Intrusion Recovery for Database-backed Web Applications

Ramesh Chandra, Taesoo Kim, Meelap Shah, Neha Narula, Nickolai Zeldovich

MIT CSAIL
Web applications routinely compromised

Teen uses worm to boost ratings on MySpace.com

It did little damage but could point to broader vulnerabilities, says a security expert

By Eric Lai
October 17, 2005 12:00 PM ET

Computerworld - Using a self-propagating worm that exploits a scripting vulnerability common to most dynamic Web sites, a Los Angeles teenager made himself the most popular member of community Web site MySpace.com earlier this month. While the attack caused little damage, the technique could be used to destroy Web site data or steal private information -- even from enterprise users behind protected networks, according to an security services firm.

The unknown 19-year-old, who used the name "Samy," put a small bit of code in his user profile on MySpace, a 32-million-member site, most of whom are under age 30. Whenever Samy's profile was viewed, the code was executed in the background, adding Samy to the viewer's list of friends and writing at the bottom of their profile, "... and Samy is my hero."
"This is an attack on the users of the Web site, using the Web site itself."
Web applications routinely compromised
Web applications routinely compromised
Recovering integrity is important

- Preventing intrusions is important, but compromises will still happen
  - Vulnerabilities are common, and new bugs are constantly being found [CVE]
    - 3-4 new vulnerabilities found per day, on average for the past 4 years
  - Administrators misconfigure policies, settings

- This talk: recovering integrity after attack
Cross-site scripting (XSS) bugs (simplified)

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Wiki pages table

Eve's browser

Alice's browser

Wiki Server
Cross-site scripting (XSS) bugs (simplified)

Eve's browser

Edit page:
<script>httpReq("/addAcl?u=Eve")</script>

Alice's browser

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&lt;script&gt;...&lt;/script&gt;</td>
</tr>
</tbody>
</table>
Cross-site scripting (XSS) bugs (simplified)

Wiki Server

Alice's browser

Eve's browser

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

httpReq("/addAcl?u=Eve")
Cross-site scripting (XSS) bugs (simplified)

Eve's browser

<script>
httpReq("/addAcl?u=Eve")
</script>

Alice's browser

Attack code runs as Alice

Wiki Server

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>
Cross-site scripting (XSS) bugs (simplified)

Wiki Server

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Add Eve to ACL: `/addAcl?u=Eve`

Legitimate requests

Eve's browser

`<script>`

httpReq(`"/addAcl?u=Eve"`)`</script>`

Alice's browser

Attack code runs as Alice
Recovering web application integrity is hard

- Web apps store data in shared data store
  - Multiple users data is commingled
  - Users access each other's data

- Makes recovering from attack complicated:
  - Attack propagates across users
  - Attack can arbitrarily corrupt user data
    - e.g., financial information
  - Attack can install backdoors
    - e.g., modify ACLs, install Google apps scripts
Limited recovery tools

- Backup-and-restore tools
  - Attack may be detected days or weeks later
  - Restoring from backup discards *all* users' changes

- Manual recovery
  - Admin spends days or weeks tracking attack's effects
  - Admin could miss a subtle backdoor or corruption
Contributions

- Warp: web application intrusion recovery
  - Undoes effects of attack but keeps legitimate changes
  - Works for real applications: MediaWiki, Drupal, Gallery2

- Key ideas:
  - *Retroactive patching* eliminates need to pinpoint attack
  - *Time-travel DB* precisely tracks causal effects
  - *DOM-level replay* preserves users' intended changes
High-level approach: rollback and re-execute

- Normal execution
  - Record *actions* in system to a log
  - Record causal dependencies between actions
  - Record checkpoints system state

- Repair
  - Identify attack action
  - Rollback *affected* system state to before attack
  - Replay all affected actions except attack action
Normal execution

Wiki Server

Eve's browser

Alice's browser

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Warp state

Dependencies

Alice's Req

Eve's Req

...
Normal execution

Eve's browser

Log HTTP requests

Log database queries

Alice's browser

Warp state

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Server actions

Dependencies

Eve's Req

Alice's Req

Time
Normal execution

Eve's browser

Alice's browser

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&lt;script&gt;...&lt;/script&gt;</td>
</tr>
</tbody>
</table>

Maintain checkpoints of database

Checkpoints
Server actions
Dependencies
Warp state

Eve's Req

Alice's Req

...
Normal execution

Wiki Server

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Wiki pages table

Dependencies

Checkpoints

Server actions

Dependencies

Client actions

Warp state

Eve's browser

Record user actions using a browser extension

Alice's browser

Client actions

Eve's Req

Alice's Req
Strawman repair

- Eve's browser
- Alice's browser

Wiki Server

Eve's Req

Alice's Req

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&lt;script&gt;...&lt;/script&gt;</td>
</tr>
</tbody>
</table>

Checkpoints

Server actions

Dependencies

Client actions

Warp state

Time
Repair: identify attack

Eve's browser

Attack action

Alice's browser

Wiki Server

Eve's Req

Alice's Req

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Checkpoints
Server actions
Dependencies
Client actions
Warp state

Time
Repair: rollback to before attack

Eve's browser

Rollback DB

Alice's browser

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Checkpoints
Server actions
Dependencies
Client actions
Warp state
Repair: skip attack action

- Eve's browser
- Alice's browser

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Warp logs

- Dependencies
- Server actions
- Client actions
- Checkpoints

Warp state
Repair: re-execute subsequent actions

Re-execute Alice's actions in shadow browser

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Checkpoints
Server actions
Dependencies
Client actions
Warp state

Alice's browser
Client actions

Eve's browser

Eve's Req

Alice's Req
Repair: re-execute subsequent actions

Wiki Server

Eve's browser

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Alice's browser

New response: Welcome!!

Alice's Req

Wiki Server

Warp logs

Dependencies

Server ID

Text

5

Welcome!!

Warp client log

Client actions

No attack code

Checkpoints

Server actions

Dependencies

Client actions

Warp state

Time
Repair: re-execute subsequent actions

Wiki Server

Eve's browser

Eve's Req

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Client actions

Preserve legitimate requests

Skip attack requests

Alice's browser

Checkpoints
Server actions
Dependencies
Client actions

Warp state

Time
Challenges to intrusion recovery

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Alice's browser

Eve's browser
Challenges to intrusion recovery

Must pinpoint intrusion in a complex app

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Alice's browser

Eve's browser

Welcome!!

Must pinpoint intrusion in a complex app
Challenges to intrusion recovery

- Must pinpoint intrusion in a complex app
- Must reduce unnecessary re-execution

Wiki Server

Eve's browser
- Welcome!!

Bob's browser
- Must reduce unnecessary re-execution

Alice's browser
- Must pinpoint intrusion in a complex app
- Welcome!!
Challenges to intrusion recovery

- Must pinpoint intrusion in a complex app
- Must reduce unnecessary re-execution
- Must reduce user involvement during repair

Alice's browser
Bob's browser
Wiki pages table
Eve's browser
Wiki Server

Welcome!!
Welcome!!
Next

Challenge 1: intrusion detection is difficult

Eve's browser

Need expert to pinpoint attack

Alice's browser

Wiki Server

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&lt;script&gt;...&lt;/script&gt;</td>
</tr>
</tbody>
</table>
Idea: retroactive patching

- Key observation: patch renders attacks harmless

- Approach:
  - Retroactively apply security patches back in time
  - Re-execute all affected requests
Retroactive patching

Eve's browser

Alice's browser

Wiki Server

Eve's Req

Alice's Req

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Welcome!!
Retroactive patching: normal execution

Eve's browser

Alice's browser

---

Wiki Server

main.php

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>&lt;script&gt;...&lt;/script&gt;</td>
</tr>
</tbody>
</table>

Eve's Req

Alice's Req

---

Eve's browser

Alice's browser
Retroactive patching: repair

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>&lt;script&gt;...&lt;/script&gt;</code></td>
</tr>
</tbody>
</table>

Alice's browser

Eve's browser

Alice's Request

Eve's Request
Rollback to before vulnerability was introduced

Wiki Server

Alice's browser

Eve's browser

Alice's Req

Eve's Req

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

main.php

...
Retroactive patching: apply patch in the past

- Eve's browser
- Alice's browser
- Wiki Server
- Wiki pages table
<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>
Retroactive patching: rerun affected requests

Alice's browser

Eve's browser

Edit page:
<script>httpReq("/addAcl?u=Eve")
</script>

Wiki Server

main.php

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Welcome!!</td>
</tr>
</tbody>
</table>

Alice's Req

Eve's Req
Retroactive patching: rerun affected requests

Wiki Server

Alice's Req

Eve's Req

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Sanitized input</td>
</tr>
</tbody>
</table>

Alice's browser

Eve's browser

Edit page:
<script>
httpReq("/addAcl?u=Eve")
</script>
Retroactive patching: rerun affected requests

Wiki pages table

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Sanitized input</td>
</tr>
</tbody>
</table>

Eve's browser

Alice's browser

New response: Sanitized input

main.php

Alice's Req

Eve's Req

Wiki Server

Time

Alice's Req

Eve's Req
Retroactive patching: rerun affected requests

Eve's browser

New response: Sanitized input

Alice's browser

Skip attack requests

Wiki Server
Retroactive patching: rerun affected requests

Wiki Server

Eve's browser

Wiki pages table

ID | Text
---|---
5  | Sanitized input

Do not need expert, just the patch
Challenge 2: reduce re-execution

- Warp re-executes requests for two reasons:
  - Request depends on attack
    - Results would be different without attack
    - Need: precise dependency tracking
  - Request re-executed to reapply legitimate changes
    - Need: avoid unnecessary rollback
Focus: database dependencies

- Dependencies arise due to shared state
- Web apps store state in database
- Must compute dependencies between SQL queries
Goals for dependency tracking

- Precise
  - Avoid false dependencies
  - Important because web applications often manage many independent pieces of data

- Fast
  - Track dependencies without re-running the queries
  - Important because web applications often handle many independent requests
Dependency tracking strawmen

- Whole-table dependencies: fast but not precise
  - Reads depend on all prior writes on same table
  - Can determine table names in queries by statically looking at query's table list
  - False dependencies: queries can access independent rows in same table

- Re-execute reads: precise but slow
  - Re-execute each read, compare results before & after
  - Slow: requires re-executing every single read query
Achieving precise and static dependency tracking is hard

- Queries name rows by different attributes (columns)
- Queries do not specify every attribute

```
UPDATE ID=5

ID | Text | Category  
---|------|----------
 5 | ...  | Science  

SELECT Category=Science
```

Database table
Solution: record write attributes at runtime

- For each write, record all attribute values of affected rows
- For reads, statically determine dependencies based on query's WHERE clause (easy + fast)
Solution: record write attributes at runtime

- For each write, record all attribute values of affected rows
- For reads, statically determine dependencies based on query's WHERE clause (easy + fast)

```
UPDATE ID=5

SELECT Category=Science

Possible dependency attributes
```
Solution: record write attributes at runtime

- For each write, record all attribute values of affected rows
- For reads, statically determine dependencies based on query's WHERE clause (easy + fast)
Challenge 2: reduce re-execution

- Warp re-executes requests for two reasons:
  - Request depends on attack
    - Results would be different without attack
    - Need: precise dependency tracking
  - Request re-executed to reapply legitimate changes
    - Need: avoid unnecessary rollback
Approach to avoiding unnecessary rollback

- Roll back only affected parts of the database
  - No need to re-apply changes to unaffected rows
  - Technique: row-level rollback

- Allow rolling back to any point in time
  - Helps avoid rolling back too far
  - No need to re-apply changes from before the attack
  - Technique: continuous checkpointing
Solution: continuous row-level checkpoints

- Keep track of all versions of every row over time
- Can roll back individual rows to any point in time

<table>
<thead>
<tr>
<th>ID</th>
<th>From</th>
<th>To</th>
<th>Text</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>∞</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution: continuous row-level checkpoints

- Keep track of all versions of every row over time
- Can roll back individual rows to any point in time

Valid time period

<table>
<thead>
<tr>
<th>ID</th>
<th>From</th>
<th>To</th>
<th>Text</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>∞</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>∞</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>9</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>∞</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

*Time-travel DB*: dependency tracking + continuous row-level checkpoints
Challenge 3: reduce user involvement during repair

- Pixel-level replay of user actions often meaningless
- Results in a *conflict*

**Sample attack text**

**Are They Your Friends Too?**

There's an easy new way to discover friends from your hometown, school, employer and more. Give it a try.

- Naina Mishra
  - Add Friend
- Christopher Kanni Ranjan
  - 6 mutual friends
  - Add Friend
- Sankar Chebiyam
  - Add Friend
- Pravin Kodi
  - 7 mutual friends
  - Add Friend

**User originally clicked this button**

**Pixel-level replay misses user intent**
Idea: DOM-level replay

- Key observation: DOM has structure
  - Changing one element does not affect other elements
  - User action's intent tied to DOM element
Idea: DOM-level replay

- Key observation: DOM has structure
  - Changing one element does not affect other elements
  - User action's intent tied to DOM element
Idea: DOM-level replay

- Normal execution
  - Record user actions on DOM elements using a browser extension

- Repair
  - Replay user actions if DOM element unchanged
  - Three-way merge for text input elements
  - If DOM element changed, flag a conflict
Putting it together

- Eve's browser
- Bob's browser
- Alice's browser

Wiki Server

- Retroactive patching
- Time-travel database
- DOM-level replay
- Wiki pages table

Time

Welcome!!
Warp: Web application repair

- Prototype implementation of Warp
  - Postgres DB: SQL query rewriting
  - PHP, Apache: log requests, non-deterministic calls
  - Firefox: browser extension, upload log, re-execution

- Total: 8,500 lines of code (C, PHP, Python, JS)
Evaluation questions

- Can Warp support real applications?
- Can Warp recover from real attacks?
- What do the admin, users have to do?
- What are the runtime overheads of Warp?
- How long does repair take?
Warp works for real applications

- Ported three applications to run on Warp
  - MediaWiki (Wikipedia software)
  - Drupal (content management system)
  - Gallery2 (photo album software)
Warp works for real applications

- Ported three applications to run on Warp
  - MediaWiki (Wikipedia software)
  - Drupal (content management system)
  - Gallery2 (photo album software)
- No application source code changes
- Tens of lines of annotations on SQL schema, to specify columns for dependency tracking
- Yet, can recover integrity after attacks
MediaWiki attack workload

- Use five real vulnerabilities
  - One attacker, 3 victims
    - Attacker injects Javascript into a page
    - Attack code runs in victim's browsers
    - Attack code edits Wiki pages, …
    - Victims also browse and edit pages
  - 96 other users browse random Wiki pages, make edits
- One admin mistake
Warp recovers from wide range of attacks on MediaWiki

<table>
<thead>
<tr>
<th>Attack</th>
<th>Initiating repair</th>
<th>User conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>SQL injection</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>Admin-initiated</td>
<td>1</td>
</tr>
<tr>
<td>CSRF</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>Retroactive patching</td>
<td>3</td>
</tr>
</tbody>
</table>
Initiating recovery requires little effort

<table>
<thead>
<tr>
<th>Attack</th>
<th>Initiating repair</th>
<th>User conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>SQL injection</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>Admin-initiated</td>
<td>1</td>
</tr>
<tr>
<td>CSRF</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>Retroactive patching</td>
<td>3</td>
</tr>
</tbody>
</table>

Retroactive patching can use real MediaWiki patches.
Warp's recovery is mostly automatic

<table>
<thead>
<tr>
<th>Attack</th>
<th>Initiating repair</th>
<th>User conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>SQL injection</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>Admin-initiated</td>
<td>1</td>
</tr>
<tr>
<td>CSRF</td>
<td>Retroactive patching</td>
<td>0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>Retroactive patching</td>
<td>3</td>
</tr>
</tbody>
</table>

Warp incurs few conflicts, corresponding to real attack side-effects.
Warp has low overheads

<table>
<thead>
<tr>
<th>Workload</th>
<th>Page visit/s without Warp</th>
<th>Page visit/s with Warp</th>
<th>Warp log / page visit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading</td>
<td>8.46</td>
<td>6.43</td>
<td>3.71 KB</td>
</tr>
<tr>
<td>Editing</td>
<td>7.19</td>
<td>5.26</td>
<td>7.34 KB</td>
</tr>
</tbody>
</table>

- 24-27% throughput reduction in the server
- 1TB disk stores one year's worth of logs, for one server at 100% load
- Negligible overhead for logging in the browser
Warp avoids significant re-execution

<table>
<thead>
<tr>
<th>Attack</th>
<th>Queries re-exec</th>
<th>Queries total</th>
<th>Repair time (s)</th>
<th>Orig time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>258</td>
<td>24,746</td>
<td>17.9</td>
<td>180.0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>293</td>
<td>24,740</td>
<td>16.7</td>
<td>179.2</td>
</tr>
<tr>
<td>SQL injection</td>
<td>524</td>
<td>24,541</td>
<td>29.7</td>
<td>177.8</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>185</td>
<td>24,326</td>
<td>10.8</td>
<td>176.5</td>
</tr>
<tr>
<td>CSRF</td>
<td>19,799</td>
<td>24,578</td>
<td>1,644</td>
<td>175.0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>23,227</td>
<td>24,641</td>
<td>1,751</td>
<td>174.3</td>
</tr>
</tbody>
</table>
Warp avoids significant re-execution

<table>
<thead>
<tr>
<th>Attack</th>
<th>Queries re-exec</th>
<th>Queries total</th>
<th>Repair time (s)</th>
<th>Orig time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>258</td>
<td>24,746</td>
<td>17.9</td>
<td>180.0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>293</td>
<td>24,740</td>
<td>16.7</td>
<td>179.2</td>
</tr>
<tr>
<td>SQL injection</td>
<td>524</td>
<td>24,541</td>
<td>29.7</td>
<td>177.8</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>185</td>
<td>24,326</td>
<td>10.8</td>
<td>176.5</td>
</tr>
<tr>
<td>CSRF</td>
<td>19,799</td>
<td>24,578</td>
<td>1,644</td>
<td>175.0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>23,227</td>
<td>24,641</td>
<td>1,751</td>
<td>174.3</td>
</tr>
</tbody>
</table>

Warp re-executes a fraction of the original execution

Warp's repair time is order of magnitude smaller
Warp avoids significant re-execution

<table>
<thead>
<tr>
<th>Attack</th>
<th>Queries re-exec</th>
<th>Queries total</th>
<th>Repair time (s)</th>
<th>Orig time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>258</td>
<td>24,746</td>
<td>17.9</td>
<td>180.0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>293</td>
<td>24,740</td>
<td>16.7</td>
<td>179.2</td>
</tr>
<tr>
<td>SQL injection</td>
<td>524</td>
<td>24,541</td>
<td>29.7</td>
<td>177.8</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>185</td>
<td>24,326</td>
<td>10.8</td>
<td>176.5</td>
</tr>
<tr>
<td>CSRF</td>
<td>19,799</td>
<td>24,578</td>
<td>1,644</td>
<td>175.0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>23,227</td>
<td>24,641</td>
<td>1,751</td>
<td>174.3</td>
</tr>
</tbody>
</table>

Some patches require re-running all requests
Warp avoids significant re-execution

<table>
<thead>
<tr>
<th>Attack</th>
<th>Queries re-exec</th>
<th>Queries total</th>
<th>Repair time (s)</th>
<th>Orig time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>258</td>
<td>24,746</td>
<td>17.9</td>
<td>180.0</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>293</td>
<td>24,740</td>
<td>16.7</td>
<td>179.2</td>
</tr>
<tr>
<td>SQL injection</td>
<td>524</td>
<td>24,541</td>
<td>29.7</td>
<td>177.8</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>185</td>
<td>24,326</td>
<td>10.8</td>
<td>176.5</td>
</tr>
<tr>
<td>CSRF</td>
<td>19,799</td>
<td>24,578</td>
<td>1,644</td>
<td>175.0</td>
</tr>
<tr>
<td>Clickjacking</td>
<td>23,227</td>
<td>24,641</td>
<td>1,751</td>
<td>174.3</td>
</tr>
</tbody>
</table>

Full re-execution slow in unoptimized prototype
**Warp's repair algorithm scales well**

<table>
<thead>
<tr>
<th>Attack</th>
<th>Orig. time (s)</th>
<th>Repair time (s)</th>
<th>Orig. time</th>
<th>Repair time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflected XSS</td>
<td>180.04</td>
<td>17.87</td>
<td>49.2X</td>
<td>2.7X</td>
</tr>
<tr>
<td>Stored XSS</td>
<td>179.22</td>
<td>16.74</td>
<td>49.3X</td>
<td>3.3X</td>
</tr>
<tr>
<td>SQL injection</td>
<td>177.82</td>
<td>29.70</td>
<td>49.9X</td>
<td>9.2X</td>
</tr>
<tr>
<td>ACL mistake</td>
<td>176.52</td>
<td>10.75</td>
<td>50.3X</td>
<td>3.9X</td>
</tr>
</tbody>
</table>

50X workload, only 3-9X repair time
Related work

- Intrusion recovery:
  - Retro [Kim10], Taser [Goel05]: OS-level recovery inefficient for database recovery
  - Akkus and Goel [Akkus10]: only recovers from mistakes, requires manual guidance

- Deterministic record and replay: ReVirt [Dunlap02], Mugshot [Mickens10]
  - Cannot replay once something changes

- Vulnerability-specific predicates [Joshi05]:
  - Manual effort for each bug
Summary

- Intrusions are commonplace and inevitable
- Few recovery tools for web applications

*Warp* restores integrity after attack
  - Retroactive patching, time-travel DB, DOM replay
  - Works for real apps: MediaWiki, Drupal, Gallery2

*Warp* recovers from wide range of attacks
Thank you!