Asynchronous intrusion recovery for interconnected web services

Ramesh Chandra, Taesoo Kim, Nickolai Zeldovich

MIT CSAIL
Today's web services are highly interconnected

• Many web services provide APIs to other sites

• Many websites integrate those APIs:
  – Authentication: Facebook Connect, Google+ ...
  – Data sharing: Dropbox ...
  – Business process management: Salesforce ...
  – ...

Example: online shopping mall

Customer Relationship Management (CRM)
Example: online shopping mall

Financial Force (Accounting Service)

Adobe Echo Sign (E-Signature Service)

Bill.ON (Invoices and Billing Service)
Example: online shopping mall

Allow Facebook users to buy our products without registration

Financial Force (Accounting Service)

Adobe Echo Sign (E-Signature Service)

Bill.ON (Invoices and Billing Service)
Example: online shopping mall

- Allow Facebook users to buy our products without registration
- Financial Force (Accounting Service)
- Adobe Echo Sign (E-Signature Service)
- Address in Facebook
- Bill.ON (Invoices and Billing Service)
Attack in one service can spread between services

- Financial Force (Accounting Service)
- Adobe Echo Sign (E-Signature Service)
- Facebook
- Twitter
- CRM
- Bill.ON (Invoices and Billing Service)

Ship purchased products to ...
Address modified by Attacker
Bugs in web services are commonplace

- Facebook (Mar 29\textsuperscript{th} 2013):
  - Attackers can intercept full permission access tokens
Bugs in web services are commonplace

- Facebook (Mar 29th 2013):
  - Attackers can intercept full permission access tokens
- Many web services have similar bugs
  - Twitter (Aug 20th 2013)
  - Instagram (May 2nd 2013)
  - Microsoft Yammer (Aug 4th 2013)

---

**SECURITY**

Bloke leaks '1000s' of Twitter login tokens, says he can hack ANY twit

Known vulnerability sat on by TV

A hacker calling himself the TV

Nir Goldshlager known as Facebook hacker and founder of Break Security, who reported many critical bugs in Facebook OAuth mechanism in past few months, today disclose a critical vulnerability in Instagram OAuth that allow an attacker to hack any account.

Successful hack allows attacker to access private photos, ability to delete victim's photos and to edit comments and also the ability to post new photos.

Microsoft's Social network Yammer vulnerable to OAuth Bypass hack

Moht Kumar, The Hacker News - Sunday, August 04, 2013

Yammer

The Enterprise Social Network
Goal

- **Recovering integrity** in interconnected services
  - Repair the state of affected services as if the attack never occurred

- State-of-the-art: manual recovery
  - Admin doesn't trust other sites for recovery
  - Require manual interaction (e.g., email other admin)
General plan for automatic recovery

- Use **rollback-and-replay** for recovering integrity in single machine
  - Prior works: Retro [OSDI '10], Warp [SOSP '11]

- **Extend** rollback-and-replay to **many web services**!
Challenges

• Rollback-and-replay requires **global coordinator**
  - Each service cannot decide what to do for repair

• All services must be **available** during recovery
  - We want to repair some services even if others are down
  - Consistency problem: some services are not repaired yet
Contributions

Enable **automatic** intrusion recovery in **distributed** web services

1. **Repair protocol** between services
   - No central coordinator
   - Each service controls its repair

2. **Asynchronous repair**
   - Proceed repair even with unavailable services
   - Consistency in partially repair state
Running example of an attack

Financial Force
(Accounting Service)

Adobe Echo Sign
(E-Signature Service)

Bill.ON
(Invoices and Billing Service)

Ship purchased products to ...

Address modified by Attacker
Running example of an attack

Bill.ON
(Invoices and Billing Service)
Running example of an attack

Attacker

Facebook

Victim

http://bit.ly/1xoTn

Bill.ON (Invoices and Billing Service)
Running example of an attack

Attacker

Facebook

Victim

http://bit.ly/1xoTn

Bill.ON
(Invoices and Billing Service)
Running example of an attack

Attacker

Facebook

Victim

http://bit.ly/1xoTn

Bill.ON
(Invoices and Billing Service)
Running example of an attack

Attacker

Facebook

Victim

http://bit.ly/1xoTn

Bill.ON
(Invoices and Billing Service)
Running example of an attack

**Attacker**

Facebook

**Victim**

Modify address

http://bit.ly/1xoTn

Address modified by Attacker

Bill.ON
(Invoices and Billing Service)
Timeline of the attack

Attacker

Victim

Facebook

Shopping Mall

Bill.ON
Timeline of the attack

Attacker

Victim

Facebook

Shopping Mall

Bill.ON

Time
Timeline of the attack:

- Attacker
- Victim
- Facebook

Events:
- Shopping Mall
- Bill.ON
Timeline of the attack
Goal: attack did not take place
Goal: attack did not take place
Overview of system execution

• Normal execution:
  – Record enough information for rollback-and-replay

• Repair:
  – Identify an attack to initiate repair
  – Repair local state: rollback and replay recorded requests
  – Propagate repair whenever local repair affects others
Overview of system execution

- **Normal execution:**
  - Record enough information for rollback-and-replay

- **Repair:**
  - Identify an attack to initiate repair
  - Repair local state: rollback and replay recorded requests
  - Propagate repair whenever local repair affects others
Strawman: repair with global coordinator using rollback-and-replay

Attacker

Victim

Facebook

Identify an attack for repair

Shopping Mall

Bill.ON

Rollback state before the attack occurred.
Strawman: repair with global coordinator using rollback-and-replay
Strawman: repair with global coordinator using rollback-and-replay

Rollback state before the attack occurred
Strawman: repair with global coordinator using rollback-and-replay
Strawman: repair with global coordinator using rollback-and-replay

Attacker
 Victim
 Facebook

Rollback state before the attack occurred

Error
Original address

Shopping Mall

Bill.ON
Strawman: repair with global coordinator using rollback-and-replay
Problems in Strawman design

• P1. All services must be available
  → Support asynchronous repair with speculation

• P2. Require global coordinator
  → Define repair APIs between services
Problems in Strawman design

• P1. All services must be available
  → Support asynchronous repair with speculation

• P2. Require global coordinator
  → Define repair APIs between services
Challenge: cooperating with unavailable web services

Wait for other services to come up?
Solution: asynchronous repair

- Asynchronously deliver repair requests
- Speculatively proceed local repair with past responses (or timeout responses)
- Expose repaired state after local repair

Intuition: why asynchronous repair works?
- Many web services are designed for independent operation, prepared for handling others failures
Example: asynchronous repair
Example: asynchronous repair

Asynchronously deliver new response

Speculatively proceed with past request
Example: asynchronous repair

- Attacker
- Victim
- Facebook
- Repair queues
- Shopping Mall
- Bill.ON

- Speculatively proceed with past request
- Asynchronously deliver new response
Example: asynchronous repair

- Attacker
- Victim
- Facebook

Speculatively proceed with past request

Asynchronously deliver new response
Example: exposing state after local repair

Attacker

Victim

Facebook

Shopping Mall

Bill.ON

Another web service

Two services are still repairing
What if speculation fails?

- If service responds differently,
  - **Restart local repair** with the new response
  - In fact, it is not different from initiating new repair

- Asynchronous repair will converge to the correctly repaired state at the end
Example: speculation failure

Message:

Mall shared a link.
September 19, 2012 near Cambridge

Ready for shipping to:

Facebook

Shopping Mall

ok
Example: speculation failure

Message:

Mall shared a link.
September 19, 2012 near Cambridge

Ready for shipping to:

Following request depends on previous request
Example: speculation failure

Message:
Mall shared a link.
September 19, 2012 near Cambridge

Ready for shipping to: [Anonymous]
Example: speculation failure

Message:

Mall shared a link.
September 19, 2012 near Cambridge

Ready for shipping to: [Anonymous]

Facebook

Shopping Mall

Message:

Mall shared a link.
September 19, 2012 near Cambridge

Ready for shipping to: [Anonymous]

ok

Respond with different result
Example: speculation failure

Facebook

Shopping Mall

Message:

Mall

Shared a link.

September 19, 2012 near Cambridge

Ready for shipping to:

ok
Example: speculation failure

Facebook

Shopping Mall

Message:
Mall shared a link.
September 19, 2012 near Cambridge

Ready for shipping to:
Example: speculation failure

Asynchronous repair makes **forward progress** in time-line graph, so it will **converge** to the correctly repaired state at the end.
Consistency problem: partially repaired state

Attacker  Victim  Facebook  Shopping Mall  Bill.ON

... Another web service

Repaired state

Two services are still repairing

Time
Consistency problem: partially repaired state

- Exposing partially repaired state might diverge global state
  - But it is not something new that our recovery mechanism introduces more
  - Most of web services *already cope with this problem*
Exposing partially repaired state might violate service invariants

- **Service invariants**: guarantees by service provider
  (e.g., locking service: when lock is held, no concurrent access)
- **In theory**: yes (for arbitrary tightly coupled systems)
- **In practice**: no
  - RESTful APIs usually provide consistency per API
  - Web services are in nature **loosely coupled**
Consistency: partial repair state

- Services and clients already deal with concurrency
- Repair of a service is modeled as:
  - Being performed by a concurrent repair actor client
  - That uses the service's regular API calls
- So, partially repaired state can be considered as state resulting from yet another concurrent operations
Problems in Strawman design

• P1. All services must be available
  → Support asynchronous repair with speculation

• P2. Require global coordinator
  → Define repair APIs between services
How to propagate repair requests without global coordinator?

Attacker  Victim  Facebook

How to ask services to initiate repair?

Shopping Mall  Bill.ON
Requesting repair with APIs (RPC)

- Tag each request in normal exec.
- Each service runs repair controller

API: modify the previous response

replace_response(#7, )
Repair APIs (RPC)

- No centralized coordinator, each server invokes following repair APIs to recover from the attack
  - \texttt{replace\_response}(tag, resp): replace past response
  - \texttt{replace\_request}(tag, req): replace past request
  - \texttt{delete}(tag): delete past request
  - \texttt{create}(req, before, after): execute new requests in the past
Repair APIs (RPC)

- No centralized coordinator, each server invokes following repair APIs to recover from the attack
  - `replace_response(tag, resp)`: replace past response
  - `replace_request(tag, req)`: replace past request
  - `delete(tag)`: delete past request
  - `create(req, before, after)`: execute new requests in the past

If service supports those 4 APIs, it can participate in decentralized recovery.
Authentication of repair APIs

- Too application specific
  - (e.g. Email service: sender can delete recipient's emails?)

- **Delegate** authentication to original web services
  - Implement application specific policy
    - (e.g. ask admin for confirmation of repair)
  - Assign a credential to repair requests
Summary of design

1. **Asynchronous repair**
   - Proceed repair with offline or unavailable services
   - Consistency in partially repair state

2. **Repair APIs** between services
   - No central coordinator
   - Each service controls its repair
   - Delegate authentication
Implementation

• Prototype implementation: **Aire**
  
  – Extend **Django** web framework
  – Support existing Django app. with **few modifications**
    • Support Askbot, Django-OAuth, and Dpaste
    • e.g., Askbot's authentication policy: 55 LoC
  
  – Total: 5700 lines of Python code
Evaluation questions

• Can Aire support real web services?
• Can Aire recover from distributed attacks?
• What are the runtime overheads of Aire?
Aire supports real web services

Askbot

OAuth Provider

Dpaste
Aire supports real web services

Askbot

OAuth Provider

Dpaste
Aire supports real web services

Askbot

OAuth Provider

Dpaste
Aire supports real web services

Askbot

OAuth Provider

Dpaste
Aire supports real web services

Askbot

OAuth Provider

Dpaste

Post code

Append a link to Dpaste

Share link: http://dpaste.com/4324
Aire supports real web services

Askbot + OAuth + Dpaste = 183K LoC!

Aire can support large Django web applications

Append a link to Dpaste

Share link: http://dpaste.com/4324
Aire enables automatic recovery

Askbot

OAuth Provider

Dpaste

Hi there! Please sign in help

ALL UNANSWERED search or ask your question

6,254 questions Sort by » by date by activity ▼ by

Can't log in into user account from previous distribution

…

Multiboot UEFI WIN8/Fedora19: GRUB not starting ==> Fedora not booting

# grub2 # fedora19 # uEFI # bootloader # laptop

Hi there! Please sign in help

ALL UNANSWERED search or ask your question

Content of grub.cfg:

Multiboot UEFI WIN8/Fedora19: GRUB not booting

# DO NOT EDIT THIS FILE
# It is automatically generated by g
# from /etc/grub.d and settings from
#
### BEGIN /etc/grub.d/99_header ###
if [ -s $prefix/grubenv ]; then
  load_env
fi

Share link: http://dpaste.com/4324

Append a link to Dpaste
Aire enables automatic recovery

Askbot

Dpaste

OAuth Provider

Post code

Append a link to Dpaste

Share link: http://dpaste.com/4324
Aire enables automatic recovery

Askbot

OAuth Provider

Dpaste

Share link: http://dpaste.com/4324

Append a link to Dpaste

Post code

Multiboot UEFI WIN8/Fedora19: GRUB not starting => Fedora not booting

# grub2  # fedora19  # uEFI  # bootloader  # laptop

Multiboot UEFI WIN8/Fedora19: GRUB not starting => Fedora not booting

Content of /etc/grub.d/00_header:

```bash
# DO NOT EDIT THIS FILE
#
# It is automatically generated by /etc/grub.d/00_header
#
### BEGIN /etc/grub.d/00_header ###
if [ -s $prefix/grubenv ]; then
  load_env
fi
```
Aire enables automatic recovery

- Askbot, OAuth, and Dpaste are correctly recovered
  - Even when Dpaste is temporary unavailable
  - Even when Dpaste goes offline

- More examples in paper:
  - Intrusion recovery (synthetic)
  - Mistakes on ACL setting
  - Misconfigured versioning spreadsheet
Aire has moderate runtime overheads

- 19-30% throughput reduction
- 5-9KB/req storage overheads

→ Moderate overheads for websites which care integrity more than performance
Aire's repair is efficient

<table>
<thead>
<tr>
<th></th>
<th>Askbot</th>
<th>OAuth</th>
<th>DPaste</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repaired Reqs</td>
<td>105 / 2196</td>
<td>2 / 9</td>
<td>1 / 496</td>
</tr>
<tr>
<td>Remote repair reqs</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Local repair time</td>
<td>84.06 sec</td>
<td>0.10 sec</td>
<td>3.91 sec</td>
</tr>
<tr>
<td>Normal exec. time</td>
<td>177.58 sec</td>
<td>0.01 sec</td>
<td>0.02 sec</td>
</tr>
</tbody>
</table>

- Experiment setting:
  - Attacker logs in as a victim user and writes a post
  - 100 legitimate users post 5 questions and navigate
  - All users are affected by the attack (read attacker's post)
Aire's repair is efficient.

Repair in Askbot propagates to OAuth and Dpaste.

<table>
<thead>
<tr>
<th></th>
<th>Askbot</th>
<th>OAuth</th>
<th>DPaste</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repaired Reqs</td>
<td>105 / 2196</td>
<td>2 / 9</td>
<td>1 / 496</td>
</tr>
<tr>
<td>Remote repair reqs</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Local repair time</td>
<td>84.06 sec</td>
<td>0.10 sec</td>
<td>3.91 sec</td>
</tr>
<tr>
<td>Normal exec. time</td>
<td>177.58 sec</td>
<td>0.01 sec</td>
<td>0.02 sec</td>
</tr>
</tbody>
</table>

- Experiment setting:
  - Attacker logins as a victim user and writes a post
  - 100 legitimate users post 5 questions and navigate
  - All users are affected by the attack (read attacker's post)
Airo's repair is efficient

5% of requests are repaired

<table>
<thead>
<tr>
<th></th>
<th>Askbot</th>
<th>OAuth</th>
<th>DPaste</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repaired Reqs</td>
<td>105 / 2196</td>
<td>2 / 9</td>
<td>1 / 496</td>
</tr>
<tr>
<td>Remote repair reqs</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Local repair time</td>
<td>84.06 sec</td>
<td>0.10 sec</td>
<td>3.91 sec</td>
</tr>
<tr>
<td>Normal exec. time</td>
<td>177.58 sec</td>
<td>0.01 sec</td>
<td>0.02 sec</td>
</tr>
</tbody>
</table>

- **Experiment setting:**
  - Attacker logins as a victim user and writes a post
  - 100 legitimate users post 5 questions and navigate
  - All users are affected by the attack (read attacker's post)
Aire's repair is efficient

<table>
<thead>
<tr>
<th></th>
<th>Askbot</th>
<th>OAuth</th>
<th>DPaste</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repaired Reqs</td>
<td>105 / 2196</td>
<td>2 / 9</td>
<td>1 / 496</td>
</tr>
<tr>
<td>Remote repair reqs</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Local repair time</td>
<td>84.06 sec</td>
<td>0.10 sec</td>
<td>3.91 sec</td>
</tr>
<tr>
<td>Normal exec. time</td>
<td>177.58 sec</td>
<td>0.01 sec</td>
<td>0.02 sec</td>
</tr>
</tbody>
</table>

Total repair takes **x2 shorter** than normal execution, although **x10 slower** in replaying a request for repair.

- 100 legitimate users post 5 questions and navigate
- All users are affected by the attack (read attacker's post)
Related work

• Intrusion recovery with selective re-execution:
  – Retro [OSDI'10], Warp [SOSP'11]
  → Use them as building blocks for asynchronous repair

• Intrusion recovery in distributed systems:
  – Heat-ray [SOSP'09], Polygraph [EuroSys'09], Dare [APsys'12]
  → Automatic recovery in loosely coupled web services
Summary

• **Aire** recovers integrity of distributed web services
  – Define a *repair protocol*
  – Support *asynchronous* and *decentralized* repair
  – Propose partial repair consistency