Research Report: Mitigating LangSec Problems With Capabilities Or: How Sandstorm Taught Me to Stop Worrying and Love the Web

Nathaniel Wesley Filardo

May 26, 2016

One-Slide Elevator Pitch

Actually two, related, pitches:

- Sandstorm's capability-based design enables *very fine-grained sandboxing* of application software, which largely (sometimes completely!) *mitigates* the majority of LangSec bugs seen in practice.
- Capability systems offer the potential to turn difficult authorization decisions into LangSec's bread and butter: syntactic constraints on requests; *every well-formed request which can be stated is authorized.*

Traditional Web Application Hosting
The Sad Story

Consider a standard LAMP-esque stack.

- Many co-hosted applications at different paths.
    - Maybe have separate kernel UIDs when executing?

Traditional Web Application Hosting
The Sad Story

Consider a standard LAMP-esque stack.

- Many co-hosted applications at different paths.
  - Maybe have separate kernel UIDs when executing?
- System design encourages *ambient authority*:
  - esp. to filesystem, network resources.

Traditional Web Application Hosting
The Sad Story

Consider a standard LAMP-esque stack.

- Many co-hosted applications at different paths.
    - Maybe have separate kernel UIDs when executing?
- System design encourages *ambient authority*:
    - esp. to filesystem, network resources.
- Database processes *per-server*
    - Own notion of users (typ. "app") and permissions.

Traditional Web Application Hosting
The Sad Story

Consider a standard LAMP-esque stack.

- Many co-hosted applications at different paths.
  - Maybe have separate kernel UIDs when executing?
- System design encourages *ambient authority*:
  - esp. to filesystem, network resources.
- Database processes *per-server*
  - Own notion of users (typ. "app") and permissions.
- Client authn, authz up to *each* hosted application.
  - Even SSO systems typically require application buy-in.
  - Groups, ACLs, etc. per application.

Traditional Web Application Hosting
The Sad Story

Consider a standard LAMP-esque stack.

- Many co-hosted applications at different paths.
  - Maybe have separate kernel UIDs when executing?
- System design encourages *ambient authority*:
  - esp. to filesystem, network resources.
- Database processes *per-server*
  - Own notion of users (typ. “app”) and permissions.
- Client authn, authz up to *each* hosted application.
  - Even SSO systems typically require application buy-in.
  - Groups, ACLs, etc. per application.
- Web's failings left to apps: XSRF, XSS, SRI, . . .

Traditional Web Application Hosting
The Impact of LangSec Bugs

In this environment, what do LangSec bugs buy an attacker?

- Outright authn/authz confusion:
  - Authn/authz cookie leak & replay
  - XSRF & XSS

Traditional Web Application Hosting
The Impact of LangSec Bugs

In this environment, what do LangSec bugs buy an attacker?

- Outright authn/authz confusion:
  - Authn/authz cookie leak & replay
  - XSRF & XSS
- Path traversals:
  - Access to intra-application resources (almost surely)
  - Access to other applications' resources (maybe)
  - Access to system configuration (likely)

Traditional Web Application Hosting
The Impact of LangSec Bugs

In this environment, what do LangSec bugs buy an attacker?

- Outright authn/authz confusion:
  - Authn/authz cookie leak & replay
  - XSRF & XSS
- Path traversals:
  - Access to intra-application resources (almost surely)
  - Access to other applications' resources (maybe)
  - Access to system configuration (likely)
- Code injection:
  - Probe file system, loopback network
  - Make remote network connections
  - Probe local *kernel* for vulnerabilities

Too easy for bug in one application to impact *entire server*.

Sandstorm Application Hosting
What a mess! Alternative design?

Sweeping changes to design of system:

- Replace web server with application supervisor.
    - *Not* “Web Application Firewall”
    - No dynamic inspection of application display content!

Sandstorm Application Hosting
What a mess! Alternative design?

Sweeping changes to design of system:

- Replace web server with application supervisor.
  - *Not* "Web Application Firewall"
  - No dynamic inspection of application display content!
- Centralize authn to supervisor.
  - Send user *display information* to application.

Sandstorm Application Hosting
What a mess! Alternative design?

Sweeping changes to design of system:

- Replace web server with application supervisor.
  - *Not* "Web Application Firewall"
  - No dynamic inspection of application display content!
- Centralize authn to supervisor.
  - Send user *display information* to application.
- Centralize *authz* to supervisor (mostly).
  - Applications enumerate possible "rights".
  - Supervisor computes agent's rights; tells application.

Sandstorm Application Hosting
What a mess! Alternative design?

Sweeping changes to design of system:

- Replace web server with application supervisor.
    - *Not* "Web Application Firewall"
    - No dynamic inspection of application display content!
- Centralize authn to supervisor.
    - Send user *display information* to application.
- Centralize *authz* to supervisor (mostly).
    - Applications enumerate possible "rights".
    - Supervisor computes agent's rights; tells application.
- Sandbox server-side resources *very tightly*.
    - Each *document* in its own container is possible!
    - Granularity up to application author and user.
    - Possible due to centralized management of sharing.

Sandstorm Application Hosting
What a mess! Alternative design?

Old world:

- As admin, install application to web server (or find host)
- Users register with *each* application (or be anonymous)
- Application juggles many documents / objects / . . .
- User rights managed within *each* application

Sandstorm Application Hosting
What a mess! Alternative design?

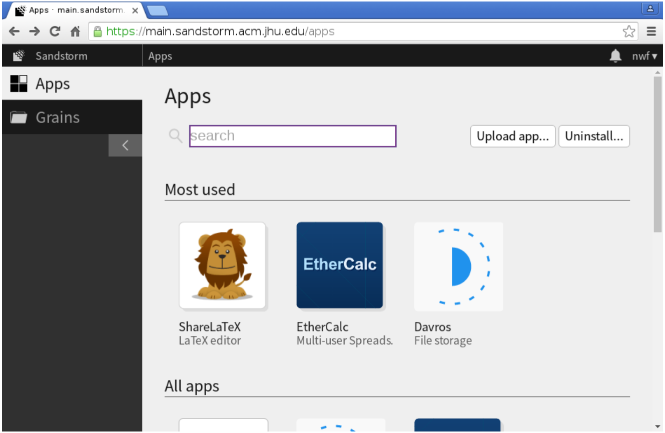
Old world:

- As admin, install application to web server (or find host)
- Users register with *each* application (or be anonymous)
- Application juggles many documents / objects / . . .
- User rights managed within *each* application

New world:

- As admin, install sandstorm server (or . . .)
- Users register once with sandstorm installation (or . . .)
- *Users* install *arbitrary* applications as desired!
- Users *instantiate* applications as "grains."
    - Each user may have zero or more grains of any app.
    - Grains begin *private to creator*.
- Users share (and revoke) appropriate access to grains.

Sandstorm Application Hosting
User's Perspective



Apps · main.sandstorm. ×

https://main.sandstorm.acm.jhu.edu/apps

Sandstorm | Apps | nwf ▾

Apps

Grains

Apps

search

Upload app... | Uninstall...

Most used

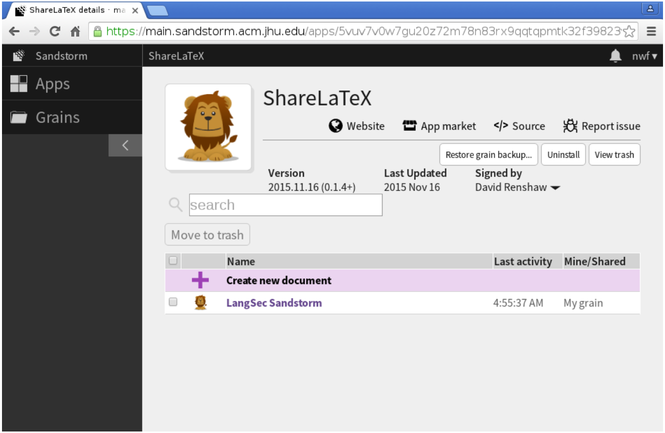
ShareLaTeX
LaTeX editor

EtherCalc
Multi-user Spreads.

Davros
File storage

All apps

Sandstorm Application Hosting
User's Perspective

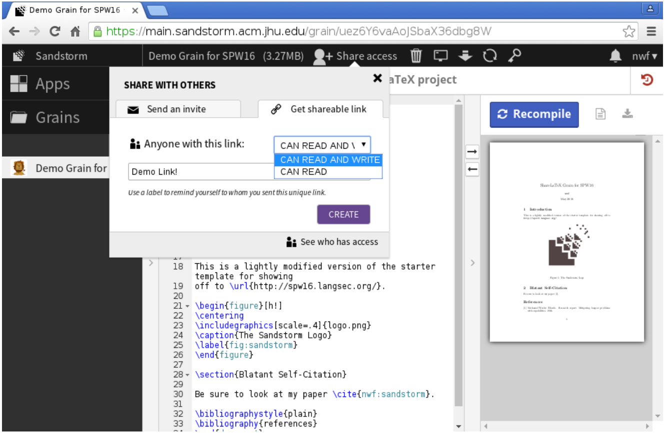


ShareLaTeX

Version	Last Updated	Signed by
2015.11.16 (0.1.4+)	2015 Nov 16	David Renshaw

□		Name	Last activity	Mine/Shared
	+	Create new document		
□		LangSec Sandstorm	4:55:37 AM	My grain

Sandstorm Application Hosting
User's Perspective



https://main.sandstorm.acm.jhu.edu/grain/uez6Y6vaAoJSbaX36dbg8W

Anyone with this link: CAN READ AND \

CAN READ AND WRITE
CAN READ

Demo Link!

Use a label to remind yourself to whom you sent this unique link.

```
This is a lightly modified version of the starter template for showing
off to \url{http://spw16.langsec.org/}.

\begin{figure}[h!]
\centering
\includegraphics[scale=.4]{logo.png}
\caption{The Sandstorm Logo}
\label{fig:sandstorm}
\end{figure}

\section{Blatant Self-Citation}

Be sure to look at my paper \cite{nwf:sandstorm}.

\bibliographystyle{plain}
\bibliography{references}
```

`https://main.sandstorm.acm.jhu.edu/shared/pruMzgByO3ReRVV9tT5uQQyhwXJulmoMCSNSFutPjXJ`

Sandstorm Application Hosting
Supervisor's Perspective

Supervisor tracks *capabilities* conveying *rights* to grains:

- Each application specifies a collection of rights.
    - ShareLaTeX: "read", "write"
    - DokuWiki: "user", "manager", "admin"
    - When grain is created, owner gets *all* rights.
    - *Nobody else gets any rights*

Sandstorm Application Hosting
Supervisor's Perspective

Supervisor tracks *capabilities* conveying *rights* to grains:

- Each application specifies a collection of rights.
  - ShareLaTeX: "read", "write"
  - DokuWiki: "user", "manager", "admin"
  - When grain is created, owner gets *all* rights.
  - *Nobody else gets any rights*
- Users *delegate* access to grains:
  - Creates a new capability object held by designated user(s) or within a sharing link.
  - Delegated access is a *subset* of delegator's access.
  - Sandstorm tracks *provenance* of rights & adjusts.

Sandstorm Application Hosting
Supervisor's Perspective

Supervisor juggles *sessions*: user's live connection to a grain.

- Grains started and stopped by supervisor as needed.

Sandstorm Application Hosting
Supervisor's Perspective

Supervisor juggles *sessions*: user's live connection to a grain.

- Grains started and stopped by supervisor as needed.
- At session startup, the grain is told what rights the initiator has to the grain.
  - Each request by a user will be part of a session. *Application just needs to check that request is permitted by session's rights.*

Sandstorm Application Hosting
Supervisor's Perspective

Supervisor juggles *sessions*: user's live connection to a grain.

- Grains started and stopped by supervisor as needed.
- At session startup, the grain is told what rights the initiator has to the grain.
    - Each request by a user will be part of a session. *Application just needs to check that request is permitted by session's rights.*
- Web sessions on *random hostnames* (anti-XSRF, -XSS).
    - Not as good as if application didn't have bugs, but ups ante to require that attacker can see client traffic.

Sandstorm Application Hosting
Application's Perspective

Grain subject to extremely fine sandboxing:

- Filesystem (private mount namespace) contains *only*:
  - grain's application mounted read-only
  - grain's data mounted read-write
  - Minimal collection of "device" nodes
- Native network access limited to "dummy" interface.
- Many syscalls are disabled via `seccomp-bpf`.

Sandstorm Application Hosting
Application's Perspective

Grain software is born (`exec()`'d) with a socket to the supervisor. All communication flows over this socket.

- Outbound network requests overseen by supervisor!
- Inbound requests, naturally, too.

Sandstorm Application Hosting
Application's Perspective

Grain software is born (`exec()`'d) with a socket to the supervisor. All communication flows over this socket.

- Outbound network requests overseen by supervisor!
- Inbound requests, naturally, too.
- Uses "Cap'n Proto" capability-based RPC.

95% of CVEs?

Sandstorm project claims

> *95% of (application) security issues automatically mitigated, before they were discovered.*

That is borne out by the data:

- 20 CVEs in sampled applications (some restrictions apply)
  - Only one, an XSS exploit, was not fully mitigated.
  - All path traversal bugs (4) mooted.
  - Most code injection bugs (2 of 3) required write access to the grain to execute; 3$^{\text{rd}}$ in typically unshared grains.
  - Authn (3) & authz (2) bugs eliminated: supervisor's job!
- Additionally: 27 (of 224) Linux kernel CVEs considered; only 3 pose threat to Sandstorm hosts.

95% of CVEs?

However, capabilities and sandboxing are not a panacea!

- Still possible to have bad authz checks in applications.
- May be difficult to draw sandbox boundaries neatly in all cases; authz, path traversal, and/or code injection bugs here could still lead to unintentional information disclosure.

The hope is that this approach...

- *rules out or confines damage from certain classes of bugs*
- *makes it easier to write secure multi-user applications*
- Provides new slogan and grounds for LangSec: "*Every well-formed request is authorized*" means that *parsers* become the place for authn checks.

The hope is that this approach...

- *rules out or confines damage from certain classes of bugs*
- *makes it easier to write secure multi-user applications*
- Provides new slogan and grounds for LangSec: "*Every well-formed request is authorized*" means that *parsers* become the place for authn checks.

Questions?