# JSgraph

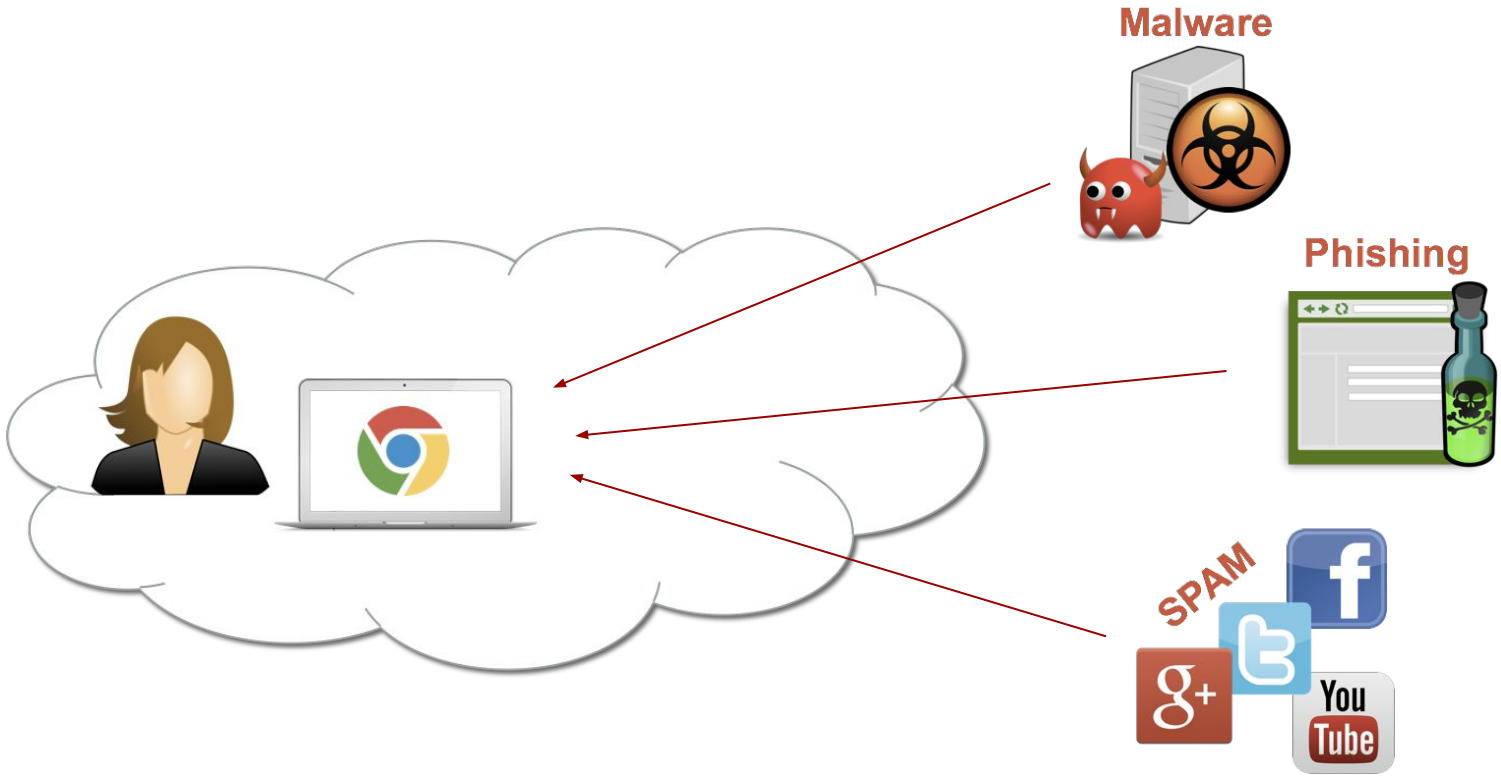
## Enabling Reconstruction of Web Attacks via Efficient Tracking of Live In-Browser JavaScript Executions

Bo Li, Phani Vadrevu, Kyu Hyung Lee, and Roberto Perdisci

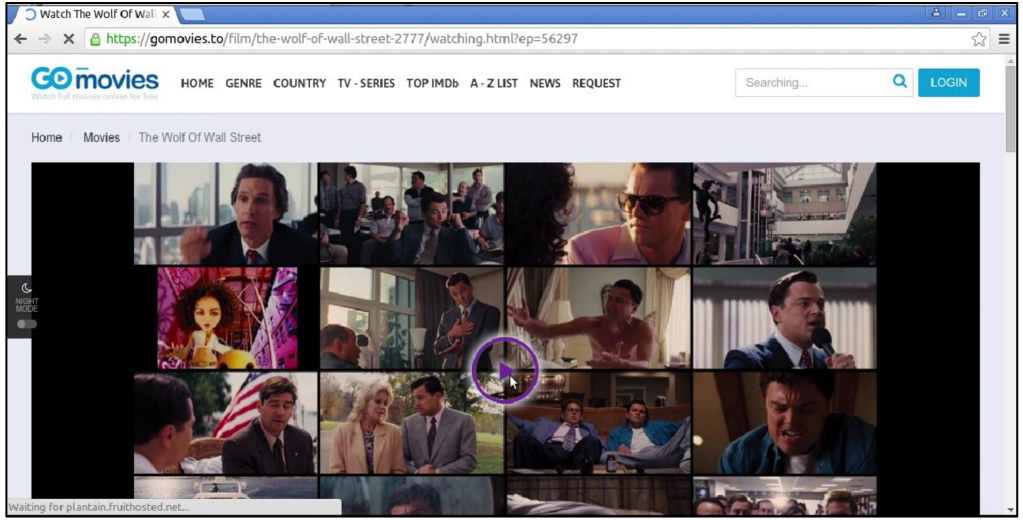
*Dept. of Computer Science - University of Georgia*


UNIVERSITY OF GEORGIA

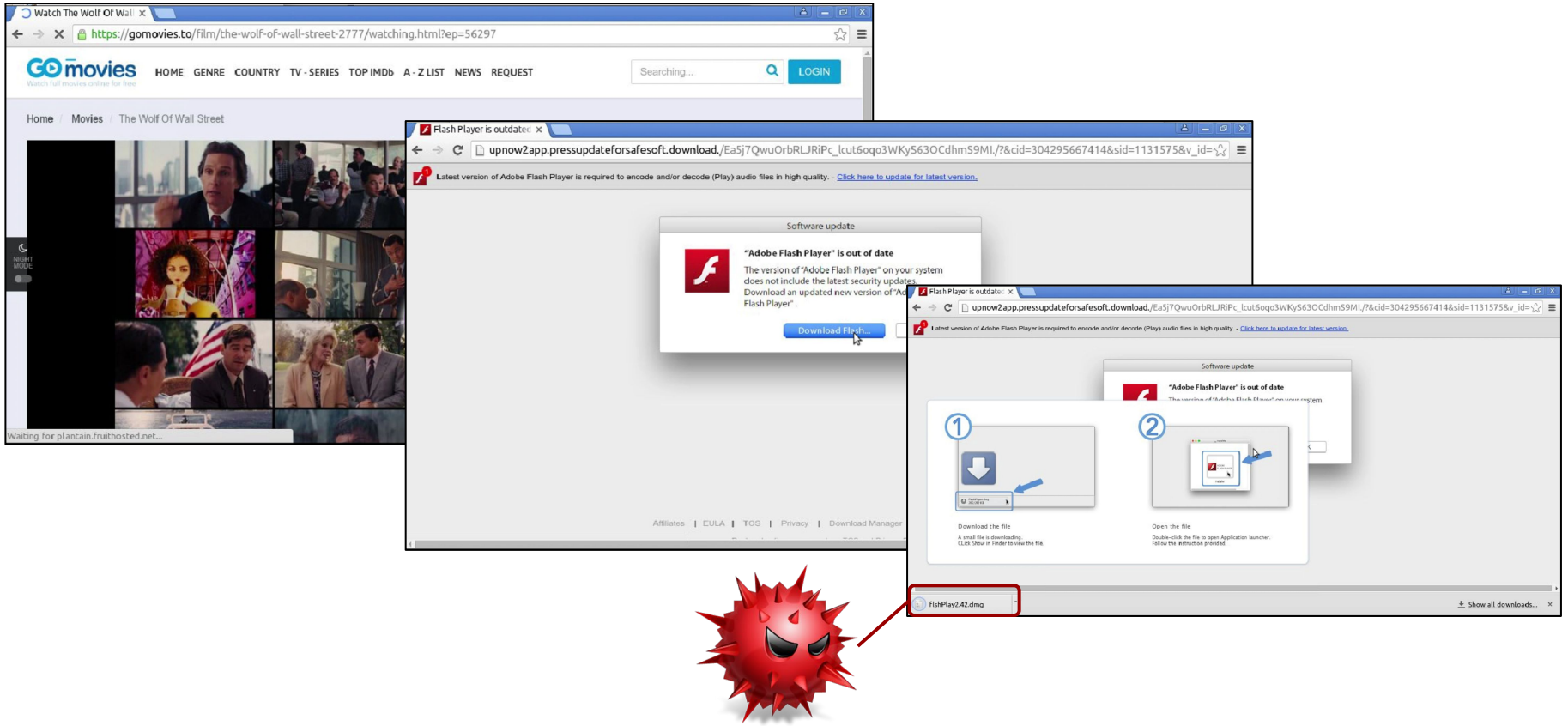
# Many security incidents originate from the Web



# Web-driven malware infections



# Web-driven malware infections

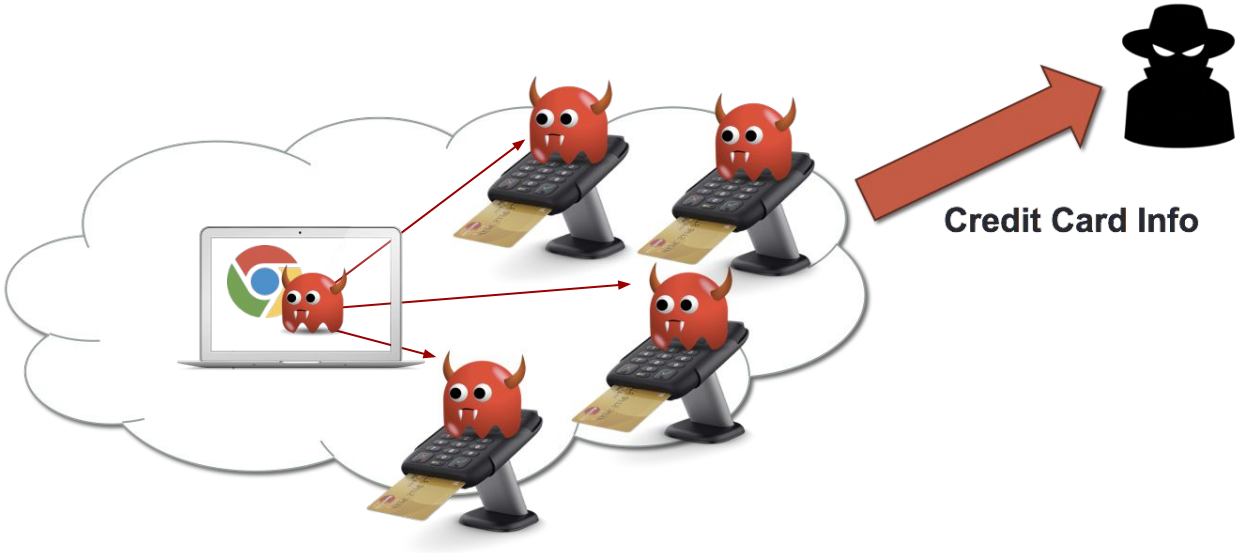



[Rafique et al] It's Free for a Reason: Exploring the Ecosystem of Free Live Streaming Services. NDSS 2016

# Malware infections can have huge consequences!

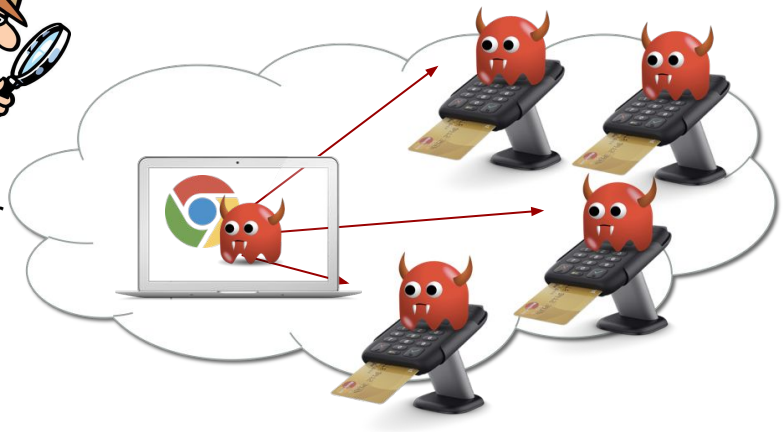
Target Data Breach

Home Depot Breach



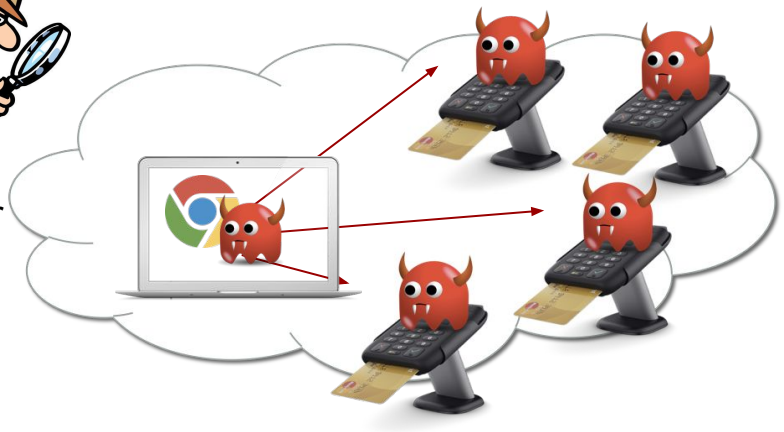
# Forensic investigation to find root causes

Forensic Investigator



# Forensic investigation to find root causes

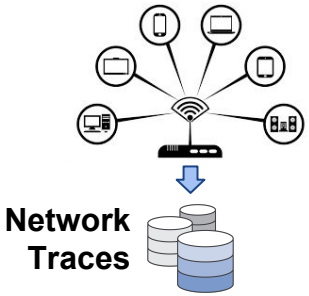
Forensic Investigator

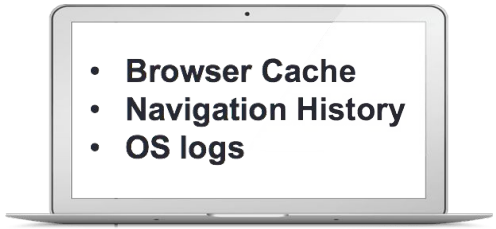



Is it possible to reconstruct exactly where the attack came from?

# Challenges to web attack reconstruction

- Existing logs are sparse, short-lived, and provide only limited information
- Semantic gap between network traces and browsing events

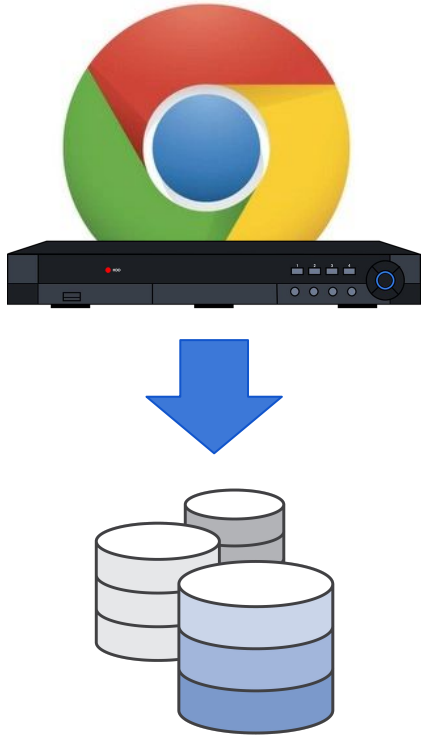


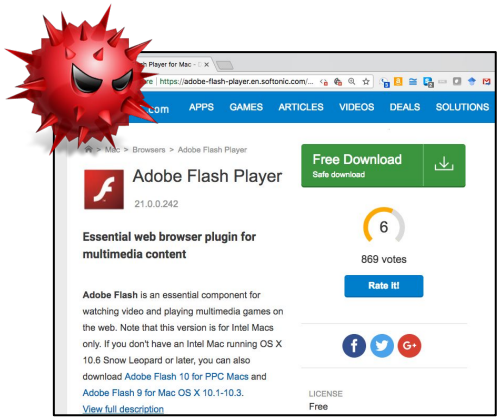

**Need more detailed and persistent web audit logs!**

# Requirements for Web log recording systems

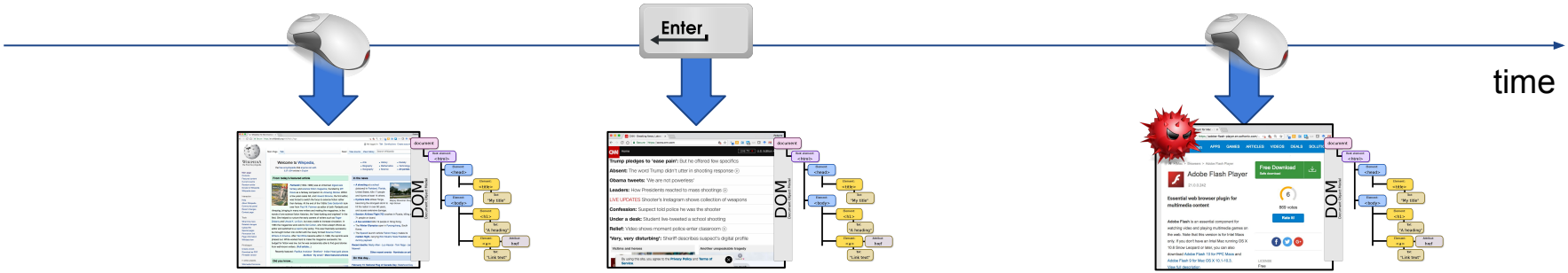
- **Always on**
  - attacks are unpredictable and ephemeral
- **Efficient**
  - recording overhead must not decrease browser usability
- **No functional interference**
  - same browser architecture and functions
- **Transparent to the user**
  - no user action needed to enable logging
- **Limited storage overhead**
  - audit logs need to be preserved for long periods of time



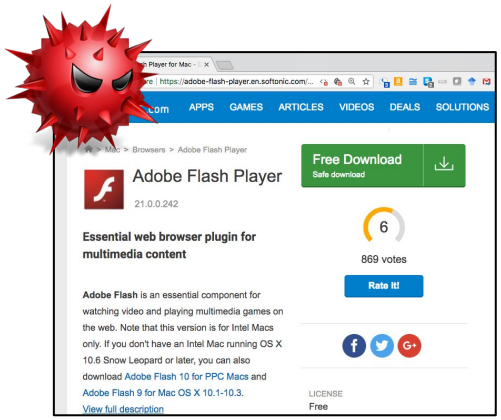
# ChromePic [NDSS 2017]




- Take *webshot* at every user interaction
  - Synchronous screenshots
  - Synchronous “deep” DOM snapshots
- Features
  - Efficient, transparent, always on recording
  - Forensic rigor (synchronous logs)

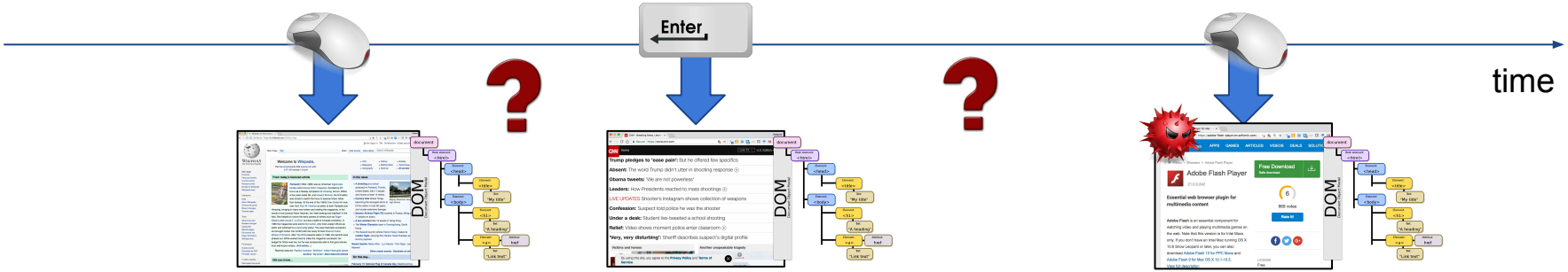


# ChromePic's main limitation



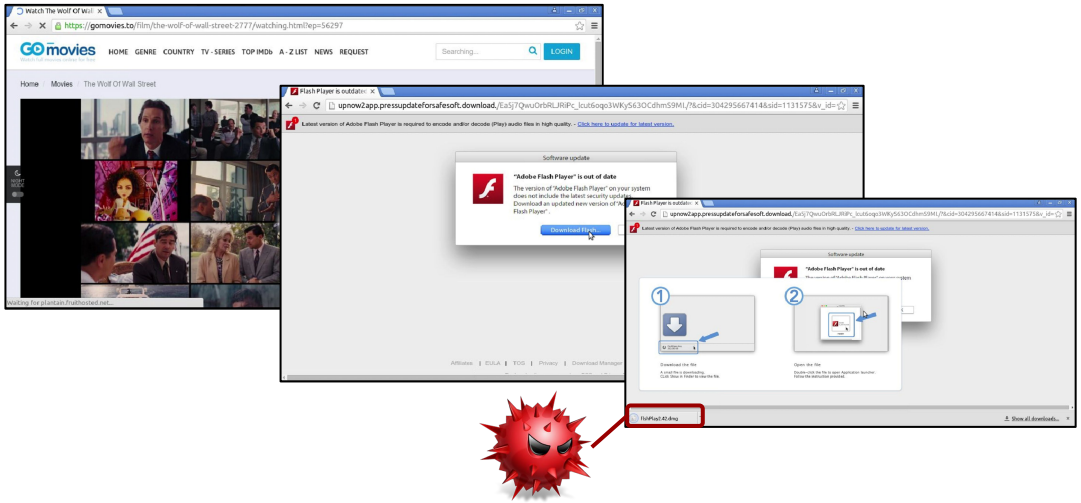

Missing info about what happens between user actions

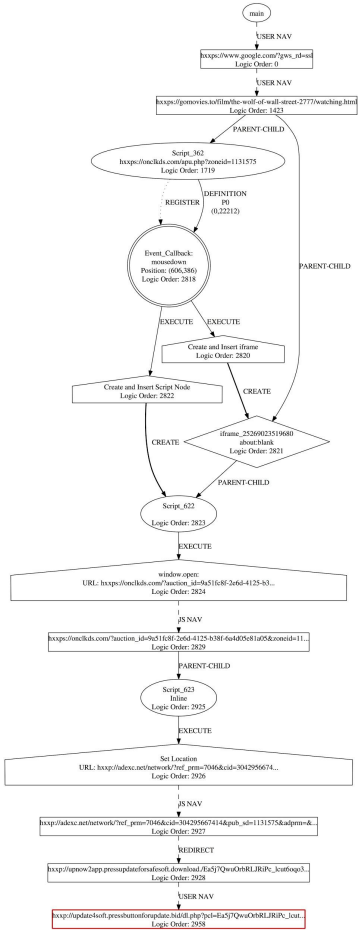
- how was the attack constructed?
- malicious JS code execution?



# JSgraph Overview

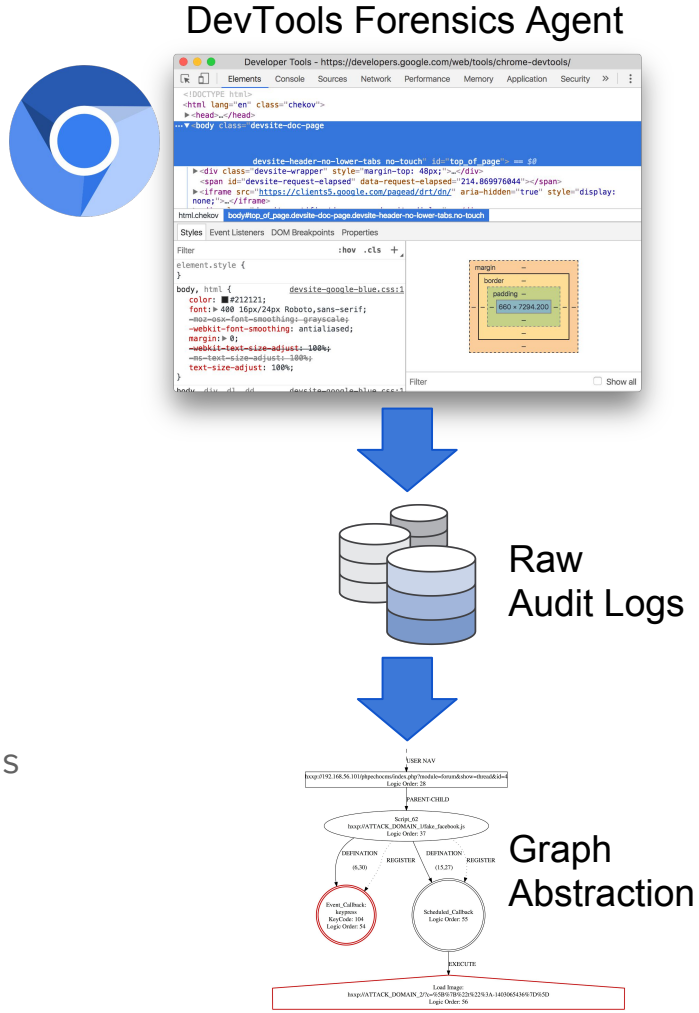
- Detailed logging of navigation events
- Continuous recording of DOM changes
- Record details of how JS code changes the DOM
- Dependences between events and JS callbacks
- Abstract detailed logs into easier-to-interpret graphs



# JSgraph System

- Extends Chromium's DevTools
  - Add custom *Inspector Agent*
  - Add *Inspector Instrumentation* hooks
- Continuously track DOM changes
  - didInsertDOMNode, willRemoveDOMNode
  - didModifyDOMAttribute
  - createdChildFrame, …
- Log JS APIs, script executions, and callbacks
  - compiledScript ➔ script ID + source code
  - runScriptBegin/End
  - callFunctionBegin/End ➔ log callback function details
    - Where was the function defined?
    - What event triggered the callback, …
  - window.open(), location.replace()
  - XMLHttpRequests (open, send, …), …



# Code Instrumentation and Example Logs

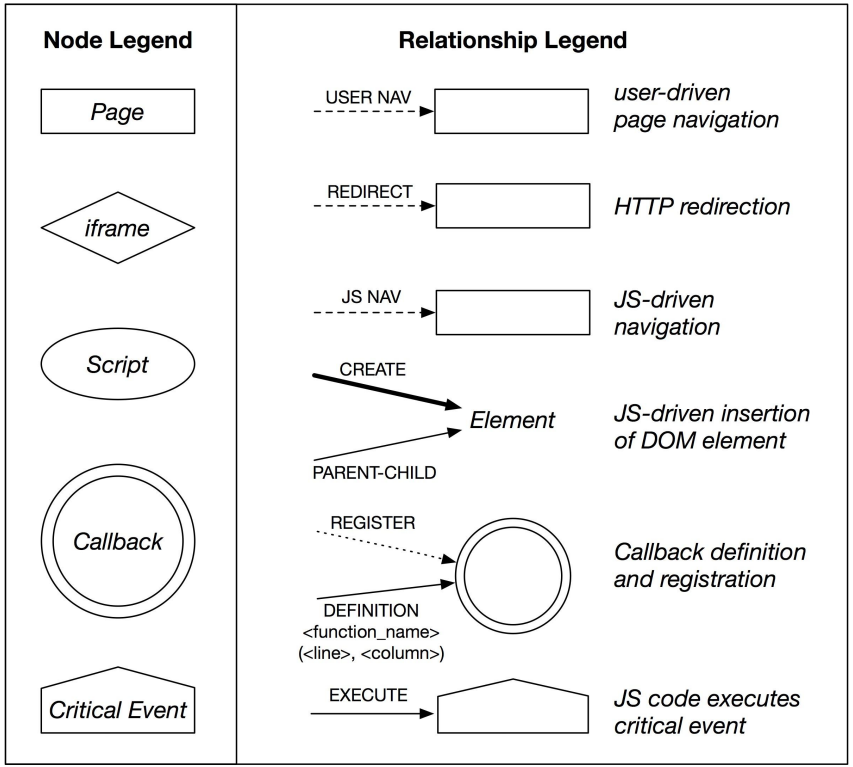
Custom *Inspector Instrumentaiton* Hook

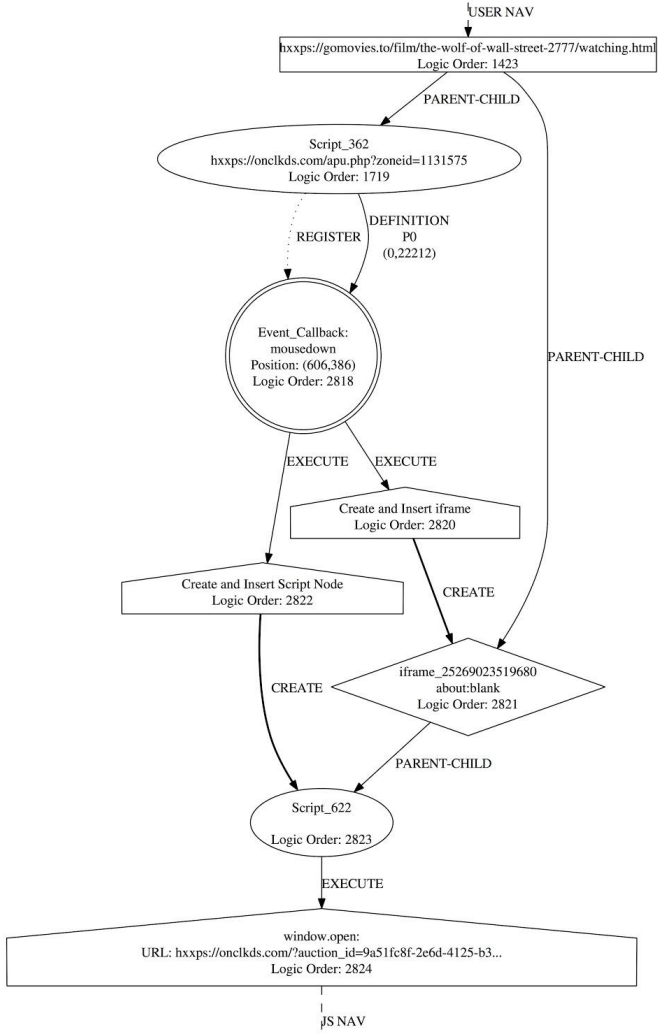
```
v8::Local<v8::Value> ScriptController::executeScriptAndReturnValue(
        v8::Local<v8::Context> context,
        const ScriptSourceCode& source,
        AccessControlStatus accessControlStatus,
        double* compilationFinishTime) {
  ...
  v8::Local<v8::Script> script;
  if (!v8Call(V8ScriptRunner::compileScript(source, isolate(), ...))
      return result;
  // :: Forensics :: BEGIN
  InspectorInstrumentation::handleCompileScriptForensics(
        frame()->document(),
        v8String(isolate(), source.source()),
        script->GetUnboundScript()->GetId(),
        source.url(),
        source.startPosition());
  // :: Forensics :: END
  if (compilationFinishTime) {
     *compilationFinishTime = WTF::monotonicallyIncreasingTime();
  }
  ...
}
```

Log Trace

```
...
InspectorForensicsAgent::handleCreateChildFrameLoaderForensics
ForensicDataStore::recordChildFrame : requestURL: about:blank, frame: 25269023519680
InspectorForensicsAgent::handleCreateChildFrameLoaderEndForensics
ForensicDataStore::recordInsertDOMNodeEvent: m_selfNode: 43987025453064,
m_parentNode: 43987026382560, m_nodeSource: <iframe style="display: none;"></iframe>
InspectorForensicsAgent::didModifyDOMAttr: m_selfNode: 43987025302224, m_nodeSource: <script type="text/javascript"></script>
ForensicDataStore::recordInsertDOMNodeEvent: m_selfNode: 43987026264856, m_parentNode: 43987025302224,
m_nodeSource: window.top = null;window.frameElement = null;
var newWin = window.open("https://onclkds.com/?auction_id=9a51fc8f-2e6d-4125- ... ", "new_popup_window_1494561683103", "");
window.parent.newWin_1494561683114 = newWin; window.parent = null; newWin.opener = null;
InspectorForensicsAgent::handleCompileScriptForensics : Thread_id:140362442277824,
Script_id:622, URL: , line: 0, column: 0, Source: window.top = null; window.frameElement = null;
var newWin = window.open("https://onclkds.com/?auction_id=9a51fc8f-2e6d-4125-  ... ", "new_popup_window_1494561683103", "");
window.parent.newWin_1494561683114 = newWin; window.parent = null; newWin.opener = null;
InspectorForensicsAgent::handleRunCompiledScriptStartForensics : Thread_id:140362442277824,
iframe: 25269023519680, Script_id: 622
InspectorForensicsAgent::handleWindowOpenForensics : URL: https://onclkds.com/?auction_id=9a51fc8f-2e6d-4125-…,
frameName: new_popup_window_1494561683103, windowFeaturesString:
...
```

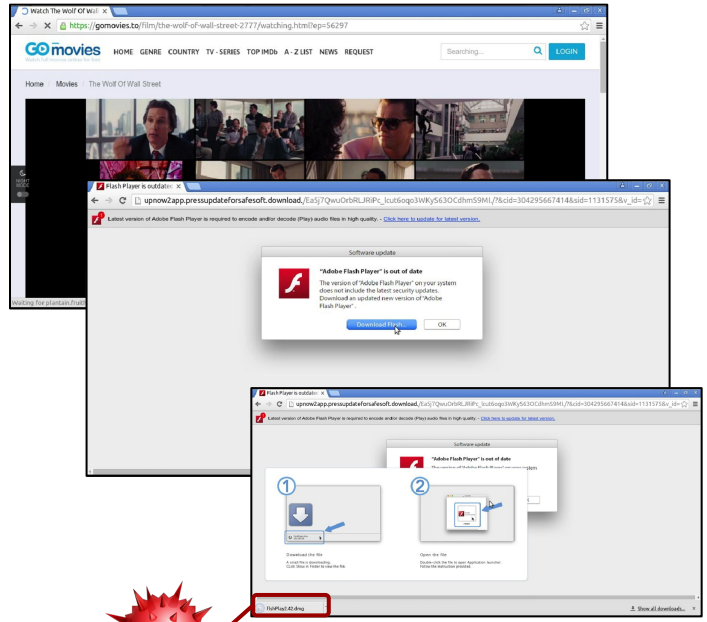
# Abstracting audit logs

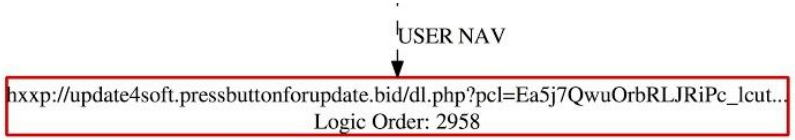


# Example attack reconstruction

- Social Engineering Malware Download
- 1st Step: identify suspicious download events
  - Forensic analyst lists all download events
  - Narrows the investigation to a set of possible target machines
  - Identifies time window of interest
  - Selects interesting file download logs as *pivot* point for analysis

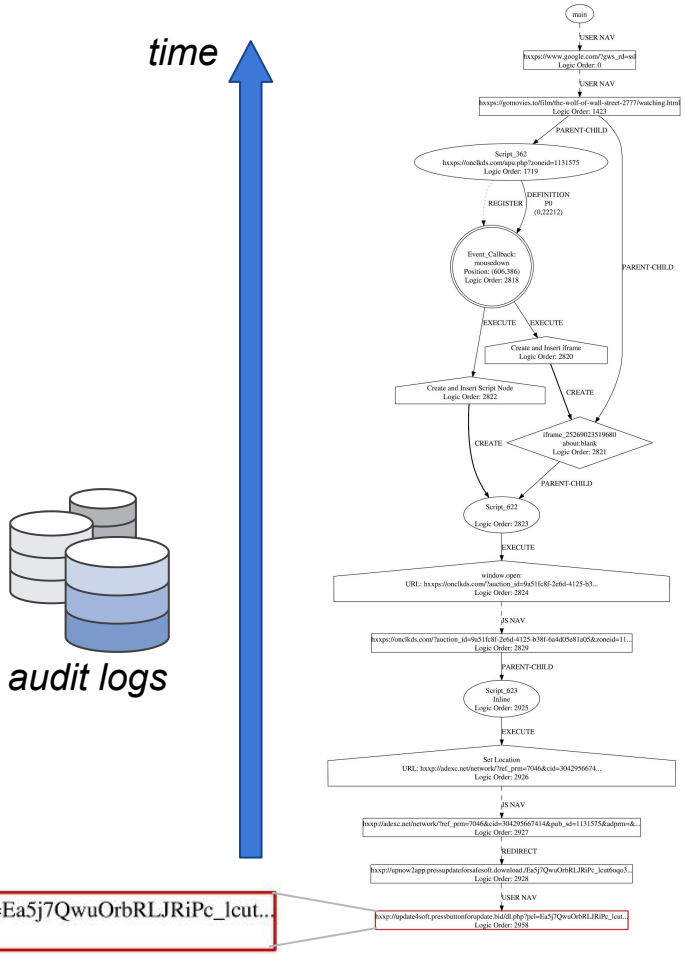


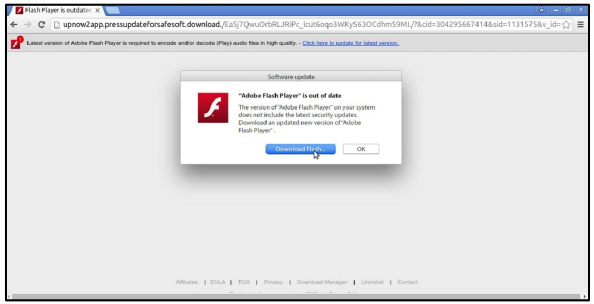

All details about file download events captured in JSgraph's audit logs!

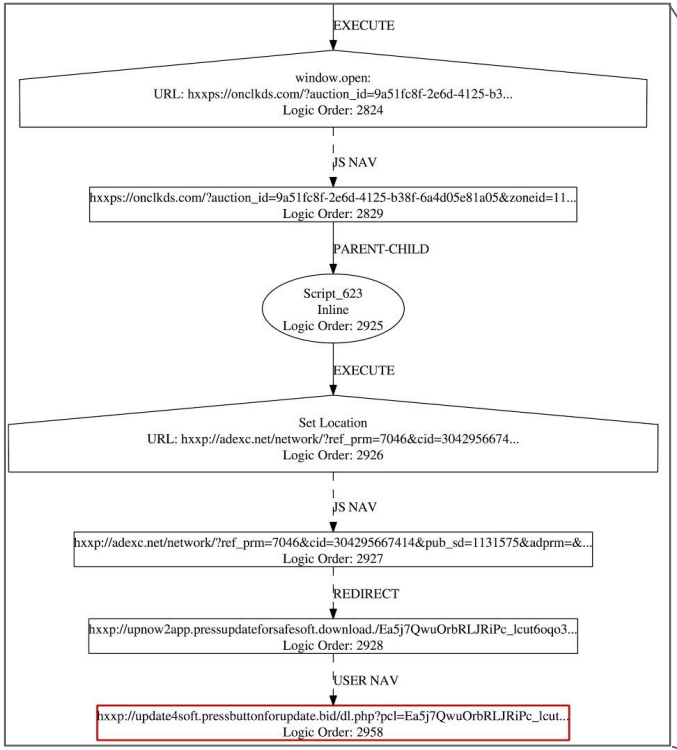
# Backward Tracking

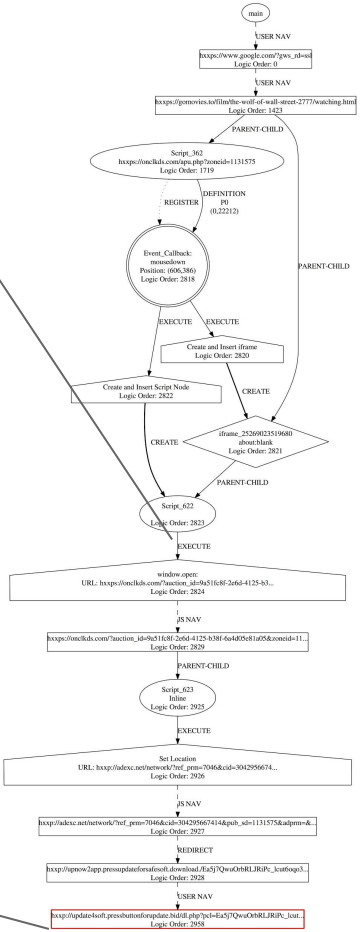
- Walk back in time
- Reconstruct sequence of audit logs
- Only consider logs for events with direct path to pivot point



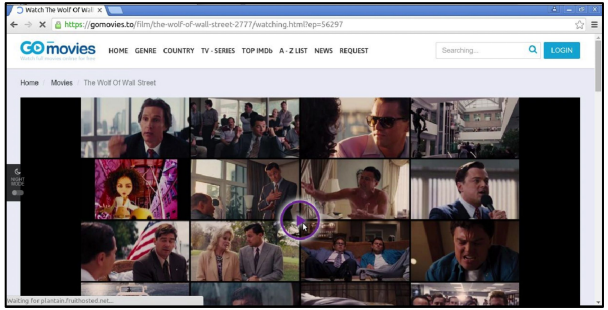
# Backward Tracking

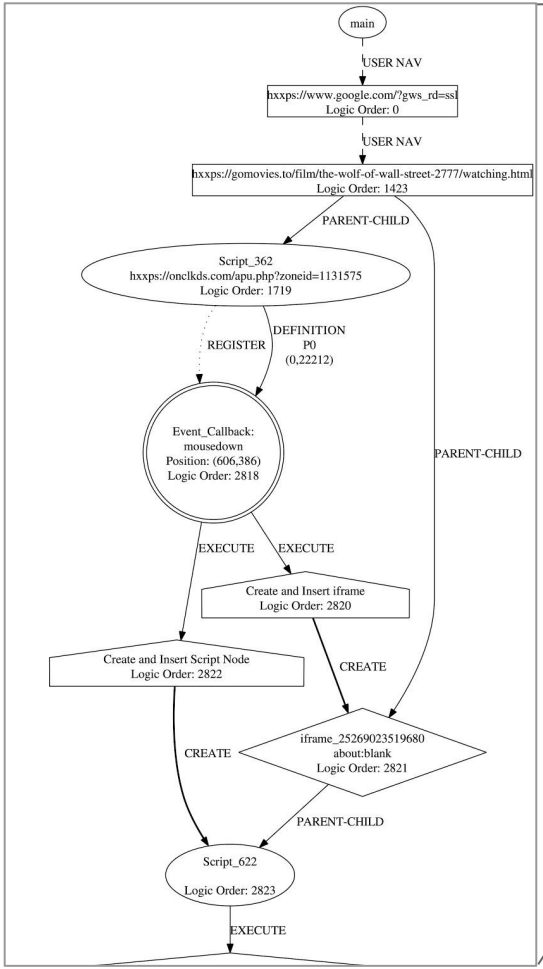


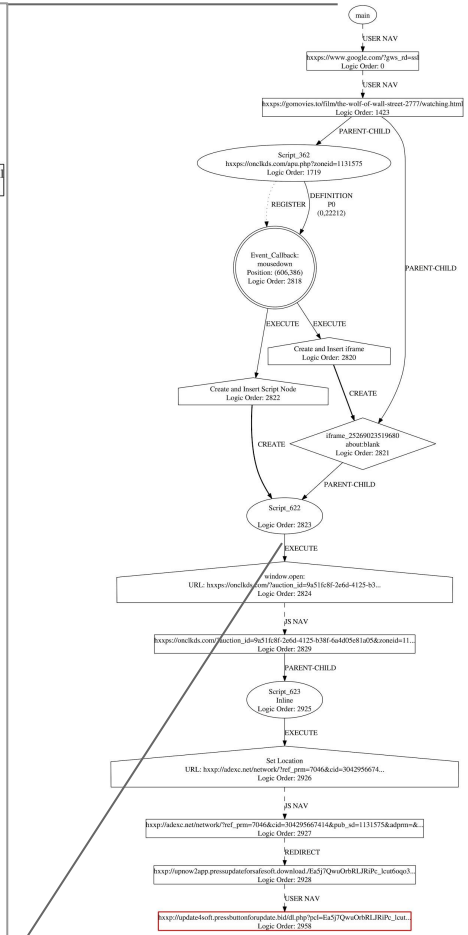



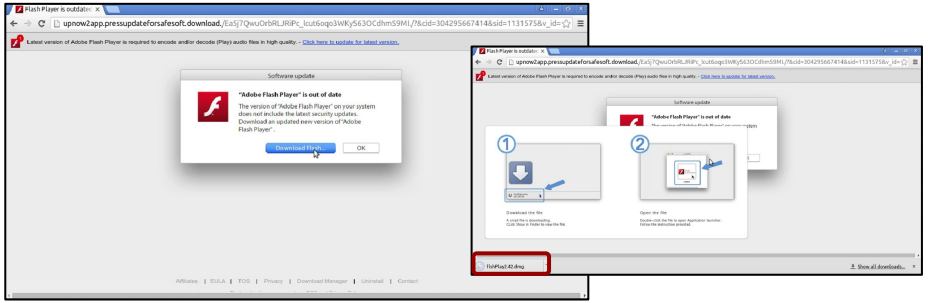
# Backward Tracking

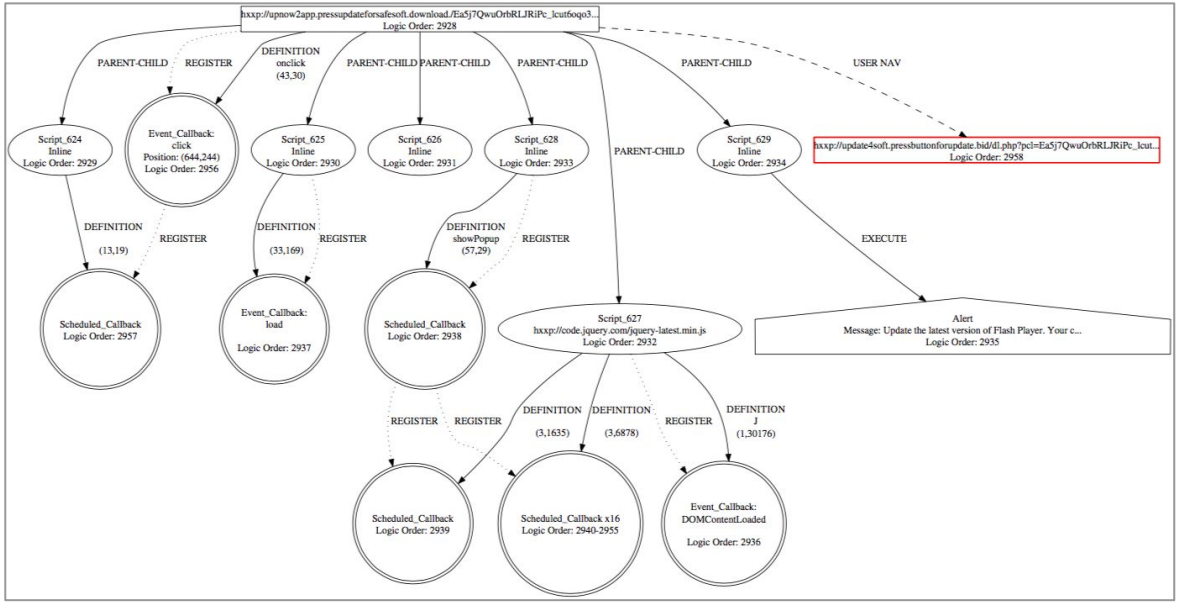


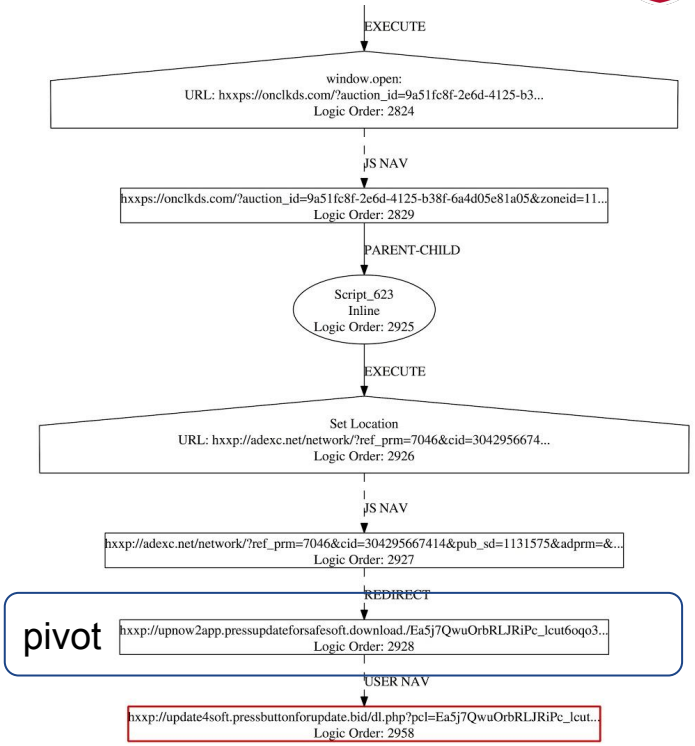



# Forward Tracking




partial backward tracking



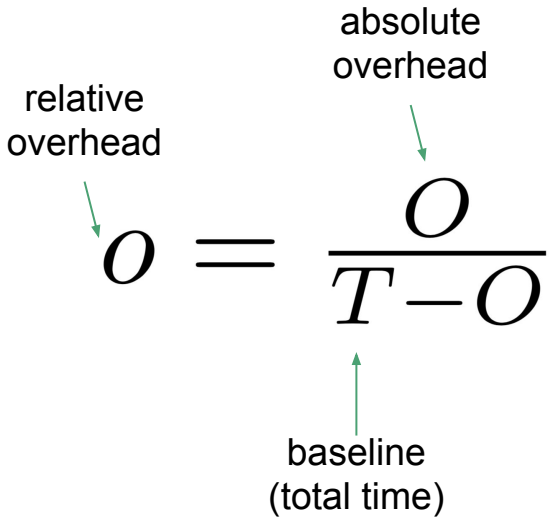
# Measuring Performance Overhead

- Use Chromium's TRACE_EVENT infrastructure
  - TRACE_EVENT0 ➔ measures the time spent within a function
  - TRACE_EVENT_BEGIN0 / _END0 ➔ measure execution time between two code points

```
1  void InspectorForensicsAgent::handleCompileScriptForensics(v8::Local<v8::String> code,
2                                                              int scriptId, const KURL& url,
3                                                              const TextPosition& startposition)
4  {
5      TRACE_EVENT0("jsgraph", "JSCapsule::InspectorForensicsAgent::handleCompileScriptForensics");
6
7      const String& code_string = String(V8StringResource<>(code));
8      m_blinkPlatform->fileUtilities()->tab_log(
9            "InspectorForensicsAgent::handleCompileScriptForensics"
10           "Thread_id:%ld, Script_id:%d, URL: %s, line: %d, column: %d,"
11           " Source: \n %s \n",
12           (long)WTF::currentThread,
13           scriptId,
14           url.string().latin1().data(),
15           startposition.m_line.zeroBasedInt(),
16           startposition.m_column.zeroBasedInt(),
17           code_string.latin1().data()
18     );
19 }
```

# Measuring Performance Overhead

- Page load
  - t(loadEventFired) - t(didStartProvisionalLoad)
- DOM construction
  - t(navigation to new page) - t(first node inserted)
  - excludes JS execution time
- JS execution
  - ∑ t(run compiled script end) - t(run compiled script begin)
  - ∑ t(call function end) - t(call function begin)
- Overall
  - t(navigation to next page) - t(didStartProvisionalLoad)




| Experiment | Overall | Page load | DOM Construction | JS Execution |
|---|---|---|---|---|
| Linux Top1K | 0.5%, 3.1% | 3.2%, 7.4% | 0.2%, 1.6% | 6.8%, 20.1% |
| Linux Top10 | 1.6%, 3.7% | 3.3%, 5.7% | 0.6%, 1.2% | 9.6 %, 17.1% |
| Android Top10 | 1.5%, 4.7% | 3.9%, 8.2% | 0.4%, 1.7% | 10.2%, 17.3% |

Relative performance overhead: **50th-** and **95th**-percentile

# Storage Overhead

- Linux top 10 experiments
  - 50 min of active browsing = 37MB compressed logs
  - = 0.74 MB/min
- Extrapolation to enterprise network
  - Assuming 8 hours of browsing / day
  - 262 work days / year
  - < 91GB of storage per user / year
  - < 91TB to keep web audit logs produced by 1,000 users for one entire year

# Conclusion

- JSgraph records audit logs to enable detailed reconstruction of web security incidents
- JSgraph is not limited to recording state of web pages only at the time of user actions (unlike ChromePic)
- Recording of critical browser-internal events, e.g., JS ↔ DOM interactions
- Post-processing module to abstract audit logs into easier-to-interpret graphs
- Acceptable performance and storage overhead

# Thank you!

https://github.com/perdisci/JSgraph