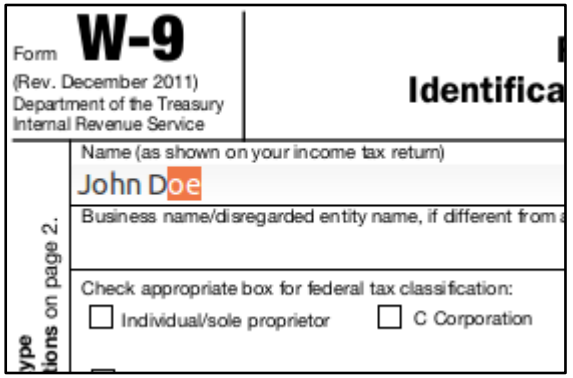
Carnegie Mellon

In-Nimbo Sandboxing

Michael Maass, Bill Scherlis,
and Jonathan Aldrich
{mmaass, wls, aldrich}@cs.cmu.edu

Let's start by looking at a motivating security problem: viewing PDFs without compromising the host machine

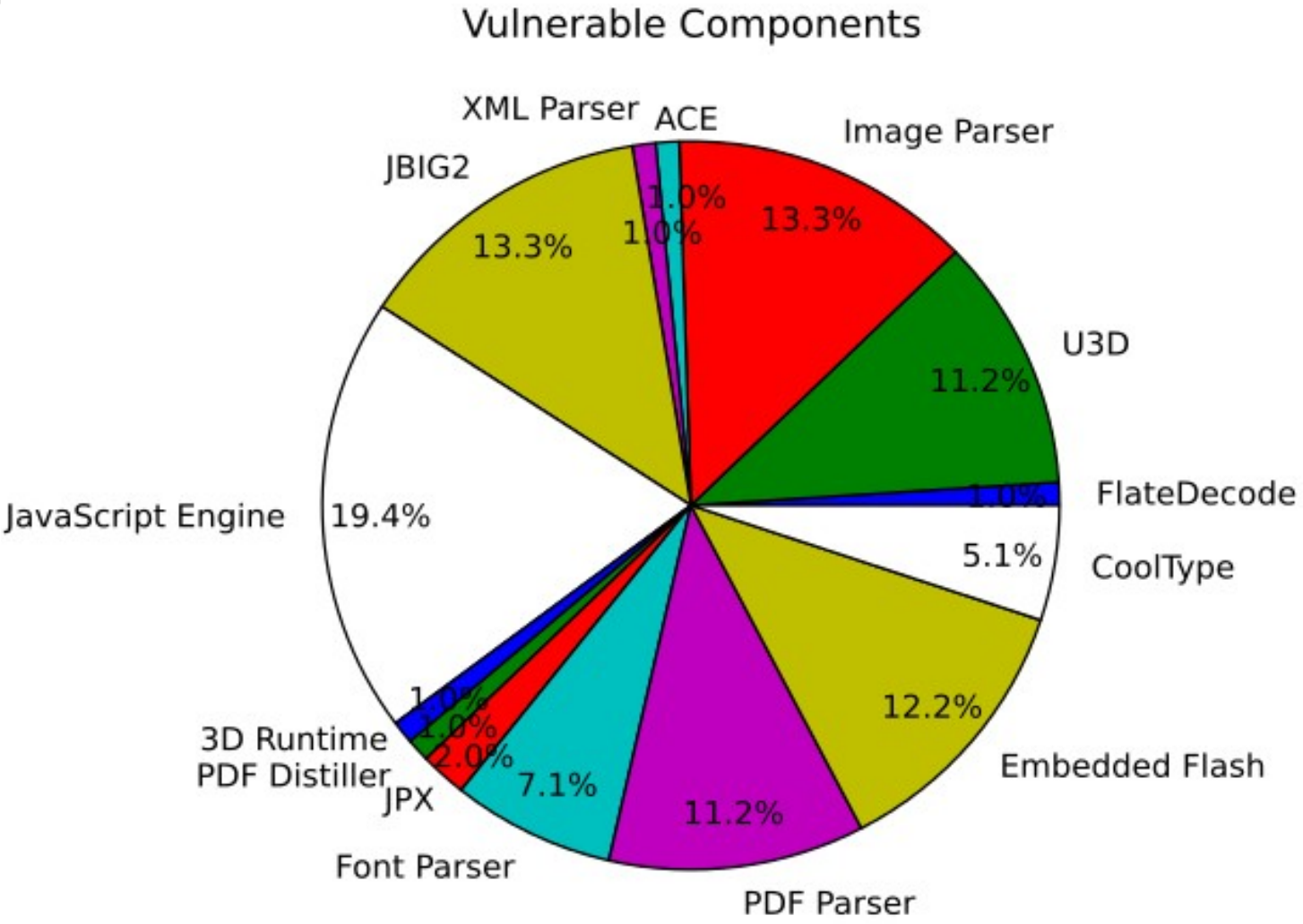
"Exotic" Features for Enterprises


Form **W-9**
(Rev. December 2011)
Department of the Treasury
Internal Revenue Service
Identifica
Name (as shown on your income tax return)
John Doe
Business name/disregarded entity name, if different from
Check appropriate box for federal tax classification:
Individual/sole proprietor
C Corporation
ype
tions on page 2.

- Embed fly-through 3-D models
- Embed full videos
- Forms with complicated workflows
- Digitally sign document subset with smart cards
- Embed obfuscated code (JavaScript, ActionScript, etc.)

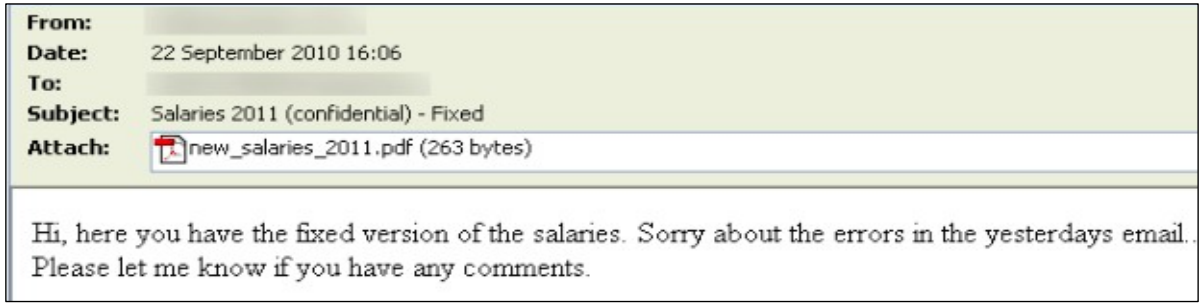
- **PDF is an <u>ACTIVE</u> container and document format**

PDF – An Exploit Vector



Typical Kill Chain (Spear Phishing)

- **Reconnaissance** – attacker finds email addresses for people holding critical information and determines what PDF reader they use
- **Weaponization** – attacker finds exploitable vulnerability in the PDF reader utilized by the intended victims
- **Delivery** – attacker emails victims a crafted PDF that exploits the vulnerability in the victim's PDF reader
  - These emails are usually spoofed to look legitimate



(wired.com)

- **Exploitation** – the victim opens the PDF, which then exploits the reader
- **C2** – the exploit drops a backdoor and executes it
- **Exfiltration** – the desired data is sent to an attacker controlled server or email address

It is difficult to **verify** that a PDF is non-malicious.

Building a **verified** reader is intractable.

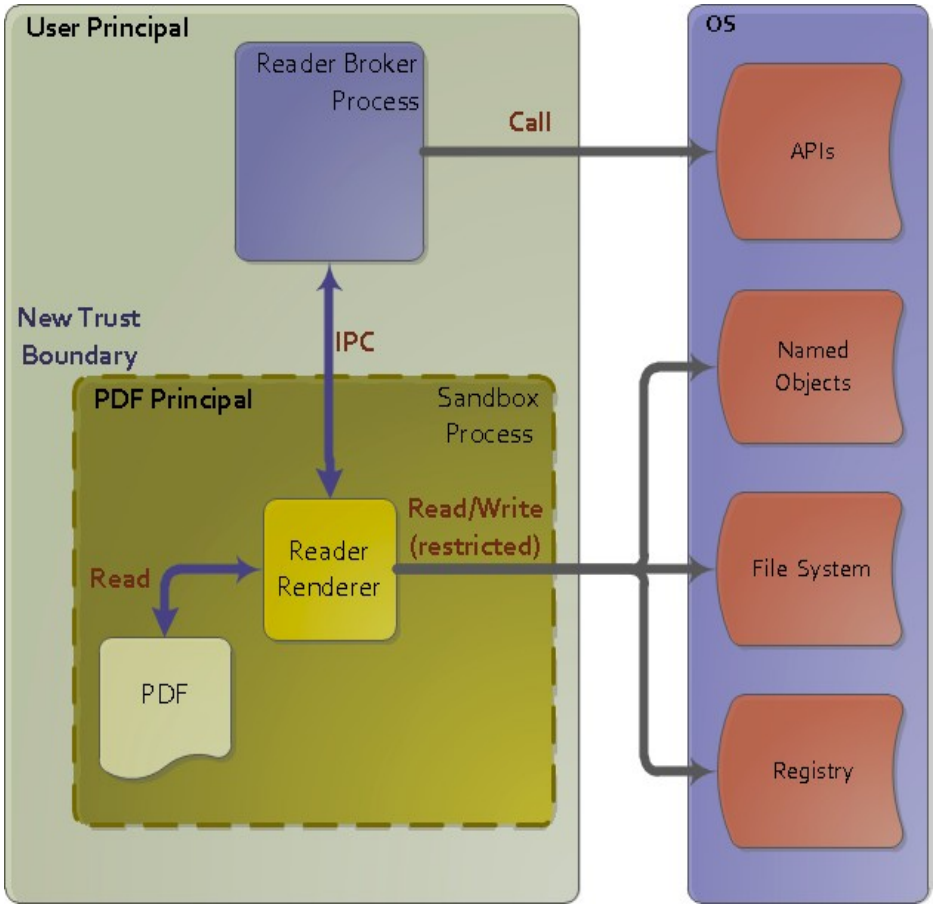
Trusting the commercial code is dubious.

Isolation is the remaining option.

(We also rely on **trust** in choosing whether to open PDFs.)

Adobe's Response to PDF Security Issues

- Formed public security teams in 2008:
  - Product Security Incident Response Team
  - Adobe Secure Software Engineering Team



- Mid-2010, stripped down and adapted Google Chrome's sandbox for subset of Adobe Reader X
  - Both sandboxes have been successfully attacked

(Security @ Adobe Blog)

Sandboxing

- A *sandbox* is a security layer that imposes a security policy on an application
  - Instead of verifying the application, verify the smaller, specialized sandbox
- Most existing sandboxes are in-situ, they encapsulate computations within the system we want to defend

In-Situ Technique	Sandbox Examples
Resource Redirection	ASLR, chroot
Rule-Constrained Execution	AppArmor
Runtime Isolation	Google NaCl, Java
Software-based Fault Isolation	Google NaCl, TruE (Payer et al 2012)
SysCall Inspection	TruE, TxBox (Jana et al 2011)
Virtualization	Cuckoo

When an in-situ sandbox is compromised, the host system is also compromised.

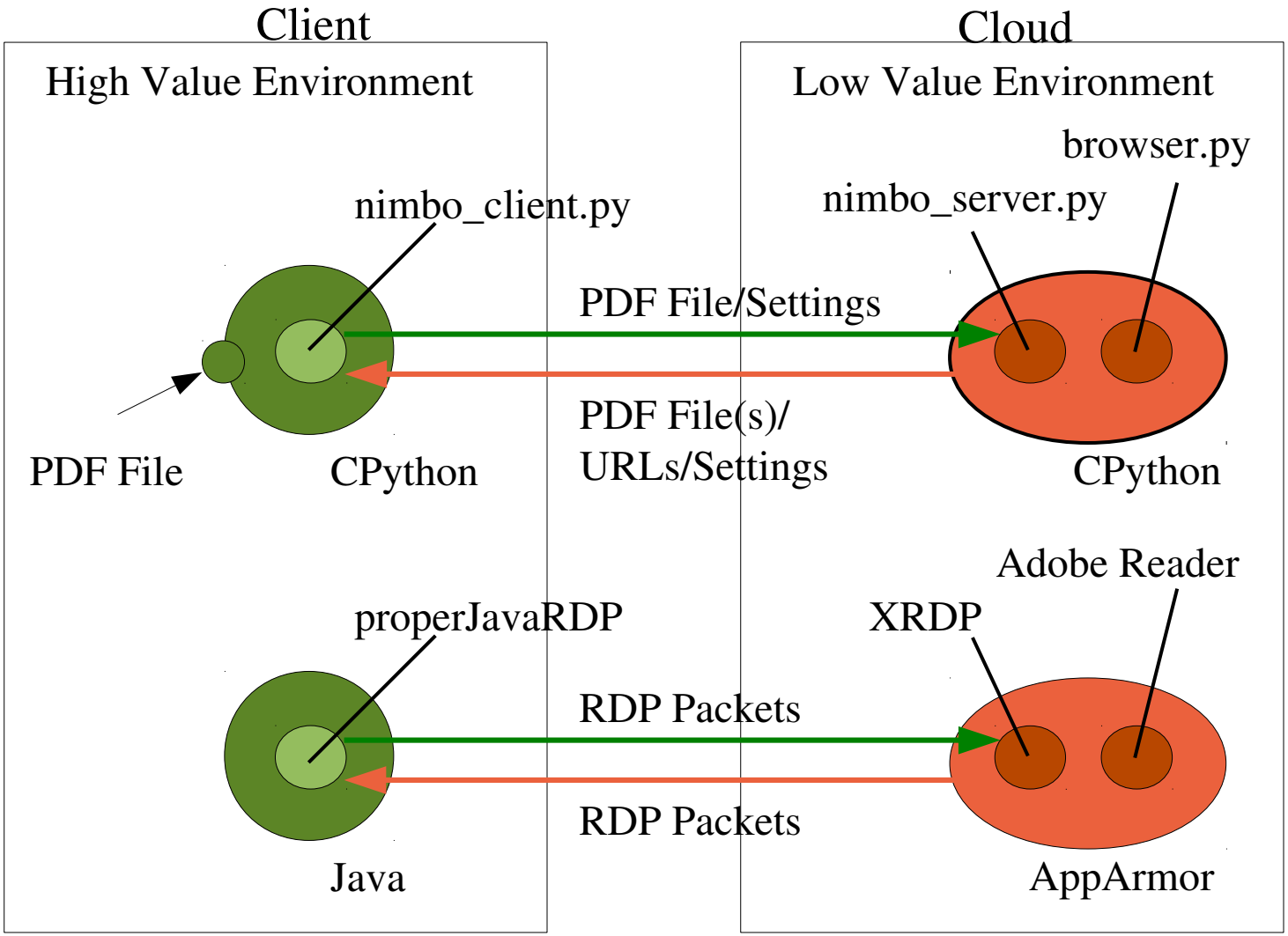
In-Nimbo Sandboxing

- **Goal:** run untrusted computations in an ephemeral environment
- **Approximation:** run untrusted computations in a cloud computing environment
  - Tear down cloud instances after each session, don't persist state (or carefully persist critical user state)
  - Continue to make use of traditional, in-situ sandboxes on the client and in the cloud

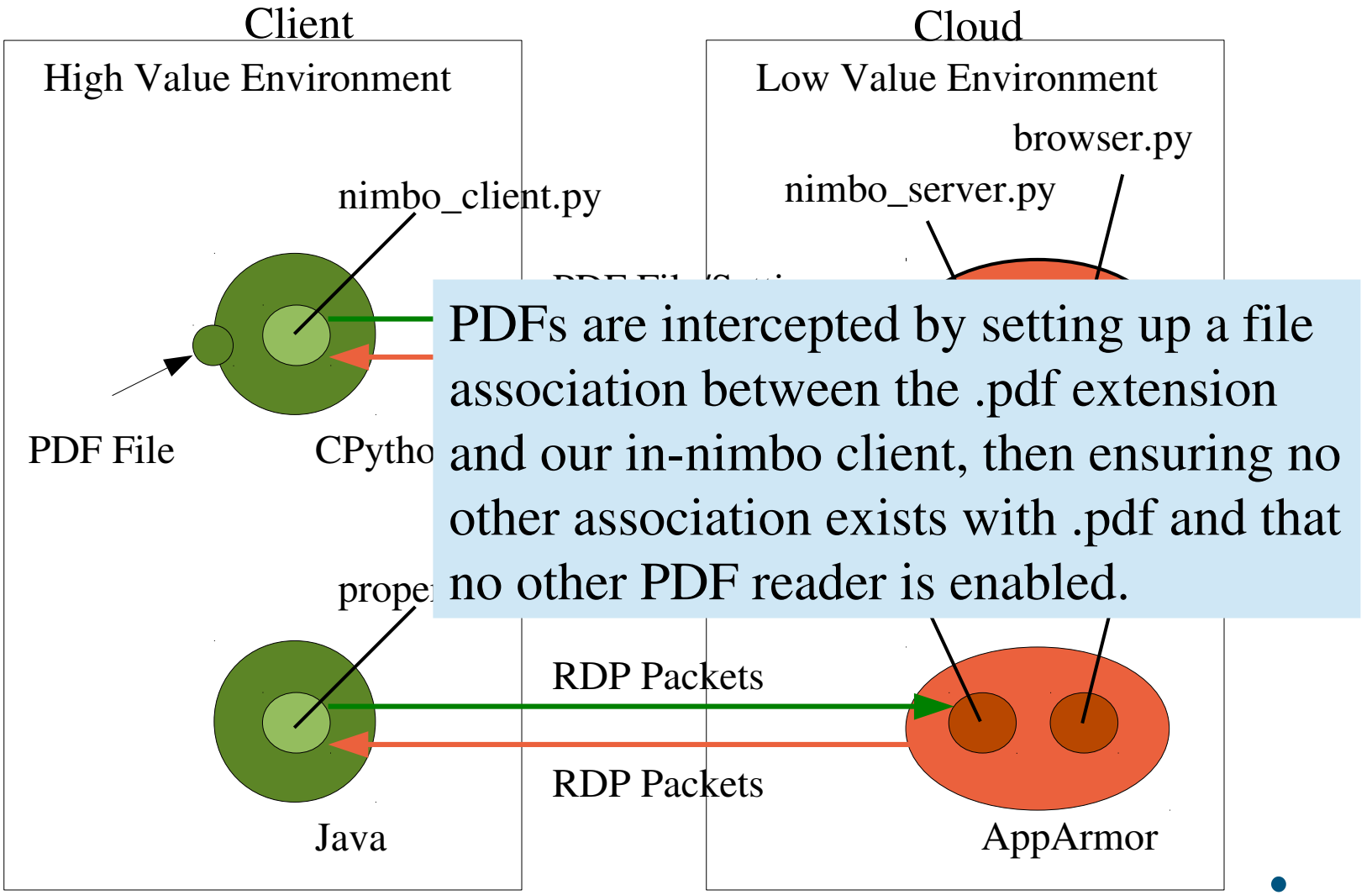
Hypotheses

- **Attack Surface Design Flexibility:**
  - In-nimbo sandboxing provides flexibility in attack surface design
    - Easier to customize the execution environment for untrusted computations
    - Can pick the path the attacker must take to compromise the system we want to defend
- **Attack Surface Extent:**
  - The technique results in encapsulated components with smaller, more defensible attack surfaces compared to the use of other techniques
- **Consequences of Attack Success:**
  - Remote encapsulation reduces the consequences of attack success
- **Additional Criteria:**
  - Performance
  - Usability

In-Nimbo Adobe Reader

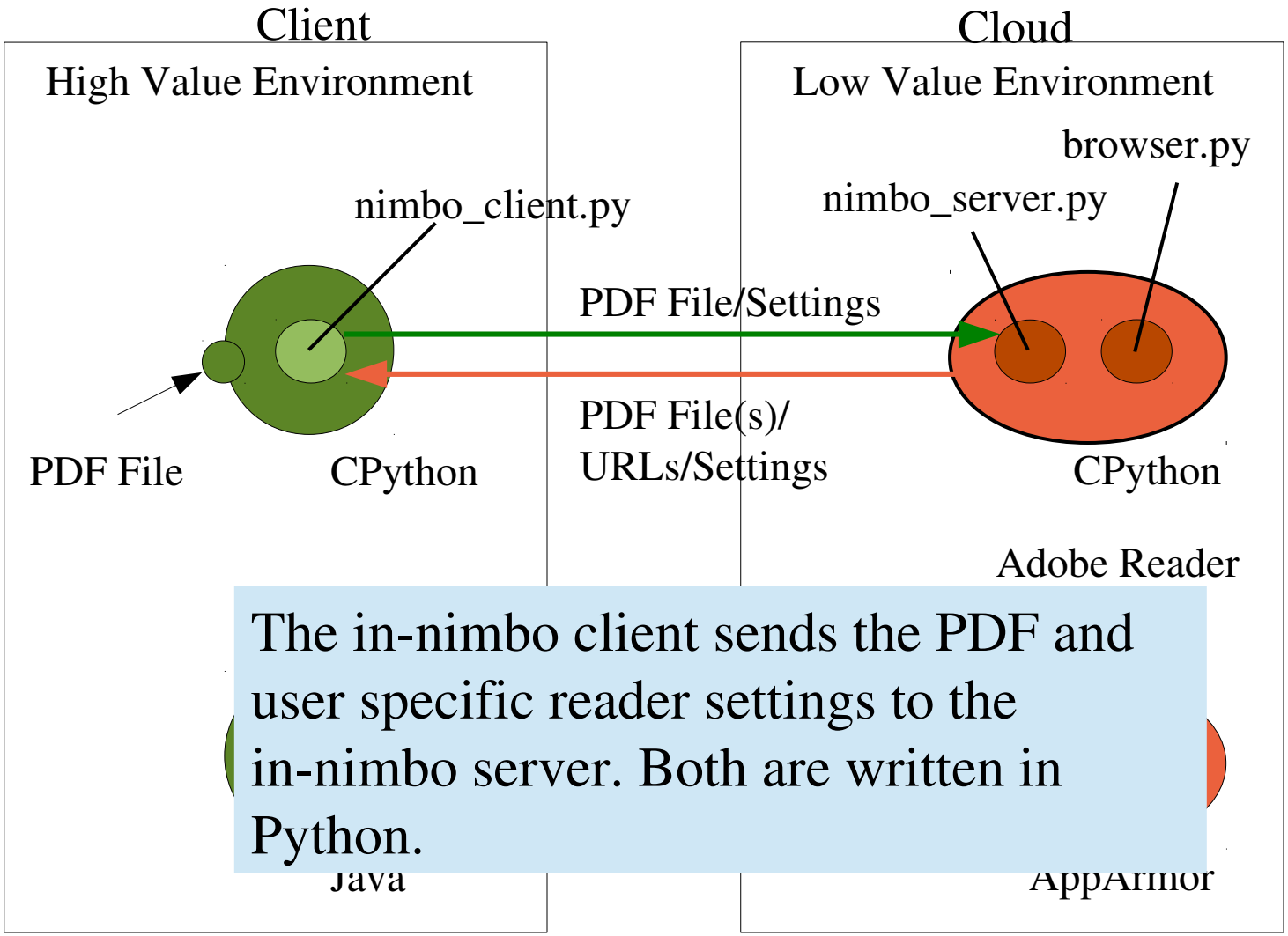


In-Nimbo Adobe Reader



PDFs are intercepted by setting up a file association between the .pdf extension and our in-nimbo client, then ensuring no other association exists with .pdf and that no other PDF reader is enabled.

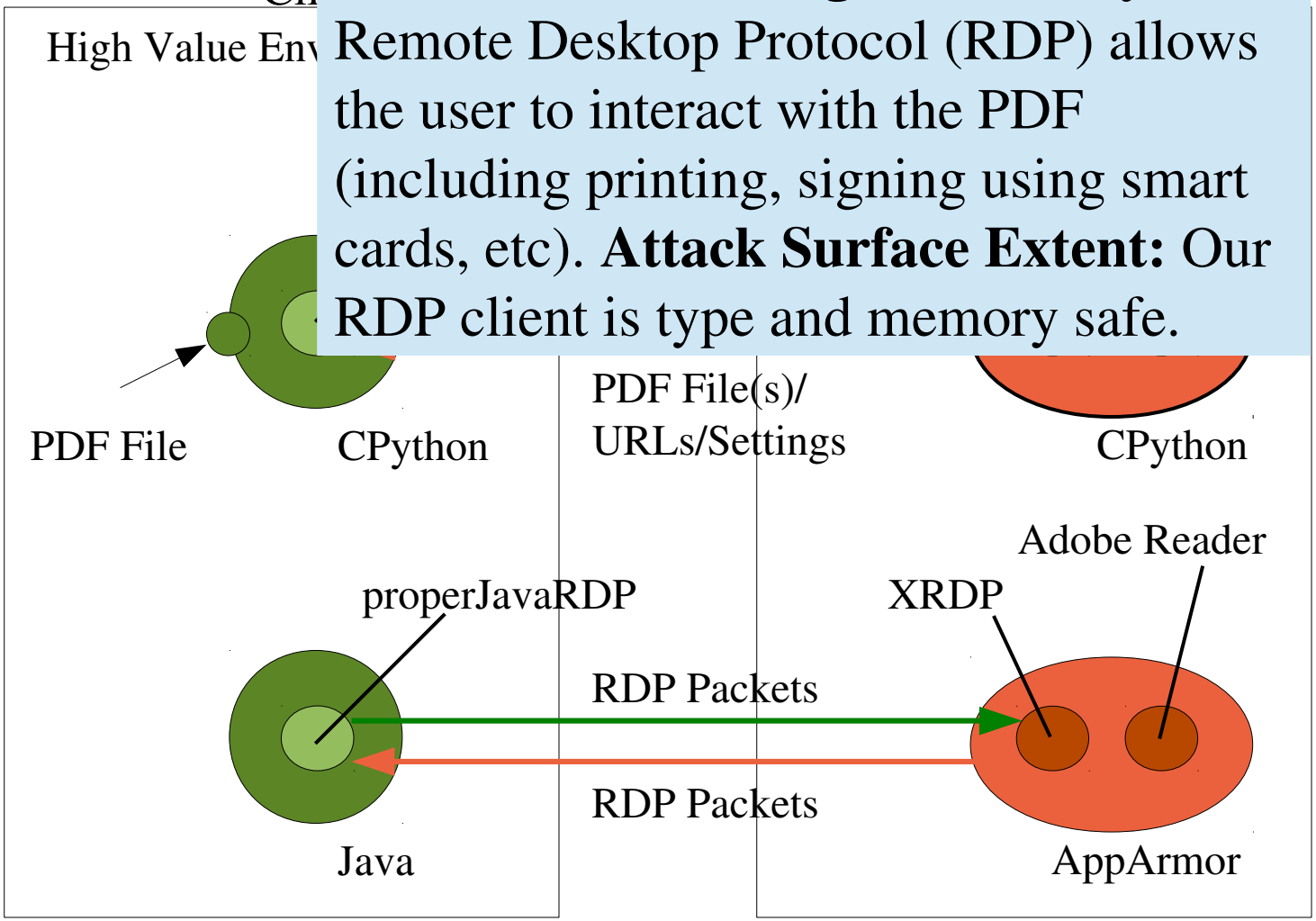
In-Nimbo Adobe Reader



The in-nimbo client sends the PDF and user specific reader settings to the in-nimbo server. Both are written in Python.

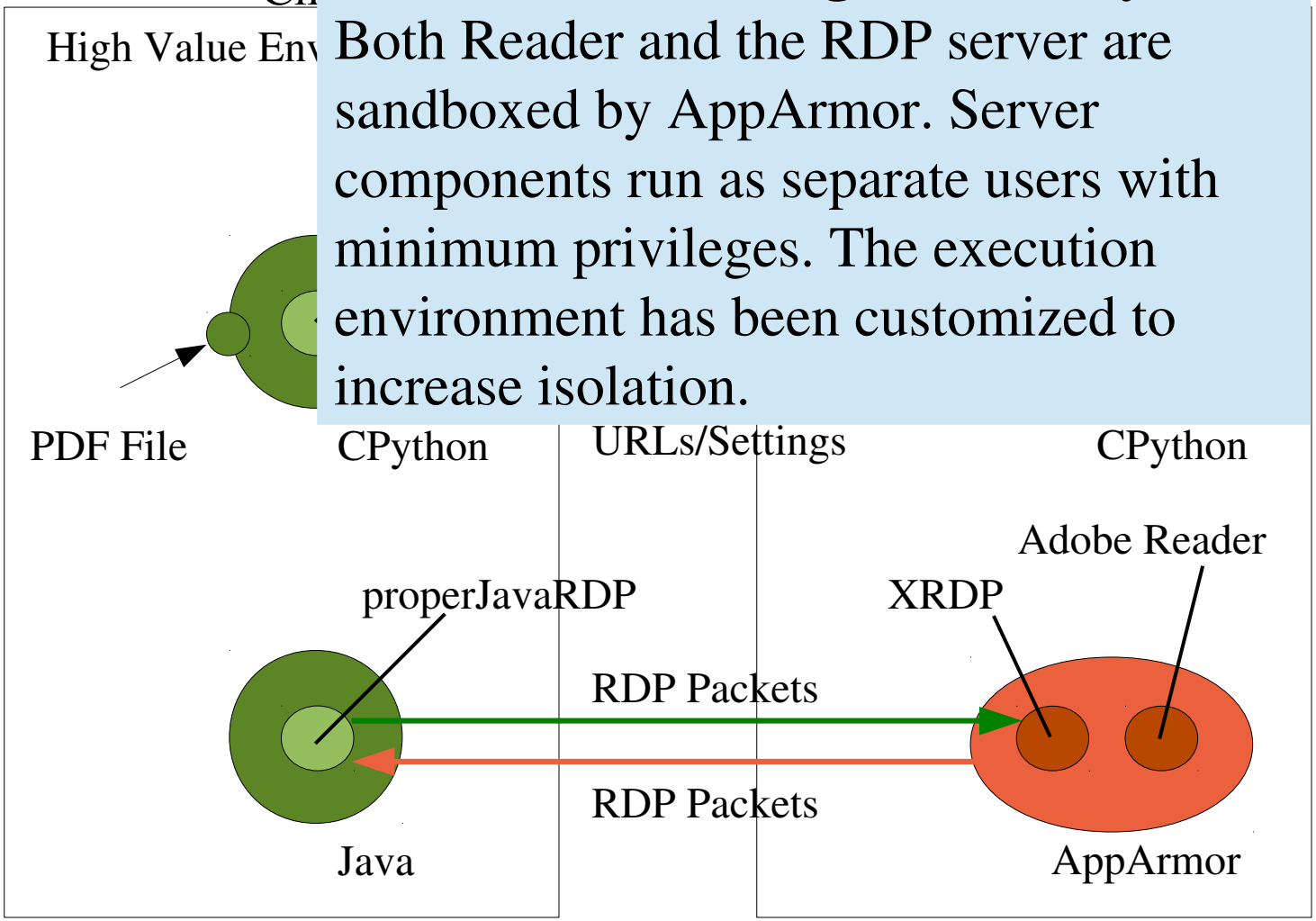
In-Nimbo Adobe Reader

Attack Surface Design Flexibility: The Remote Desktop Protocol (RDP) allows the user to interact with the PDF (including printing, signing using smart cards, etc). **Attack Surface Extent:** Our RDP client is type and memory safe.

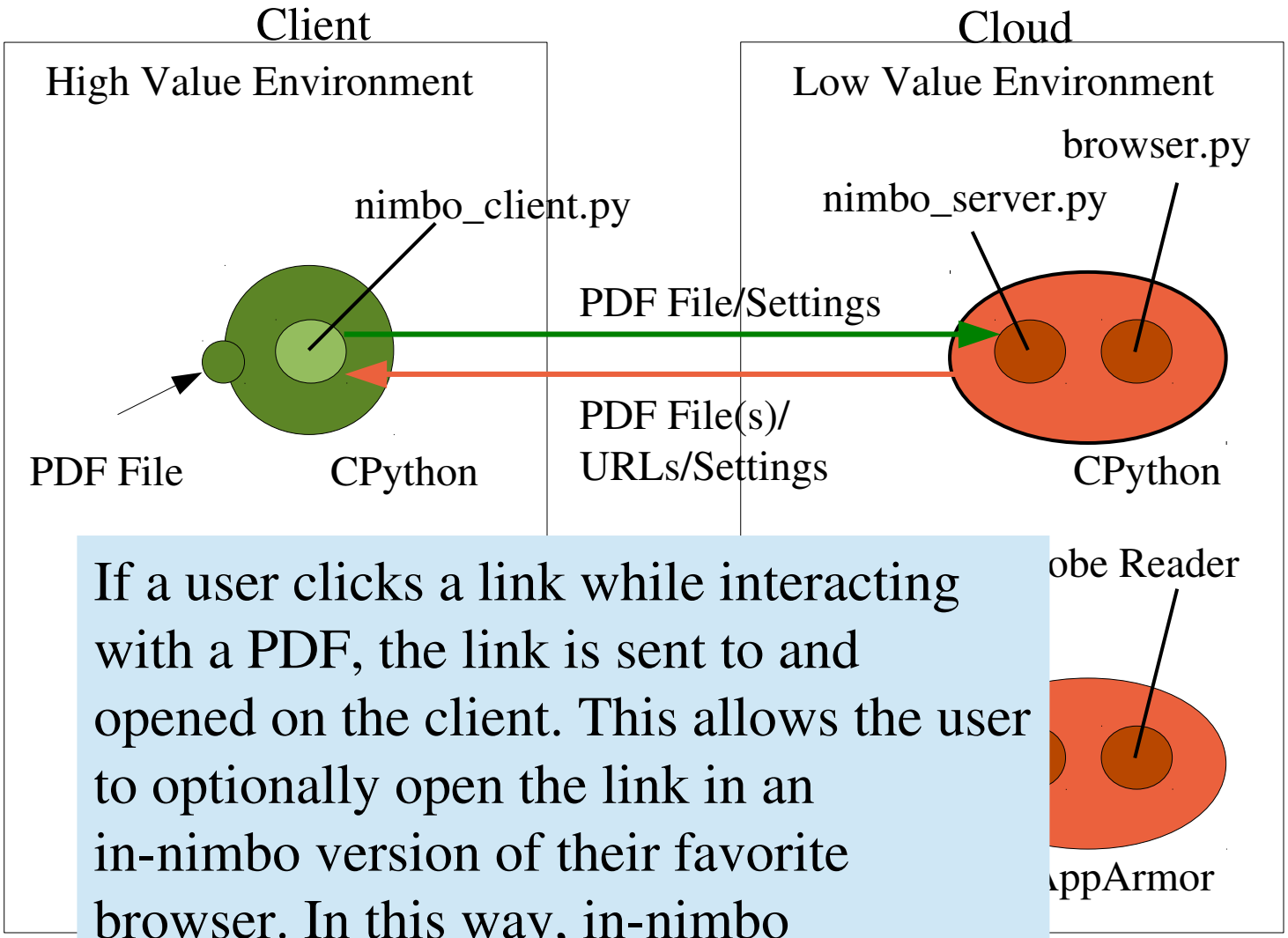


In-Nimbo Adobe Reader

Attack Surface Design Flexibility: Both Reader and the RDP server are sandboxed by AppArmor. Server components run as separate users with minimum privileges. The execution environment has been customized to increase isolation.

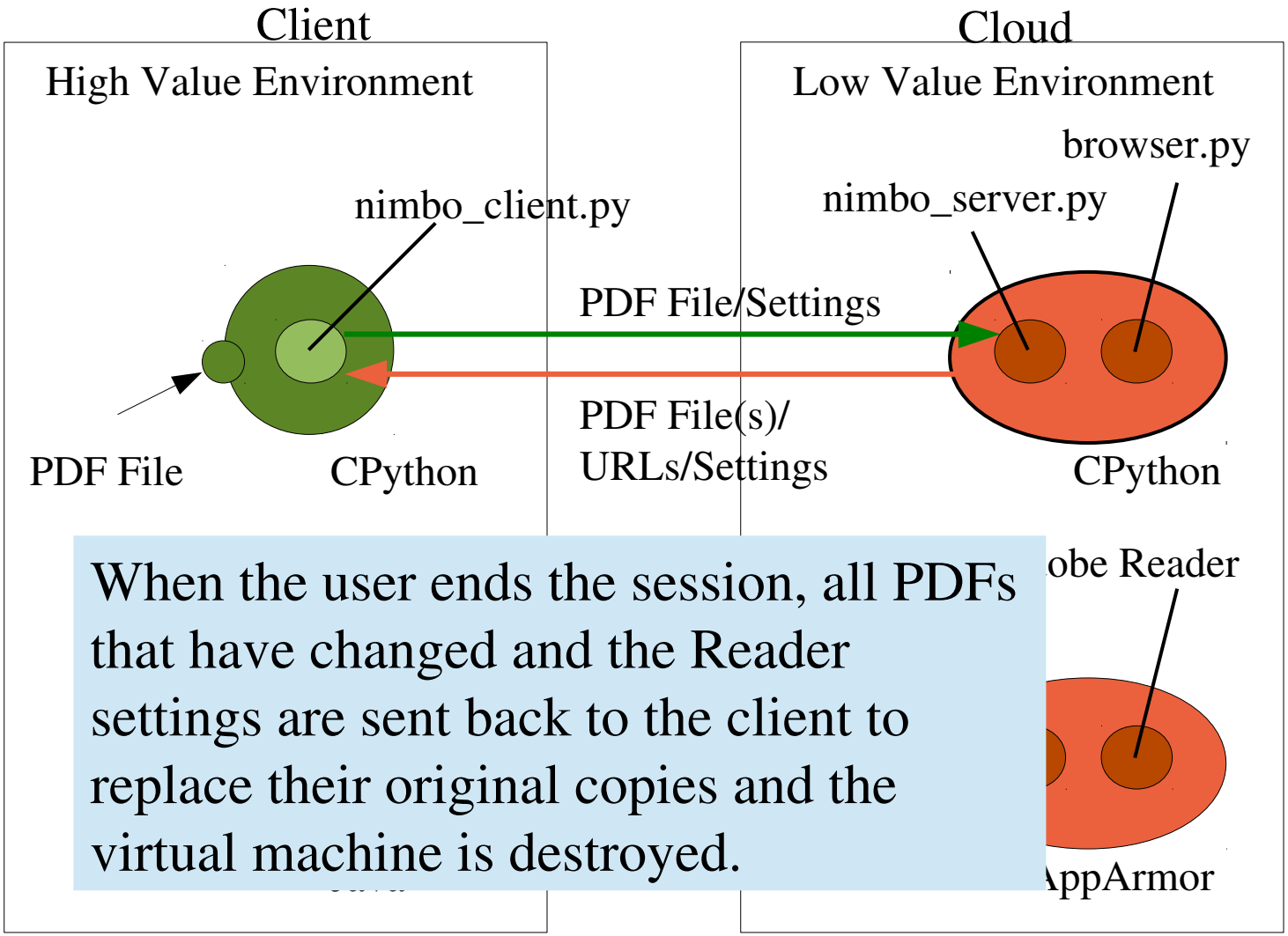


In-Nimbo Adobe Reader



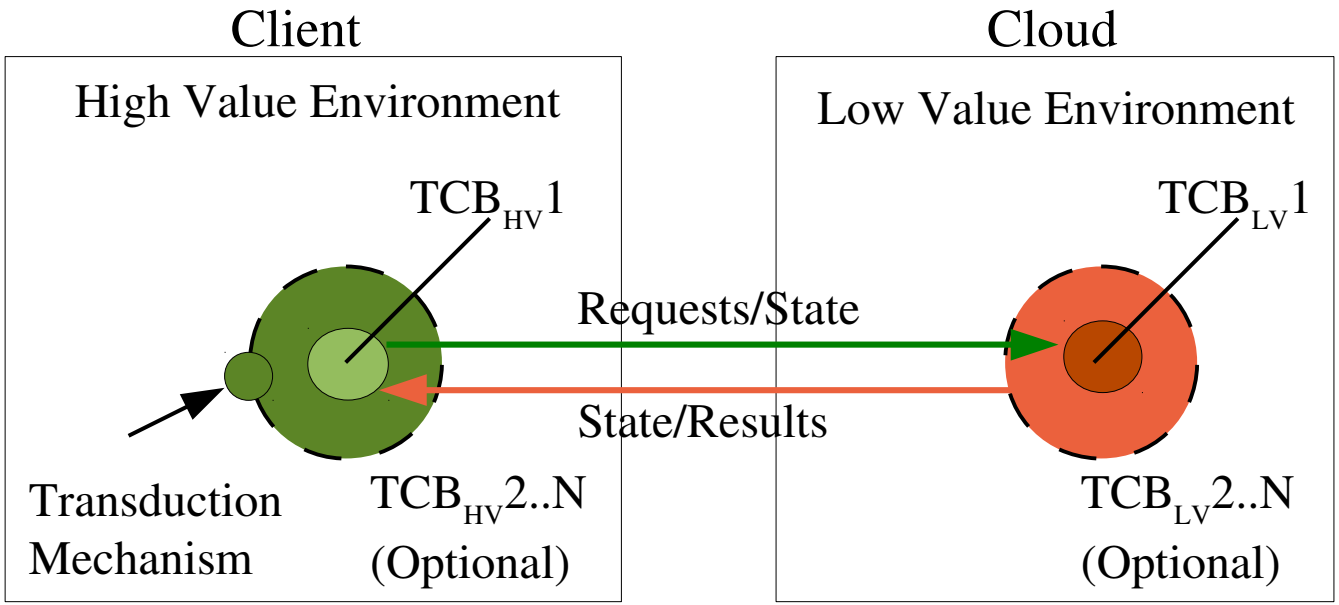
If a user clicks a link while interacting with a PDF, the link is sent to and opened on the client. This allows the user to optionally open the link in an in-nimbo version of their favorite browser. In this way, in-nimbo sandboxes can be composed.

In-Nimbo Adobe Reader



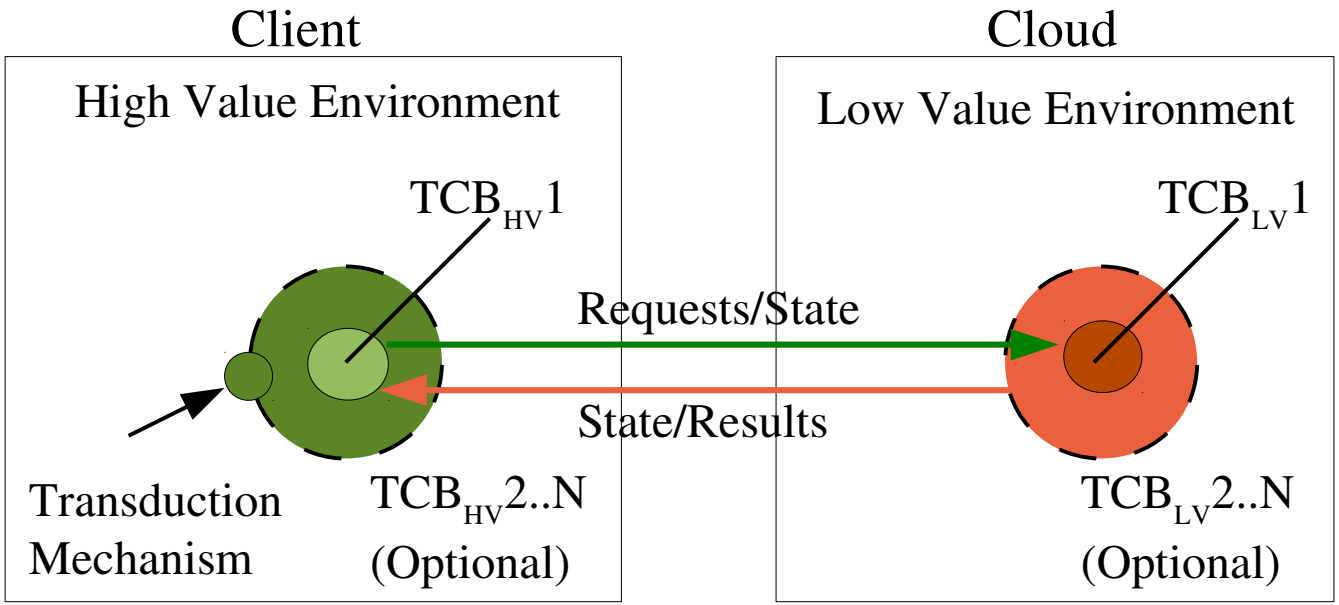
When the user ends the session, all PDFs that have changed and the Reader settings are sent back to the client to replace their original copies and the virtual machine is destroyed.

In-Nimbo Sandboxing



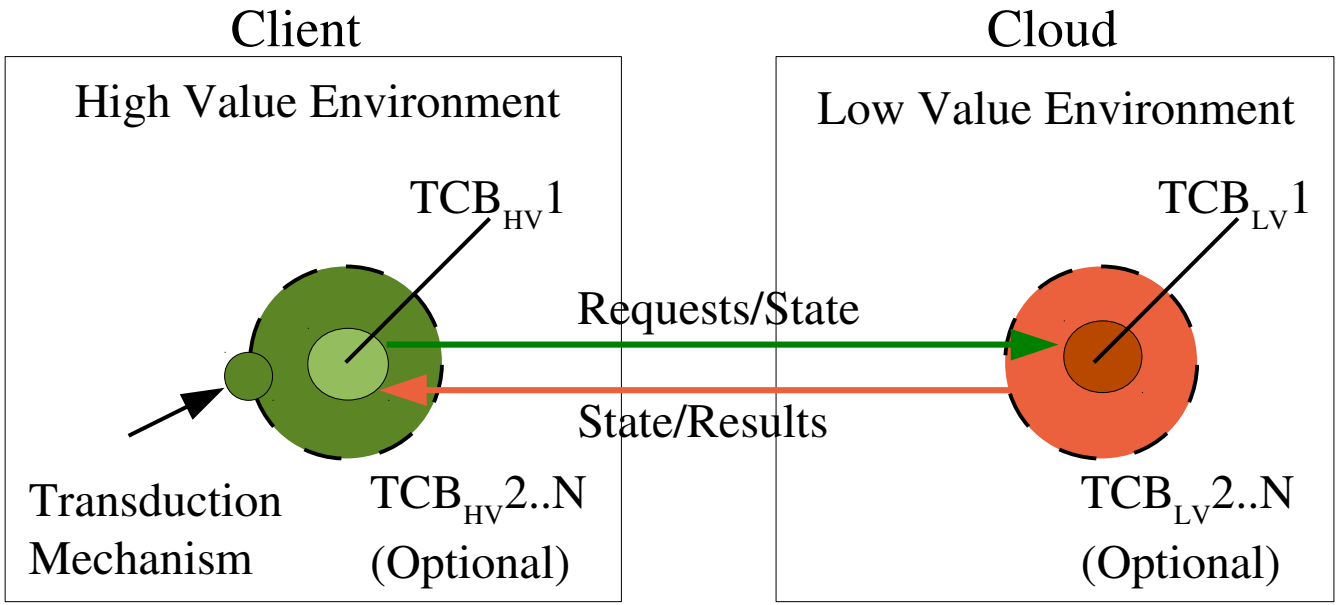
In-Nimbo Sandboxing

1. Capture an untrusted computation
2. Transfer the computation and any required state to the High Value TCB



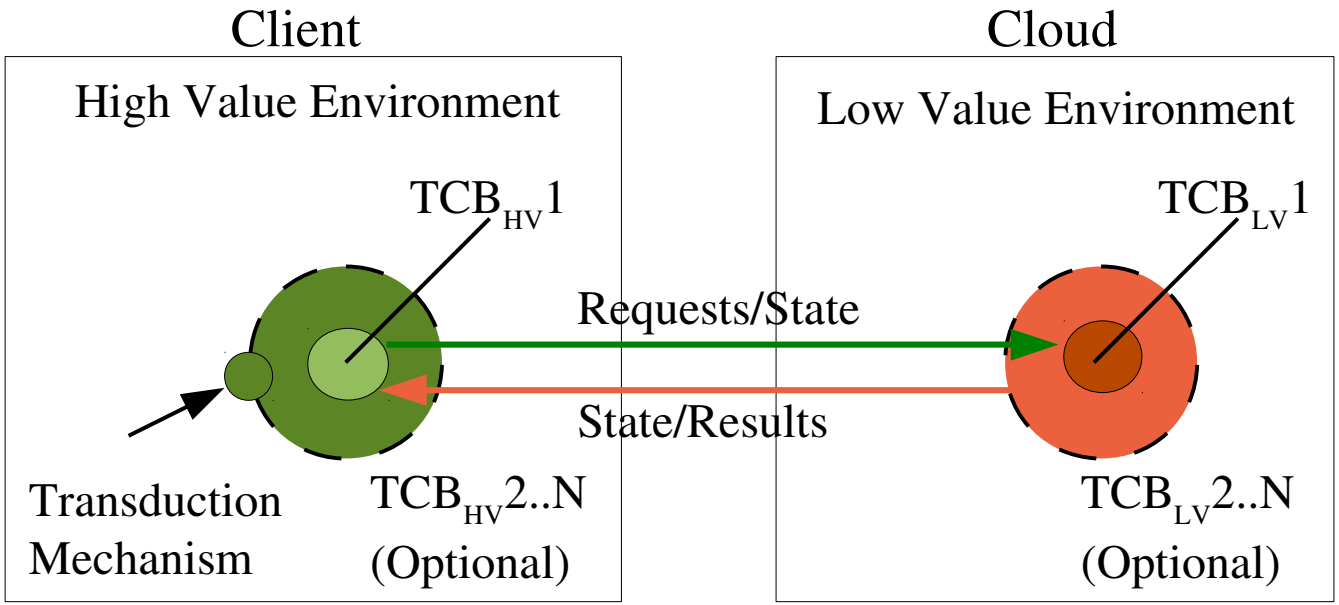
In-Nimbo Sandboxing

3. Transfer the computation and state to the Low Value TCB



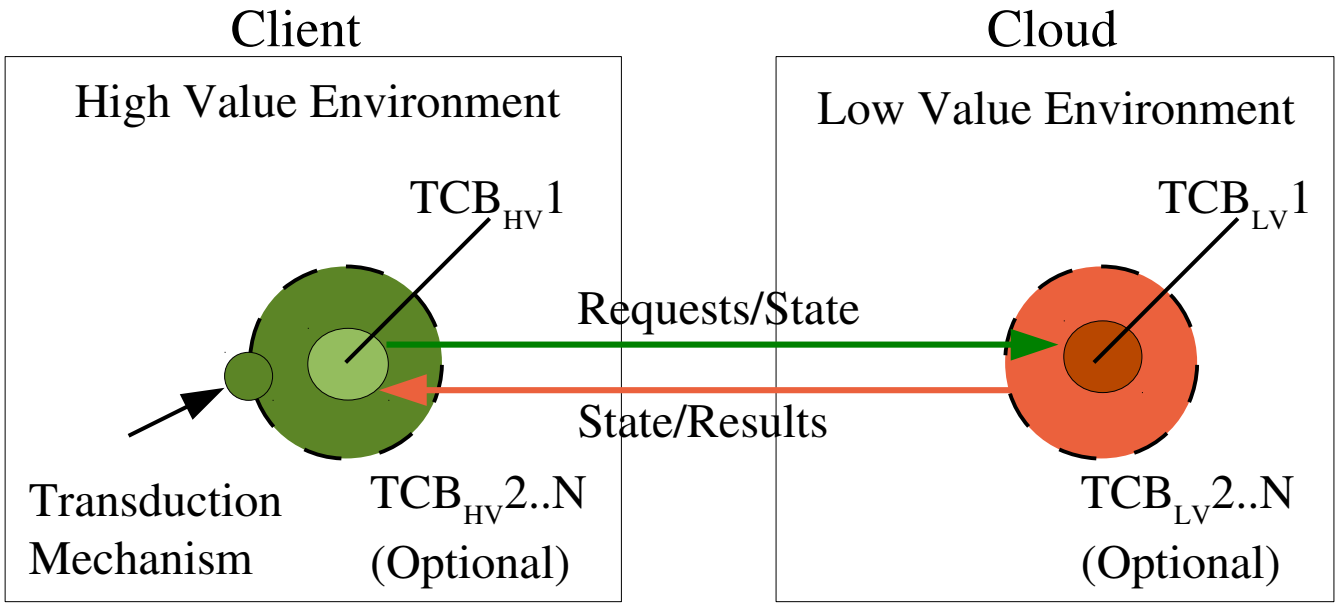
In-Nimbo Sandboxing

4. Execute the untrusted computation
5. Return updated state and results



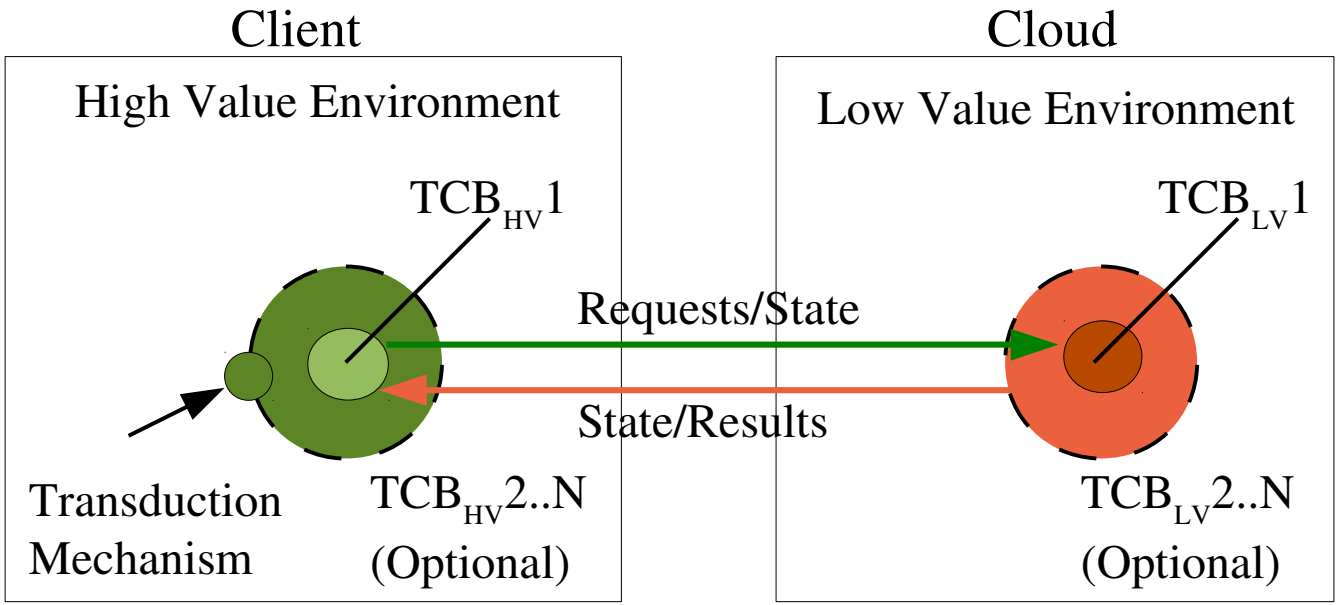
In-Nimbo Sandboxing

6. When necessary, check results and state
7. Transfer results and state to the transduction mechanism



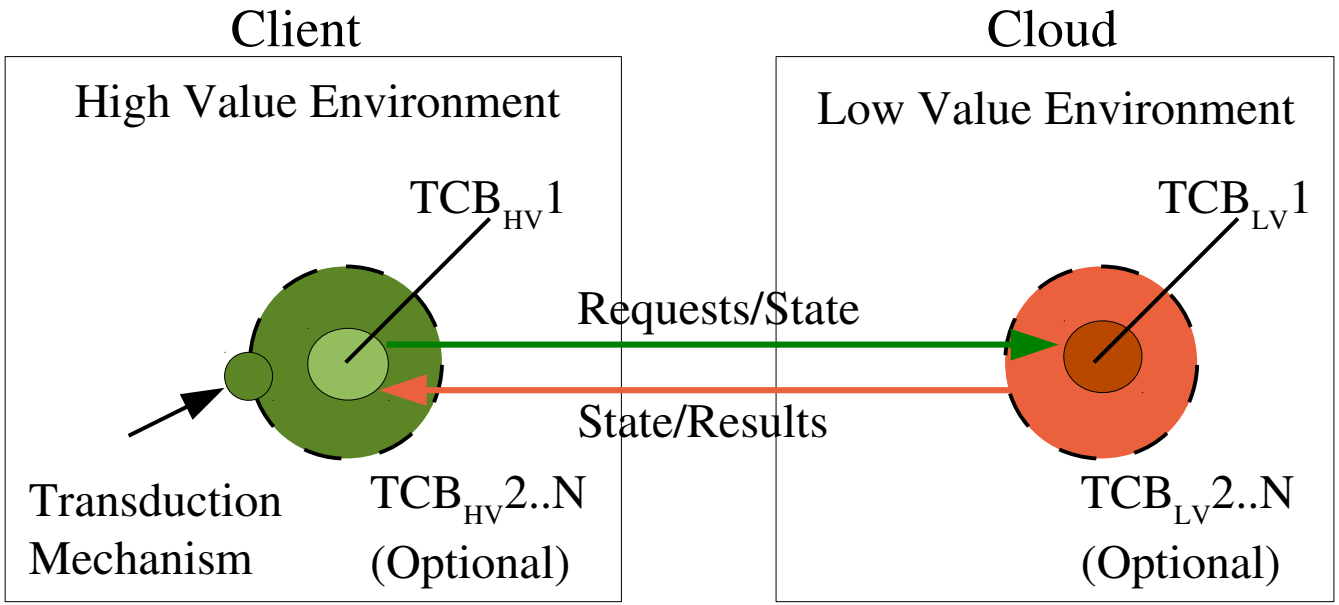
In-Nimbo Sandboxing

8. Integrate results and state



In-Nimbo Sandboxing

- Existing sandboxes can be used to encapsulate each TCB to form a TCB architecture.
- Nested TCBs add *Defense in Depth*



Field Trial

- The sandbox as designed supports the full range of PDF features:
  - 3-D models, printing, DRM, forms, and opening URLs
- We built a prototype of the sandbox and deployed it in cooperation with a large aerospace company
- The trial deployment was used to test and measure the sandbox in a real environment with real users

Performance and Usability

- Rich PDF interaction functions with no lag
- In-nimbo Reader visually appears exactly as it would if the user had run Reader locally

PDF Size	1 MB
Average upload time	2.1 seconds +/- 0.3s (Confidence level: 95%)
Average Adobe Reader start time in-nimbo	0.5 seconds
Average time to establish RDP channel	1.5 seconds
Average time until user can interact	**2.1 seconds**
Distance from client to cloud	1,800 miles
Average Adobe Reader start time in-situ	1.5 seconds

Consequences of Attack Success: Structured Evaluation

- Using a priori knowledge of how a sandbox functions, consider threats that exist when the sandbox *holds*, *fails*, or is *bypassed*
  - A **bypass** occurs when an attacker can accomplish his goals by jumping from a sandboxed component to an unsandboxed component
    - Can occur because an insufficient security policy is imposed
  - A **failure** occurs when the attacker must directly attack the sandbox to accomplish their goals
    - Requires an exploitable vulnerability in the sandbox

Risk Considerations

- **Enforced Security Policy:**
  - (-) Reader X's sandbox still allows for exfiltration*
  - (+) In-Nimbo Reader allows no access to sensitive files
- **Path to Critical Consequences:**
  - (-) Reader X exploits compromise the high value environment
  - (+) In-Nimbo Reader exploits must compromise Reader and VM sandboxes and survive long enough to compromise the small, typesafe client to get to the high value environment
  - (+) Known malicious PDF's do not escape the sandbox or survive across sessions
- **Attack Surface Extent:**
  - (-) Reader X's attack surface is the entire application
  - (+) In-Nimbo Reader's attack surface is the small, typesafe client

*Reader XI's sandbox fixes this, but has also been exploited

Caveat

- We don't consider attacks on the cloud itself:

1. The ability to outsource risk that cannot be eliminated to a willing third party is a key advantage of in-nimbo sandboxing
2. In-nimbo sandboxing does not add any new threats to cloud environments in general
3. In-nimbo sandboxing is a use of the cloud, not a cloud implementation

Conclusion

- **In-Nimbo Sanboxing**: execute untrusted computations on ephemeral virtual machine instances in the cloud
- **Attack Surface Design Flexibility**
  - Case Study: In our prototype sandbox for Adobe Reader, we were able to choose the path the attacker must take to compromise our high value environment and customize the execution environment for untrusted code
- **Attack Surface Extent**
  - Case Study: To compromise In-Nimbo Reader you must ultimately exploit a small, typesafe client as opposed to any vulnerability in Reader
- **Consequences of Attack Success**
  - Structured Evaluation: Exploiting In-Nimbo Reader compromises a sacrificial virtual machine instead of our workstation
- **Questions?**

Criteria Matrix

	Reader X	Local VM	Cloud VM
Performance	+ users don't notice it	- resource intensive - slow start-up	+ resources provisioned from cloud + improved by pooling but - dependent on load - network latency
Coverage	- sandboxes only a subset of the application's architecture	+ VMM contains (most) of the untrusted computation's execution - degree of coverage depends on type of VMM	+ only carefully persisted state isn't contained in the cloud - carefully persisted state is dealt with by a small client
Effectiveness	- not yet reliable	+ VMM adds an additional layer to compromise	+ VMM and cloud add several additional layers + critical attack surface is a small client

Criteria Matrix

	Reader X	Local VM	Cloud VM
Usability	+ users don't notice it	- VMM management decentralized	+ sandbox fully managed by a central authority