Programming Languages and Analyses for Reliable, Available, and Secure Software

Michael Hicks
University of Maryland, College Park
Software runs the world
Software runs the world

• We (sometimes indirectly) interact with devices running (lots of) software every day
Software runs the world

• We (sometimes indirectly) interact with devices running (lots of) software every day
  - Desktops, laptops, routers, smartphones, tablets
Software runs the world

- We (sometimes indirectly) interact with devices running (lots of) software every day
  - Desktops, laptops, routers, smartphones, tablets
  - Coffee makers, TVs, energy meters, medical devices
Software runs the world

• We (sometimes indirectly) interact with devices running (lots of) software every day
  ▪ Desktops, laptops, routers, smartphones, tablets
  ▪ Coffee makers, TVs, energy meters, medical devices
  ▪ Cars, aircraft, weapon systems, nuclear centrifuges
Software failures are disruptive
Software failures are disruptive

• 3/11: Mizuho FG’s ATM system goes down
  ▪ 5,600 machines offline for 24 hours
Software failures are disruptive

• 3/11: Mizuho FG’s ATM system goes down
  ▪ 5,600 machines offline for 24 hours
• 8/10: Toyota Prius brakes fail due to software glitch
  ▪ Ford also issues patch for similar problem
Software failures are disruptive

- 3/11: Mizuho FG’s ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems
**Software failures are disruptive**

- 3/11: Mizuho FG’s ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems
- 3/08: Heartland exposes 134M credit cards
  - SQL injection used to install spyware
Software failures are disruptive

• 3/11: Mizuho FG’s ATM system goes down
  ▪ 5,600 machines offline for 24 hours
• 8/10: Toyota Prius brakes fail due to software glitch
  ▪ Ford also issues patch for similar problem
• 6/10: Stuxnet malware
  ▪ Exploits flaws in industrial control systems
• 3/08: Heartland exposes 134M credit cards
  ▪ SQL injection used to install spyware
• 8/07: LAX offline due to faulty network card
  ▪ 17,000 planes grounded for eight hours
Software failures are disruptive

- 3/11: Mizuho FG’s ATM system goes down
  - 5,600 machines offline for 24 hours
- 8/10: Toyota Prius brakes fail due to software glitch
  - Ford also issues patch for similar problem
- 6/10: Stuxnet malware
  - Exploits flaws in industrial control systems
- 3/08: Heartland exposes 134M credit cards
  - SQL injection used to install spyware
- 8/07: LAX offline due to faulty network card
  - 17,000 planes grounded for eight hours
- 8/03: Northeast, multi-state blackout
  - Race condition in power plant management software cascades
Software updates are disruptive too

- Typically require restarting the program
- interrupts active users / processing
- makes services unavailable
Software updates are disruptive too

• Typically require restarting the program
• interrupts active users / processing
• makes services unavailable
Software updates are disruptive too

- Typically require restarting the program
- Interrupts active users / processing
- Makes services unavailable
Software updates are disruptive too

• Typically require restarting the program
  • interrupts active users / processing
  • makes services unavailable
Software updates are disruptive too

- Typically require restarting the program
- Interrupts active users / processing
- Makes services unavailable
Programming Languages
A vehicle to a solution

• The language **facilitates** and **constrains** software’s implementation
  ▪ To make it **easy to implement a given design**
  ▪ While **discouraging/disallowing poor coding idioms**

• Software tools can play a similar role
  ▪ Enforce/encourage good coding practice
  ▪ Simplify addition of useful features
  ▪ **Apply to existing software in existing languages**
My research

• Tackles problems of software
  ▪ **reliability**: software does what it should
  ▪ **security**: software free from vulnerability
  ▪ **availability**: avoid downtime by updating on the fly
    - and avoid delayed use of security-critical patches and upgrades

• Two-pronged approach
  ▪ **Formalize** and prove key idea is correct
  ▪ **Implement** and **evaluate** idea on real software
    - Using existing software, or write new software in new language
Roadmap

• Dynamic software updating (DSU)
  ▪ Kitsune: Flexible and Efficient DSU for C programs

• Program analysis for security and reliability
  ▪ Knowledge-based security: quantitatively tracking information

• Quick tour of some other work
Dynamic Software Updating (DSU)
Dynamic Software Updating (DSU)

- Goal: **Update programs while they run**
  - Avoid interruptions
    - Overwhelming number of security breaches due to unpatched software
  - Preserve critical program state
Dynamic Software Updating (DSU)

• Goal: **Update programs while they run**
  - Avoid interruptions
    - Overwhelming number of security breaches due to unpatched software
  - Preserve critical program state

• Useful for:
  - **Non-stop services**
    - E.g., Financial processing, air traffic control, network infrastructure
  - **Programs with long-lived connections**
    - E.g., OpenSSH and media streaming
  - **Long-running programs with large in-memory state**
    - E.g., operating systems, caching servers, in-memory databases
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...
Dynamic Software Updating (DSU)

- Run program at the old version
- At some point update to the new version, preserving and updating existing program state
  - existing connections, important data on the stack and heap, program counter, ...
Many forms of DSU now mainstream
Many forms of DSU now mainstream

language run-times
Many forms of DSU now mainstream

Erlang Programming

JAVA

APPLICATION ENHANCER

smallTALK

.NET

LiveRebel

language run-times

app. tools
Many forms of DSU now mainstream

- language run-times
- app. tools
- OSes

Bought by Oracle in 2011
DSU research challenges

• Which mechanisms should we use to update a running program/service?
  ▪ Compilers, binary rewriters, run-time systems, VMs, process migration, ...

• How do we ensure a dynamic update is correct?
  ▪ Formal specifications, static analyses, testing tools, ...

• How do we balance various competing concerns?
  ▪ Flexibility, efficiency, ease-of-use, portability, ...
Our research in DSU

• We have thoroughly researched these questions
  ▪ We have built DSU implementations for C and Java
    [PLDI’06, PLDI‘09x2, HotSWUp’10, OOPSLA’12]
  ▪ We have experience performing dozens of real-world
    updates on a wide variety of programs
  ▪ We have developed methods for systematic testing
    and static analysis to reason about dynamic updates
    [POPL’05, TOPLAS’07, POPL’08, HotSWUp’10, VSTTE’12]
  ▪ We have developed and empirically validated a variety
    of automatic safety checks for ensuring safety [TSE’11]
• Next: Kitsune, new DSU system for C [OOPSLA’12]
DSU state of the art: Transparency

- Goal: work on any program, with no changes
- Assessment: Laudable, but highly impractical
  - At odds with the reasons people use C
    - Control over low-level data representations, explicit resource management, legacy code, high performance
  - Empirical study shows existing transparent update approaches allow incorrect updates [TSE’11]
  - Not as transparent as they seem
    - Often requires refactoring to permit future updates
    - and/or requires satisfying a conservative static pointer analysis
New approach: Kitsune

• Favors **explicitness** over **transparency**
  - Kitsune treats DSU as a **program feature** and helps developers implement and maintain it as such

• Having the developer orchestrate DSU allows:
  - simpler DSU mechanisms
  - easier developer reasoning
  - full flexibility
  - better performance and control

• **Principle:** Pay for what you use
  - Design carefully builds on lessons from earlier work

*Kitsune* (fox) - a shapeshifter according to Japanese folklore
Results

• Applied Kitsune to six open-source programs
  ▪ memcached, redis, icecast, snort: 3-6 mos. of releases
  ▪ Tor, vsftpd: 2, and 4, years of releases, respectively

• Performance overhead in the noise

• Update times typically less than 40ms

• Programmer effort manageable
  ▪ 50-160 LOC per program (largely one-time effort)
    - Program sizes from 5KLOC up to 220KLOC
  ▪ 27-200 LOC of xfgen specs across all releases
    - xfgen is our DSL for writing state transformer functions
Kitsune: whole-program updates

- driver
Kitsune: whole-program updates

1. Load first version
Kitsune: whole-program updates

1. Load first version
2. Run it
1. Load first version
2. Run it
Kitsune: whole-program updates

1. Load first version
2. Run it
3. Call back to driver when update ready
Kitsune: whole-program updates

1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
Kitsune: whole-program updates

1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
Kitsune: whole-program updates

1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
5. Migrate and transform state
Kitsune: whole-program updates

1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
5. Migrate and transform state
6. Free up old resources
Kitsune: whole-program updates

1. Load first version
2. Run it
3. Call back to driver when update ready
4. Load second version
5. Migrate and transform state
6. Free up old resources
7. Continue with new version
Summary:
• For each source file
  • replace gcc -c with composition of kitc and gcc
  • Add -shared flag to linker and include kit-rt.a
  • Allows us to update the entire program at once
To implement DSU as a program feature, Kitsune requires the programmer to:

- Choose **update points**: where updates may take place
- Code for **data migration**: Identify the state to be transformed, and where it should be received in the new code
- Code for **control migration**: Ensure execution reaches the right event loop when the new version restarts
Example single-threaded server

typedef int data;
data *mapping;
int l_fd;

void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    // ... process client requests
  }
}

int main() {
  mapping = malloc(...);
  l_fd = setup_conn();
  while (1) {
    client_loop();
  }
}
Example single-threaded server

typedef int data;
data *mapping;
int l_fd;

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        // ... process client requests
    }
}

int main() {
    mapping = malloc(...);
l_fd = setup_conn();

    while (1) {
        client_loop();
    }
}
typedef int data;
data *mapping;
int l_fd;

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        // ... process client requests
    }
}

int main() {
    mapping = malloc(...);
    l_fd = setup_conn();
    while (1) {
        client_loop();
    }
}

1. Choose update points
   One per long running loop
typedef int data;
data *mapping;
int l_fd;

void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    // ... process client requests
  }
}

int main() {
  mapping = malloc(...);
  l_fd = setup_conn();

  // after modification for Kitsune
  kitsune_update("main");
  client_loop();
}
typedef int data;
data *mapping;
int l_fd;

void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    kitsune_update("client");
    // ... process client requests
  }
}

int main() {
  mapping = malloc(...);
  l_fd = setup_conn();
}

1. Choose update points
One per long running loop

while (1) {
  kitsune_update("main");
  client_loop();
}
typedef int data;
data *mapping;
int l_fd;

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        kitsune_update("client");
        // ... process client requests
    }
}

int main() {
    mapping = malloc(...);
l_fd = setup_conn();

    while (1) {
        kitsune_update("main");
        client_loop();
    }
}

2. Add data migration code
   Globals migrated by default
   Initiate at start of main()
typedef int data;
data *mapping;  // automigrated
int l_fd;     // automigrated

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        kitsune_update("client");
        // ... process client requests
    }
}

int main() {
    kitsune_do_automigrate();

    mapping = malloc(...);
    l_fd = setup_conn();

    while (1) {
        kitsune_update("main");
        client_loop();
    }
}

2. Add data migration code
   Globals migrated by default
   Initiate at start of main()
Example single-threaded server

typedef int data;
data *mapping; // automigrated
int l_fd; // automigrated

void client_loop() {
  int cl_fd = get_conn(l_fd);
  while (1) {
    kitsune_update("client");
    // ... process client requests
  }
}

int main() {
  kitsune_do_automigrate();

  mapping = malloc(...);
  l_fd = setup_conn();

  /* after modification for Kitsune */

  while (1) {
    kitsune_update("main");
    client_loop();
  }

3. Add control migration code
   Avoid reinitialization
   Redirect control to update point
Example single-threaded server

```c
typedef int data;
data *mapping; // automigrated
int l_fd; // automigrated

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        kitsune_update("client");
        // ... process client requests
    }
}

int main() {
    kitsune_do_automigrate();
    if (!kitsune_is_updating()) {
        mapping = malloc(...);
        l_fd = setup_conn();
    }

    while (1) {
        kitsune_update("main");
        client_loop();
    }
}
```

3. Add control migration code
   Avoid reinitialization
   Redirect control to update point

after modification for Kitsune
Example single-threaded server

typedef int data;
data *mapping; // automigrated
int l_fd; // automigrated

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        kitsune_update("client");
        // ... process client requests
    }
}

int main() {
    kitsune_do_automigrate();
    if (!kitsune_is_updating()) {
        mapping = malloc(...);
        l_fd = setup_conn();
    }
    Kitsune after modification for Kitsune

3. Add control migration code
   Avoid reinitialization
   Redirect control to update point

    if (kitsune_is_updating_from ("client")) {
        client_loop();
    }
    while (1) {
        kitsune_update("main");
        client_loop();
    }
}
Example single-threaded server

typedef int data;
data *mapping; // automigrated
int l_fd; // automigrated

void client_loop() {
    int cl_fd = get_conn(l_fd);
    while (1) {
        kitsune_update("client");
        // ... process client requests
    }
}

int main() {
    kitsune_do_automigrate();
    if (!kitsune_is_updating()) {
        mapping = malloc(...);
        l_fd = setup_conn();
    }
}

We also support migration of locals
Generalizes to multi-threaded programs
Migrating and transforming state

• State may need to be transformed to work with the new program
  
  ▪ Transformation piggybacks on top of migration

old

```
typedef int data;
data *mapping;
```

new

```
typedef char *data;
data *mapping;
```
Migrating and transforming state

• State may need to be transformed to work with the new program

  ▪ Transformation piggybacks on top of migration

```
old
typedef int data;
data *mapping;
```

```
new
typedef char *data;
data *mapping;
```

```
Xform
For each value x of type data in the running program
and its corresponding location p in the new program

do
  *p = malloc(N);
  snprintf(*p,N,"%d",x);
end
```
Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

```
old typedef int data;
data *mapping;
new typedef char *data;
data *mapping;

new::mapsz = old::mapsz;
new::mapping = malloc(new::mapsz*sizeof(char*));
for (int i=0;i<new::mapsz;i++) {
    old::data x = old::mapping[i];
    new::data *p = &new::mapping[i];
    *p = malloc(N);
    snprintf(*p,N,”%d”,x);
}
```
Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

```
Xform tool
- Require programmer to write relevant xform code using high-level specs
- Automate generation of transformation code
  - requires some additional type annotations
```
Migrating and transforming state

- State may need to be transformed to work with the new program
  - Transformation piggybacks on top of migration

```c
typedef int data;
data *mapping;
```

```c
typedef char *data;
data *mapping;
```

**Xform**
```
typedef data → typedef data: {
  $out = malloc(N);
  snprintf($out, N, "%d", $in);
}
```
Using Kitsune and xfgenv

- Using Kitsune and xfgenv with the following command:

```bash
gcc -c -fPIC -fvis...=
```

- The compiled files are then linked using:

```bash
gcc -shared
```

- The resulting shared object is:

```bash
.so
```
Using Kitsune and xfgen

- Transformation specs in per-update .xf file
- Linked in with new version and invoked by kitsune_do_automigrate() and MIGRATE_LOCAL()
Kitsune benchmarks: changes required
## Kitsune benchmarks: changes required

<table>
<thead>
<tr>
<th>Program</th>
<th># Vers</th>
<th>Range</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsftpd</td>
<td>14</td>
<td>1.1.0–2.0.6</td>
<td>12,202</td>
</tr>
<tr>
<td>redis</td>
<td>5</td>
<td>2.0.0–2.0.4</td>
<td>13,387</td>
</tr>
<tr>
<td>Tor</td>
<td>13</td>
<td>0.2.1.18–0.2.1.30</td>
<td>76,090</td>
</tr>
<tr>
<td>memcached</td>
<td>3</td>
<td>1.2.2–1.2.4</td>
<td>4,181</td>
</tr>
<tr>
<td>icecast</td>
<td>5</td>
<td>2.2.0–2.3.1</td>
<td>15,759</td>
</tr>
<tr>
<td>snort</td>
<td>4</td>
<td>2.9.2–2.9.2.3</td>
<td>214,703</td>
</tr>
</tbody>
</table>

*Multi-threaded
### Kitsune benchmarks: changes required

<table>
<thead>
<tr>
<th>Program</th>
<th># Vers</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsftpd</td>
<td>14</td>
<td>12,202</td>
</tr>
<tr>
<td>redis</td>
<td>5</td>
<td>13,387</td>
</tr>
<tr>
<td>Tor</td>
<td>13</td>
<td>76,090</td>
</tr>
<tr>
<td>memcached*</td>
<td>3</td>
<td>4,181</td>
</tr>
<tr>
<td>icecast*</td>
<td>5</td>
<td>15,759</td>
</tr>
<tr>
<td>snort*</td>
<td>4</td>
<td>214,703</td>
</tr>
</tbody>
</table>

*Multi-threaded

<table>
<thead>
<tr>
<th>Program</th>
<th>Upd</th>
<th>Ctrl</th>
<th>Data</th>
<th>E_*</th>
<th>Oth</th>
<th>Σ</th>
<th>v→v</th>
<th>t→t</th>
<th>Σ</th>
<th>xf</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>vsftpd</td>
<td>6</td>
<td>26</td>
<td>17+8</td>
<td>6+14</td>
<td>28+8</td>
<td>83+30</td>
<td>9</td>
<td>21</td>
<td>30</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>redis</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>43</td>
<td>8</td>
<td>57</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Tor</td>
<td>1</td>
<td>39</td>
<td>37+6</td>
<td>19</td>
<td>57</td>
<td>153+6</td>
<td>16</td>
<td>15</td>
<td>31</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>memcached*</td>
<td>4</td>
<td>9</td>
<td>13</td>
<td>20</td>
<td>66</td>
<td>112</td>
<td>12</td>
<td>10</td>
<td>22</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>icecast*</td>
<td>11+1</td>
<td>22+3</td>
<td>14+9</td>
<td>32+3</td>
<td>39</td>
<td>118+16</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>snort*</td>
<td>2</td>
<td>90+18</td>
<td>110+2</td>
<td>158</td>
<td>66</td>
<td>426+20</td>
<td>111</td>
<td>64</td>
<td>175</td>
<td>197</td>
<td></td>
</tr>
</tbody>
</table>
Performance overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>Orig (siqr)</th>
<th>Kitsune</th>
<th>Ginseng</th>
<th>UpStare</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>64-bit, 4×2.4Ghz E7450 (6 core), 24GB mem, RHEL 5.7</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsftpd 2.0.6*</td>
<td>6.55s (0.04s)</td>
<td>+0.75%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>memchd 1.2.4</td>
<td>59.30s (3.25s)</td>
<td>+0.51%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>redis 2.0.4</td>
<td>46.83s (0.40s)</td>
<td>-0.31%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>icecast 2.3.1</td>
<td>10.11s (2.27s)</td>
<td>-2.18%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>32-bit, 1×3.6Ghz Pentium D (2 core), 2GB mem, Ubuntu 10.10</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsftpd 2.0.3*</td>
<td>5.96s (0.01s)</td>
<td>+2.35%</td>
<td>+11.3%</td>
<td>+41.6%</td>
</tr>
<tr>
<td>vsftpd 2.0.3†</td>
<td>14.03s (0.02s)</td>
<td>+0.29%</td>
<td>+1.47%</td>
<td>+6.64%</td>
</tr>
<tr>
<td>memchd 1.2.4</td>
<td>101.40s (0.35s)</td>
<td>-0.49%</td>
<td>+18.4%</td>
<td>–</td>
</tr>
<tr>
<td>redis 2.0.4</td>
<td>43.88s (0.16s)</td>
<td>-1.21%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>icecast 2.3.1</td>
<td>35.71s (0.68s)</td>
<td>+1.18%</td>
<td>-0.28%</td>
<td>–</td>
</tr>
</tbody>
</table>

*CD+LS benchmark, †file download benchmark

- 21 runs each, median, siqr reported
- Overall: -2.18% to 2.35% overhead (in the noise)
- (No performance measurements for snort yet)
### Update times

<table>
<thead>
<tr>
<th>Program</th>
<th>Med.</th>
<th>(siqr)</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>64-bit, 4×2.4Ghz E7450 (6 core), 24GB mem, RHEL 5.7</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsftpd → 2.0.6</td>
<td>2.99ms</td>
<td>(0.04ms)</td>
<td>2.62</td>
<td>3.09</td>
</tr>
<tr>
<td>memcached → 1.2.4</td>
<td>2.50ms</td>
<td>(0.05ms)</td>
<td>2.27</td>
<td>2.68</td>
</tr>
<tr>
<td>redis → 2.0.4</td>
<td>39.70ms</td>
<td>(0.98ms)</td>
<td>36.14</td>
<td>82.66</td>
</tr>
<tr>
<td>icecast → 2.3.1</td>
<td>990.89ms</td>
<td>(0.95ms)</td>
<td>451.73</td>
<td>992.71</td>
</tr>
<tr>
<td><em>icecast-nsp</em> → 2.3.1</td>
<td>187.89ms</td>
<td>(1.77ms)</td>
<td>87.14</td>
<td>191.32</td>
</tr>
<tr>
<td>tor → 0.2.1.30</td>
<td>11.81ms</td>
<td>(0.12ms)</td>
<td>11.65</td>
<td>13.83</td>
</tr>
<tr>
<td><strong>32-bit, 1×3.6Ghz Pentium D (2 core), 2GB mem, Ubuntu 10.10</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vsftpd → 2.0.3</td>
<td>2.62ms</td>
<td>(0.03ms)</td>
<td>2.52</td>
<td>2.71</td>
</tr>
<tr>
<td>memcached → 1.2.4</td>
<td>2.44ms</td>
<td>(0.08ms)</td>
<td>2.27</td>
<td>3.12</td>
</tr>
<tr>
<td>redis → 2.0.4</td>
<td>38.83ms</td>
<td>(0.64ms)</td>
<td>37.69</td>
<td>41.80</td>
</tr>
<tr>
<td>icecast → 2.3.1</td>
<td>885.39ms</td>
<td>(7.47ms)</td>
<td>859.00</td>
<td>908.87</td>
</tr>
<tr>
<td>tor → 0.2.1.30</td>
<td>10.43ms</td>
<td>(0.46ms)</td>
<td>10.08</td>
<td>12.98</td>
</tr>
</tbody>
</table>

- < 40ms in all cases but icecast
  - Icecast includes 1s sleeps; icecast-nsp removes these
Key idea #1: Update points
Key idea #1: Update points

- Competing approach: update anywhere
  - (when code to be changed not running)
  - Used by Ksplice, K42 (OS), OPUS
Key idea #1: Update points

• Competing approach: update anywhere
  ▪ (when code to be changed not running)
  ▪ Used by Ksplice, K42 (OS), OPUS

• Benefits of update points
  ▪ Simplifies reasoning for programmers
    - Particularly for multithreaded programs
  ▪ May accelerate update times
    - As opposed to waiting for updated code to become inactive
  ▪ Simplifies updating mechanism
Key idea #2: Whole program updates
Key idea #2: Whole program updates

• Competing approach
  - Program keeps running the current code, and subsequent function calls to new versions
  - Used by Ginseng, POLUS, OPUS, Ksplice, K42
Key idea #2: Whole program updates

• Competing approach
  ▪ Program keeps running the current code, and subsequent function calls to new versions
  ▪ Used by Ginseng, POLUS, OPUS, Ksplice, K42

• Benefits of whole-program updates:
  ▪ Can update active code (e.g., long-running loops) in an arbitrary manner
    - very important in practice
  ▪ Explicit control migration simplifies reasoning, maintenance
  ▪ More efficient implementation
    - No need to insert levels of indirection, use trampolines, etc.
    - No need to compile datastructures differently
Ongoing work
Ongoing work

• Means to specify and verify the correctness of dynamic software updates [VSTTE’12]
  ▪ Reuse specifications for each version individually
  ▪ Explicate acceptable backward-incompatible behaviors
Ongoing work

• Means to specify and verify the correctness of dynamic software updates [VSTTE’12]
  ▪ Reuse specifications for each version individually
  ▪ Explicate acceptable backward-incompatible behaviors

• Means to automatically generate state transformations from dynamic analysis [OOPSLA’12]
  ▪ E.g., automatically correct leaks in running heap
Ongoing work

- Means to specify and verify the correctness of dynamic software updates [VSTTE'12]
  - Reuse specifications for each version individually
  - Explicate acceptable backward-incompatible behaviors

- Means to automatically generate state transformations from dynamic analysis [OOPSLA'12]
  - E.g., automatically correct leaks in running heap

- Adapt Kitsune methodology to Java
  - Contrast to our earlier VM-based approach [PLDI'09]
Ongoing work

• Means to specify and verify the correctness of dynamic software updates [VSTTE’12]
  ▪ Reuse specifications for each version individually
  ▪ Explicate acceptable backward-incompatible behaviors

• Means to automatically generate state transformations from dynamic analysis [OOPSLA’12]
  ▪ E.g., automatically correct leaks in running heap

• Adapt Kitsune methodology to Java
  ▪ Contrast to our earlier VM-based approach [PLDI’09]

• Implement lazy state transformation for Kitsune
DSU project team

- Former students / post-docs
  - Manuel Oriol, post-doc 2005-06, @University of York (UK) and ABB
  - Gareth Stoyle, Ph.D. (Cambridge) 2007, @UBS (UK)
  - Iulian Neamtiu, Ph.D. 2008, @UC Riverside
  - Suriya Subramanian, Ph.D. (UT Austin) 2011, @Intel
  - Stephen Magill, post-doc 2010-11, @IDA/CCS (Gov. lab)
  - Chris Hayden, Ph.D. 2012, @Washington Post Labs

- Current students
  - Karla Saur (3rd year), Ted Smith (undergrad), Luis Pina (3rd year, visiting)

- Profs/researchers
  - Kathryn McKinley, Prof @UT, MSR; Jeff Foster, Prof @Maryland;
  - Nate Foster, Prof @Cornell; Peter Sewell, Prof @Cambridge; Gavin Bierman, @MSR Cambridge
Roadmap

- Dynamic software updating (DSU)
  - Kitsune: Flexible and Efficient DSU for C programs

- Program analysis for security and reliability
  - Knowledge-based security: quantitatively tracking information

- Quick tour of some other work
Program analysis to improve quality

• Software is ubiquitous, and critically important
  ▪ Yet it is often unreliable and insecure

• So: build tools to analyze software automatically
  ▪ **Static analysis** applied before running the program
    - Examples: Type checkers/inferencers, tools like FindBugs
    - Pros: Complete coverage (considers all runs), no run-time overhead
    - Cons: problems are undecidable, so often false alarms
  ▪ **Dynamic analysis** observes actual executions
    - Pros: Very precise, no false alarms
    - Cons: Less coverage, instrumentation adds run-time overhead, discovered problems hard to remediate in deployment
Hybrid analysis: best of both worlds

• **Dynamic analysis, optimized by static analysis**
  - Eliminate redundant checks; no false alarms
  - *Ex: concurrency error checking [POPL’10], atomicity enforcement [TX’06]*

• **Dynamic analysis, proved correct statically**
  - Prove that necessary checks take place for all possible executions
  - *Ex: Fable/SELinks for security checking [Oakland’08, SIGMOD’09]*

• **Static analysis, made more precise by dynamic analysis**
  - Added contextual information reduces false alarms
  - *Ex: Synthesis of DSU state transformers [OOPSLA’12], Knowledge-based security [CSF’11, PLAS’12], Rubydust [POPL’11, STOP’11]*
Hybrid analysis: best of both worlds

• Dynamic analysis, optimized by static analysis
  ▪ Eliminate redundant checks; no false alarms
  ▪ *Ex: concurrency error checking* [POPL’10], *atomicity enforcement* [TX’06]

• Dynamic analysis, proved correct statically
  ▪ Prove that necessary checks take place for all possible executions
  ▪ *Ex: Fable/SELinks for security checking* [Oakland’08, SIGMOD’09]

• Static analysis, made more precise by dynamic analysis
  ▪ Added contextual information reduces false alarms
  ▪ *Ex: Synthesis of DSU state transformers* [OOPSLA’12], *Knowledge-based security* [CSF’11, PLAS’12], *Rubydust* [POPL’11, STOP’11]
No privacy: They have your data

This is the status quo
Alternative: Maintain your own data

querier

query

response

you
Alternative: Maintain your own data

The question then becomes: Which queries should you answer and which should you refuse?
Query 1: Useful and non-revealing

out = 24 ≤ Age ≤ 30
& Female?
& Engaged? *

* real query used by a Facebook advertiser

querier

true

you
Query 2: Reveals too much!

out = (gender, zip-code, birth-date) *

* - gender, zip-code, birth-date can be used to uniquely identify 87% of Americans
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.
When to accept, when to reject

- Maintain a representation of the querier’s **belief** about secret’s possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values.
- Each query result revises the belief; reject if actual secret becomes too likely.
  - Cannot let rejection defeat our protection.

Belief $\triangleq$ probability distribution

...
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.
When to accept, when to reject

- Maintain a representation of the querier’s **belief** about secret’s possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

Bayesian reasoning to revise belief
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

OK (answer)
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.
When to accept, when to reject

- Maintain a representation of the querier’s **belief** about secret’s possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - **Cannot let rejection defeat our protection.**
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

... Q1 Q2...

OK (answer)  Reject

time
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

... Q1 ... Q2 ...

OK (answer)   Reject

time
When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.

When to accept, when to reject

- Maintain a representation of the querier’s belief about secret’s possible values
- Each query result revises the belief; reject if actual secret becomes too likely
  - Cannot let rejection defeat our protection.
When to accept, when to reject

- Maintain a representation of the querier’s **belief** about secret’s possible values
- Each query result **revises** the belief; reject if actual secret becomes too likely
  - **Cannot let rejection defeat our protection.**

```
Q1
⇒ Reject
Q2
⇒ OK (answer)
Q3
⇒ OK (answer)
```

*time*
Meet Bob

Bob (born September 24, 1980)
bday = 267
byear = 1980

0 ≤ bday ≤ 364
1956 ≤ byear ≤ 1992
each equally likely

Assumption: this is accurate
bday-query1

today := 260;
if bday ≤ today && bday < (today + 7)
then out := 1
else out := 0

= (out = 0)
Problem
Policy: Is this acceptable?

```plaintext
bday-query1
today := 260;
if bday ≤ today && bday < (today + 7)
    then out := 1
else out := 0
```

= (out = 0)

1956
1992
0 259 267 364
Idea: policy as knowledge threshold

- Answer a query if, for querier’s revised belief, \( \Pr[\text{my secret}] < t \)
  - Call \( t \) the knowledge threshold

- Choice of \( t \) depends on the risk of revelation
Bob’s policies

Bob (born September 24, 1980)

\[ bday = 267 \]
\[ byear = 1980 \]

\[
0 \leq bday \leq 364 \\
1956 \leq byear \leq 1992 \\
\text{each equally likely}
\]

\[
\Pr[bday = 267] \ldots
\]

\[
\text{Policy}
\]
\[
\Pr[bday] < 0.2 \\
\Pr[bday,byear] < 0.05
\]

\[
\text{Currently}
\]
\[
\Pr[bday] = 1/365 \\
\Pr[bday,byear] = 1/(365*37)
\]
Back to the query ...

day-query1
today := 260;
if bday ≤ today && bday < (today + 7)
then out := 1
else out := 0
Potentially
Pr[bday] = 1/358 < 0.2
Pr[bday, byear] = 1/(358*37) < 0.05

```plaintext
bday-query1
today := 260;
if bday ≤ today && bday < (today + 7)
    then out := 1
else out := 0
```

= ☠ (out = 0)
Next day ...

```plaintext
bday-query2
today := 261;
if bday ≤ today && bday < (today + 7)
then out := 1
else out := 0
```

So reject?
Querier’s perspective

Assume querier knows policy

if bday $\neq$ 267

1992

1956

0 259 268 364

will get answer

if bday = 267

1992

1956

0 267

will get reject
Rejection problem

- Policy: $\Pr[bday = 267 \mid out = o] < t$
- Rejection, intended to protect secret, reveals secret!
Rejection revised

• Policy: $\Pr[bday = 267 \mid out = o] < t$

• Solution?
  • Decide policy independently of secret
  • Revised policy

  • for every possible output $o$,
    • for every possible bday $b$,
      • $\Pr[bday = b \mid out = o] < t$

  • So the real $bday$ in particular
bday-query1

```plaintext
today := 260;
if bday ≤ today && bday < (today + 7)
    then out := 1
    else out := 0
```

accept

initial belief
bday-query2

today := 261;
if bday ≤ today && bday < (today + 7)
    then out := 1
    else out := 0

reject

(regardless of what bday actually is)
bday-query3
today := 266;
if bday ≤ today && bday < (today + 7)
then out := 1
else out := 0

accept

This is acceptable since it is five days after the last accept, keeping the probability within \( t = 0.2 \); i.e., \( \Pr[266 \leq \text{bday} \leq 270] = 1/5 \) if \( \text{out} = 1 \), \( \Pr[\text{bday}] = 1/353 \) otherwise.
Implementation

• Our query analysis in the style of abstract interpretation
  ▪ We developed a novel probabilistic polyhedral domain
  ▪ Scales far better than monte carlo sampling

• Precisely analyzes a particular sequence of queries, rather than all possible sequences
  ▪ Far less conservative than considering all possible sequences of queries
Illustration of improved scalability

$0 \leq \text{bday} \leq 364$

$1956 \leq \text{byear} \leq 1992$

each equally likely

bday1 small

$0 \leq \text{bday} \leq 364$

$1910 \leq \text{byear} \leq 2010$

each equally likely

bday 1 large
Illustration of improved scalability

Our approach
best precision
time indep. of state size

0 ≤ bday ≤ 364
1910 ≤ byear ≤ 2010
each equally likely
bday 1 large

0 ≤ bday ≤ 364
1956 ≤ byear ≤ 1992
each equally likely
bday1 small
Illustration of improved scalability

Sampling
same precision
much slower

= 0 ≤ bday ≤ 364
1956 ≤ byear ≤ 1992
each equally likely

bday1 small

= 0 ≤ bday ≤ 364
1910 ≤ byear ≤ 2010
each equally likely

bday 1 large
Related work

• Significant work on database-oriented privacy, e.g., differential privacy. Key differences:
  ▪ Trusts third party data provider to run safe aggregate queries. We work with individual data directly
  ▪ DP’s powerful adversary severely compromises utility, particularly for queries specific to individuals
  ▪ Does not perform on-the-fly query analysis

• Also work on quantifying information flow
  ▪ Tracks “bits leaked” but not relevant policies
  ▪ Considers all possible query streams; too conservative
Current activities

• Application to secure *multiparty computation* [PLAS’12]
  - Two parties $p_1, p_2$ have secrets $s_1, s_2$ and compute $f(s_1, s_2) = x$, revealing only $x$ to each
  - How much does $x$ reveal about $s_1$ and $s_2$?

• Time-indexed data: protect predictive features
  - Cooperative computations over coalition sensor networks
  - Ensuring anonymity of location traces [CCS’12]

• General direction: Privacy as a right
Collaborators (on analyses/tools)

- Former students / post-docs
  - Nikhil Swamy, Ph.D. 2008, @MSR Redmond
  - Polyvios Pratikakis, Ph.D. 2008, @FORTH Labs (Crete, Greece)
  - Avik Chaudhuri, post-doc 2009-10, @Adobe Research
  - Saurabh Srivastava, Ph.D. 2010, @Berkeley (CIFellow post-doc)
  - Martin Ma, Ph.D. 2011, @Amazon
  - Nataliya Guts, post-doc 2011-12, @Google

- Current students/post-docs
  - Khoo Yit Phang (7th year), Piotr Mardziel (4th year), Aseem Rastogi (4th year), Matt Hammer (post-doc)

- Profs/researchers
  - Jeff Foster (Maryland); Jonathan Katz (Maryland); Mudhakar Srivatsa (IBM T.J. Watson); Miguel Castro et al. (MSR Cambridge); Daan Leijen (MSR Redmond)
Other research

• Systems/networking research

  ▪ Pavlos Papageorgiou (Ph.D, 2008), Passive-Aggressive Measurement with MGRP [SIGCOMM’09]

  ▪ Justin McCann (Ph.D., 2012), Automating Performance Diagnosis in Networked Systems

• SCORE: Agile method for academic research [CACM’10]
Maryland Cybersecurity Center (MC2)

- MC2 Director (since Oct 2011)
  - Two new CMSC faculty (Shi and Feamster)
  - Fifteen corporate partners (SAIC, NGC, Sourcefire, ...)
  - First MC2 Symposium, May 2011
  - Google Cybersecurity Seminars
  - ACES honors program, Prof. Masters, new courses

- Several new research initiatives underway
  - Privacy as a right
  - Anti-censorship
Summary: Building better software

• Along with colleagues and students, I am working to understand how to construct software that is available, reliable, and secure; i.e., software that
  ▪ never crashes
  ▪ adapts to changing circumstances and requirements
  ▪ properly protects data
  ▪ nevertheless provides useful and efficient services

• Programming languages, tools, and analyses, utilizing theory and implementation, are a powerful mechanism to this end