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
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

- 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

\[
\text{SELECT employee_id, CONCAT(firstname, lastname) FROM employees WHERE MAX(salary) > 1,000,000}
\]
Protocol Parsers: Simple Object Access Protocol

Simple Object Access Protocol (SOAP) enables remote procedure calls.

It requires a web services description language (WSDL) file to parse messages, which defines the API for SOAP messages.
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

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

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
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

ImageTragick: 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 reverse shell on system

ImageMagick runs identify on file, executing code to create reverse shell

```
X.X.X.X
WasGeneratedBy

HTTP Request

httpd worker 4435
WasGeneratedBy

uploads/rsh.jpg
WasGeneratedBy

libMagickCore.so.2.0.0
WasGeneratedBy

sh -c identify uploads/rsh.jpg
WasGeneratedBy

identify uploads/rsh.jpg
WasGeneratedBy

sh -c curl -s -k -o /tmp/magic
WasGeneratedBy

bash -c /dev/tcp/X.X.X.X/9999 0>&1
WasGeneratedBy

vi htdocs/reverse-shell.php
WasGeneratedBy

reverse-shell.php
WasGeneratedBy
```
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?