OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis

Wajih Ul Hassan, Mohammad A. Noureddine, Pubali Datta, Adam Bates

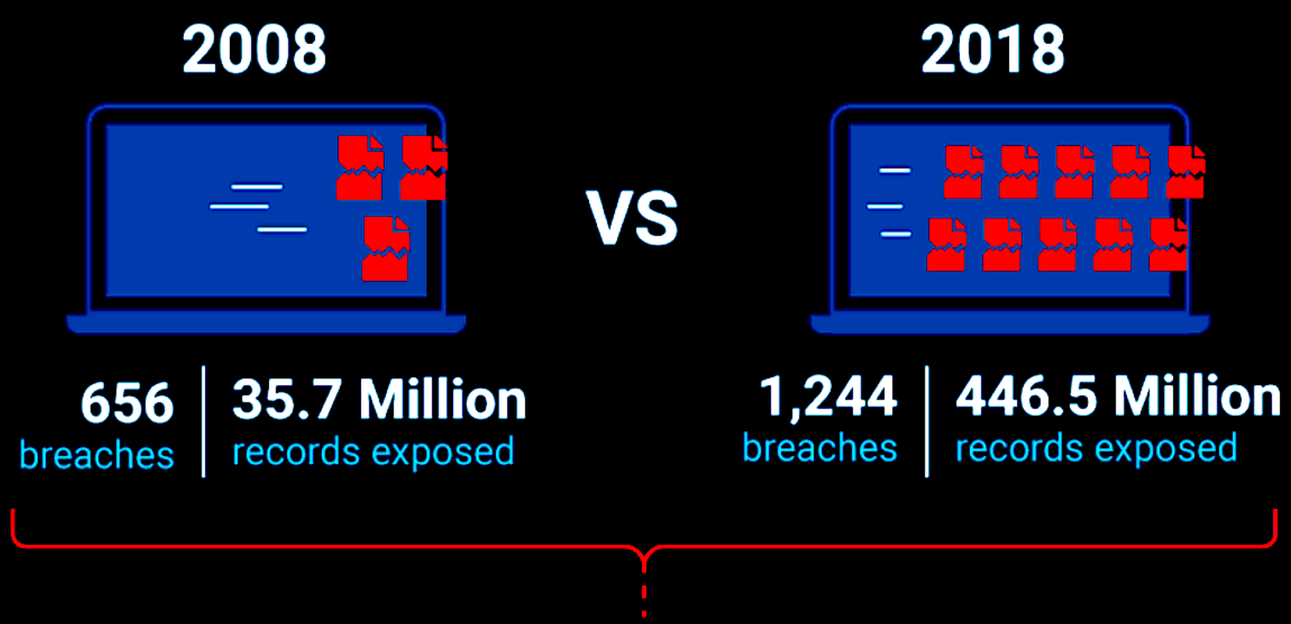
Network and Distributed System Security Symposium (NDSS) 2020

26 February 2020

ILLINOIS

State of Data Breaches

Data breaches in the United States:



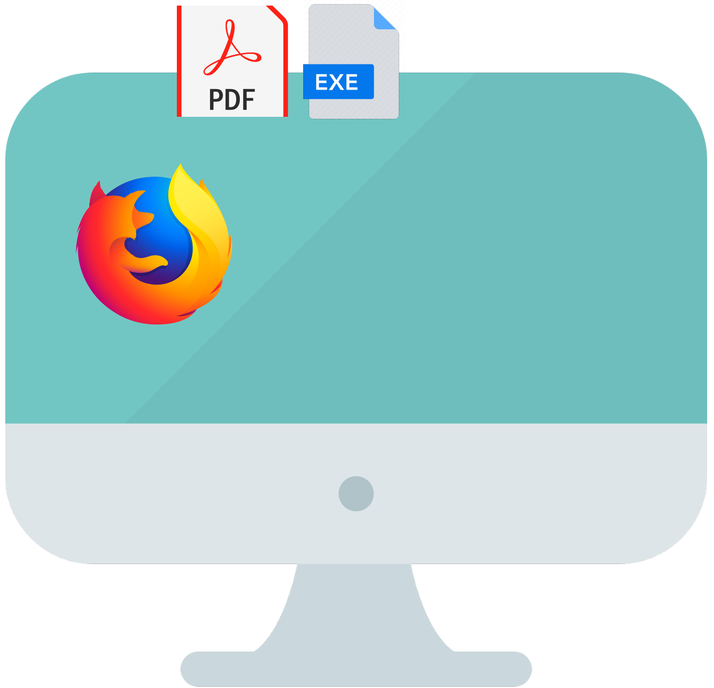
According to a survey by RSA 73% of cyber analysts have inadequate levels of capability to detect/respond to attack[2]

[1] Infographic from: https://link.medium.com/5Omijdiyg4

[2] Survey and image from: https://www.rsa.com/content/dam/en/infographic/rsa-poverty-index-2016-update.pdf

Threat Investigation

- Audit logs
  - Maintain a history of events that occur during system execution
  - System-Level Logs (e.g., Linux Audit) record events at the system call granularity



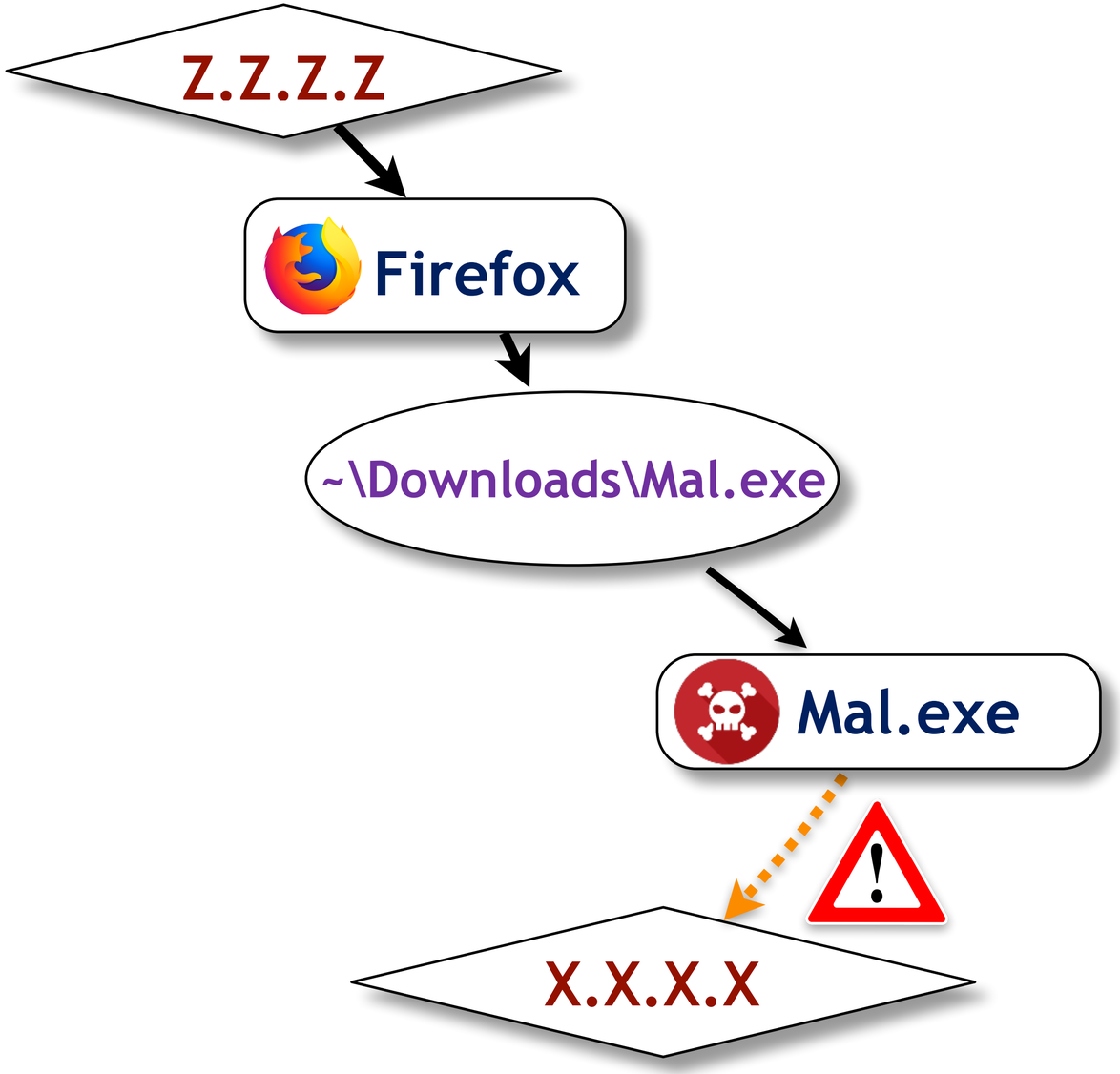
System-level Log

```
Process 1234 created from firefox.exe
……
Process 1234 reads from IP y.y.y.y
Process 1234 writes file ~\Downloads\A.pdf
……
Process 1234 reads from IP z.z.z.z
Process 1234 writes file ~\Downloads\Mal.exe
……
```

Data Provenance

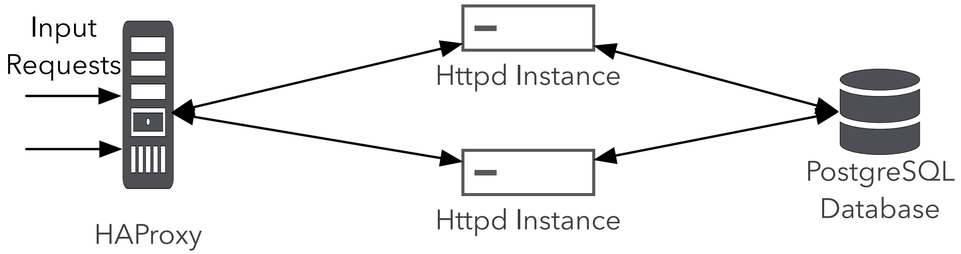
- To simplify investigation, we can parse system logs into *data provenance* graphs
  - Vertex**:** File, Socket, Process, etc.
  - Edge**:** Causal event (i.e., syscall)

- Find root cause of the attack symptom
  - *Backward Tracing*
- Find the ramification of the attack
  - *Forward Tracing*



Case Study: SQL Injection Attack

- A simple WordPress website hosted on a web server



- In addition to system logs, the different components (load balancer, server, database) also log *application events*.

Attacker performed SQL injection to steal credentials and used Wordpress file plugin to change website content.

Investigation using Application Logs

- Investigator knows that “accounts” table was accessed by attack
- Grep **PostgreSQL** query logs to find out which query read the “accounts” table content.
- It returned the following query from the PostgreSQL logs:

PostgreSQL

```
…
SELECT * FROM users WHERE user_id=123
UNION SELECT password FROM accounts;
…
```

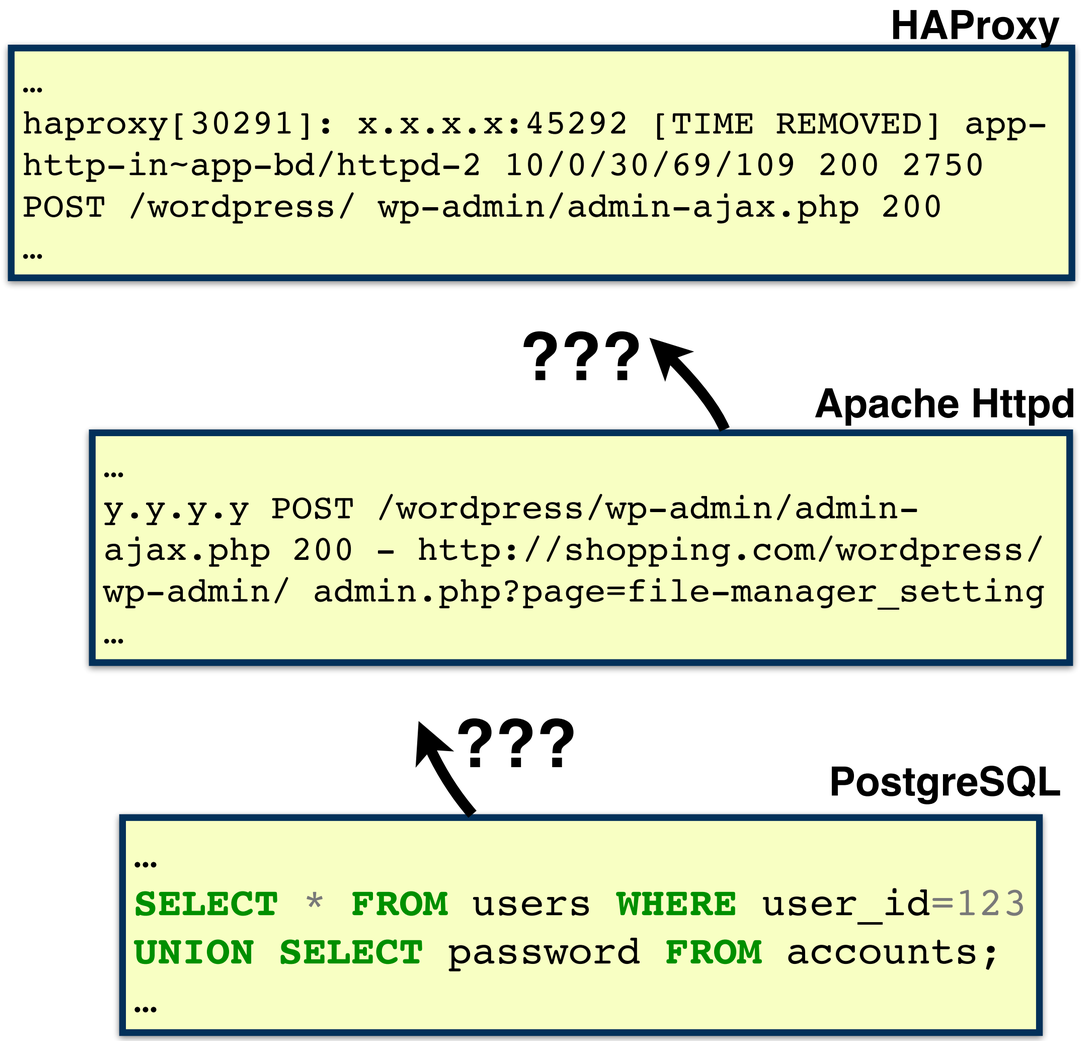
- Query indicates SQL injection attack

Investigation using Application Logs

- However, admin is unable to proceed further in the investigation using application event logs alone.
- **HAProxy** and **Apache logs** contain important evidence related to SQL injection attack
  - Cannot associate with PostgreSQL log
  - Do not capture workflow dependencies between applications
  - Grep will not work on these logs because SQL query was not in URL

Investigation using Application Logs

- However, admin is unable to proceed further in the investigation using application event logs alone.
- **HAProxy** and **Apache logs** contain important evidence related to SQL injection attack
  - Cannot associate with PostgreSQL log
  - Do not capture workflow dependencies between applications
  - Grep will not work on these logs because SQL query was not in URL



Investigation using System Logs

- To proceed investigation, now admin uses a system-level provenance graph
  - It allows admin to trace dependencies across applications.
- Malicious query read database file: `/usr/local/db/datafile.db`
- Admin issues backward tracing query from that file
  - Return provenance graph

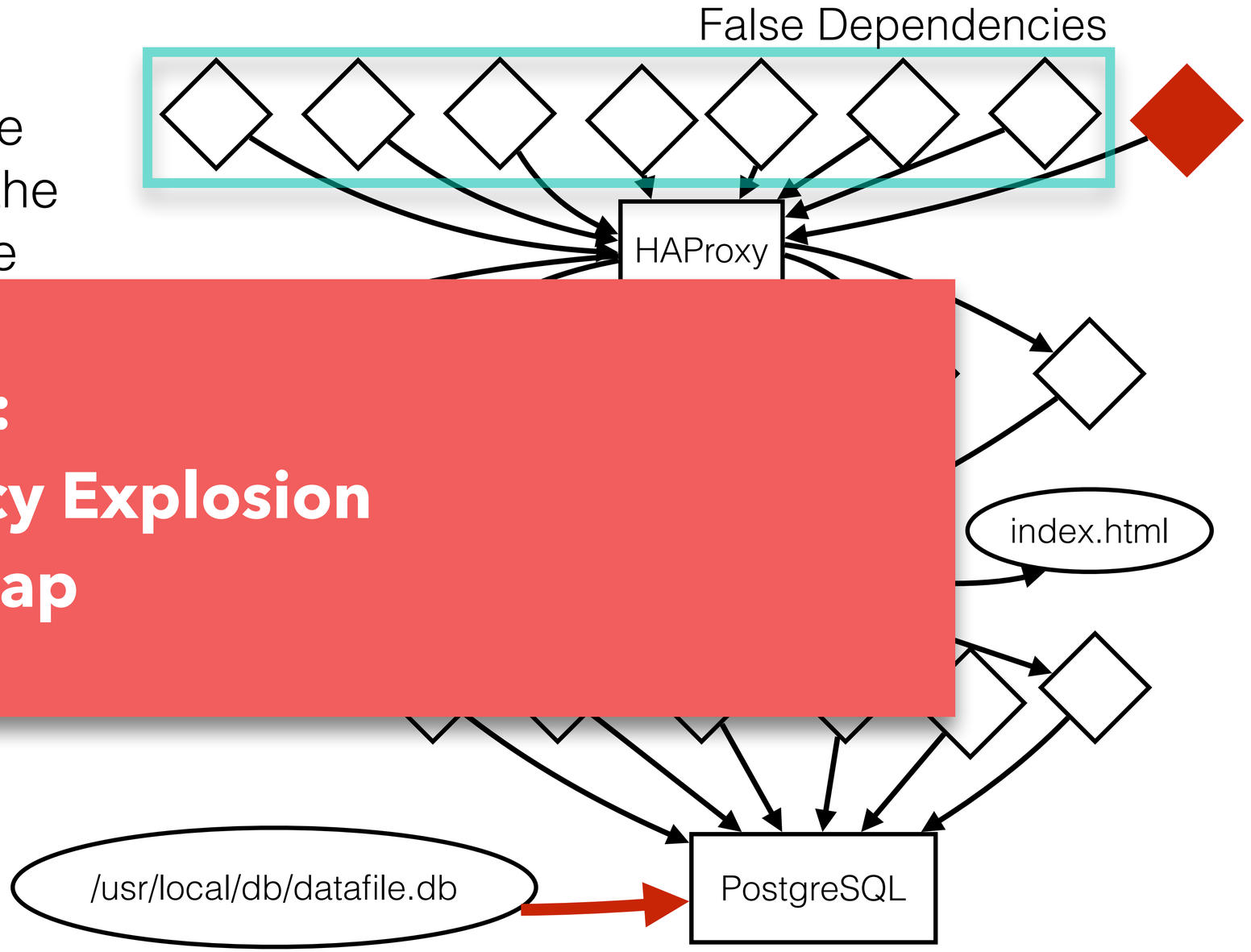
Investigation using System Logs

- **Dependency Explosion:** One output event depends on all the preceding input events on the same
- There cause injecti
- **Sema** seman present in application logs

Two Challenges:

1) Dependency Explosion

2) Semantic Gap



OmegaLog

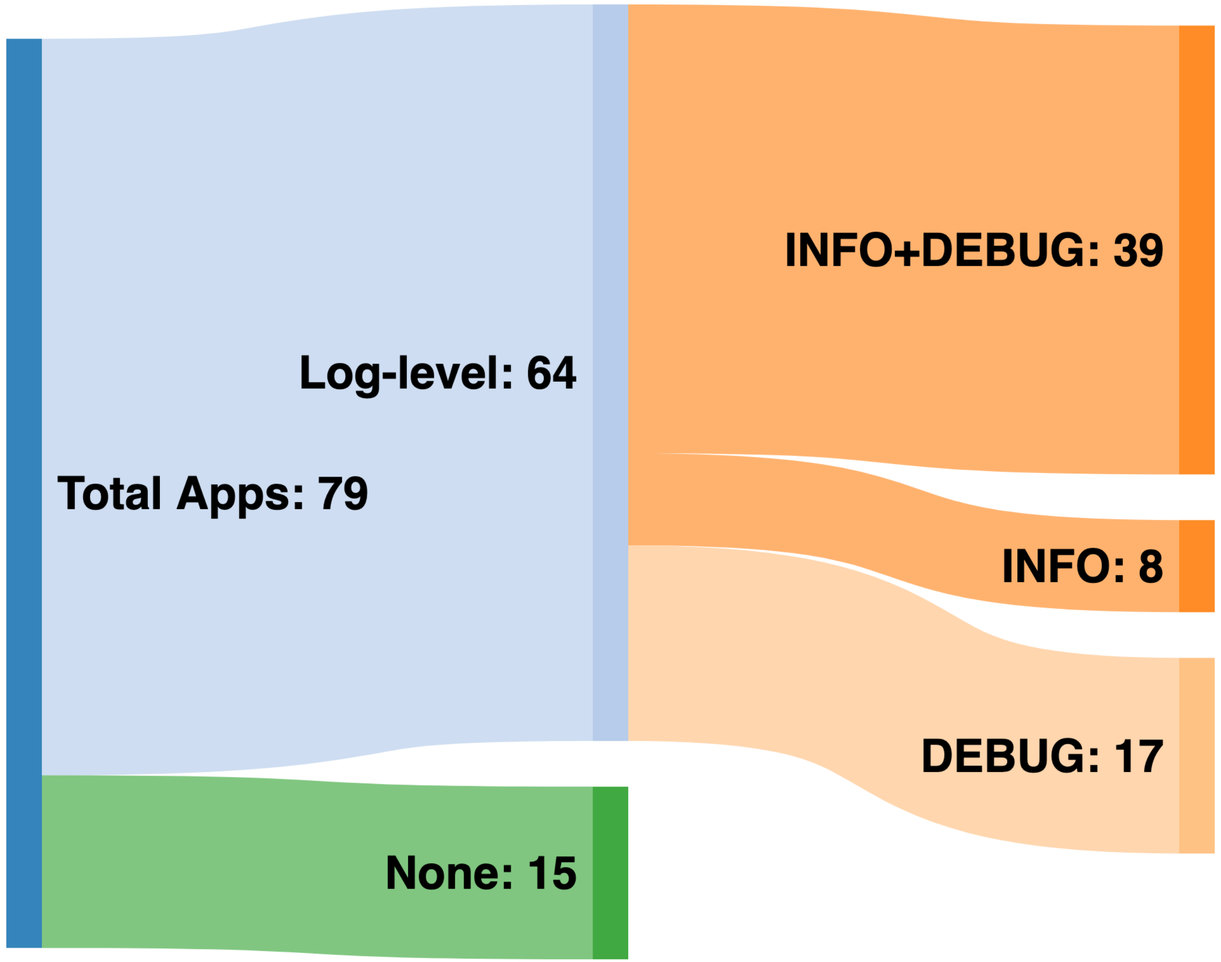
A provenance tracker that transparently solves both the dependency explosion and semantic gap problems

OmegaLog

- Solves dependency explosion problem by identifying event-handling loop through the application log sequences
  - Each iteration of event-handling loop is considered one semantically independent execution unit (BEEP NDSS'13)…
  - *But unlike BEEP, no instrumentation or training is required!*

- Tackles semantic gap problem by grafting application event logs onto the system-level provenance graphs

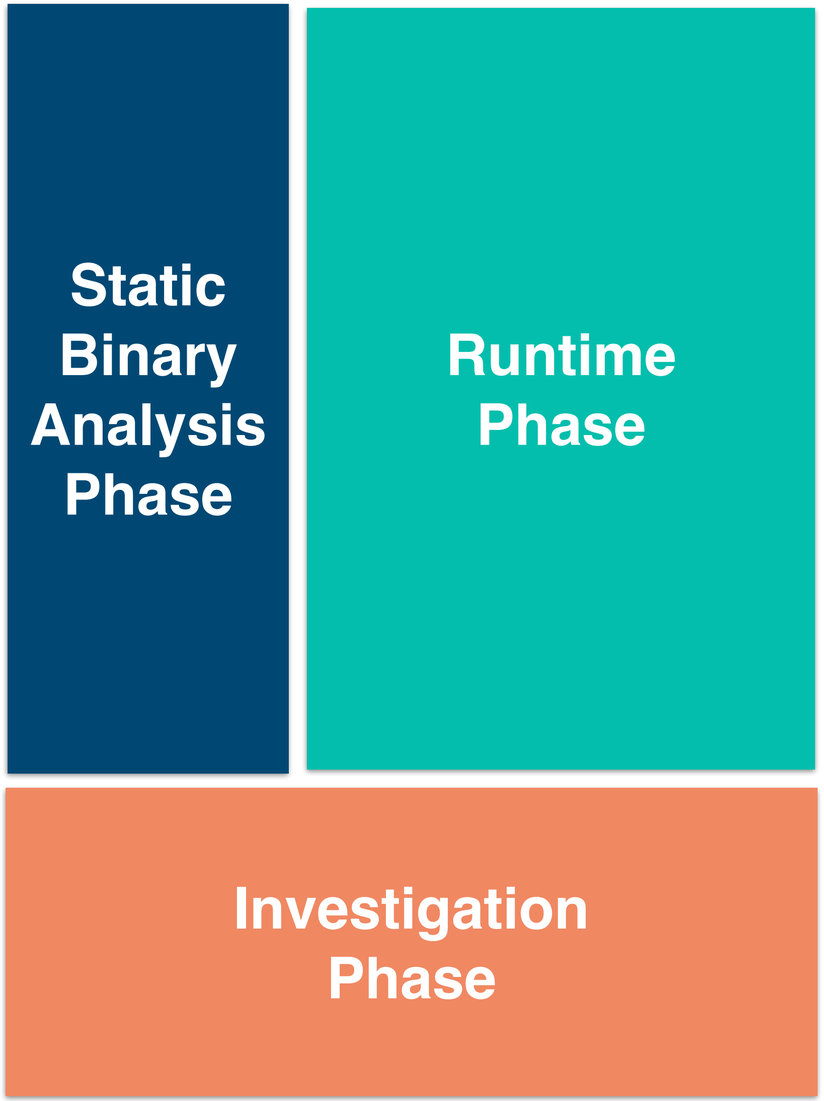
Do appli
insi
event-har



- 15 applications with no logging:
  - Light-weight apps
  - GUI apps

OmegaLog Workflow

Consist of 3 Phases:



Static Binary Analysis Phase

1. Identify log message printing functions
   - Separate normal file writes from log file writes

     e.g., `logMsg(…); ap_log_error(…);`
   - Used heuristics to find them
     - Well-known logging libraries (log4c) functions
     - Functions writing to `/var/log/`

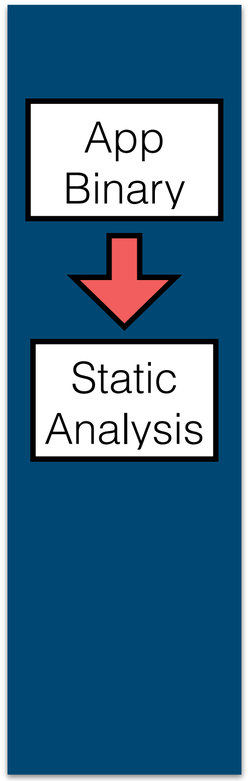
Static Binary Analysis Phase

2. Find call sites to those functions and concretize log message string (LMS) passed as argument

   - Use symbolic execution

     `“Opened file “%s””`

     `“Accepted connection with id %d”`



Static Binary Analysis Phase

2. Find call sites to those functions and concretize log message string (LMS) passed as argument

   - Use symbolic execution

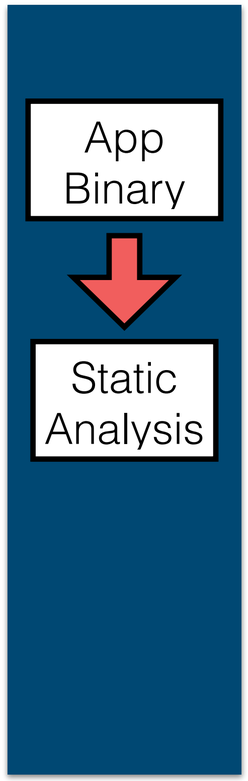
     `“Opened file “%s””`

     `“Accepted connection with id %d”`

3. Build regex from concretized log message strings for runtime matching

   `“Opened file “.*””`

   `“Accepted connection with id [0-9]+”`



Static Binary Analysis Phase

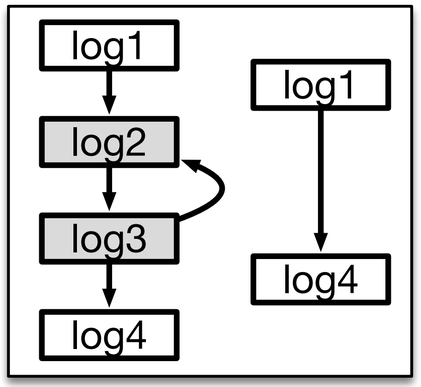
4. Perform control flow analysis

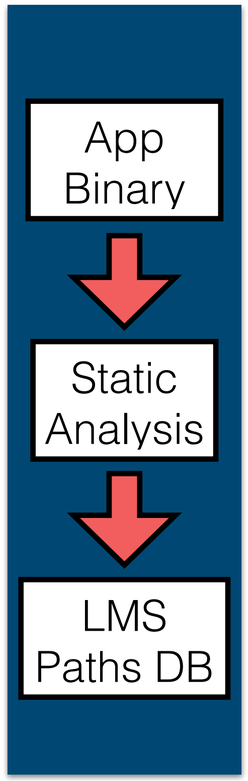
   - Generate a set of all valid log message control flow paths that can occur during execution

Code Snippet

```
log("Server started"); // log1
while(...) {
 log("Accepted Connection"); // log2
 ... /*Handle request here*/
 log("Closed Connection"); // log3
}
log("Server stopped"); // log4
```

Control flow paths

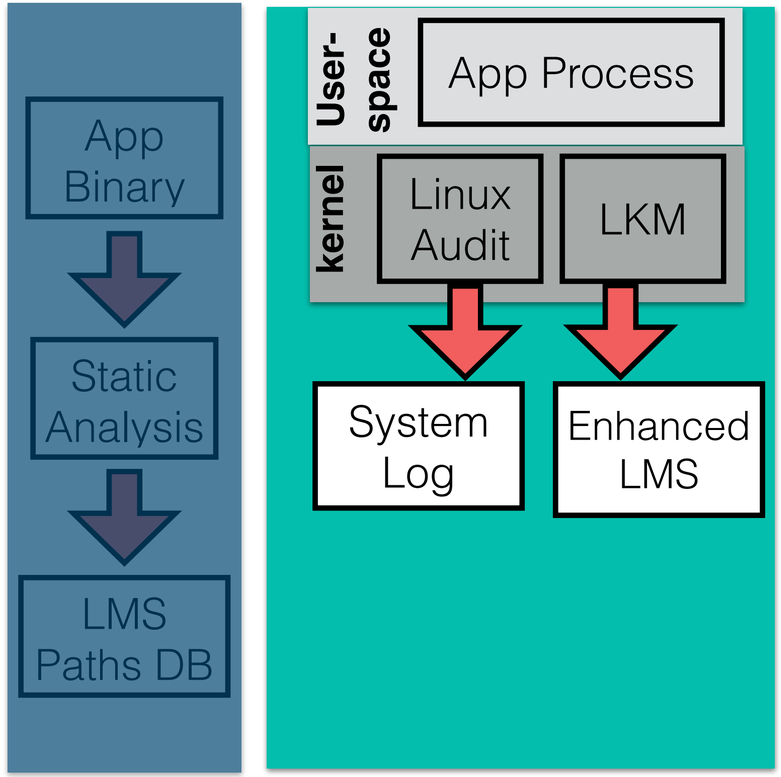




Log message control flow paths will guide OmegaLog to identify event-handling loop and partition execution of application into execution units

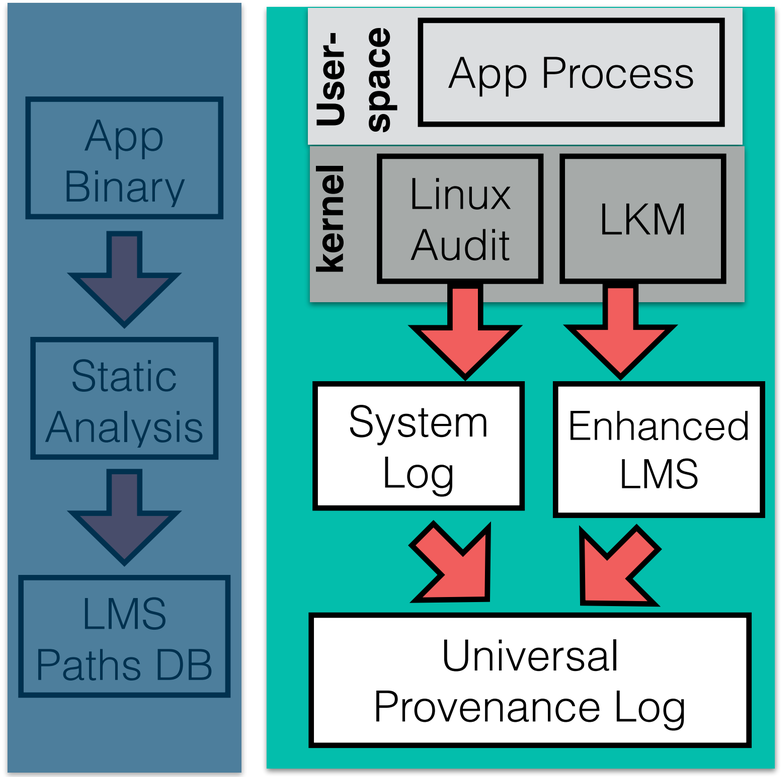
Runtime Phase

- We collect whole-system logs using Linux Audit Module
- A custom Linux Kernel Module (LKM)
  - Intercepts write system calls
  - Catch application log messages
  - Add PID/TID to log message
  - Allow us to combine log message with corresponding system-level log entry.



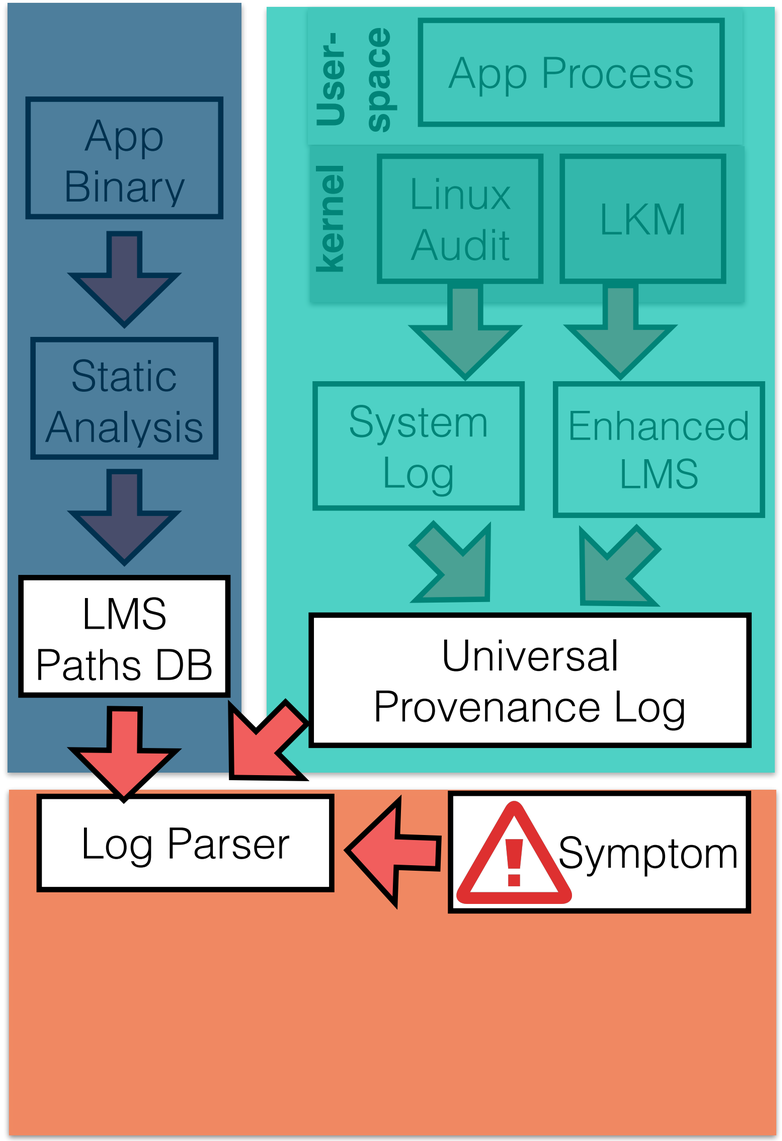
Runtime Phase

- We collect whole-system logs using Linux Audit Module
- A custom Linux Kernel Module (LKM)
  - Intercepts write system calls
  - Catch application log messages
  - Add PID/TID to log message
  - Allow us to combine log message with corresponding system-level log entry.
- Unify system logs and runtime log messages into universal provenance log



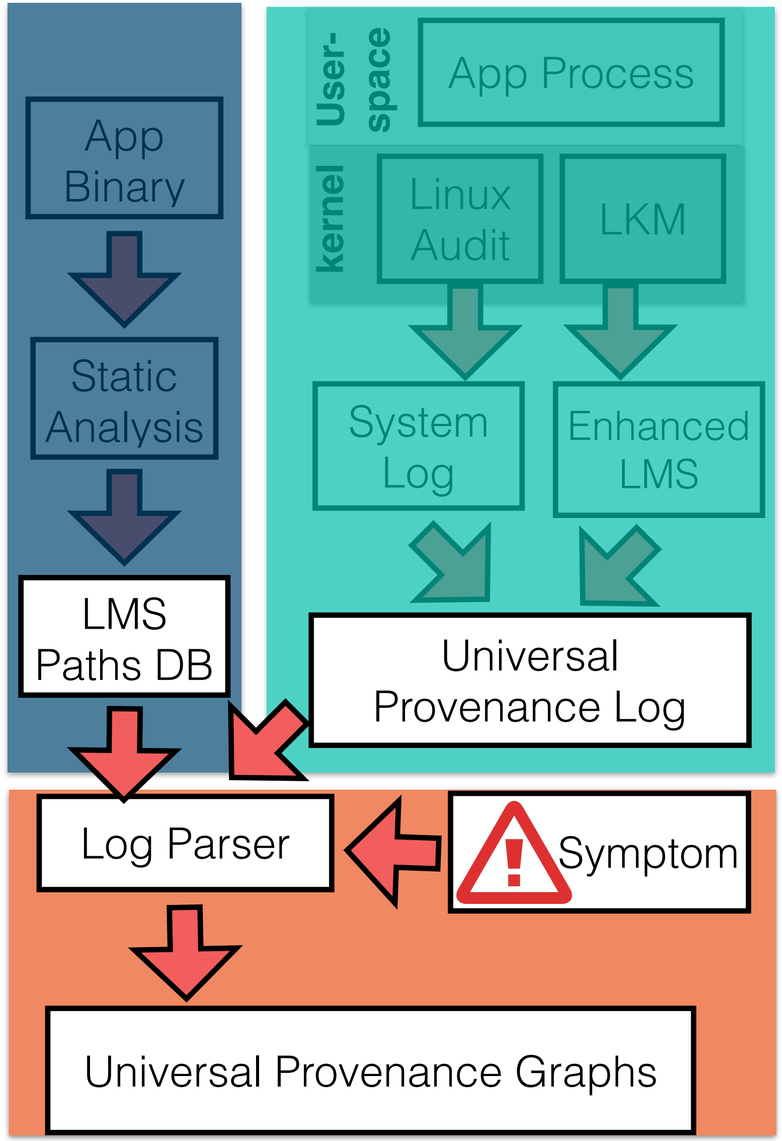
Investigation Phase

- Given a symptom of an attack, OmegaLog uses
  - Log message control flow paths database
  - Universal provenance log
- Log parser partitions the system log into units
  - By matching application log messages in universal provenance log with log message string control flow paths
  - Generates execution partition graph



Investigation Phase

- Given a symptom of an attack, OmegaLog uses
  - Log message control flow paths database
  - Universal provenance log
- Log parser partitions the system log into units
  - By matching application log messages in universal provenance log with log message string control flow paths
  - Generates execution partition graph
- Then add application log messages vertices to execution-partitioned provenance graph
- Final output: universal provenance graph



Back to our case study

Application Logs

HAProxy

```
…
haproxy[30291]: x.x.x.x:45292 [TIME REMOVED] app-
http-in~app-bd/httpd-2 10/0/30/69/109 200 2750
POST /wordpress/ wp-admin/admin-ajax.php 200
…
```

???

Apache Httpd

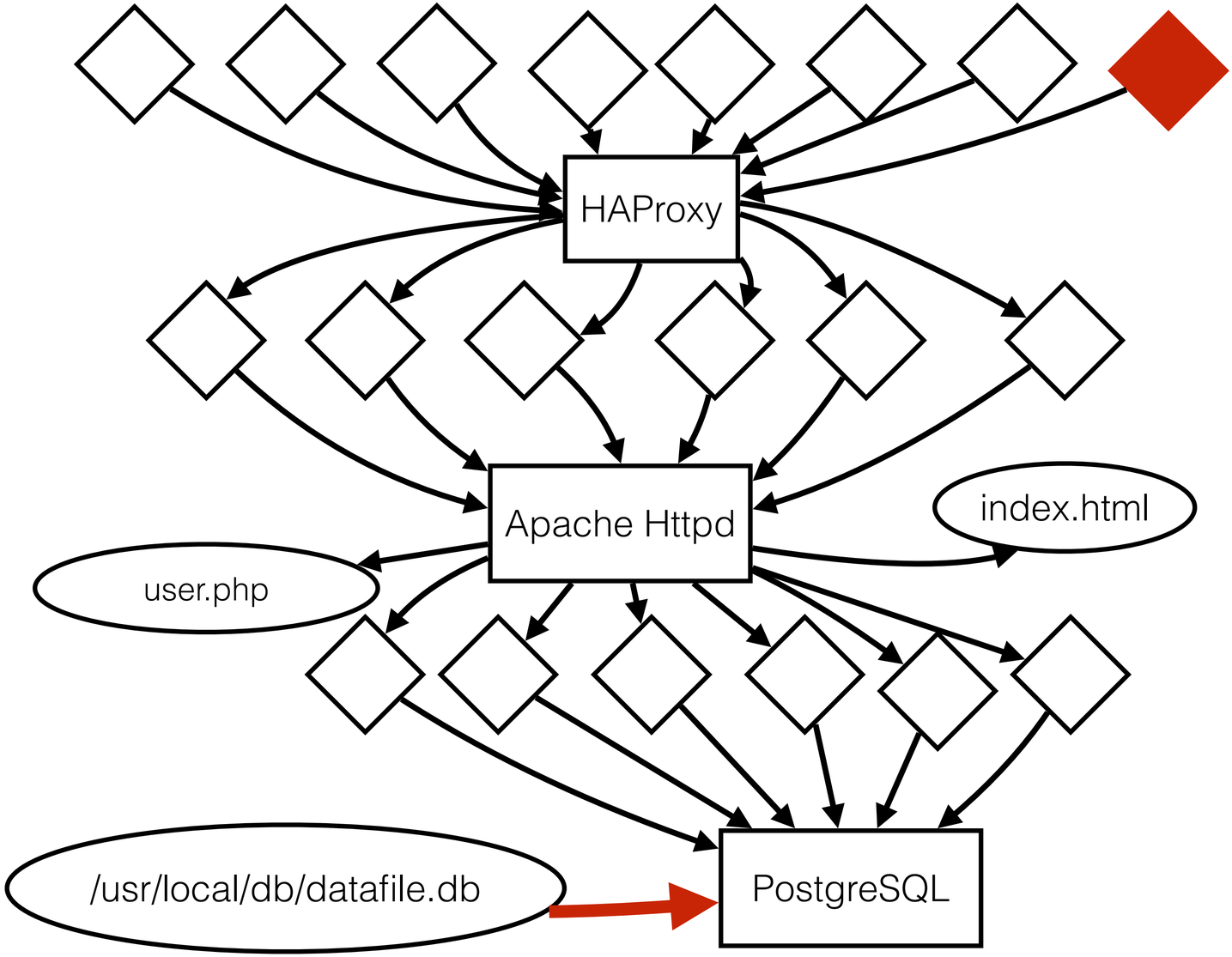
```
…
y.y.y.y POST /wordpress/wp-admin/admin-
ajax.php 200 - http://shopping.com/wordpress/
wp-admin/ admin.php?page=file-manager_setting
…
```

???

PostgreSQL

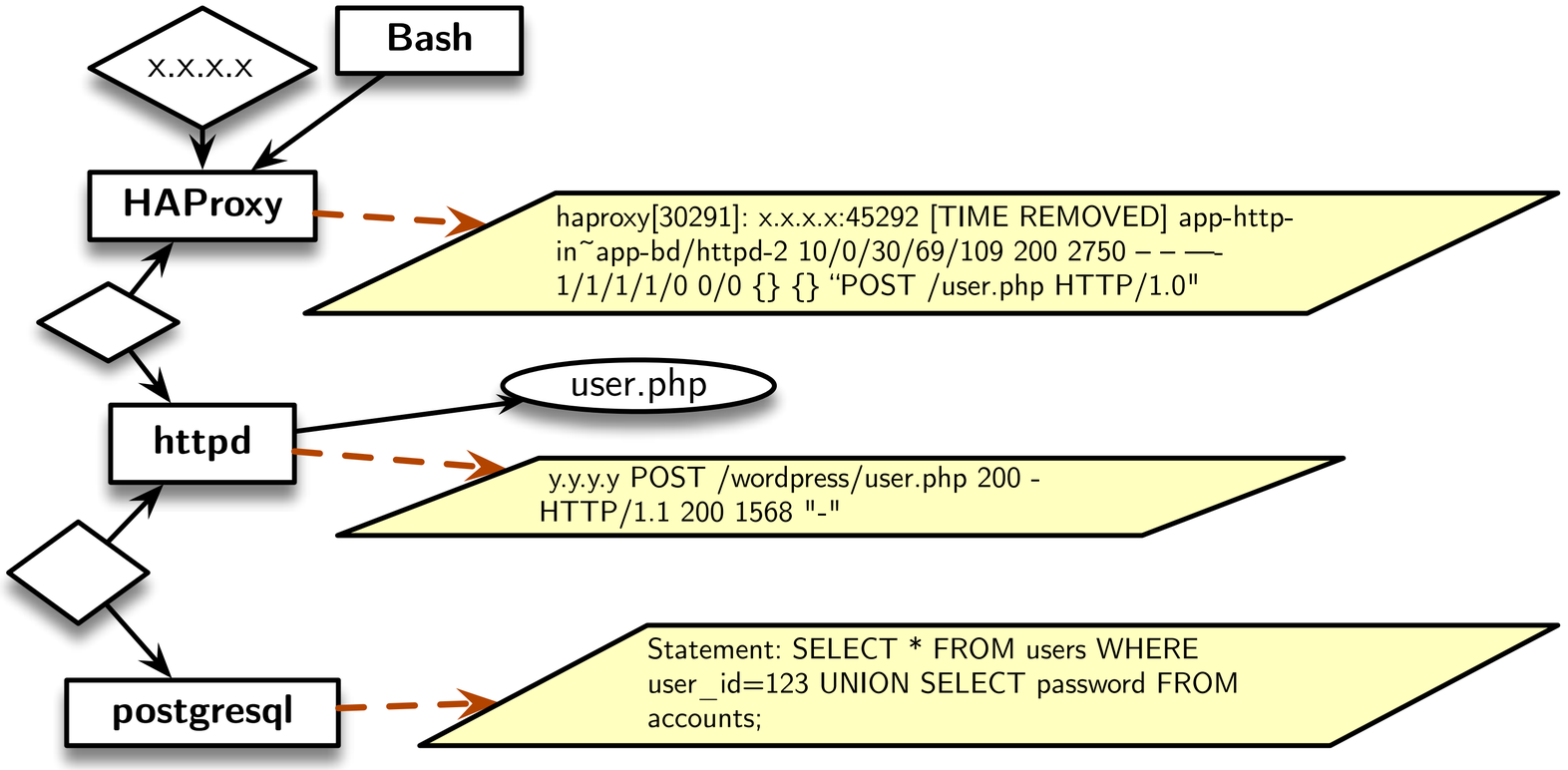
```
…
SELECT * FROM users WHERE user_id=123
UNION SELECT password FROM accounts;
…
```

Provenance graph



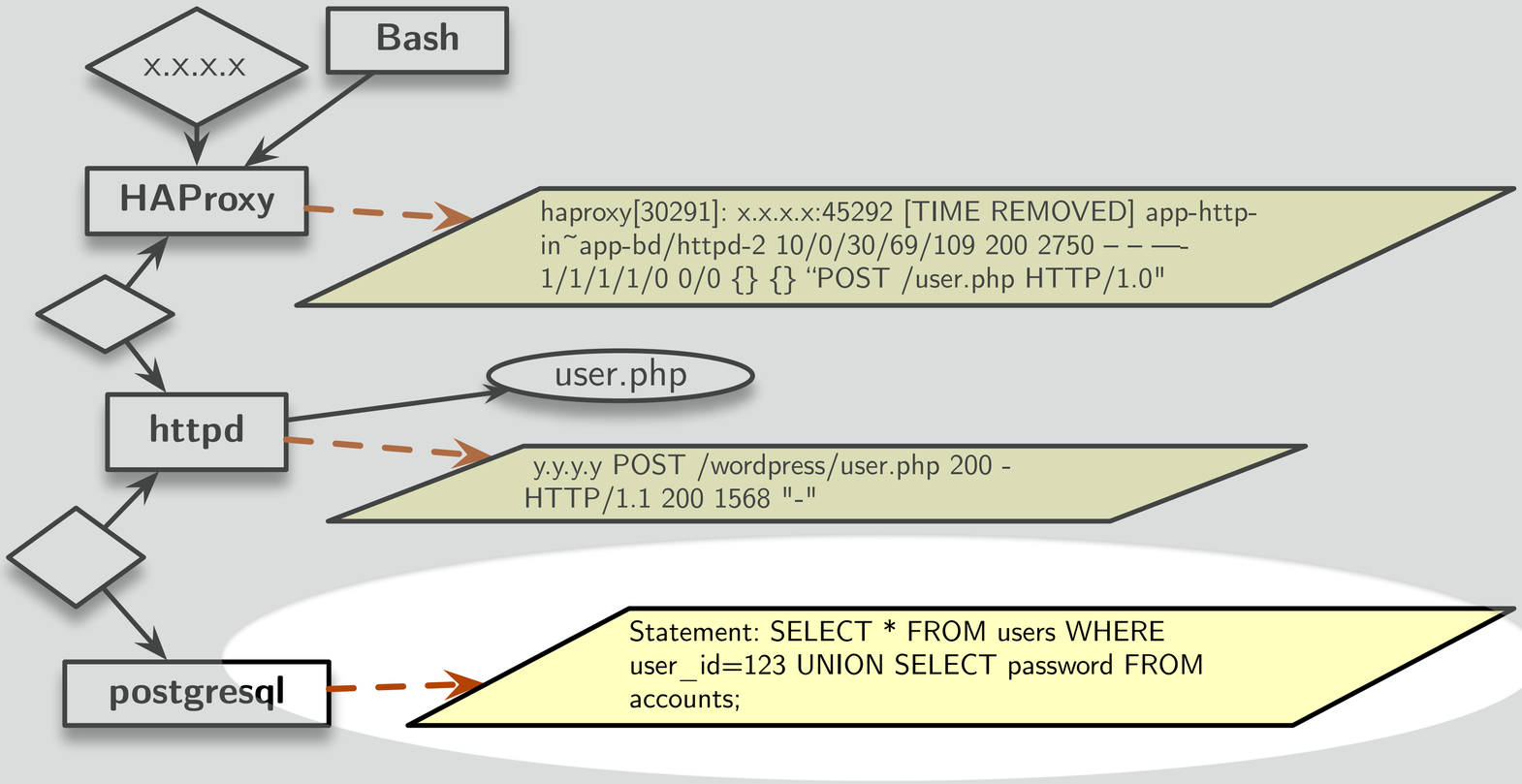
Universal Provenance Graph

1. Identifies which web request (root-cause) led to data exfiltration



Universal Provenance Graph

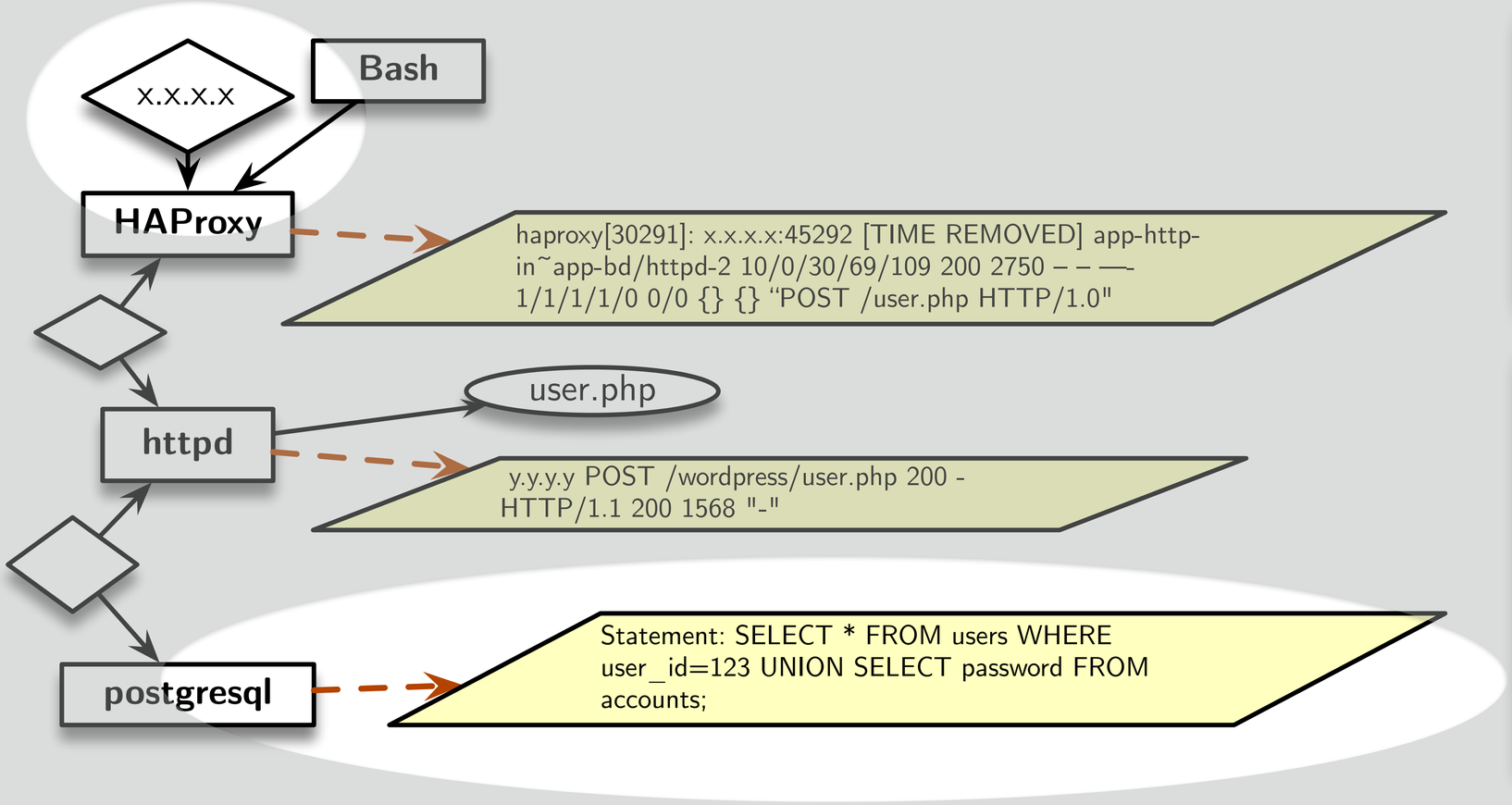
1. Identifies which web request (root-cause) led to data exfiltration



Account credentials were stolen using SQL injection attack

Universal Provenance Graph

1. Identifies which web request (root-cause) led to data exfiltration

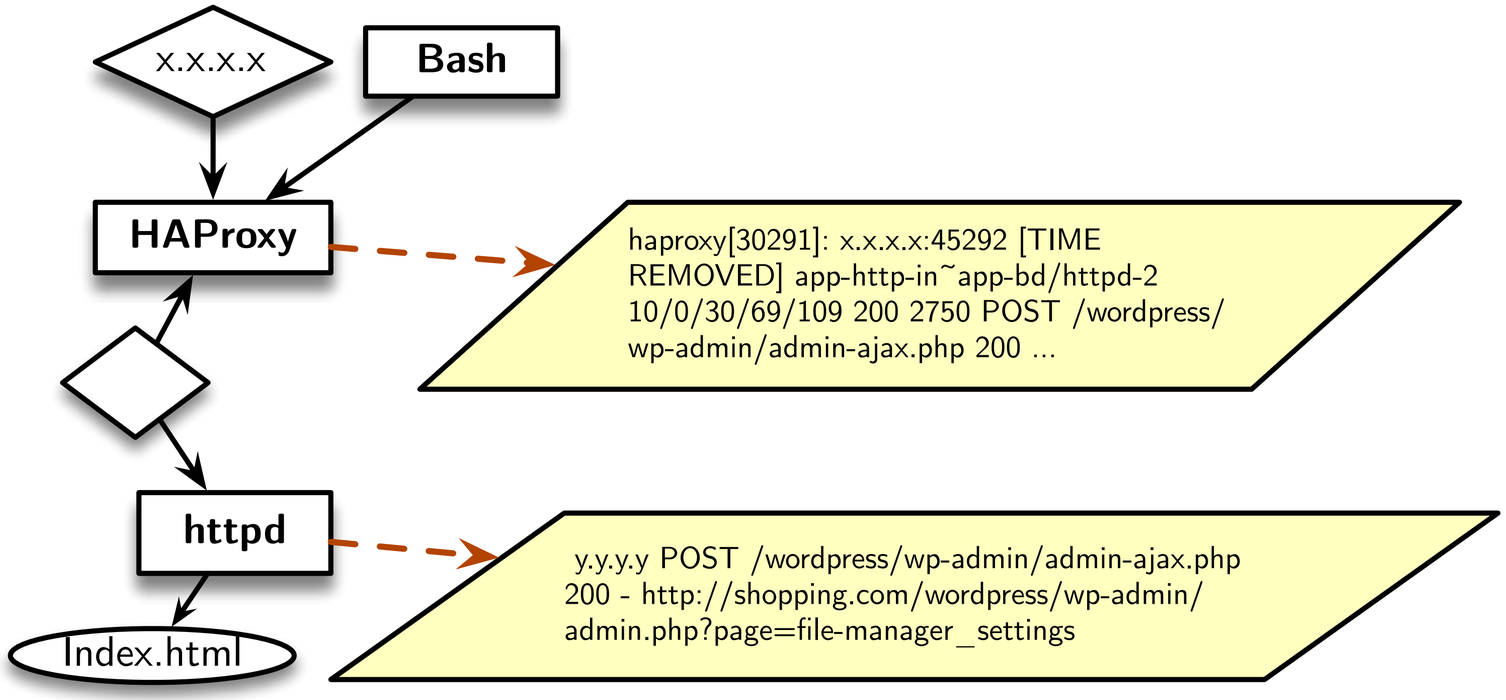


Web request from IP: X.X.X.X started the attack

Account credentials were stolen using SQL injection attack

Universal Provenance Graph

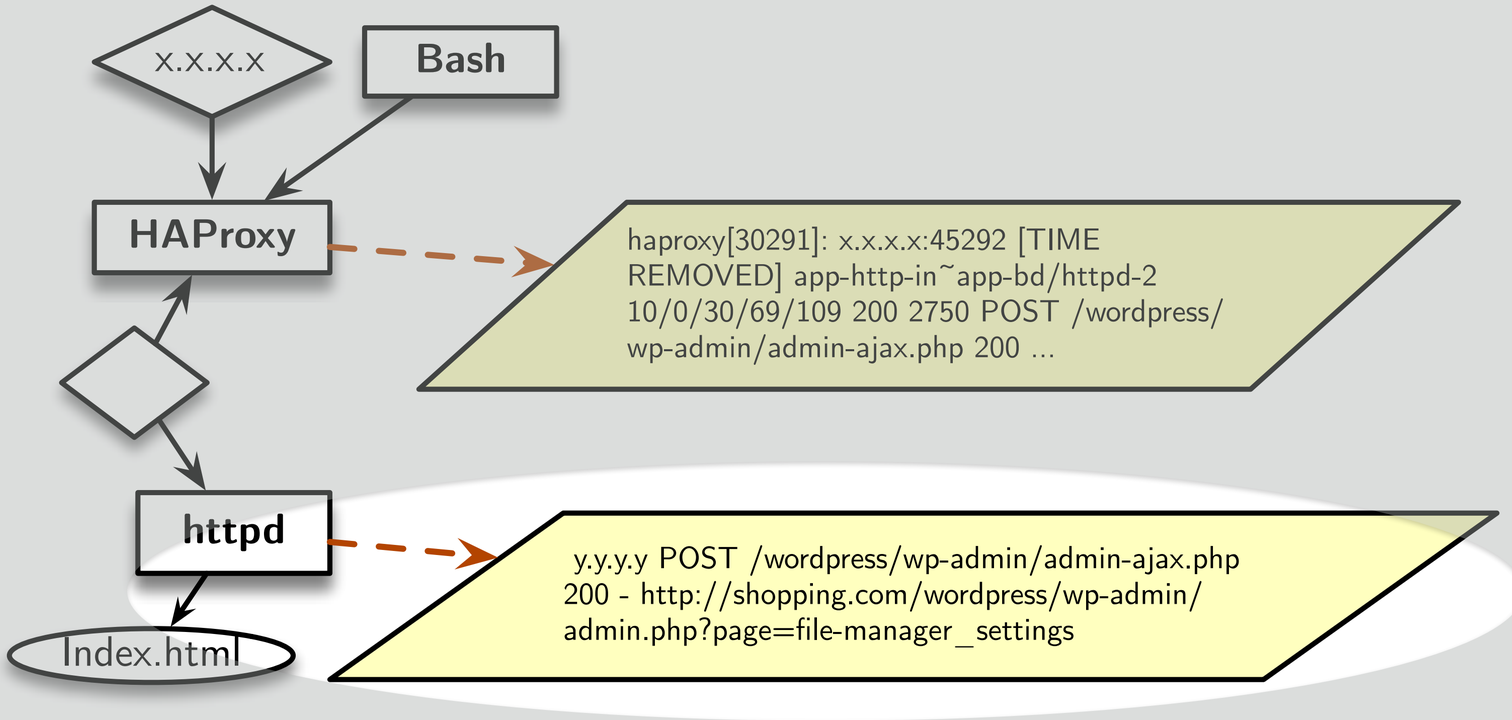
1. Identifies which web request (root-cause) led to data exfiltration
2. Reason about how the website was defaced



Universal Provenance Graph

1. Identifies which web request (root-cause) led to data exfiltration
2. Reason about how the website was defaced

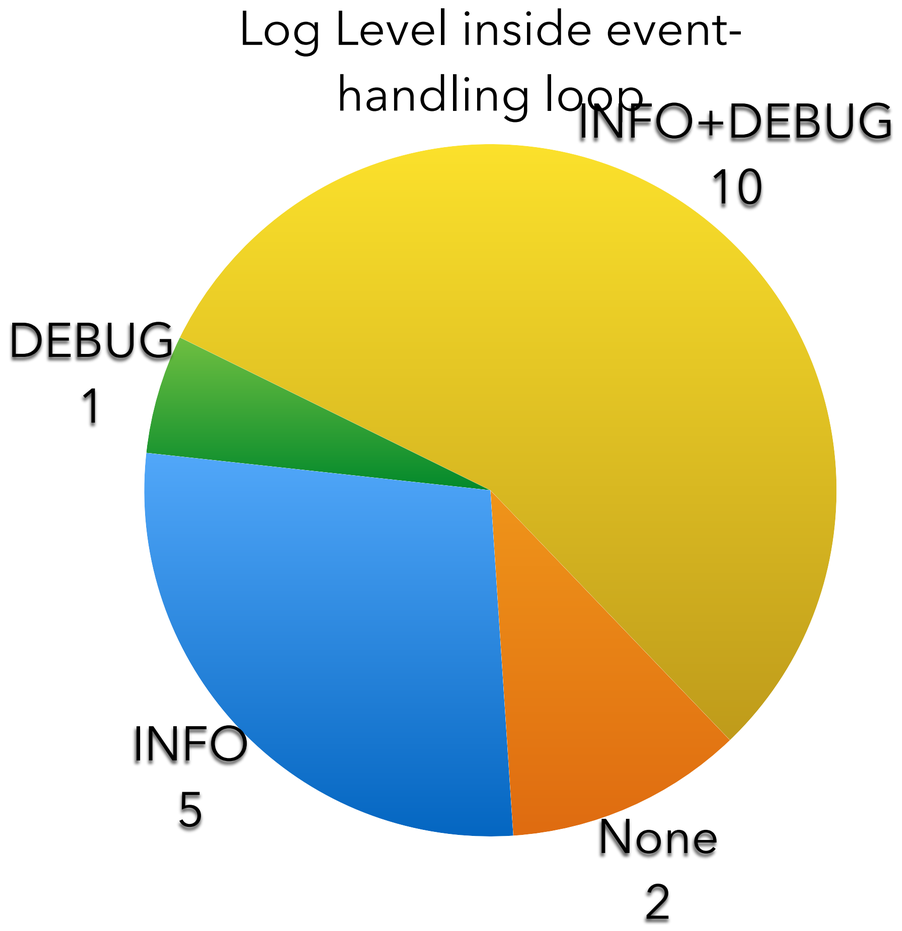
A WordPress file manager plugin used to change index.html.



Evaluation

Evaluation Setup

Program	Binary Size (kB)
Squid	64,250
PostgreSQL	22,299
Redis	8,296
HAProxy	4,095
ntpd	3,503
OpenSSH	2,959
NGINX	2,044
Httpd	1,473
Proftpd	1,392
Lighttpd	1,212
CUPSD	1,210
yafc	1,007
Transmission	930
Postfix	900
memcached	673
wget	559
thttpd	105
skod	47



Evaluation: Static Analysis

Applications	Time to concretize log message (sec)	Time to generated log message control path (sec)
Squid	831	46
PostgreSQL	3880	258
Redis	495	7
…	…	…
Wget	200	3
thttpd	157	8
Skod	12	0

12 secs to 1 hour to concretize log message string

1 sec to 4 mins to generate log message string control flow paths

One time effort to concretize log message string and generate control flow paths

Evaluation: Static Analysis

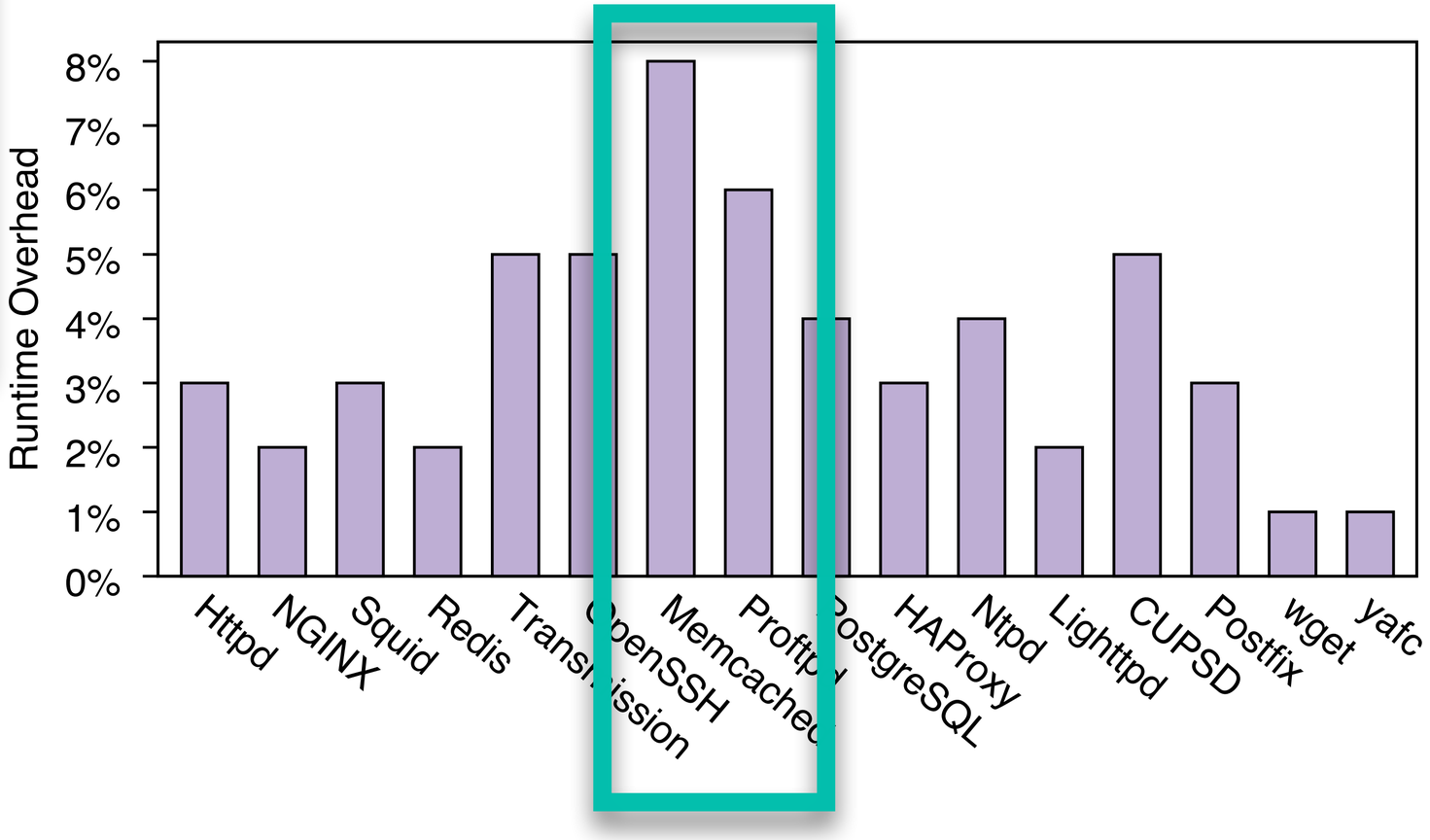
Program	Completeness	
	Callsites	Cov. %
Squid	70	91
PostgreSQL	5,529	64
Redis	394	95
HAProxy	56	95
ntpd	518	95
OpenSSH	869	97
NGINX	925	100
Httpd	211	100
Proftpd	718	100
Lighttpd	358	97
CUPSD	531	100
yafc	60	95
Transmission	227	78
Postfix	98	98
memcached	69	93
wget	275	31
thttpd	5	80
skod	25	100

Coverage: Concretized log message strings relative to identified call sites of log printing functions

>95% Coverage except for four applications

Evaluation: Runtime Overhead

Write intensive applications



Average runtime overhead of around 4%

Limitations

- OmegaLog requires at least one log message inside event-handling loop
  - Good logging practice
- Works on C/C++ application binaries
- Does not work on programs that use asynchronous I/O programming model

Conclusion

- A new approach to
  - Execution partition long-running processes
  - Encode semantic information in system-level logs
- Program analysis to reconcile application event logs with system-level logs
- Evaluation
  - Low overhead
  - High-fidelity attack investigation

Conclusion

- A new approach to
  - Execution partition long-running processes
  - Encode semantic information in system-level logs
- Program analysis to reconcile application event logs with system-level logs
- Evaluation
  - Low overhead
  - High-fidelity attack investigation

Thanks & Questions


whassan3@illinois.edu

Backup Slides

Examples

```
/* src/main.c */
static void daemon_loop(void) {
  ...
  while (TRUE){
  ...

listen_conn=pr_ipbind_accept_conn(&listenfds,&fd
);
  ...
  fork_server(fd,listen_conn,no_forking);
  ...
  }}
static void fork_server(int fd, conn_t *l, ...){
  ...
  pr_log_pri(PR_LOG_INFO,"%s session opened.",

pr_session_get_protocol(PR_SESS_PROTO_FL_LOGOUT)
);
  ...
}
```

Proftpd

```
/* /src/networking.c */
while(...) {
  /* Wait for TCP connection */
  cfd = anetTcpAccept(server.neterr, fd, cip,
sizeof(cip), &cport);

  serverLog(LL_VERBOSE,"Accepted %s:%d", cip,
cport);

 ... /*Process request here*/

  serverLog(LL_VERBOSE, "Client closed
connection");
}
```

Redis

Used software categories from BEEP (NDSS'13)

Picked famous applications for each category

18 of those applications were used in previous work on provenance

TABLE II: Logging behavior of long-running applications.

Category		Total Apps	Apps with Log Verbosity of			
			IN+DE	INFO	DEBUG	None
Client-Server	Web server	9	7	1	0	1
	Database server	9	7	1	1	0
	SSH server	5	5	0	0	0
	FTP server	5	4	0	1	0
	Mail server	4	3	1	0	0
	Proxy server	4	3	1	0	0
	DNS server	3	2	0	1	0
	Version control server	2	0	1	1	0
	Message broker	3	2	0	1	0
	Print server	2	1	0	1	0
	FTP client	6	0	1	4	1
	Email client	3	1	0	1	1
	Bittorrent client	4	3	1	0	0
	NTP client	3	0	1	2	0
GUI	Audio/Video player	8	1	0	3	4
	PDF reader	4	0	0	0	4
	Image tool	5	0	0	1	4
Total		**79**	**39**	**8**	**17**	**15**