Where is the policy enforced?

e.g. with CFI policy is enforced by instructions are placed inside the application, Same Origin Policy is at Browser level and not something a webapp does itself.

- **System (OS/Hypervisor)** -- The policy is enforced by a system component and enforcement of the particular policy is dependent on the presence of the sandbox component (e.g. enforcement of the policy doesn't travel with the computation if it's migrated to another system)
    - e.g. ASLR, DEP, chroot, TxBox (IEEE 2011), Giuffrida (Usenix 2012)
    - "TxBox consists of a relatively simple, policy-agnostic security monitor running in the OS kernel and a user-level policy manager. " --TxBox (IEEE 2011) ¶ 2, § 1
- **Application** -- The policy is enforced by the application itself because the application has been instrumented with the enforcement code. The policy is enforced on any system the application runs on.
    - e.g. SFI/CFI, Safe Loading (IEEE 2012), Zhang (Usenix 2013), Sehr (Usenix 2010)
    - e.g. BLUEPRINT (IEEE 2009) (webapps)
    - "Our implementation uses a source-to-source transformation on C programs. Note that a particular randomization isn't hardcoded into the transformed code. Instead, the transformation produces a self-randomizing..." --Bhatkar (Usenix 2005) ¶ 2, § 1.1
- ~~**Web Application -- The policy is enforced by the application itself because the application was either rewritten to enforce the policy or utilizes a sandboxing framework/library.**~~
    - ~~**e.g. BLUEPRINT (IEEE 2009)**~~Merged with application category because we made browser category more generic to include Java
- **Application Host (e.g. browser)** -- The host enforces the policy for any sandboxed application that runs in it.
    - e.g. Same Origin Policy -- automatically applied to all web apps
        - Pivot (IEEE 2014)
    - e.g. Content Security Policy -- user must set headers to activate and configure it
    - e.g. Java
    - e.g. Adobe Reader
    - e.g. Native Client (IEEE 2009)
    - "We present librando, the first comprehensive technique to harden JIT compilers in a completely generic manner by randomizing their output transparently ex post facto." --librando (CCS 2013) ¶ 3, Abstract

~~Data~~
~~User Input?~~

When is the policy imposed (maybe applied instead?)?

- **Dynamic** -- The policy is imposed at runtime
    - e.g. through the use of a runtime monitor or through the use of dynamic binary rewriting
    - e.g. W^X
    - “It hooks into the memory protections of the target OS and randomizes newly generated code on the fly when marked as executable.“ --librando (CCS 2013) ¶ 3, Abstract
- **Static** -- The policy is encoded in the computation itself by modifying the computation before it is run.
    - Compile (compiler inserts checks)
    - Retrofit (binary rewriting -- often to add dynamic checks, insert checks without source code)
    - e.g. SFI/CFI
    - e.g. Chang (CCS 2008)
    - “Our implementation uses a source-to-source transformation on C programs." --Bhatkar (Usenix 2005) ¶ 2, § 1.1
- **Hybrid** -- Information is gathered statically to aid in imposing the policy and/or policy is partially imposed statically. Policy is not fully imposed until runtime.
    - e.g. Relocation table added by compiler, but code is only randomized at runtime
    - e.g. CANDID (CCS 2007)
    - “Our binary-rewriting tools analyze binaries and add system call location information to them, without requiring 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.” --Linn (Usenix 2005) ¶ 4 § 1

What is protected by the policy? (Fine Grained)

- **Files** -- access to the filesystem is limited by the policy
    - e.g. chroot
- **Memory** -- memory corruption is mitigated by the policy or memory access is restricted.
    - e.g. ASLR, Cling (Usenix 2010)
- **Communication** --network and inter-process communications are constrained by the policy
    - e.g. Using the Java SM to constrain what addresses and ports an outgoing socket can connect to
    - e.g. Encrypt data on a previously plaintext channel

    - e.g. Since Windows Vista there have been constraints on when a process can send another process a window message to prevent shatter attacks
- **Code/Instructions** -- control flow and code integrity are defended by the policy
    - e.g. CFI (intent to is to ensure that the designed control flow is the one that is actually executed)
    - e.g. DEP/X^W to prevent memory pages from being both writable and executable
- **User Data** -- a security policy is imposed on user input itself or interaction with user data
    - e.g. BLUEPRINT (IEEE 2009) and CLAMP (IEEE 2009)

What is protected by the policy? (Coarse Grained)

- **Targeted Application** -- the policy is enforced on user selected applications to defend or limit those applications' behaviour.
    - e.g. Native Client (IEEE 2009)
- **Class of Applications** -- the policy is enforced on applications that meet an abstract description (e.g. all python applications, all native applications, applications with debug information, etc.)
    - e.g. MapBox (not in our set right now), Tang (CCS 2011)
- **System Level Component** -- the policy is intended to mitigate system-level vulnerabilities and does not have a direct security impact on userland.
    - Kernel
        - KCoFI (IEEE 2014), kGuard (Usenix 2012)
    - Hypervisor
        - Hypersafe (IEEE 2010)

~~How is the policy enforced?~~

~~CFI~~
~~SFI~~
~~Full System Virtualization~~
~~… What else?~~

We think the interesting points here will come out in other meta-categories and that it will be difficult to come up with interesting categories here

---

Requirements of person applying the sandbox

- **Select a pre-made security policy** - the user selects a security policy to apply from a list maintained by the sandbox creators.
    - e.g. AppArmor
- **Write a security policy** - the user must create a security policy using the primitives provided by the sandbox
    - e.g. BLUEPRINT, custom Java SecurityManager
- **Run a tool -** the user runs a tool usually providing the to-be sandboxed application as input or installs a component that functions as the sandbox for all targeted applications.
    - e.g. Compact Control Flow Integrity (IEEE 2013)
- **Install a tool** - User must install the sandbox, but all targeted applications are encapsulated after that (e.g. install a hypervisor, install a browser addon, etc.).
    - e.g. Wurster (CCS 2010)
- **None** - the sandbox requires no effort from the user.
    - e.g. ASLR

Requirements of the application

- **Annotated source code** - the source code of the application must include sandbox-specific annotations. These annotations usually specify the security policy.
    - e.g. JIF
- **Have source code** - the source code of the application must be available to the sandbox, usually for analysis.
    - e.g. Compact Control Flow Integrity (IEEE 2013)
- **Use special compiler** - the source code must be translated into machine code with a sandbox-specific compiler.
    - e.g. NaCl (IEEE 2009)
    - Any sandbox in this category also occupies the “Have source code” category.
- **Have compiler-introduced metadata** (e.g. relocation, debug, etc.) - the binary must include metadata, introduced by a standard compiler (e.g. GCC), for use by the sandbox.
    - e.g. ASLR
- **Use sandbox as framework/library** - the application must implement the security policy by writing a plugin to the sandbox’s framework or calling the sandbox’s library.
    - e.g. BLUEPRINT, NaCl -- Secure Runtime, Conscript, Flexible Access Control for JavaScript -- Misc Tab)
    - Interesting that this category appears primarily (only?) in web settings. Probably because other kinds of applications have the full power of the OS and fully capable programming languages, but in the web they can only do the things a modern browser and supported protocols allow. These sandboxes give webapps the power back that a desktop application would already have
- **No additional requirements** - the application does not need to prepare to be sandboxed in any way.

    - e.g. virtualization, Pappas (IEEE 2012)

Security Policy Type

- **Fixed Policy -** the same policy is applied to every application sandboxed.
    - e.g. ASLR
- **User-defined Policy** - the security policy applied by the sandbox is selected or created by the applier of the sandbox.
    - e.g. AppArmor
- **Application-defined Policy** - the policy is embedded in the sandbox application. The policy cannot be changed by the user.
    - e.g. BLUEPRINT

~~Classes of Policies (e.g. all python apps, all e-commerce websites)~~ (struck here because it's actually about policy management)

Policy enforcements placement in the kill chain

- **Pre-exploit** - the policy is enforced before an exploit can run (an in-scope exploit can't successfully run)
    - Zhang (IEEE 2013), GATEKEEPER (Usenix 2009)
- **Post-exploit** - the policy is enforced after an exploit is run: it can limit the behavior of the exploit, but it can't stop the exploit from fully executing unless the exploit is dependent on a constrained behavior
    - AppArmor

Policy Management

- **Central policy repository** - security policies are available for download from a repository maintained by the sandbox's owners or sharing of policies is otherwise trivially possible due to how the sandbox stores the policy.
    - e.g. AppArmor, Safety Oriented Platform for Web Applications -- IEEE 2006)
- **Classes of policies** (e.g. policy applies to all python apps) - policies can be applied to user-defined categories of applications.
    - e.g. MapBox
- **No management** - policies are not managed by the sandbox. Sharing policies, if possible, is the responsibility of sandbox users.
    - e.g most sandboxes, ASLR

~~Policy Level~~

~~Application Specific~~
~~System (Struck because captured by Security Policy Tyoes)~~

Policy Construction

- **Encoded in the logic of the sandbox** -- The tool that applies the sandbox follows a process for encoding policy enforcement checks into a computation
    - (e.g. all CFI/SFI tools)
- **Encoded in the logic of the application** - the security policy is implemented by the application code.
    - e.g. BLUEPRINT
- **Manually written policies** -- the user personally writes the policy that is to be enforced in some policy language
    - AppArmor, Provos (Usenix 2003 -- systrace), GATEKEEPER (Usenix 2009)
    - Only class that can have management

Validated claims

- **Performance** -- the sandbox does not substantially reduce security.
    - e.g. TxBox (IEEE 2011)
- **Security** -- the sandbox mitigates some vulnerabilities or encapsulates some threats.
    - e.g. Pappas (IEEE 2012)
- **Applicability** -- the sandbox applies to more applications within the “type” of application targeted than previous sandboxes (e.g. works in binaries without any metadata where metadata was previously required) or the applied techniques can be translated to other systems.
    - e.g. TxBox (IEEE 2011)

Validation

- **Benchmark suite** -- a standard benchmark suite is run within the sandbox to generate results that are easily comparable to other tools. We include benchmark suite subsets in this category (e.g. 10 programs from SPEC CPU 2000).

    - e.g. TxBox (IEEE 2011)
- **Case Studies** -- the claims are demonstrated through case studies that demonstrate the use or efficacy of the sandbox.
    - e.g. TxBox (IEEE 2011)
- **Argumentation** -- an informal argument is made to demonstrate the claims are met.
    - e.g. Cappos (CCS 2010) -- see section 7.2
- **Analytical Analysis** -- a structured, mathematical analysis is presented to demonstrate the efficacy of a technique.
    - e.g. Giuffrida (Usenix 2012) -- see section 7.3
- **Proof** -- mathematical proof is presented to demonstrate the claims are met.
    - e.g. subset of KCoFI (IEEE 2014) is formally verified
- **Public Data** -- the sandbox was validated using public data or code. Cases where public code was modified or ported in non-public ways are excluded.

Availability

- **Source Code** -- the source code the sandbox is available for anyone to download.
- **Binaries** -- a compiled version of the sandbox is available for anyone to download.
- **Not Available** -- the source code or binaries have not been made available or no mention of their availability was mde.
