite included with the compiler.”
“Second, we benchmarked the HotSpot client compiler for Java, using the Computer Language Shootout Game benchmarks [8].”

Validation -- Argumentation (Security):

“To randomize code layout, we randomly insert NOP instructions (instructions without effects) into the diversified blocks, between the existing instructions. This technique has been used successfully in other work to change instruction or block alignment to improve performance [12], security [18, 31, 14, 11], or provide contention mitigation [28]. NOP insertion pushes each proper instruction forward by a random offset (the total length of all preceding inserted NOPs), making the location of each instruction more difficult to predict.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Moshchuk, CCS 2013 -

Where is the policy enforced? -- System:

“In this work, we let the OS take over the burden of content isolation from applications. By consolidating content isolation logic in the OS, we reduce the trusted computing base from trusting many applications’ isolation logic to trusting just that of the OS.”
“We present a design that achieves these goals and describe our prototype system called ServiceOS, implemented as a reference monitor between the kernel and applications in Windows.”

When is the policy imposed? -- Hybrid:

“The UI passes a newly typed URL to the monitor, which fetches the content, picks a content processor, and admits this content processing stack into the right isolation container, following the semantics of Sections 4 and 5.”

See Requirements of the application

What is protected by the policy? (fine grained) -- User Data:

“In this paper, we advocate a content-based principal model in which the OS treats content owners as its principals and isolates content of different owners from one another.”

What is protected by the policy? (coarse grained) -- Class of Applications:

NOTE: These applications have to be wrapped or in some circumstances have their code modified. The wrapping can be done automatically in many cases.

“With Drawbridge, we are theoretically able to support all user-space-only Windows applications on our system, though in practice, Drawbridge is not yet mature enough to support certain application features (such as DLLs necessary to run macros in Office documents).”

Requirements of the person applying the sandbox -- Install a Tool:

“Isolation mechanisms. We adopted Drawbridge [36] as our main sandboxing mechanism. Drawbridge can run unmodified Windows applications in a highly isolated mode by refactoring Windows into a library OS and virtualizing all high-level OS components, such as windowing libraries, files, or registry.”

Requirements of the application -- Use the sandbox as library:

“We extended Wordpad with the same ServiceOS support as for Word and Excel. For example, we modified the document parser to recognize special objects representing remote content and to call Embed(), and we modified UI code to make room for embedded content frames when rendering the document.”

Security Policy Type -- Fixed Policy, User-defined policy (optional trust lists):

“A principal is the unit of isolation. Program execution instances with different principal labels are isolated in separate isolation containers.”

“URL resource R at the server. The trust list contains a set of URLs with which R trusts to coexist in the same isolation container. This is one-way trust, meaning that R trusting to coexist with S does not mean that S trusts to coexist with R.”

Policy enforcements place in kill chain -- Post-exploit:

“Word treats documents opened from the web as untrusted and does not run macros by default, but offers users a choice to trust the document via a single click on a yellow security button above it. The attack document tricks the victim to click on this button by pretending to be a greeting card that needs permission to be customized.”

Policy Management -- No management:

None specified

Policy Construction -- Encoded in the logic of the sandbox, manually written:

See Security Policy Type

Validation Claim -- Applicability:

“We demonstrate that ServiceOS is practical by successfully adapting several large applications, such as Microsoft Word, Outlook, and Internet Explorer, onto ServiceOS with a relatively small amount of effort.”

Validation Claim -- Security/Performance:

“Our evaluation shows that ServiceOS eliminates a large percentage of existing security vulnerabilities by design and has acceptable overhead.”

Validation -- Case Studies (Applicability):

See Validation Claims -- Applicability

Validation -- Case Studies (Security):

“We analyzed vulnerabilities of three applications published during 2008-2011 [32, 2], and evaluated whether ServiceOS’s design mitigated them by checking whether each vulnerability was related to parsing or other content processing errors.”
“To verify that our system can indeed stop exploits of content processing flaws we analyzed above, we examined two real-world Word 2010 exploits.”

Validation -- Case Studies (Performance):

“We present results for three applications: Excel 2010, Internet Explorer (IE), and Wordpad. Excel and Wordpad experiments used 10KB, 10MB and 100MB documents; IE was used to open a simple test page on an Intranet web server.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Wartell, CCS 2012 -

Where is the policy enforced? -- Application:

“This paper introduces binary stirring, a new technique that imbues x86 native code with the ability to self-randomize its instruction addresses each time it is launched.”

When is the policy imposed? -- Statically:

“The architecture of STIR is shown in Fig. 3. It includes three main components: (1) a conservative disassembler, (2) a lookup table generator, and (3) a load-time reassembler. At a high level, our disassembler takes a target binary and transforms it to a randomizable representation. An address map of the randomizable representation is encoded into the new binary by the lookup table generator. This is used by the load-time reassembler to efficiently randomize the new binary’s code section each time it is launched.

What is protected by the policy? (fine grained) -- Memory:

“The output is a new binary whose basic block addresses are dynamically determined at load-time. Therefore, even if an attacker can find code gadgets in one instance of the binary, the instruction addresses in other instances are unpredictable.”

What is protected by the policy? (coarse grained) -- Targeted Application:

“This makes it easily deployable; software vendors or end users need only apply STIR to their binaries to generate one self-randomizing copy, and can thereafter distribute the binary code normally.”

Requirements of the person applying the sandbox -- Run a Tool:

See What is protected by the policy? (coarse grained)

Requirements of the application -- None:

“STIR is a fully automatic, binary-centric solution that does not require any source code or symbolic information for the target binary program.”

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation claims -- Performance:

“Evaluation of STIR for both Windows and Linux platforms shows that stirring introduces about 1.6% overhead on average to application runtimes.”

Validation claims -- Applicability:

“It is therefore fully transparent, and there is no modification to the OS or compiler.”

See What is protected by the policy? (coarse grained)

Validation claims -- Security:

See What is protected by the policy? (fine grained)

Validation -- Benchmark Suite/Case Studies (Performance, Applicability):

“On Windows, we tested STIR against the SPEC CPU 2000 benchmark suite as well as popular applications like Notepad++ and DosBox. For the Linux version, we evaluated our system against the 99 binaries in the coreutils toolchain (v7.0) for the Linux version.”

Validation -- Benchmark Suite (Security):

“There are several tools available for such evaluation, including Mona [20] on Windows and RoPGadget [48] on Linux. We used Mona to evaluate the stirred Windows SPEC2000 benchmark programs.”

Validation -- Analytical Analysis (Security):

“On a 64-bit architecture with 14-bit aligned pages and 1 bit reserved for the kernel (i.e., n = 50), the expected number of probes for a g=3-gadget attack is therefore over 7.92 × 1028 (≈ 249 248/2) times greater with STIR than with re-randomizing ASLR.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

FlowFox, CCS 2012 -

Where is the policy enforced? -- Application Host:

“We present FlowFox, the first fully functional web browser that implements a precise and general information flow control mechanism for web scripts based on the technique of secure multi-execution.”

When is the policy imposed? -- Dynamically:

“FlowFox can enforce general information flow based confidentiality policies on the interactions between web scripts and the browser API. Information entering or leaving scripts through the API is labeled with a confidentiality label chosen from a partially ordered set of labels, and FlowFox enforces that information can only flow upward in a script.”

What is protected by the policy? (fine grained) -- Communication, Code/Instructions, User Data:

“Here are some concrete examples of threats that can be mitigated by FlowFox. We will return to these examples further in the paper.

Session Hijacking through Session Cookie Stealing.
...
Malicious Advertisements.
...
History Sniffing and Behavior Tracking.”

What is protected by the policy? (coarse grained) -- Class of Applications:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Write a Policy, Install a Tool:

“Policies are specified as a sequence of policy rules, and associate a level and default value with any given DOM API invocation as follows.”

See Where is the policy enforced?

Requirements of the application -- None:

See Policy Management -- Central Repository

Security Policy Type -- User-defined Policy:

See Requirements of the person applying the sandbox

Policy enforcements place in kill chain -- Post-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- Central Repository (the defined policy applies to all webapps run in FlowFox):

"For an invocation of DOM API method D, if there is a policy rule for D, that rule is used to determine level and default value. If there is no rule in the policy for D, that call is considered to have level L, with default value undefined. The default value for invocations classified at L is irrelevant, as the SME rules will never require a default value for such invocations."

Policy Construction -- Manually written policy:

See Requirements of the person applying the sandbox

Validation claims -- Security:

"We demonstrate how FlowFox subsumes many ad-hoc script containment countermeasures developed over the last years."

Validation claims -- Applicability:

"We also show that FlowFox is compatible with the current web, by investigating its behavior on the Alexa top-500 web sites, many of which make intricate use of JavaScript."

Validation claims -- Performance:

"The performance and memory cost of FlowFox is substantial (a performance cost of around 20% on macro benchmarks for a simple two level policy), but not prohibitive."

Validation -- Benchmark Suite (Performance/Applicability):

"We used the Google Chrome v8 Benchmark suite version 6 5 – a collection of pure JavaScript benchmarks used to tune the Google Chrome project – to benchmark the JavaScript interpreter of our prototype."
"In a first experiment, we measure what impact FlowFox has for users on the visual appearance of websites. We construct an automated crawler that instructs two Firefox browser and one FlowFox browser to visit the Alexa top 500 websites."

Validation -- Case Studies (Performance/Applicability):

"For each category, we randomly picked a prototypical web site from this top-15 list for which we worked out and recorded a specific, complex use case scenario of an authenticated user interacting with that web site."
"We used the web application testing framework Selenium to record and automatically replay six scenarios from our second compatibility experiment for both the unmodifiedMozilla Firefox 8.0.1 browser and FlowFox."

Validation -- Argumentation (Security):

"For (2), – given the size and complexity of the code base of our prototype – we can't formally guarantee the absence of any implementation vulnerabilities. However, we can provide some assurance: the ECMAScript specification assures us that I/O can only be done in JavaScript by means of the browser API. Core JavaScript – as defined by the ECMAScript specification – doesn't provide any input or output channel to the programmer [20, §I]. Since all I/O operations have to pass the translation layer to be used by the DOM implementation (see Section 4.2), we have high assurance that all operations are correctly intercepted and handled according to the SME I/O rules.
Finally, we have extensively manually verified whether FlowFox behaves as expected on malicious scripts attempting to leak information (we discuss some example policies in Section 5.2.2). We believe all these observations together give a reasonable amount of assurance of the security of FlowFox."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Binaries:

"FlowFox is available for download, and can successfully browse to complex web sites including Amazon, Google, Facebook, Yahoo! and so forth."

ScriptGard, CCS 2011 -

Where is the policy enforced? -- Application:

"Instead, we use binary rewriting of server code to embed a browser model that determines the appropriate browser parsing context when HTML is output by the web application."

When is the policy imposed? -- Dynamically:

“Then at runtime, ScriptGard detects which path is actually executed by the program. If the path has been seen in the training phase, then ScriptGard can look up and apply the correct sanitizer sequence from the cache, obviating the need for the full taint flow instrumentation.”

What is protected by the policy? (fine grained) -- Communication, Code/Instructions, User Data:

“Web applications are explosively popular, but they suffer from cross-site scripting (XSS) [4, 32] and cross-channel scripting (XCS) [5]. At the core of these attacks is injection of JavaScript code into a context not originally intended. These attacks lead to stolen credentials and actions performed on the user’s behalf by an adversary.”

What is protected by the policy? (coarse grained) -- Class of Applications:

“To address these errors, we propose ScriptGard, a system for ASP.NET applications which can detect and repair the incorrect placement of sanitizers.”

Requirements of the person applying the sandbox -- Install a Tool:

“Using the encapsulation features offered by the language, we have implemented the taint status for each string object rather than keeping a bit for each character. The taint status of each string object maintains metadata that identifies if the string is untrusted and if so, the portion of the string that is untrusted.”
“ScriptGard employs a web browser to determine the contexts in which untrusted data is placed, in order to check if the sanitization sequence is consistent with the required sequence.
In our implementation, we use an HTML 5 compliant parser used in the C3 browser, that has been developed from scratch using code contracts to be as close to the current specification as possible.”

Requirements of the application -- None:

“Our system requires no changes to web browsers or to server side source code.”

Security Policy Type -- Fixed Policy:

“We develop ScriptGard, a system for detecting these sanitization errors, and repairing them by automatically choosing the appropriate sanitizer.”

Policy enforcements place in kill chain -- Pre-exploit:

See When is the policy imposted?

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposted?

Validation claims -- Performance:

“With our optimizations, when used for mitigation, ScriptGard incurs virtually no statistically sig- nificant overhead.”

Validation claims -- Security:

“While mitigations for cross site scripting attacks have seen intense prior research, we consider both server and browser context, none of them achieve the same degree of precision, and many other mitigation techniques require major changes to server side code or to browsers.”

Validation -- Case Studies (Security):

“We performed our security testing on a set of 53 large web pages derived from 7 sub-applications built on top of our test application. Each page contains 350-900 DOM nodes. Out of 25, 209 total paths exercised, we found context-mismatched sanitization on 1,207 paths ScriptGard analyzed, 4.7% of the total paths analyzed.”

Validation -- Case Studies (Performance):

“We took nine URLs, each of which triggered complicated processing on the server to create the resulting web page. For each URL we first warmed the server cache by requesting the URL 13 times.”

Availability -- Not Available:

No mention in paper

Zeng, CCS 2011 -

Where is the policy enforced? -- Application:

“Static analysis can be used to verify the result of binary rewriting and optimizations. The verification checks whether the rewritten and optimized code obeys the desired security policy, removing the binary rewriter and the optimizer from the TCB.”

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions:

“In many software attacks, inducing an illegal control-flow transfer in the target system is one common step. Control-Flow Integrity (CFI [1]) protects a software system by enforcing a pre-determined control-flow graph.”

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

“Our implementation is built in LLVM 2.8 [20], a widely used compiler infrastructure. We inserted a pass for CFI rewriting, a pass for data sandboxing rewriting and opti-mization, and a pass for CFI and data-sandboxing verification.”

Requirements of the application -- Use special compiler:

“Our ideas have been implemented in LLVM and fully evaluated using benchmark programs.”

Security Policy Type -- Fixed Policy:

“The expected control-flow graph serves as a specification of control transfers allowed in the program. A software-based CFI implementation inserts runtime checks to enforce the specification. The runtime checks will catch and prevent illegal control transfers attempted by attacks.”

Policy enforcements place in kill chain -- Pre-exploit:

See Security Policy Type

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Security Policy Type

Validation claims -- Security, Performance:

“Our results show that the combination of CFI and static analysis has the potential of bringing down the cost of general inlined reference monitors, while maintaining strong security.”

Validation -- Benchmark Suite (Performance):

“On top of CFI, our system adds only 2.7% runtime overhead on SPECint2000 for sandboxing memory writes and adds modest 19% for sandboxing both reads and writes.”

Validation -- Argumentation (Security):

“Our verifier is robust in the sense it can verify many more optimizations, including those we have not implemented.”
“During the development, the verifier helped us catch several implementation errors in early versions of the optimizer; these errors would be hard to find by hand.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Reis, CCS 2011 -

Where is the policy enforced? -- Application Host:

“However, most users access the web with only one browser. We explain the security benefits that using multiple browsers provides in terms of two concepts: entry-point restriction and state isolation. We combine these concepts into a general app isolation mechanism that can provide the same security benefits in a single browser.”

When is the policy imposed? -- Dynamically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Communication, Code/Instructions, User Data:

“Table 1: Cross-origin attacks mitigated by entry-point restriction, Same-origin attacks, such as stored XSS, are not mitigated.”

What is protected by the policy? (coarse grained) -- Targeted Application:

“To remain compatible with web sites that desire this sharing, we employ an opt-in policy that lets web developers decide whether to isolate their site or web application from the rest of the browser.”

Requirements of the person applying the sandbox -- Write a Policy, Install a Tool:

“We implement app isolation in the Chromium browser and verify its security properties using finite-state model checking.”

What is protected by the policy? (coarse grained)

Requirements of the application -- None:

NOTE: Mechanism is only appropriate for applications that don’t do deep linking, but when it can be turned on manually by writing a policy there are no special requirements for the application.

Security Policy Type -- Fixed Policy, User-Defined (enable for site):

“We have shown that a single browser can achieve the security benefits of using multiple browsers, by implementing entry-point restriction and state isolation to isolate sensitive apps.”

See What is protected by the policy?

Policy enforcements place in kill chain -- Pre-exploit/Post-exploit (exploit dependent):

See What is protected by the policy? (fine grained)

Policy Management -- No Management:

None specified

Policy Construction -- Encoded in the logic of the sandbox, Manually Written (enable for site):

See Security Policy Type and What is protected by the policy?

Validation claims -- Security:

"While not appropriate for all types of web sites, many sites with high-value user data can opt in to app isolation to gain defenses against a wide variety of browser-based attacks."

Validation claims -- Performance:

"While extra disk space is required for isolated caches, the overhead is generally far less than using multiple browsers."

Validation -- Proof (Security):

"To evaluate the security benefits of app isolation, we model our proposals in the Alloy language, leveraging previous work on modeling web security concepts in Alloy [9]."

Validation -- Benchmark Suite (Performance):

"We measured the load times of the Alexa Top 100 Web sites with and without entry-point restriction enabled."

Validation -- Case Studies (Performance):

"To see the impact of state isolation, we measured the disk and memory space required for visiting 12 popular sites in their own tabs, similar to the sites used in Figure 4."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Wurster, CCS 2010 -

Where is the policy enforced? -- System:

"Our prototype design consists of two main elements: a kernel extension and a user-space daemon. The user-space daemon is responsible for the bulk of the work, namely, ensuring that one application cannot modify files related to a different application. The kernel is responsible for denying (or forwarding) requests to modify protected file-system objects, by which we mean files (including binaries), directories, symbolic links, and other objects that are part of the file-system."

When is the policy imposed? -- Dynamically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Files:

"We address the problem of restricting root's ability to change arbitrary files on disk, in order to prevent abuse on most current desktop operating systems."

What is protected by the policy? (coarse grained) -- Class of Applications:

"While we focus primarily on installers in this paper (since they perform the vast majority of system configuration changes), the protection mechanism remains in force past install time, restricting system modifications while applications (including Trojans) are running."

Requirements of the person applying the sandbox -- Install a tool:

See Where is the policy enforced?

Requirements of the application -- None:

“Backwards compatibility is therefore critical for incremental deployability. Our prototype did not change the Debian package structure at all, maintaining backwards compatibility with versions of dpkg not designed to work with configd.”

Security Policy Type -- Fixed Policy:

“Our testing confirmed that the above rule set allows package installs and upgrades to be automatically allowed, while providing encapsulation.”

Policy enforcements place in kill chain -- Pre-exploit:

See Where is the policy enforced?

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Security Policy Type

Validation claims -- Security:

“Our architecture exposes a control point available for use to enforce policies that prevent one application from modifying another’s file-system objects.”

Validation claims -- Applicability:

“While our discussion and prototype focus on Linux, we believe the approach can be adapted to Windows, Mac OS X, BSD, and other operating systems. Indeed, configd implemented on Windows could also protect the Windows registry (since it is stored on disk).”

Validation claims -- Performance:

“For day to day operations which do not involve heavy file-system activity, we expect the overhead of configd to be well under 4.8%.”

Validation -- Case Studies (Performance):

“To test the performance of our kernel modifications on file-system intensive day-to-day operations, we performed a complete compile of the Linux 2.6.31.5 kernel.”

Validation -- Case Studies (Security):

“To test how well the mechanism presented in this paper protect a system when exposed to malware, we became root on a system with configd running and kernel protections enabled.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Robusta, CCS 2010 -

Where is the policy enforced? -- Application Host:

Starting from software-based fault isolation (SFI), Robusta isolates native code into a sandbox where dynamic linking/loading of libraries is supported and unsafe system modification and confidentiality violations are prevented. It also mediates native system calls according to a security policy by connecting to Java’s security manager. Our prototype implementation of Robusta is based on Native Client and OpenJDK.

When is the policy imposed? -- Hybrid:

“The runtime overhead of Robusta can roughly be put into two classes. First, there is the SFI cost. For NaCl, this is the cost of masking indirect jump instructions and the cost of making the program properly aligned at 32-byte blocks. The second class of runtime overhead happens during context switches. In Robusta, the execution context may switch between the JVM and the sandbox in a number of situations: when the JVM invokes a native method, the context is switched into the sandbox; when native code finishes execution, the context is switched outside of the sandbox; when native code invokes a JNI call or a system call, the context is

switched outside of the sandbox to invoke trusted wrappers and is then switched back into the sandbox."
"The program was then fed to the NaCl toolchain to produce NaCl-compliant binaries and was run in Robusta."

What is protected by the policy? (fine grained) -- Files, Memory, Communication, Code/Instructions:

See Where is the policy enforced? (SFI gets Memory/Code instructions and the use of the Java sandbox gets the rest).

What is protected by the policy? (coarse grained) -- Class of Applications:

See Where is the policy enforced? (Java applications that use native code)

Requirements of the person applying the sandbox -- Run a Tool, Write a Policy:

See When is the policy imposed? (Need to use NaCl tool chain for builds, write a policy for the Java sandbox)

Requirements of the application -- Use special compiler:

See When is the policy imposed?

Security Policy Type -- Fixed Policy (SFI), User-defined Policy (Java sandbox):

See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit (SFI)/Post (Java sandbox):

See Where is the policy enforced?

Policy Management -- Classes of Applications (Java policies can be applied to codebases that match an abstract description -- e.g. digsig):

See Where is the policy enforced?

Policy Construction -- Encoded in the logic of the sandbox (SFI), Manually written policies (Java sandbox):

See Where is the policy enforced?

Validation claims -- Performance:

“Our prototype implementation of Robusta is based on Native Client and OpenJDK. Experiments in this prototype demonstrate Robusta is effective and efficient, with modest runtime overhead on a set of JNI benchmark programs.”

Validation claims -- Security:

“Robusta can be used to sandbox native libraries used in Java’s system classes to prevent attackers from exploiting bugs in the libraries. It can also enable trustworthy execution of mobile Java programs with native libraries.”

Validation claims -- Applicability:

“The design of Robusta should also be applicable when other type-safe languages (e.g., C#, Python) want to ensure safe interoperation with native libraries.”

Validation -- Argumentation (Security):

“We next discuss what kinds of security policies Robusta enforces despite attacks described in the threat model.
Our discussion will be based on a lightweight formal notation.”

Validation -- Case Studies (Security):

“We created a set of microbenchmarks for testing the functionality and testing the security of Robusta.”

Validation -- Case Studies (Performance):

“Therefore, an interesting question is to explore the relationship between the runtime overhead and how frequent context switches happen. An answer helps to understand what kinds of applications should be put under the control of Robusta. We compiled a set of medium-sized JNI programs, explained as follows.
• Java classes in java.util.zip invoke the popular Zlib C library for performing general-purpose data compression/decompression. We extracted from OpenJDK the Java classes in java.util.zip, the Zlib 1.2.3 library, and the JNI glue code that links Zlib with Java.
• libec is a C library for elliptic curve cryptography. OpenJDK provides JNI bindings for interfacing with the library.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

HookSafe, CCS 2009 -

Where is the policy enforced? -- System (Hypervisor):

"To address the above challenges, in this paper, we present HookSafe, a hypervisor-based lightweight system that can protect thousands of kernel hooks in a guest OS from being hijacked."

When is the policy imposed? -- Dynamically:

"As such, we can relocate those kernel hooks to a dedicated page-aligned memory space and then regulate accesses to them with hardware-based page-level protection."

What is protected by the policy? (fine grained) -- Memory:

"In this paper, we consider kernel data as control data if it is loaded to processor program counter at some point in kernel execution. There are two main types of kernel control data: return addresses and function pointers."

What is protected by the policy? (coarse grained) -- System Level Component:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Install a tool:

See Where is the policy enforced?

Requirements of the application -- No additional requirements:

None stated

Security Policy Type -- Fixed Policy:

"First, an offline hook profiler component profiles the guest kernel execution and outputs a hook access profile for each protected hook. A hook access profile

includes those kernel instructions that read from or write to a hook and the set of values assigned to it. In the next step, a hook's access profile will be used to enable transparent hook indirection. For simplicity, we refer to those instructions that access a hook as Hook Access Points (HAPs).
Second, taking hook access profiles as input, an online hook protector creates a shadow copy of all protected hooks and instruments HAP instructions such that their accesses will be transparently redirected to the shadow copy. The shadow hooks are aggregated together in a central location and protected from any unauthorized modifications."

Policy enforcements place in kill chain -- Pre-exploit:

See Security Policy Type

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Security Policy Type

Validation claims -- Security:

"Our experiments with nine real-world rootkits show that HookSafe can effectively defeat their attempts to hijack kernel hooks."

Validation claims -- Performance:

"We also show that HookSafe achieves such a large-scale protection with a small overhead (e.g., around 6% slowdown in performance benchmarks)."

Validation -- Case Studies (Security):

See Validation claims -- Security

Validation -- Benchmark Suite/Case Studies (Performance):

"We evaluated HookSafe on benchmark programs (e.g., UnixBench [29] and ApacheBench[6]) and real-world applications. Our experimental results show that the performance overhead introduced by HookSafe is around 6%."

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Chang, CCS 2008 -

Where is the policy enforced? -- Application:

“Our system uses a compiler to transform untrusted programs into policy-enforcing programs, and our system can be easily reconfigured to support new analyses and policies without modifying the compiler or runtime system.”

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Files, Memory:

“Our system comes with predefined policies for taint and file disclosure, and our system can be easily extended to handle other problems and security policies without modifying our system implementation.”

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Select a pre-made security policy:

“The input is an untrusted program. The output is an enhanced program that enforces some specified security policy, which is selected by the end-user at compile time.”

Requirements of the application -- Use special compiler:

“The policy itself is defined in an annotation file that describes the policy and the effects of standard library calls on the policy. Thus, the policy is entirely separate from the data flow tracking mechanism, so in addition to the existing security policies that we have already defined, new security policies can be specified without modifying either the compiler or the runtime system."

See Where is the policy enforced?

Security Policy Type -- User-Define Policy:

See Requirements of the application

Policy enforcements place in kill chain -- Pre/Post (policy dependent):

See What is protected by the policy? (fine grained)

Policy Management -- Central Policy Repository (policies easily shared):

See Requirements of the application

Policy Construction -- Manually written policy:

See Requirements of the application

Validation claims -- Performance, Applicability:

"Current taint tracking systems suffer from high overhead and a lack of generality. In this paper, we solve both of these issues with an extensible system that is an order of magnitude more efficient than previous software taint tracking systems and is fully general to dynamic data flow tracking problems."

Validation claims -- Security:

"We now examine the security-related assumptions and advantages of our system."

Validation -- Argumentation (Security):

"Although there are security implications [45] to trusting the compiler, the additional trust required by our approach is mitigated by two factors. First, in typical modern environments, the compiler (usually gcc or some other widely used compiler) is already trusted to compile the server programs that are actually run. Second, our source-to-source translator relies on the user's already trusted compiler for generating binary code."
"The design of our system makes it difficult in practice for an attacker to subvert the enforcement mechanism itself. First, like other compiler-based systems [48,

31], the original program is written before the enforcement code is added, so the original program cannot directly access enforcement data. Moreover, unlike taint...”

Validation -- Proof (security):

“The soundness of our analysis prevents any attacks that violate the policy.”

Validation -- Benchmark Suite (Performance), Case Studies (Security, Performance):

“We verify attack prevention, measure static code expansion, and measure runtime overhead for five open-source server programs and four compute-bound SPECint 2000 benchmarks.”
“We first evaluate our system’s ability to detect attacks. Four of our benchmark programs contain known vulnerabilities that are exploitable.”

Validation -- Argumentation (Applicability):

“Although our current implementation is a source-to-source translator for the C language, our techniques are applicable to other modern languages and even binary code. For example, our static data flow analysis could be implemented in a static binary rewriting system, producing a system that protects binary code from attacks while using static analysis to reduce the runtime cost.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

SOMA, CCS 2008 -

Where is the policy enforced? -- Application Host:

“To evaluate our proposal, we have developed a Firefox SOMA add-on.”

When is the policy imposed? -- Statically:

“To participate in SOMA, browsers have to make minimal code changes and web sites must create small, simple policy files.”

What is protected by the policy? (fine grained) -- Code/Instructions, User Data:

“By requiring site operators to specify approved external domains for sending or receiving information, and by requiring those external domains to also approve interactions, we prevent page content from being retrieved from malicious servers and sensitive information from being communicated to an attacker.”

What is protected by the policy? (coarse grained) -- Target Application:

See What is protected by the policy? (fine grained)

Requirements of the person applying the sandbox -- Write a Policy, Install a Tool:

See Where is the policy enforced? and When is the policy imposed?

Requirements of the application -- None:

NOTE: Site creator has to make a policy and user needs an addon, but no changes need to be made to the app.

Security Policy Type -- User-Defined Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Policies are specific to particular webapps.

Policy Construction -- Manually written policy:

See When is the policy imposed?

Validation claims -- Security:

See What is protected by the policy? (fine grained)

Validation claims -- Applicability:

“SOMA is compatible with current web applications and is incrementally deployable, providing immediate benefits for clients and servers that implement it.”

Validation claims -- Performance:

“SOMA has an overhead of one additional HTTP request per domain accessed and can be implemented with minimal effort by application and web browser developers.”

Validation -- Benchmark Suite (Performance):

“First, we determined the average HTTP request round-trip time for each of 40 representative web sites6 on a per-domain basis using PageStats [9]. We used this per-domain average as a proxy for the time to retrieve a soma-approval from a given domain.”

Validation -- Benchmark Suite (Applicability):

“To test compatibility with existing web pages, the global top 45 sites as ranked by Alexa [2] were visited in the browser with and without the SOMA add-on.”

Validation -- Case Studies (Security):

“In order to verify that SOMA actively blocks information leakage, cross-site request forgery, cross-site scripting, and content stealing, we created examples of these attacks.”

Validation -- Public Data (Performance, Applicability):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Chen, CCS 2007 -

Where is the policy enforced? -- Application Host:

“The basic idea is to introduce domain-specific “accents” to scripts and HTML object names so that two frames cannot communicate/interfere if they have different accents. The mechanism has been prototyped on Internet Explorer.”

When is the policy imposed? -- Dynamically:

“We keep a lookup table in the HTML engine (mshtml.dll) to map each domain name to an accent key. The keys are generated in a Just-In-Time fashion: immediately after the document object is created for each frame, we look up the table to find the key associated with the domain of the frame (if not found, create a new key for the domain), and assign the key to the window object (i.e., the frame containing the document).”
“We observed that internally a common function called by execScript, setTimeout and setInterval is InvokeMemberFunc, and a common function called for all Javascript URL navigations is InvokeNavigation. Therefore, we insert the accenting operation before InvokeMemberFunc and InvokeNavigation.”

What is protected by the policy? (fine grained) -- Code/Instructions, User Data:

“IE implements a security mechanism to guarantee that scripts from one frame can access documents in another frame if and only if the two frames are from the same domain.”

NOTE: The sandbox is hardening this mechanism.

What is protected by the policy? (coarse grained) -- Class of Applications (WebApps):

See Validation Claims -- Applicability.

Requirements of the person applying the sandbox -- Install a Tool (custom browser):

See Where is the policy enforced?

Requirements of the application -- None:

See When is the policy imposed?

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Pre-exploit:

“Without needing an explicit check for the domain IDs, the accenting mechanism naturally implies that two frames cannot interfere if they have different accent keys.”

Policy Management -- No Management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation claims -- Security:

“The evaluation showed that all known crossframe attacks were defeated.”

Validation claims -- Applicability:

“Moreover, because the accenting mechanism only slightly changes the interface between the script engine and the HTML engine, it is fully transparent to web applications.”

Validation claims -- Performance:

“Our stress test showed a 3.16% worst-case performance overhead, but the measurement of the end-to-end browsing time did not show any noticeable slowdown.”

Validation -- Argumentation (Security):

“We now revisit the attack scenarios discussed in Section 4 and demonstrate how the script accenting mechanism can defeat all these attacks. Also, these examples support our argument that the correct implementation of the accenting/de-accenting operations is significantly more robust than that of the current frame-based isolation mechanism.”
“Although the above probing attack seems plausible at the first glance, it is not effective for two reasons. First, we observe that scripts in IE are always represented using wide-characters, which means the string “//” is already four-byte long. It requires 2564 attempts to guess.”

Validation -- Case Studies (Applicability):

"To verify the transparency of our implementation, our modified IE executable has been tested on many web applications. Table 1 shows a number of representative examples. We intentionally selected the web applications with rich user interaction capabilities in order to test the transparency of the mechanism. We observed that all these applications run properly in our IE executable."

Validation -- Case Studies (Performance):

"To measure the upper bound of the performance overhead, we queried window.document.body.innerText for 400,000 times."
"To estimate how the performance overhead affects the end-to-end browsing time, we measured the page initialization time of popular websites."

Availability -- Not Available:

No mention in paper

CANDID, CCS 2007 -

Where is the policy enforced? -- Application:

"Candid consists of two components: an offline Java program transformer that is used to instrument the application, and an (online) SQL parse tree checker."

When is the policy imposed? -- Hybrid:

"In this paper we offer a solution, dynamic candidate evaluation, a technique that automatically (and dynamically) mines programmer-intended query structures at each SQL query location, thus providing a robust solution to the retrofitting problem."

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions:

"A characteristic diagnostic feature of SQL injection attacks is that they change the intended structure of queries issued."

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

See Where is the policy enforced?

Requirements of the application -- Use sandbox as framework/library:

"The automated transformation was implemented for Java byte-code using an extension to the SOOT optimization framework [22]. SOOT provides a three-address intermediate byte-code representation, Jimple, suitable for code analysis and optimization. Class files of the uninstrumented applications were processed using the SOOT framework with CANDID to generate instrumented class files for deployment."
"Immediately preceding this command location, the Candid instrumentation calls the parse tree comparison checker."

Security Policy Type -- Fixed Policy:

"In this section, we formalize SQL injection attacks and, through a series of gradual refinements and approximations, we derive the detection scheme used by Candid. In order to simplify and concentrate on the main ideas in this analytic"

Policy enforcements place in kill chain -- Pre-Exploit:

"As mentioned earlier, we compare the parse trees of the real and candidate queries for attack detection. It is worthwhile to mention here that even the slightest mismatch of the parse trees is detected as an attack."

Policy Management -- No Management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Requirements of the application and Security Policy Type

Validation claims -- Security:

"Candid's natural and simple approach turns out to be very powerful for detection of SQL injection attacks."

Validation claims -- Applicability:

“A fully automated, program transformation mechanism for Java programs that employs this technique, with a discussion of practical issues and resilience to various artifacts of Web applications.”
“Our tool, Candid, is implemented to defend applications written in Java, and works for any web application implemented through Java Server Pages or as Java servlets.”

Validation claims -- Security, Performance:

“A comprehensive evaluation of the effectiveness of attack detection and performance overheads.”

Validation -- Argumentation (Applicability):

“The transformation of programs to dynamically detect intentions of the programmer using candidate inputs as presented above is remarkably resilient in a variety of scenarios. We outline some interesting input manipulations Web applications perform, and illustrate how Candid handles them.”

Validation -- Benchmark Suite (Performance, Security):

“We evaluated our technique using a suite of applications that was obtained from an independent research group [11].”
“The attack test suite was also obtained from the authors of [11]. It consists, for each application, both attack and non-attack inputs that test several kinds of SQL injection vulnerabilities.”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Petroni, CCS 2007 -

Where is the policy enforced? -- System (Hypervisor):

“We have implemented SBCFI as part of the Xen and VMware Workstation virtual machine monitors.”

When is the policy imposed? -- Hybrid:

“As described in Section 2.2, to verify that the kernel’s control-flow has not been modified, the kernel monitor performs two tasks: (1) it validates that the kernel’s text has not been modified and (2) it verifies that all reachable function pointers are in accord with the kernel’s CFG.”
“The generated monitor takes as input trusted copies of the kernel and LKM binaries for runtime comparison (this is shown by the dashed line in Figure 2).”
“With this, a traversal algorithm can start at the roots and transitively follow the pointers embedded in objects it reaches until all function pointers have been discovered.
We gather the necessary inputs via static analysis of the kernel’s source code and compiled binary, and the monitor generator constructs the traversal code in three steps:”

What is protected by the policy? (fine grained) -- Code/Instructions:

See When is the policy imposed?

What is protected by the policy? (coarse grained) -- System Level Component:

“This paper presents a new approach to dynamically monitoring operating system kernel integrity, based on a property called state-based control-flow integrity (SBCFI).”

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- Have source code:

See When is the policy imposed?

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Post-exploit:

“When running at large intervals (currently three seconds or greater), the second “validation” run is commenced within three seconds, rather than waiting for the entire monitor period to expire. This narrows the window between detection and notification, while still allowing the performance tuning to remain in place.”

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation claims -- Security, Performance:

“SBCFI enforcement as part of the Xen and VMware virtual machine monitors. Our implementation detected all the control-flow modifying rootkits we could install, while imposing unnoticeable overhead for both a typical web server workload and CPU-intensive workloads when operating at 10 second intervals.”

Validation -- Case Studies (Security):

“To demonstrate the effectiveness of SBCFI at detecting kernel attacks, we collected as many publicly available rootkits as we could and tested them on our target platform. Of the 25 that we acquired, we were able to install 18 in our virtual test infrastructure.”

Validation -- Benchmark Suite (Performance):

“To evaluate the performance impact of SBCFI monitoring, we measured the performance of the target VMM using subsets of the SPECweb2005 and SPECCPU2006 benchmark suites [33].”

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Abadi, CCS 2005 -

Where is the policy enforced? -- Application:

"Whereas CFI enforcement can potentially be done in several ways, we rely on a combination of lightweight static verification and machine-code rewriting that instruments software with runtime checks."

When is the policy imposed? -- Statically:
See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Code/Instructions:

"The CFI security policy dictates that software execution must follow a path of a Control-Flow Graph (CFG) determined ahead of time. The CFG in question can be defined by analysis—source-code analysis, binary analysis, or execution profiling. For our experiments, we focus on CFGs that are derived by a static binary analysis."

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced? and Requirements of the person applying the sandbox.

Requirements of the person applying the sandbox -- Run a Tool:

"We have implemented inlined CFI enforcement for Windows on the x86 architecture. Our implementation relies on Vulcan [52], a mature, state-of-the art instrumentation system for x86 binaries that requires neither recompilation nor source-code access."

Requirements of the application -- No additional requirements:

See Requirements of the person applying the sandbox.

Security Policy Type -- Fixed Policy:

See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Where is the policy enforced?

Validation claims -- Security:

"Current software attacks often build on exploits that subvert machine-code execution. The enforcement of a basic safety property. Control-Flow Integrity (CFI), can prevent such attacks from arbitrarily controlling program behavior."

Validation claims -- Applicability, Performance:

"Moreover, CFI enforcement is practical: it is compatible with existing software and can be done efficiently using software rewriting in commodity systems."

Validation -- Argumentation (Applicability):

"This system addresses the challenges of machine-code rewriting in a practical fashion—as evidenced by its regular application to software produced by Microsoft."

Validation -- Benchmark Suite (Performance):

"We measured the overhead of our inlined CFI enforcement on some of the common SPEC computation benchmarks [54]."

Validation -- Case Studies (Security):

"Even so, in order to assess the effectiveness of CFI, we examined by hand some well-known security exploits (such as those of the Blaster and Slammer worms) as well as several recently reported vulnerabilities (such as the Windows ASN.1 and GDI+JPEG flaws)."

Validation -- Benchmark Suite (Security):

"For a final set of experiments, we ported to Windows a suite of 18 tests for dynamic buffer-overflow prevention developed by Wilander and Kamkar [61]."

Validation -- Proof (Security):

“With these definitions, we obtain formal results about our instrumentation methods. Those results express the integrity of control flow despite memory modifications by an attacker.”

Validation -- Public Data (Performance):

See other validation quotes. Security experiments not reasonably comparable.

Availability -- Not Available:

No mention in paper

Ringenburg, CCS 2005 -

Where is the policy enforced? -- Application:

“We use static dataflow analysis to determine automatically which addresses should be in the white-lit at any given time. Our source-to-source transformation then uses the knowledge gleaned from static analysis to insert the code that maintains and checks the white-list. Thus the programmer merely needs to update the Makefile and recompile.”

When is the policy imposed? -- Static:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory:

“We propose a simple, flexible, and direct way to control the memory modified by a function (such as printf): An explicit, dynamic white-list of address ranges can control writes that may be unsafe, such as those exploited by format-string attacks.”

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Run a Tool:

See Where is the policy enforced?

Requirements of the application -- Have source code:

See Where is the policy enforced?

Security Policy Type -- Fixed Policy:

"The dynamic nature of our white-lists provides the flexibility necessary to encode a very precise security policy—namely, that %n-specifiers in printf-style functions should modify a memory location x only if the programmer explicitly passes a pointer to x."

Policy enforcements place in kill chain -- Pre-exploit:

"To check the white-list, before executing a %n qualifier the printing function must first verify that the location it is about to write is in a registered address range."
"If a white-list check fails, we choose to abort the program, but other choices are possible (such as silently skipping the write or sending a signal)."

Policy Management -- No Management:

Fixed Policy

Policy Construction -- Encoded in the logic the sandbox:

See Where is the policy enforced?

Validation claims -- Security, Performance:

"We have implemented a white-list based approach to preventing format-string attacks, and determined that the performance overhead is reasonable."

Validation -- Case Studies (Security):

"We tested our approach on four programs with known format string vulnerabilities: ..."

Validation -- Case Studies (Performance):

"To determine our overhead per printf call, we ran a series of simple microbenchmarks consisting of a single loop containing a single sprintf call."

“We also searched for a real, printf-intensive application to test our performance.”

Validation -- Public Data (Security):

See other validation quotes.

Availability -- Source Code:

“Our tool is available for download from our website [26].”

VirtuOS, SOSP 2013 -

Where is the policy enforced? -- System:

“VirtuOS exploits virtualization to isolate and protect vertical slices of existing OS kernels in separate service domains.”
“We have implemented a prototype based on the Linux kernel and Xen hypervisor.”

When is the policy imposed? -- Statically:

“VirtuOS allows its user processes to interact directly with service domains through an exceptionless system call interface [48], which can avoid the cost of local system calls in many cases.”
“VirtuOS intercepts system calls using a custom version of the C library that dispatches system calls to the appropriate service domains.”

What is protected by the policy? (fine grained) -- Files, Memory, Code/Instructions:

“VirtuOS supports failure recovery for any faults occurring in service domains, including memory access violations, interrupt handling routine failure and dead-locks.”

What is protected by the policy? (coarse grained) -- System-level component:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- Use the sandbox as a framework/library:

See When is the policy imposed?

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Post-exploit:

“We designed an experiment to demonstrate that (1) a failure of one service domain does not affect programs that use another one; (2) the primary domain remains viable, and it is possible to restart affected programs and domains.”

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation Claim -- Security:

“Unlike competing solutions that merely isolate device drivers, or cannot protect from malicious and vulnerable code, VirtuOS provides full protection of isolated system components.”

Validation Claim -- Performance/Applicability:

“Thus, VirtuOS may provide a suitable basis for kernel decomposition while retaining compatibility with existing applications and good performance.”

Validation -- Case Studies (Applicability, Performance):

“Our current prototype implementation uses the Linux 3.2.30 kernel for all domains. We tested it with Alpine Linux 2.3.6, x86 64 (a Linux distribution which uses uClibc 0.9.33 as its standard C library) using a wide range of application binaries packaged with that distribution, including OpenSSH, Apache 2, mySQL, Firefox, links, lynx, and Busybox (which includes ping and other networking

utilities). In addition, we tested compilation toolchains including GCC, make and abuild."
"Our first microbenchmark repeatedly executes the fcntl(2) call to read a flag for a file descriptor that is maintained by the storage domain."
"The benchmark writes 16MB in chunks of 32, 64, up to 2MB to a file created in a tmpfs filesystem, which is provided by a native kernel in the baseline case and by a storage domain in VirtuOS."
"We first measured streaming TCP network performance by sending and receiving requests using the TTCP tool [1], using buffer sizes from 512 bytes to 16 KB."
"We also tested VirtuOS with single-threaded, multiple process applications such as Apache 2, and compared performance with native Linux."

Validation -- Benchmark Suite (Performance):

"We use the OLTP/SysBench macrobenchmark [5] to evaluate the performance of VirtuOS's network domain."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Source Code:

"VirtuOS's source code is available at http://people.cs.vt.edu/ rnikola/ under various open source licenses."

Hails, OSDI 2011 -

Where is the policy enforced? -- Application:

"A principled approach to code confinement could allow the integration of untrusted code while enforcing flexible, end-to-end policies on data access. This paper presents a new web framework, Hails, that adds mandatory access control and a declarative policy language to the familiar MVC architecture."

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- User Data:

See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Targeted Application:

See Where is the policy enforce? (Those applications written to use the framework)

Requirements of the person applying the sandbox -- Write a policy, Install a Tool:

“Browser-level confinement As previously noted, wecannot expect all users to install the Hails browser extension which provides confinement in the browser.”

See Where is the policy enforced?

Requirements of the application -- Use the sandbox as a framework/library:

See Where is the policy enforced?

Security Policy Type -- User-defined Policy:

See Where is the policy enforced?

Policy enforcements place in kill chain -- Pre-exploit:

See Where is the policy enforced?

Policy Management -- No management:

NOTE: Uses labels that are necessarily not portable to other applications that use the framework.

Policy Construction -- Manually written policy:

See Where is the policy enforced?

Validation Claim -- Applicability:

“A criticism of past MAC systems has been the perceived difficulty for application programmers to understand the security model. Hails offers a new design point in this space by introducing MAC to the popular MVC pattern and binding access control policy to the model component in MPVC.”

Validation Claim -- Performance:

"We compare the performance of the Hails framework against existing web frameworks, and report on the experience of application authors not involved in the design and implementation of the framework."

Validation Claim -- Security:

"To address these problems, we have developed an alter-
nate approach for confining untrusted apps."

Validation -- Case Studies (Applicability):

"We built and deployed GitStar.com, a Hails platform centered around source code hosting and project management."

Validation -- Benchmark Suite (Performance):

"We use httperf [31] to measure the throughput of each server setup when 100 client connections continuously make requests in a closed-loop—we report the average responses/second."

Validation -- Proof (Security):

"The Hails runtime, including the confinement mechanism, HTTP server, and libraries are part of the TCB. Parts of the system, namely our labels and confinement mechanism, have been formalized in [30, 39-41]. We remark that different from other work, our language-level concurrent confinement system is sound even in the presence of termination and timing covert channels [41]."

Validation -- Public Data (Performance):

See other validation quotes. Security experiments not reasonably comparable.

Availability -- Not Available:

No mention in paper

Dunn, OSDI 2012 -

Where is the policy enforced? -- System:

“Lacuna executes private sessions in a virtual machine (VM) under a modified QEMU-KVM hypervisor on a modified host Linux kernel.”

When is the policy imposed? -- Dynamically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- User Data:

“In this paper, we describe the design and implementation of Lacuna, a system that protects privacy by erasing all memories of the user’s activities from the host machine. Inspired by the “private mode” in Web browsers, Lacuna enables a “private session” abstraction for the whole system.”

What is protected by the policy? (coarse grained) -- Targeted Applications:

“Using a VM helps protect applications that consist of many executables communicating via inter-process communication (IPC), e.g., most modern Web browsers.”

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- None:

See Validation Claim -- Applicability, Performance

Security Policy Type -- Fixed Policy:

See Validation Claim -- Applicability, Performance

Policy enforcements place in kill chain -- Pre-exploit:

“This adversary should not be able to extract any usable evidence of activities conducted in a private session, except (1) the fact that the machine ran a private session at some point in the past (but not which programs were executed during the session), and (2) which devices were used during the session.”

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Validation Claim -- Applicability, Performance

Validation Claim -- Applicability, Performance:

“Lacuna can run unmodified applications that use graphics, sound, USB input devices, and the network, with only 20 percentage points of additional CPU utilization.”

Validation Claim -- Security:

“We design and implement Lacuna, a system that allows users to run programs in “private sessions.” After the session is over, all memories of its execution are erased.”

Validation -- Case Studies (Security):

“Following the methodology of [8], we inject 8-byte “tokens” into the display, audio, USB, network, and swap subsystems, then examine physical RAM for these tokens afterwards. Without Lacuna (but with QEMU and PaX), the tokens are present after the applications exit.”

Validation -- Case Studies (Performance, Applicability):

“We measure the overhead of Lacuna on a number of full-system tasks: watching a 854 × 480 video with mplayer across the network, browsing the Alexa top 20 websites, and using LibreOffice, a full-featured office suite, to create a document with 2,994 characters and 32 images.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Cells, SOSP 2011 -

Where is the policy enforced? -- System:

“This model enables a new device namespace mechanism and novel device proxies that integrate with lightweight operating system virtualization to multiplex phone hardware across multiple virtual phones while providing native hardware device performance.”

When is the policy imposed? -- Statically:
“VPs are created and configured on a PC and downloaded to a phone via USB. A VP can be deleted by the user, but its configuration is password protected and can only be changed from a PC given the appropriate credentials.”

What is protected by the policy? (fine grained) -- Files, Communication, User Data:

“Each VP can be configured to have different access rights for different devices. For each device, a VP can be configured to have no access, shared access, or exclusive access. Some settings may not be available on certain devices; shared access is, for example, not available for the framebuffer since only a single VP is displayed at any time. These per device access settings provide a highly flexible security model that can be used to accommodate a wide range of security policies.”

What is protected by the policy? (coarse grained) -- Targeted Application:

“No access means that applications running in the VP cannot access the given device at any time.”

Requirements of the person applying the sandbox -- Write a Policy, Install a Tool:

See Where is the policy enforced? and What is protected by the policy? (fine grained)

Requirements of the application -- None:

See Validation Claim -- Performance, Applicability

Security Policy Type -- User-defined Policy:

See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Post-exploit:

See Validation Claim -- Security

Policy Management -- Central Policy Repository:

“On the other hand, IT administrators can also create VPs that users can download or remove from their phones, but cannot be reconfigured by users.”

Policy Construction -- Manually Written Policy:

See What is protected by the policy? (fine grained)

Validation Claim -- Security:

“We present Cells, a virtualization architecture for enabling multiple virtual smartphones to run simultaneously on the same physical cellphone in an isolated, secure manner.”
“Cells isolates VPs from one another, and ensures that buggy or malicious applications running in one VP cannot adversely impact other VPs.”

Validation Claim -- Performance, Applicability:

“Cells imposes only modest runtime and memory overhead, works seamlessly across multiple hardware devices including Google Nexus 1 and Nexus S phones, and transparently runs Android applications at native speed without any modifications.”

Validation -- Argumentation (Security):

“Cells uses four techniques to isolate all VPs from the root namespace and from one another, thereby securing both system and individual VP data from malicious reads or writes. First, user credentials, virtualized through UID namespaces, isolate the root user in one VP from the root user in the root namespace or any other VP.”

Validation -- Case Studies (Performance, Applicability):

“We have implemented a Cells prototype using Android and demonstrated its complete functionality across different Android devices, including the Google Nexus 1 [8] and Nexus S [9] phones.”
“We further quantitatively measured the performance of our unoptimized prototype running a wide range of applications in multiple VPs.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

CloudVisor, SOSP 2011 -

Where is the policy enforced? -- System:

"A tiny security monitor is introduced underneath the commodity VMM using nested virtualization and provides protection to the hosted VMs."

When is the policy imposed? -- Dynamically:

"The goal of CloudVisor is to prevent the malicious VM management stack from inspecting or modifying a tenant's VM states, thus providing both secrecy and integrity to a VM's states, including CPU states, memory pages and disk I/O data. CloudVisor guarantees that all accesses not from a VM itself (e.g., the VMM, other VMs), such as DMA, memory dumping and I/O data, can only see the encrypted version of that VM's data."

What is protected by the policy? (fine grained) -- User Data:

"As a result, our approach allows virtualization software (e.g., VMM, management VM and tools) to handle complex tasks of managing leased VMs for the cloud, without breaking security of users' data inside the VMs."

What is protected by the policy? (coarse grained) -- System-level component:

See When is the policy imposed?

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- None:

"Further, CloudVisor can then be separately designed and verified, and be orthogonal to the evolvement of the VMM and management software."

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Post-exploit:

See When is the policy imposed?

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic the sandbox:

“As the essential protection logic for VM resources is quite fixed, CloudVisor can be small enough to verify its security properties (e.g., using formal verification methods [34]).”

Validation Claim -- Security:

See Where is the policy enforced?

Validation Claim -- Performance:

“Performance evaluation shows that CloudVisor incurs moderate slowdown for I/O intensive applications and very small slowdown for other applications.”

Validation -- Argumentation (Security):

“The code base is only around 5.5K lines of code (LOCs), which should be small and simple enough to verify.”

Validation -- Benchmark Suite/Case Studies (Performance):

“The application benchmarks for Linux VMs include: 1) Kernel Build (KBuild) that builds a compact Linux kernel 2.6.31 to measure the slowdown for CPU-intensive workloads; 2) Apache benchmark (ab) on Apache web server 2.2.15 [10] for network I/O intensive workloads; 3) memcached 1.4.5 [19] for memory and network I/O intensive workloads; 4) dbench 3.0.4 [65] for the slowdown of disk I/O workloads. SPECjbb [59] is used to evaluate the server side performance of Java runtime environment in the Windows VM. To understand the overhead in encryption and hashing and the effect of the VM read and VM write optimization, we also present a detailed performance analysis using KBuild and dbench.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Mao, SOSP 2011 -

Where is the policy enforced? -- System:

“This paper proposes LXFI, a system which isolates kernel modules from the core kernel so that vulnerabilities in kernel modules cannot lead to a privilege escalation attack.”

When is the policy imposed? -- Statically:

See Requirements of the application

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:

“To enforce control flow integrity on function returns, the LXFI runtime pushes the return address onto the shadow stack at the wrapper’s entry, and validate its value at the exit to make sure that the return address is not corrupted. ”

What is protected by the policy? (coarse grained) -- System-level component:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Write a policy:

See Requirements of the application

Requirements of the application -- Have source code, Annotated source code, Use special compiler:

“Programmers specify principals and API integrity rules through capabilities and annotations. Using a compiler plugin, LXFI instruments the generated code to grant, check, and transfer capabilities between modules, according to the programmer’s annotations.

Security Policy Type -- User-defined Policy:

See Requirements of the application

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained

Policy Management -- No management:

No official management, annotations are stored in program source code

Policy Construction -- Manually written policy:

See Requirements of the application

Validation Claim -- Security:

See Where is the policy enforced?

Validation Claim -- Performance:

"Stress tests of a network driver module also show that isolating this module using LXFI does not hurt TCP throughput but reduces UDP throughput by 35%, and increases CPU utilization by 2.2–3.7×."

Validation -- Case Studies (Security):

"To answer the first question we inspected 3 privilege escalation exploits using 5 vulnerabilities in Linux kernel modules revealed in 2010 that can lead to privilege escalation. Figure 8 shows three exploits and the corresponding vulnerabilities. LXFI successfully prevents all of the listed exploits as follows."

Validation -- Benchmark Suite (Performance):

"To measure the enforcement overhead, we measure how much LXFI slows down the SFI microbenchmarks [23]."

Validation -- Case Study (Performance):

"To evaluate the overhead of LXFI on an isolated kernel module, we run netperf [14] to exercise the Linux e1000 driver as a kernel module."

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

SPORC, OSDI 2010 -

Where is the policy enforced? -- Application:

“SPORC provides a framework for building collaborative applications that need to synchronize different kinds of state between clients. It consists of a generic server implementation and client-side libraries that implement the SPORC protocol, including the sending, receiving, encryption, and transformation of operations, as well as the necessarily consistency checks and document membership management. To build applications within the SPORC framework, a developer only needs to implement clientside functionality that (i) defines a data type for SPORC operations, (ii) defines how to transform a pair of operations, and (iii) defines how to combine multiple document operations into a single one. The server need not be modified, as it always deals with operations on encrypted data.”

When is the policy imposed? -- Statically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Files, Communication, User Data:

See Where is the policy enforced?

What is protected by the policy? (coarse grained) -- Targeted application:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- None:

See Where is the policy enforced? (NOTE: what the user has to define is not security policy)

Requirements of the application -- Use the sandbox as a framework/library:

See Where is the policy enforced?

Security Policy Type -- Fixed Policy:

NOTE: While the users of applications that use SPORC have some control over a security policy, this is may be required for the use of some application that uses SPORC, not the sandbox itself.

“SPORC provides a generic collaboration service in which users can create a document, modify its access control list, edit it concurrently, experience fully automated merging of updates, and even perform these operations while disconnected. The SPORC framework supports a broad range of collaborative applications. Data updates are encrypted before being sent to a cloud-hosted server. The server assigns a total order to all operations and redistributes the ordered updates to clients. If a malicious server drops or reorders updates, the SPORC clients can detect the server’s misbehavior, switch to a new server, restore a consistent state, and continue. The same mechanism that allows SPORC to merge correct concurrent operations also enables it to transparently recover from attacks that fork clients’ views.”

Policy enforcements place in kill chain -- Pre-exploit/Post-exploit:

See Security Policy Type

Policy Management -- No Management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Security Policy Type

Validation Claim -- Security:

“To overcome this strict tradeoff, we present SPORC, a generic framework for building a wide variety of collaborative applications with untrusted servers.”

Validation Claim -- Applicability:

“SPORC allows concurrent, low-latency editing of shared state, permits disconnected operation, and supports dynamic access control even in the presence of concurrency.”

Validation -- Case Studies (Applicability):

“We demonstrate SPORC’s flexibility through two prototype applications: a causally-consistent key-value store and a browser-based collaborative text editor.”

Validation -- Case Studies (Performance):

“To measure SPORC’s latency, we conducted three minute runs with between one and sixteen clients for both key-value and text editor operations.”

Validation -- Argumentation (Security):

“SPORC clients use sequence numbers and a hash chain to ensure that operations are properly serialized and that the server is well behaved. Every operation has two sequence numbers: a client sequence number (clntSeqNo)...”

NOTE: There are several other similar quotes that pertain to security the application that uses the framework.

Availability -- Not Available:

No mention in paper

Castro, SOSP 2009 -

Where is the policy enforced? -- System:

“We present BGI (Byte-Granularity Isolation), a new software fault isolation technique that addresses this problem. BGI uses efficient byte-granularity memory protection to isolate kernel extensions in separate protection domains that share the same address space.”

When is the policy imposed? -- Statically:

See Requirements of the application

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:

“Previous work on fine-grained memory protection [45] relies on special hardware to achieve good performance. BGI achieves good performance with a software implementation by using a combination of compile time changes to the layout of data, careful design of the data structures that store ACLs, static analysis, and judicious tradeoffs between performance and isolation. Like other systems [11, 42, 43], BGI does not check ACLs before reads, and checks before other types of access are not performed atomically with the access [11]. This enables efficient but still effective isolation of extensions that communicate frequently with the kernel. Additionally, BGI can detect common types of errors inside domains, for example, corruption of return addresses and exception handler pointers, and sequential buffer overflows and underflows.”

What is protected by the policy? (coarse grained) -- System-level component:

See Where is the policy enforced?

Requirements of the person applying the sandbox --

Requirements of the application -- Use special compiler, Use sandbox as framework/library:

“BGI is implemented as a compiler plug-in that generates instrumented code for kernel extensions, and an interposition library that mediates communication between the extensions and the kernel. BGI runs extensions in separate protection domains that share the same address space.”

Security Policy Type -- Fixed Policy:

“BGI runs extensions in separate protection domains that share the same address space. It associates an access control list (ACL) with each byte of virtual memory that lists the domains that can access the byte and how they can access it. Access rights are granted and revoked by code inserted by our compiler and by the interposition library according to the semantics of the operation being invoked.”

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See Security Policy Type

Validation Claim -- Performance, Applicability:

"Our results show that BGI is practical: it can isolate Windows drivers without requiring changes to the source code and it introduces a CPU overhead between 0 and 16%."

Validation Claim -- Security:

"BGI can prevent errors in isolated drivers from corrupting state elsewhere in the operating system, it can prevent attackers from exploiting these errors, and it can recover drivers with errors."

Validation -- Case Studies (Security):

"We injected faults into the fat and intelpro drivers to measure BGI's effectiveness at detecting faults before they propagate outside a domain."
"We selected 50 buggy drivers at random from the set of buggy drivers with escaping blue screens that BGI can contain. We loaded them into a BGI domain with recovery support and then activated the injected faults."

Validation -- Benchmark Suite (Performance):

"We also measured the overhead introduced by BGI. For the disk, file system, and USB drivers, we used the PostMark [23] file system benchmark that simulates the workload of an email server."

Validation -- Case Studies (Performance):

"Next, we measured the overhead introduced by BGI to isolate the network card drivers. For our TCP tests, we used socket buffers of 256KB and 32KB messages with intelpro and socket buffers of 1MB and 64KB messages with xframe.

Validation -- Case Studies (Applicability):

"In our last experiment, we tested 6 of the extensions with BGI. We used existing test suites that achieve good code coverage. BGI found 28 new bugs in these widely used Windows extensions. Table 8 shows the different types of bugs found by BGI."

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Williams, OSDI 2008 -

Where is the policy enforced? -- System:

“Therefore, in our driver architecture, a global, trusted reference validation mechanism (RVM) [3] mediates all interaction between device drivers and devices. The RVM invokes a device-specific reference monitor to validate every interaction between a device driver and its associated device, thereby ensuring the driver conforms to a device safety specification (DSS), which defines allowed and, by extension, prohibited behaviors.”

When is the policy imposed? -- Dynamically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory, Communication, Code/Instructions:

“The RVM protects the integrity, confidentiality, and availability of the system, by preventing:
• Illegal reads and writes: Drivers cannot read or modify memory they do not own.
• Priority escalation: Drivers cannot escalate their scheduling priority.
• Processor starvation: Drivers cannot hold the CPU for more than a pre-specified number of time slices.
• Device-specific attacks: Drivers cannot exhaust device resources or cause physical damage to devices.”
“In addition, given a suitable DSS, an RVM can enforce site-specific policies to govern how devices are used. For example, administrators at confidentiality-sensitive organizations might wish to disallow the use of attached microphones or cameras; or administrators of trusted networks might wish to disallow promiscuous (sniffing) mode on network cards.”
“In sum, this paper shows how to use standard mem-
ory protection and device-specific reference monitors to

execute device drivers with limited privilege and in user
space. ”

What is protected by the policy? (coarse grained) -- System-level component:

“This paper introduces a practical mechanism for executing device drivers in user space and without privilege.”

Requirements of the person applying the sandbox -- Write a Policy:

See What is protected by the policy? (fine grained)

Requirements of the application -- None:

NOTE: Assumes the drivers are written to work with their OS.

Security Policy Type -- Used-defined policy:

See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Pre-exploit/Post-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No Management:

None specified

Policy Construction -- Encoded in the logic of the sanbox/Manually written policy:

See What is protected by the policy? (fine grained) NOTE: Parts of the policy apply to all drivers others are manually written by an admin for a specific driver.

Validation Claim -- Security:

“This paper describes how to move them out of the trusted computing base, by running them without supervisor privileges and constraining their interactions with hardware devices.”

Validation Claim -- Performance:

“These Nexus drivers exhibit performance nearly as fast as earlier in-kernel, trusted drivers.”

Validation -- Case Studies (Performance):

“To gain insight into the performance of our user-space device drivers, we tested each at idle and under load.”

Validation -- Case Studies (Security):

“Nevertheless, to establish the security of our RVM and reference monitors, we used two approaches others have used. First, we simulated unanticipated malicious drivers by randomly perturbing the interactions between drivers and the RVM, resulting in potentially invalid operations being submitted to the reference monitor and possibly to the device. Second, we built specific drivers that perpetrate known attacks on the kernel using interrupt and DMA capabilities.”

Availability -- Not Available:

No mention in paper

Wang, SOSP 2007 -

Where is the policy enforced? -- Application Host:

“Our prototype is based on Internet Explorer 7 (IE) and runs on both Windows XP SP2 and Windows Server 2003 SP1, but our methodology and techniques can also be applied to other browsers.
Instead of modifying IE’s source code directly, we leverage browser extensions and public interfaces exported by IE.”

When is the policy imposed? -- Statically:

“We introduce two new HTML tags for integrators to include unauthorized content: <Sandbox> for private unauthorized content that is hosted at and belongs to the integrator and <OpenSandbox> that may be hosted by any domain:

..

Because the content in <Sandbox> is private, when the src attribute indicates a path from a different domain (principal), the enclosing page cannot access the content in the sandbox; only when the content comes from the same domain can the enclosing page access the content fully. In contrast, for <OpenSandbox>, no matter which domain hosts the content, the enclosing page can access the content fully including the HTML content. We use the term "sandbox" to loosely refer to either element."

What is protected by the policy? (fine grained) -- Communication, Code/Instructions, User Data:

"Among the myriad of operating system issues, we focus on the most imminent needs of today's browsers: abstractions for protection and communication. The goal of protection is to prevent one principal from compromising the confidentiality and integrity of other principals, while communication allows them to interact in a controlled manner."

What is protected by the policy? (coarse grained) -- Targeted Applications:

NOTE: Applications that use specific HTML tags:

"We introduce <Sandbox> and <OpenSandbox> abstractions and a provider-browser protocol to enable content providers to publish and integrators to consume unauthorized content without liability and overtrusting, providing both security and ease in creating client mashups."

Requirements of the person applying the sandbox -- Install a Tool, Write a Policy:

See Where is the policy enforced? and When is the policy imposed?

Requirements of the application -- Annotated source code:

See When is the policy imposed?

Security Policy Type -- User-defined policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Post-exploit:

See When is the policy imposed? NOTE: Doesn't stop an exploit from running, but does stop it from accessing protected content.

Policy Management -- No management:

None specified

Policy Construction -- Manually written policy:

See When is the policy imposed?

Validation Claim -- Applicability:

"We have designed our abstractions to be backward compatible and easily adoptable."

Validation Claim -- Security, Performance:

"Our evaluation shows that our abstractions make it easy to build more secure and robust client-side Web mashups and can be easily implemented with negligible performance overhead."

Validation -- Case Study (Applicability):

"In this section, we first demonstrate the ease of programming robust Web services with MashupOS protection abstractions by showcasing an example application in Section 10.1."

Validation -- Benchmark Suite (Performance):

"We found a JavaScript and DHTML script performance benchmark called BenchJS [2]. The benchmark contains the following 7 JavaScript and DHTML tests."

Validation -- Case Study (Security):

"PhotoLoc puts Google's map library along with the <Div> display element that the library needs into "g.uhtml" and serves "g.uhtml" as private unauthorized content. PhotoLoc's main service page (index.htm) uses <Sandbox> to contain "g.uhtml". PhotoLoc can access everything inside the sandbox, but Google's map library cannot reach out of the sandbox."

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Criswell, SOSP 2007 -

Where is the policy enforced? -- System:

“A virtual machine implementing SVA achieves these goals by using a novel approach that exploits properties of existing memory pools in the kernel and by preserving the kernel’s explicit control over memory, including custom allocators and explicit deallocation.”

When is the policy imposed? -- Hybrid:

“The SAFECode compiler and run-time system together enforce the following safety properties for a complete, standalone C program with no manufactured addresses [11, 12, 10]:”

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:

“SVA aims to enforce fine-grain (object level) memory safety, control-flow integrity, type safety for a subset of objects, and sound analysis.”

What is protected by the policy? (coarse grained) -- System-level component, Class of Applications:

“This paper describes an efficient and robust approach to provide a safe execution environment for an entire operating system, such as Linux, and all its applications.”

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- Use special compiler:

“SVA is also designed to ensure that the (relatively complex) safety checking compiler does not need to be a part of the trusted computing base.”

Security Policy Type -- Fixed Policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation Claim -- Security:

“SVA is able to prevent 4 out of 5 memory safety exploits previously reported for the Linux 2.4.22 kernel for which exploit code is available, and would prevent the fifth one simply by compiling an additional kernel library.”

Validation -- Proof (Security):

“The SVA guarantees provided to a kernel vary for different (compiler-computed) partitions of data, or equivalently, different metapools. The strongest guarantees are for partitions that are proven type-homogeneous and complete. For partitions that lack one or both of these properties, the guarantees are correspondingly weakened.”

Validation -- Case Studies (Security):

“To see how well our system detects exploits that use memory error vulnerabilities, we tried five different exploits on our system that were previously reported for this version of the Linux kernel, and which occur in different subsystems of the kernel.”

Validation -- Public Data (Security):

See other validation quotes.

Availability -- Not Available:

No mention in paper

SecVisor, SOSP 2007 -

Where is the policy enforced? -- System:

"We propose SecVisor, a tiny hypervisor that ensures code integrity for commodity OS kernels."

When is the policy imposed? -- Dynamically:

"We implement SecVisor as a tiny hypervisor that uses hardware memory protections to ensure kernel code integrity. SecVisor virtualizes the physical memory, which allows it to set hardware protections over kernel memory, that are independent of any protections set by the kernel."

What is protected by the policy? (fine grained) -- Code/Instructions:

"In particular, SecVisor ensures that only user-approved code can execute in kernel mode over the entire system lifetime. This protects the kernel against code injection attacks, such as kernel rootkits."

What is protected by the policy? (coarse grained) -- System-level component:

See Where is the policy enforced?

Requirements of the person applying the sandbox -- Install a Tool:

See Where is the policy enforced?

Requirements of the application -- None:

NOTE: Some minor coding changes to port to hypervisor in some cases, but nothing to code.

Security Policy Type -- Fixed Policy:

See When is the policy imposted?

Policy enforcements place in kill chain -- Pre-exploit:

See When is the policy imposted?

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic the sandbox:

See When is the policy imposted?

Validation Claim -- Security:

“Further, SecVisor can even defend against attackers with knowledge of zero-day kernel exploits.”

Validation Claim -- Applicability:

“It is easy to port OS kernels to SecVisor.”

Validation -- Case Study (Applicability):

“We port the Linux kernel version 2.6.20 by adding 12 lines and deleting 81 lines, out of a total of approximately 4.3 million lines of code in the kernel.”

Validation -- Argumentation (Security):

“The hypercall interface is small which reduces the attack surface available to the attacker through the kernel. Also, the parameters passed in each hypercall are well-defined, making it possible for SecVisor to ensure the validity of these arguments.”

Availability -- Not Available:

No mention in paper

Krohn, SOSP 2007 -

Where is the policy enforced? -- System:

“We present a user-space implementation of Flume for Unix, with some extensions for managing data for large numbers of users (as in Web sites). Flume’s user space design is influenced by other Unix systems that build confinement in user space, such as Ostia [12] and Plash [29]. ...

Flume's Linux implementation, like Ostia's, runs a small component in the kernel: a Linux Security Module (LSM) [36] implements Flume's system call interposition (see Section 5.2)."

When is the policy imposed? -- Statically:

"Flume's approach for enhancing Moin's read and write protection is to factor out security code into a small, isolated security module, and leave the rest of Moin largely unchanged. The security module needs to configure only a Flume DIFC policy and then run Moin according to that policy."

What is protected by the policy? (fine grained) -- Communication, User Data:

"As applied to privacy, DIFC allows untrusted software to compute with private data while trusted security code controls the release of that data."

What is protected by the policy? (coarse grained) -- Targeted Application:

"The Flume system provides DIFC at the granularity of processes, and integrates DIFC controls with standard communication abstractions such as pipes, sockets, and file descriptors, via a user-level reference monitor."

Requirements of the person applying the sandbox -- Write a Policy:

See When is the policy imposed?

Requirements of the application -- Use the sandbox as a framework/library:

"Though wrapper programs like wikilaunch could be expressed in other DIFC systems like Asbestos or HiStar, the integration within Moin would be difficult without an application-level API like the one presented here."

Security Policy Type -- User-defined policy:

See When is the policy imposed?

Policy enforcements place in kill chain -- Pre-exploit:

"Moin can pull third-party plug-ins into its address space, but with end-to-end integrity protection, users can enforce that selected plug-ins never touch (and potentially corrupt) their sensitive data, either on input or output."

Policy Management -- No management:

None specified

Policy Construction -- Manually written policy:

See When is the policy imposed?

Validation Claim -- Security:

“Flume eases DIFC’s use in existing applications and allows safe interaction between conventional and DIFC-aware processes.”

Validation Claim -- Applicability:

“ Refinements to Flume DIFC required to build real systems, such as machine cluster support, and DIFC primitives that scale to large numbers of users.”

Validation -- Case Study (Security):

“The most important evaluation criterion for Flume is whether it improves the security of existing systems. Of the five recent ACL bypass vulnerabilities [25, 26], three are present in the MoinMoin version (1.5.6) we forked to create FlumeWiki.”

Validation -- Case Study (Applicability):

“To evaluate Flume’s programmability, we ported a complex and popular application, MoinMoin wiki [22], to the Flume system. MoinMoin is a feature-rich Web document sharing system (91,000 lines of Python code), with support for access control lists, indexing, Web-based editing, versioning, syntax highlighting for source code, downloadable “skins”, etc.”

Availability -- Not Available:

No mention in paper

Castro, OSDI 2006 -

Where is the policy enforced? -- Application:

“To enforce data-flow integrity at runtime, our implementation instruments the program to compute the definition that actually reaches each use at runtime.”

When is the policy imposed? -- Statically:

“We implemented data-flow integrity enforcement using the Phoenix compiler infrastructure [29].”

What is protected by the policy? (fine grained) -- Memory:

“For example, attackers exploit buffer overflows and format string vulnerabilities to write data to unintended locations. We present a simple technique that prevents these attacks by enforcing data-flow integrity.”

What is protected by the policy? (coarse grained) -- Targeted Application:

See When is the policy imposed?

Requirements of the person applying the sandbox -- None:

NOTE: Just use their compiler

Requirements of the application -- Use special compiler:

See When is the policy imposed?

Security Policy Type -- Fixed Policy:

“The analysis relies on the same assumptions that existing compilers rely on to implement standard optimizations. These are precisely the assumptions that attacks violate and data-flow integrity enforcement detects when they are violated.”

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See When is the policy imposed?

Validation Claim -- Applicability, Security, Performance:

"This implementation can be used in practice to detect a broad class of attacks and errors because it can be applied automatically to C and C++ programs without modifications, it does not have false positives, and it has low overhead."

Validation -- Benchmark Suite (Performance):

"We used several programs from the SPEC CPU 2000 benchmark suite to measure the overhead added by our instrumentation. We chose these programs to facilitate comparison with other techniques that have been evaluated using the same benchmark suite, for example, [5]."

Validation -- Benchmark Suite/Argumentation (Applicability):

NOTE: Implicit -- they do their analysis and instrumentation on the high-level intermediate representation in their modified compiler.

Validation -- Benchmark Suite/Case Studies (Security):

"We used a benchmark with synthetic exploits [40] and several exploits of real vulnerabilities in existing programs. This section describes the programs, the vulnerabilities, and the exploits."

Validation -- Public Data (Security, Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

~~Ta-Min, OSDI 2006 -~~

NOTE: Struck due to the requirements of the application. This sandbox requires so much implementation work in the private OS side that it doesn't encompass a reasonable number of real world applications.

~~**Where is the policy enforced? -- System:**~~

~~"This ability is provided by running both commodity and private OSs on a VMM, and using a thin operating system proxy, called Proxos, which we have designed."~~

~~When is the policy imposed? -- Statically:~~

~~“Using high-level system call routing rules specified by the application developer, Proxos transparently routes each system call made by the application to the commodity OS if the request does not need to be trusted, or to the private OS if it does.”~~

~~What is protected by the policy? (fine grained) --~~

~~What is protected by the policy? (coarse grained) -- Targeted Application:~~

~~See When is the policy imposed?~~

~~Requirements of the person applying the sandbox -- Install a Tool, Write a Policy:~~

~~See Where is the policy enforced? and When is the policy imposed?~~

~~Requirements of the application -- Use the sandbox as a framework/library:~~

~~“With this knowledge, the developer identifies the system calls that access these objects and specifies that they are to be forwarded to the private OS using the routing language described in Section 2.3. The private OS methods can be implemented especially for the application by the developer, or even obtained from a library of generic private OS methods provided by a third-party.”~~

~~Security Policy Type -- User-defined Policy:~~

~~See When is the policy imposed?~~

~~Policy enforcements place in kill chain --~~

~~Policy Management --~~

~~Policy Construction -- Manually written policy:~~

~~See When is the policy imposed?~~

~~Validation Claim -- Performance:~~

~~“In addition, applications in Proxos incur only modest performance overhead, with most of the cost resulting from inter-VM context switches.”~~

~~Validation Claim -- Security:~~

~~“Consequently, the application developer is able to remove the entire commodity OS from the TCB of their application while maintaining reasonable performance.”~~

~~Validation --~~

BrowserShield, OSDI 2006 -

Where is the policy enforced? -- Application:

“We avoid this undecidability problem by rewriting web pages and any embedded scripts into safe equivalents, inserting checks so that the filtering is done at run-time. The rewritten pages contain logic for recursively applying run-time checks to dynamically generated or modified web content, based on known vulnerabilities.”

When is the policy imposed? -- Dynamically:

“To this end, we have designed BrowserShield, a system that performs dynamic instrumentation of embedded scripts and that admits policies for changing web page behavior. A vulnerability signature is one such policy, which sanitizes web pages according to a known vulnerability.”

What is protected by the policy? (fine grained) -- Memory, Code/Instructions, User Data:

“The BrowserShield design is focused on HTML, script, and ActiveX controls, and it can successfully handle all 12 of these vulnerabilities. This includes vulnerabilities where the underlying programmer error is at a higher layer of abstraction than a buffer overrun, e.g., a cross-domain scripting vulnerability.”

What is protected by the policy? (coarse grained) -- Class of Applications:

“Because BrowserShield protects web browsers by transforming their inputs, not the browser itself, the BrowserShield logic injector can be deployed at client or edge firewalls, browser extensions, or web publishers that republish third-party content such as ads.”

Requirements of the person applying the sandbox -- Write a Policy, Install a Tool:

See When is the policy imposed? and What is protected by the policy? (coarse grained)

Requirements of the application -- None:

See Validation Claim -- Security

Security Policy Type -- User-defined Policy:

"Policy functions are given the chance to inspect and modify script behavior at all interposition points, including property reads and writes, function and method invocations, and object creations. We also allow policy writers to introduce new global state and functions as part of the global bshield object, or introduce local state and methods for all objects or for specific objects."

Policy enforcements place in kill chain -- Pre-exploit:

See When is the policy imposed?

Policy Management -- No management:

None specified

Policy Construction -- Manually written policy:

See Security Policy Type

Validation Claim -- Security:

"We have designed BrowserShield to adhere to well established principles for protection systems: complete interposition of the underlying resource (i.e., the HTML document tree), tamper-proofness and transparency [3, 10, 33]."

Validation Claim -- Performance:

"We evaluated BrowserShield's performance on realworld pages containing over 125 KB of JavaScript. Our evaluation shows a 22% increase in firewall CPU utilization, and client rendering latencies that are comparable to the original page latencies for most page."

Validation -- Case Studies (Security):

“We evaluated BrowserShield’s ability to protect IE against all critical vulnerabilities for which Microsoft released patches in 2005 [1]. Of the 29 critical patches that year, 8 are for IE, corresponding to 19 IE vulnerabilities.”

Validation -- Case Study (Performance):

“We evaluated BrowserShield’s performance by scripting multiple IE clients to download web pages (and all their embedded objects) through an ISA server running the BrowserShield firewall plugin.”
“We designed microbenchmarks to measure the overhead of individual JavaScript operations after translation.”
“We designed macrobenchmarks to measure the overall client experience when the BrowserShield framework is in place.”

Validation -- Public Data (Security):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Zeldovich, OSDI 2006 -

Where is the policy enforced? -- System:

“HiStar is a new operating system designed to minimize the amount of code that must be trusted. HiStar provides strict information flow control, which allows users to specify precise data security policies without unduly limiting the structure of applications.”

When is the policy imposed? -- Statically:

See Where is the policy imposed?

What is protected by the policy? (fine grained) -- Files, Memory, Communication, Code/Instruction, User Data:

See Validation Claim -- Security, Performance

What is protected by the policy? (coarse grained) -- Class of Applications:

"HiStar tracks and enforces information flow using Asbestos labels [5]. All operating system abstractions are layered on top of six low-level kernel object types described in the next section—threads, address spaces, segments, gates, containers, and devices. Every object has a label. The label specifies, for each category of taint, whether the object has untainting privileges for that category (threads and gates can have such privileges), and, if not, how tainted the object is in that category."

Requirements of the person applying the sandbox -- Write a Policy:

"The equivalent of setting Unix permissions bits is for Bob to allocate two categories, br and bw , to restrict read and write access to his files, respectively. Bob labels his data {br 3, bw 0, 1}."

Requirements of the application -- None:

NOTE: To most effectively use HiStar an application needs to be decomposed into components of different privileges, but this is not a requirement to use HiStart.

Security Policy Type -- User-defined Policy:

See Requirements of the person applying the sandbox

Policy enforcements place in kill chain -- Pre/Post-exploit:

See What is protected by the policy? (coarse grained)

Policy Management -- No management:

"SELinux [11] lets Linux support MAC; like most MAC systems, policy is centrally specified by the administrator. In contrast, HiStar lets applications craft policies around their own categories of information."

Policy Construction -- Manually written policy:

See Requirements of the person applying the sandbox

Validation Claim -- Security, Performance:

“HiStar’s security features make it possible to implement a Unix-like environment with acceptable performance almost entirely in an untrusted user-level library. The system has no notion of superuser and no fully trusted code other than the kernel.”

Validation -- Case Studies (Security):

“This section presents some applications that take advantage of HiStar to provide security guarantees not achievable on typical Unix systems.”

Validation -- Argumentation (Security):

“An object’s label controls information flow to and from the object. In particular, the kernel interface was designed to achieve the following property:
The contents of object A can only affect object B if, for every category c in which A is more tainted than B, a thread owning c takes part in the process.”

Validation -- Benchmark Suite/Case Studies (Performance):

“To evaluate the performance of specific aspects of Hi-Star, we chose four microbenchmarks: LFS small-file and large-file benchmarks [20], an IPC benchmark which measures the latency of communication over a Unix pipe, and a fork/exec benchmark that measures the latency of executing /bin/true using fork and exec.”
“For an application-level benchmark, we built the HiStar kernel using GNU make 3.80 and GCC 3.4.5 on the three operating systems; Figure 13 summarizes the results.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

XFI, OSDI 2006 -

Where is the policy enforced? -- Application:

“For this purpose, XFI combines static analysis with inline software guards and a two-stack execution model. We have implemented XFI for Windows on the x86 architecture using binary rewriting and a simple, stand-alone verifier; the implementation's correctness depends on the verifier, but not on the rewriter."

When is the policy imposed? -- Static:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Memory, Code/Instructions:

"An XFI module complies with the following policy in its interactions with its system environment:
P1 Memory-access constraints: Memory accesses are either into (a) the memory of the XFI module, such as its global variables, or (b) into contiguous memory regions to which the host system has explicitly granted access. ...
P2 Interface restrictions: Control can never flow outside the module's code, except via calls to a set of prescribed support routines, and via returns to external call-sites. ...
P3 Scoped-stack integrity: The scoped stack is always
well formed. ...
P4 Simplified instruction semantics: Certain machine-code instructions can never be executed.
P5 System-environment integrity: Certain aspects of the system environment, such as the machine model, are subject to invariants. For instance, the x86 segment registers cannot be modified, nor can the x86 flags register—except for condition flags, which contain results of comparisons.
P6 Control-flow integrity: Execution must follow a static, expected control-flow graph, even on computed calls and jumps.
tions must return to their callers.
P7 Program-data integrity: Certain module-global and function-local variables can be accessed only via static references from the proper instructions in the XFI module."

What is protected by the policy? (coarse grained) -- Targeted Application:

"We produce XFI binary modules from Windows x86 executables with a rewriter based on the Vulcan library [37]; XFI rewriters could as easily be created using similar libraries for other architectures [37, Section 7]."

Requirements of the person applying the sandbox -- Run a tool:

See What is protected by the policy? (coarse grained)

Requirements of the application -- Have compiler introduce metadata:

“Although our x86 rewriter requires neither recompilation nor source-code access, it makes use of debug information (PDB files), for instance to distinguish code from data.”

Security Policy Type -- Fixed Policy:

See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Pre-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See What is protected by the policy? (fine grained)

Validation Claim -- Applicability:

“We have applied XFI to software such as device drivers and multimedia codecs.”

Validation Claim -- Security, Performance:

“The resulting modules function safely within both kernel and user-mode address spaces, with only modest enforcement overheads.”

Validation -- Argumentation (Security):

“As discussed in Section 3, the correctness of XFI protection depends on the load-time verification of XFI modules. Our verifier was written from scratch and is self-contained. In particular, it is independent from our rewriter, and from any specific strategy for creating XFI modules. It is 3000 lines of straightforward, commented C++ code, most of which are tables for x86 opcode decoding. The verifier needs only a basic understanding of x86 behavior, modeling nothing more complex than integer comparisons and how instructions copy registers. Its simplicity contributes to our confidence in the verifier.”

Validation -- Case Studies (Applicability):

“XFI has a wide range of potential applications. To date, we have applied our XFI implementation to dynamic libraries, device drivers, and multimedia codecs. In this section we describe some of these applications; for brevity, we focus on device drivers.”

Validation -- Benchmark Suite/Case Studies (Performance):

“We applied our x86 XFI implementation to WDF device drivers, SFI benchmarks [36], the JPEG decoder [20], and Mediabench kernels [22], all compiled using Microsoft VC++ 8.0, with optimizations.”

Validation -- Public Data (Performance):

See other validation quotes.

Availability -- Not Available:

No mention in paper

Efstathopoulos, SOSP 2005 -

Where is the policy enforced? -- System:

“Asbestos, a new prototype operating system, provides novel labeling and isolation mechanisms that help contain the effects of exploitable software flaws.”

When is the policy imposed? -- Statically:

See Requirements of the application

What is protected by the policy? (fine grained) -- Files, Communication, User Data:

“Applications can express a wide range of policies with Asbestos’s kernel-enforced label mechanism, including controls on inter-process communication and system-wide information flow. A new event process abstraction provides lightweight, isolated contexts within a single process, allowing the same process to act on behalf of multiple users while preventing it from leaking any single user’s data to any other user.”

What is protected by the policy? (coarse grained) -- Targeted Application:

See Requirements of the application

Requirements of the person applying the sandbox -- None:

See Requirements of the application (the policy is set by the application dev and carried with the application)

Requirements of the application -- Use the sandbox as a framework/library:

“Figure 3 summarizes the notation developed in earlier sections, and Figure 4 gives the final versions of the label operations associated with the send, new port, and set port label system calls.”
“Appropriate access is defined by an application policy: the application defines which of its parts should be isolated, and how.”

Security Policy Type -- Application-defined Policy:

See Requirements of the application

Policy enforcements place in kill chain -- Post-exploit:

See Where is the policy enforced?

Policy Management -- No management:

None specified

Policy Construction -- Encoded in the logic of the application:

See Requirements of the application

Validation Claim -- Security:

See Where is the policy enforced?

Validation Claim -- Performance:

“A Web server that uses Asbestos labels to isolate user data requires about 1.5 memory pages per user, demonstrating that additional security can come at an acceptable cost.”

Validation -- Case Study (Performance):

“The performance measurements were conducted on a gigabit local network with a Linux HTTP client generating requests.”

Availability -- Not Available:

No mention in paper

Terra, SOSP 2003 -

Where is the policy enforced? -- System:

“Terra achieves this synthesis by use of a trusted virtual machine monitor (TVMM) that partitions a tamper-resistant hardware platform into multiple, isolated virtual machines (VM), providing the appearance of multiple boxes on a single, general-purpose platform.”

When is the policy imposed? -- Dynamically:

See Where is the policy enforced?

What is protected by the policy? (fine grained) -- Files, Communication, User Data:

“To each VM, the TVMM provides the semantics of either an “open box,” i.e. a general-purpose hardware platform like today’s PCs and workstations, or a “closed box,” an opaque special-purpose platform that protects the privacy and integrity of its contents like today’s game consoles and cellular phones.”

What is protected by the policy? (coarse grained) -- Class of Applications:

NOTE: Terra is general purpose and can therefore be used for essentially any software system.

“The TVMM mechanisms allow Terra to partition the platform into multiple, isolated VMs. Each VM can tailor its software stack to its security and compatibility requirements.”

Requirements of the person applying the sandbox -- Install a Tool:

See What is protected by the policy? (coarse grained)

Requirements of the application -- None:

See What is protected by the policy? (coarse grained)

Security Policy Type -- Fixed Policy:

See What is protected by the policy? (fine grained)

Policy enforcements place in kill chain -- Post-exploit:

See What is protected by the policy? (fine grained)

Policy Management -- No management:

Fixed Policy

Policy Construction -- Encoded in the logic of the sandbox:

See What is protected by the policy? (fine grained)

Validation Claim -- Security, Applicability:

“Our architecture, called Terra, provides a simple and flexible programming model that allows application designers to build secure applications in the same way they would on a dedicated closed platform.”

Validation -- Argumentation/Case Studies (Security, Applicability):

“In this section we describe the Terra prototype and provide an in-depth discussion of several applications that we built using the prototype. We also look at how these applications demonstrate the capabilities and the limitations of the closed-box abstraction that Terra provides.”

Availability -- Not Available:

No mention in paper