SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data[1]

Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete,
Rigel Gjomemo, R. Sekar, Scott D. Stoller, and V.N. Venkatakrishnan
Stony Brook University and University of Illinois at Chicago

[1]This work was primarily supported by DARPA (contract FA8650-15-C-7561) and in part by NSF (CNS-1319137, CNS-1421893, CNS-1514472 and DGE-1069311) and ONR (N00014-15-1-2208 and N00014-15-1-2378).

Challenges of Advanced Persistent Threat (APT) Campaigns

- APTs combine social engineering (e.g., spearphishing) with advanced exploits
  - Get past first-line defenses, e.g., ASLR, DEP, and sandboxes
- Enterprises forced to rely on second-line defenses
  - Intrusion detection systems (IDS), Security incident and event management (SIEM), ...
- Key challenges
  - *"Needle in a haystack"* — spot the minuscule fraction of *real attacks* within vast quantities of data emitted by these systems.
  - *"Connecting the dots"* — stitch isolated steps together into a larger campaign.

Result: Many APT campaigns remain undetected for months.

Previous Research

Attack detection: Numerous intrusion detection techniques have been developed.

- Real-world use hampered by high false positive rates

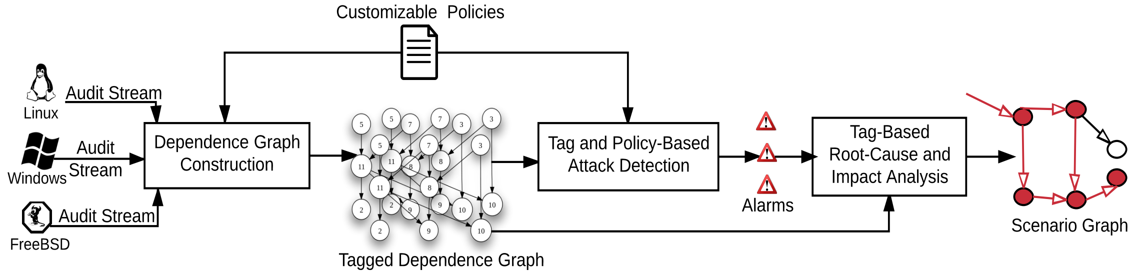
Linking attack campaign steps: Backtracker [King and Chen] and subsequent works use dependencies recorded in system logs to stitch together attacker activities

- Forensic tool — does not help analyst to understand ongoing attacks in real-time.
- Result can include many irrelevant events due to explosion of (false) dependencies.
  - Fine-grained dependency tracking techniques developed to address this problem, but have performance and compatibility costs.

Goals and Challenges

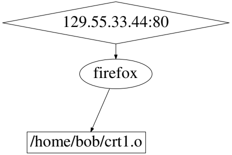
- Real-time reconstruction of APT campaign from audit logs
  - Provide compact visual summary of the campaign
- Key challenges
  - *Data volume:* hundreds of millions to billions per day
  - *"Needle in a haystack"* — only a small fraction of these are attacks, perhaps one in a million
    - Avoid being swamped in false positives
  - *"Connecting the dots"* — link successive steps of an APT campaign
- Part of the DARPA Transparent Computing program
  - Our adversarial evaluation relies on Red Team engagements organized by DARPA.

SLEUTH Architecture and Contributions



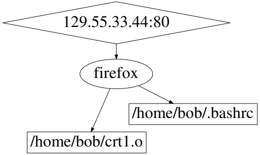
- Space-efficient in-memory dependence graph representation
- Effective attack detection based on trustworthiness and confidentiality tags
- Customizable policy framework for tag assignment and propagation
- Highly effective and efficient tag-based backward and impact analysis
- Experimental evaluation: fast, accurate and compact visual representation of APT campaigns

Illustrative Example



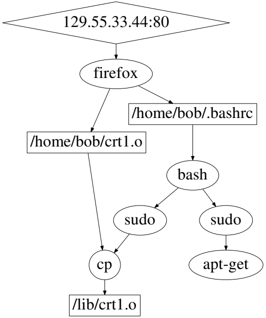
- **Attacker goal:** Insert backdoor into a vendor's software
- **Steps:**
  1. Use a browser vulnerability to drop a malicious version of `crt1.o` in `/home/bob`

Illustrative Example



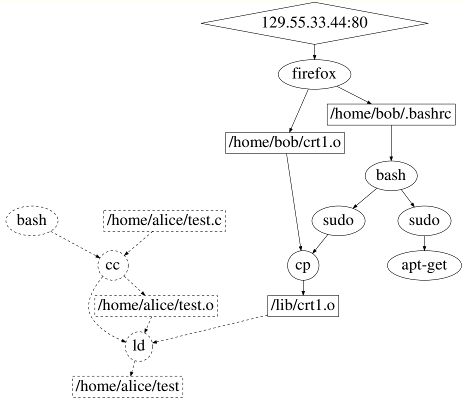
- **Attacker goal:** Insert backdoor into a vendor's software
- **Steps:**
  1. Use a browser vulnerability to drop a malicious version of `crt1.o` in `/home/bob`
  2. Modify Bob's `.bashrc` to redefine sudo

Illustrative Example



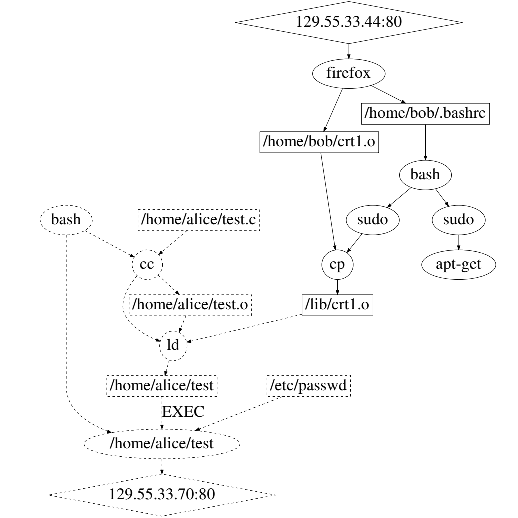
- **Attacker goal:** Insert backdoor into a vendor's software
- **Steps:**
  1. Use a browser vulnerability to drop a malicious version of `crt1.o` in `/home/bob`
  2. Modify Bob's `.bashrc` to redefine sudo
  3. Next time Bob uses sudo, it copies `/home/bob/crt1.o` to `/lib/crt1.o`

Illustrative Example



- **Attacker goal:** Insert backdoor into a vendor's software
- **Steps:**
  1. Use a browser vulnerability to drop a malicious version of `crt1.o` in `/home/bob`
  2. Modify Bob's `.bashrc` to redefine sudo
  3. Next time Bob uses sudo, it copies `/home/bob/crt1.o` to `/lib/crt1.o`
  4. When Alice builds her software, malicious `crt1.o` code is included in her executable.

Illustrative Example



- **Attacker goal:** Insert backdoor into a vendor's software
- **Steps:**
  1. Use a browser vulnerability to drop a malicious version of `crt1.o` in `/home/bob`
  2. Modify Bob's `.bashrc` to redefine sudo
  3. Next time Bob uses sudo, it copies `/home/bob/crt1.o` to `/lib/crt1.o`
  4. When Alice builds her software, malicious `crt1.o` code is included in her executable.
  5. When this software is run, it exfiltrates sensitive data (password file)

Adversarial Engagement Overview

Campaign	Length (hours)	# of events	Drop & load	Gather intel.	Insert backdoor	Escalate privilege	Data exfil.	Clean-up
Win-1	06:22	100K	✓	✓			✓	✓
Win-2	19:43	401K	✓	✓	✓		✓	✓
Lin-1	07:59	2.68M	✓	✓	✓		✓	✓
Lin-2	79:06	38.5M	✓	✓	✓	✓	✓	✓
Lin-3	79:05	19.3M	✓	✓	✓	✓	✓	✓
Bsd-1	08:17	701K			✓		✓	
Bsd-2	78:56	5.86M	✓	✓	✓		✓	
Bsd-3	79:04	5.68M	✓	✓			✓	

Attack Detection Using Provenance Tags

Provenance Tags

Trustworthiness (t-tag)

Benign authentic: Data from strongly *authenticated* sources believed to be *benign*.

Benign: Believed to be benign, but sources not well-authenticated.

Unknown: No good basis to trust this source.

Provenance Tags

Trustworthiness (t-tag)

Benign authentic: Data from strongly *authenticated* sources believed to be *benign*.

Benign: Believed to be benign, but sources not well-authenticated.

Unknown: No good basis to trust this source.

Code Vs Data Trustworthiness

- Processes have two t-tags: *code t-tag* and *data t-tag*
- Separation (a) aids detection and (b) speeds analysis by focusing on fewer root causes

Provenance Tags

Trustworthiness (t-tag)

Benign authentic: Data from strongly *authenticated* sources believed to be *benign*.

Benign: Believed to be benign, but sources not well-authenticated.

Unknown: No good basis to trust this source.

Code Vs Data Trustworthiness

- Processes have two t-tags: *code t-tag* and *data t-tag*
- Separation (a) aids detection and (b) speeds analysis by focusing on fewer root causes

Confidentiality (c-tag)

Secret: Highly sensitive, e.g., `/etc/shadow`

Sensitive: Disclosure has security impact, but less than disclosed secrets.

Private: Loss may not pose a direct security threat.

Public: Widely available, e.g., on public web sites

Attack Detection Using Provenance Tags

Approach: Focus on *motive* and *means*

Motive: Does an act advance attacker's high-level objectives?

- Deploy and run attacker code
- Replace/modify important files, e.g., `/etc/passwd`, ssh keys, ...
- Steal and exfiltrate sensitive data

Means: Can the attacker control the action?

- Is the process performing the action trustworthy?

Attack Detection Using Provenance Tags

Approach: Focus on *motive* and *means*

Motive: Does an act advance attacker's high-level objectives?

- Deploy and run attacker code
- Replace/modify important files, e.g., `/etc/passwd`, ssh keys, ...
- Steal and exfiltrate sensitive data

Means: Can the attacker control the action?

- Is the process performing the action trustworthy?

Benefits

- Application-independent
- No need for training
- Resists attacker manipulation (assuming provenance isn't compromised)

Attack Detection Policies

Untrusted exec (UE): Subject w/ high code trustworthiness execs lower t-tag object.

Suspicious modification (SM): Subject with lower code tag modifies higher t-tag file.

Data leak (DL): Untrusted subject writes confidential data to network.

Untrusted execution preparation (UP): Memory/file objects with low data trustworthiness made executable.

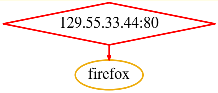
Attack Detection Policies

Untrusted exec (UE): Subject w/ high code trustworthiness execs lower t-tag object.

Suspicious modification (SM): Subject with lower code tag modifies higher t-tag file.

Data leak (DL): Untrusted subject writes confidential data to network.

Untrusted execution preparation (UP): Memory/file objects with low data trustworthiness made executable.



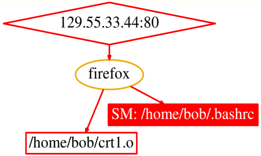
Attack Detection Policies

Untrusted exec (UE): Subject w/ high code trustworthiness execs lower t-tag object.

Suspicious modification (SM): Subject with lower code tag modifies higher t-tag file.

Data leak (DL): Untrusted subject writes confidential data to network.

Untrusted execution preparation (UP): Memory/file objects with low data trustworthiness made executable.



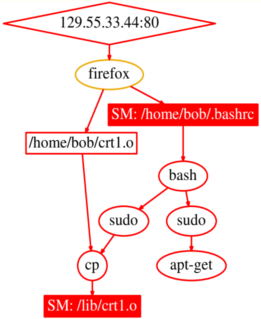
Attack Detection Policies

Untrusted exec (UE): Subject w/ high code trustworthiness execs lower t-tag object.

Suspicious modification (SM): Subject with lower code tag modifies higher t-tag file.

Data leak (DL): Untrusted subject writes confidential data to network.

Untrusted execution preparation (UP): Memory/file objects with low data trustworthiness made executable.



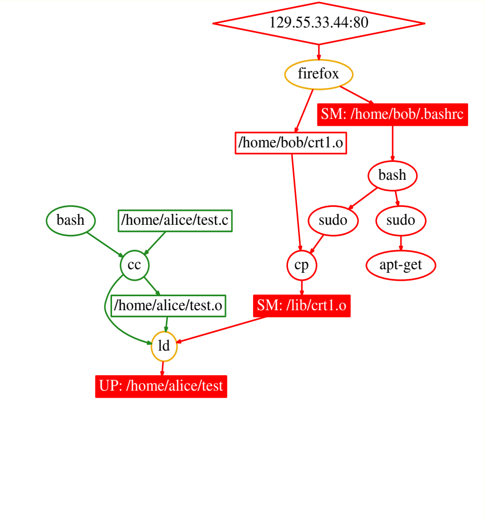
Attack Detection Policies

Untrusted exec (UE): Subject w/ high code trustworthiness execs lower t-tag object.

Suspicious modification (SM): Subject with lower code tag modifies higher t-tag file.

Data leak (DL): Untrusted subject writes confidential data to network.

Untrusted execution preparation (UP): Memory/file objects with low data trustworthiness made executable.



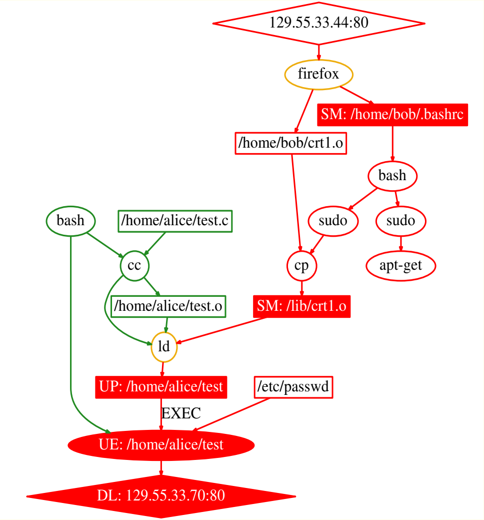
Attack Detection Policies

Untrusted exec (UE): Subject w/ high code trustworthiness execs lower t-tag object.

Suspicious modification (SM): Subject with lower code tag modifies higher t-tag file.

Data leak (DL): Untrusted subject writes confidential data to network.

Untrusted execution preparation (UP): Memory/file objects with low data trustworthiness made executable.



Flexible Policy Framework

- *Tag assignment* and *propagation* can be customized using policies.
  - Policies invoked at trigger points:
    - object creation, removal, read, write, load, execute, chmod, and chown
  - Can refer to subject, object and event attributes
- *Tag initialization* example:

$$init(o)\text{: } match(o.name, \texttt{"\^{}(file|IP:(10\textbackslash.0|127))"}) \rightarrow o.ttag = \text{BENIGN\_AUTH}$$

$$init(o) : match(o.name, \texttt{"\^{}IP:"}) \rightarrow o.ttag = \text{UNKNOWN}$$

- *Tag propagation*:
  - Default is to propagate tags from input to output
  - Custom policies created to capture exceptions, e.g., upgrade tag after a hash/signature verification.

Attack Detection Summary

Data Set	# of Events	Untrusted Execution	Suspicious Modification	Execution Preparation	Data Leak
Win-1	0.1M	3	3	0	11
Win-2	0.4M	2	108	0	18
Lin-2	39M	5	1	8	159
Lin-3	19M	5	2	0	5300

Key Point

- Almost zero false positives and negatives (except for data leak)
  - Typically filters out 99.99% to 99.9999% of events

Effectiveness of Split Trustworthiness Tags

Dataset	Untrusted Exec		Suspicious Modification		Untrusted Exec Prep		Data Leak	
	Single	Split	Single	Split	Single	Split s	Single	Split
Win-1	21	3	1.2 K	3	0	0	6.1 K	11
Win-2	44	2	3.7 K	108	0	0	20.2 K	18
Lin-1	60	2	53	5	1	1	19	6
Lin-2	1.5 K	5	19.5 K	1	280	8	122 K	159
Lin-3	695	5	26.1 K	2	270	0	62.1 K	5.3 K
Average Reduction		**45.39x**		**517x**		**6.24x**		**112x**

Key Point

- Without separating code and data tags, we will have 5x to 500x more alarms

False Positives in Benign Environment

- Untrusted execution (download+execute) plays a critical role in detection
  - What happens in an environment with legitimate software downloads?
- Experiment: Linux servers with automated security updates and manual upgrades
- Approach: Use custom policy to upgrade downloaded files before `apt-get` invokes dpkg
  - Note: `apt-get` verifies signatures, so this is safe.

Dataset	# of Events	Duration hh:mm:ss	Packages Updated	Binary Files Written
Server 1	2.17M	00:13:06	110	1.8K
Server 2	4.67M	105:08:22	4	4.2K
Server 3	20.9M	104:36:43	4	4.3K
Server 4	5.09M	119:13:29	4	4.3K

No (false) alarms reported.

Tag-Based Backward and Forward Analysis

Backward Analysis

Goal: Identify entry point of an attack.

- Entry point is a *source*, i.e., vertex with in-degree zero.

Starting points: Suspect vertices marked by attack detectors.

Problem: Find source vertices from which a suspect vertex is reachable.

Backward Analysis

Goal: Identify entry point of an attack.

- Entry point is a *source*, i.e., vertex with in-degree zero.

Starting points: Suspect vertices marked by attack detectors.

Problem: Find source vertices from which a suspect vertex is reachable.

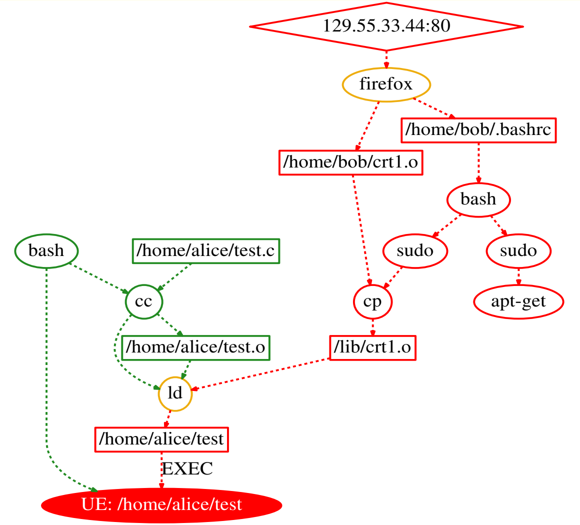
Complications:

Multiple sources: Suspect vertex is reachable from multiple sources.

Multiple suspect nodes: Typically, many detectors go off during attacks, and numerous vertices end up looking suspicious.

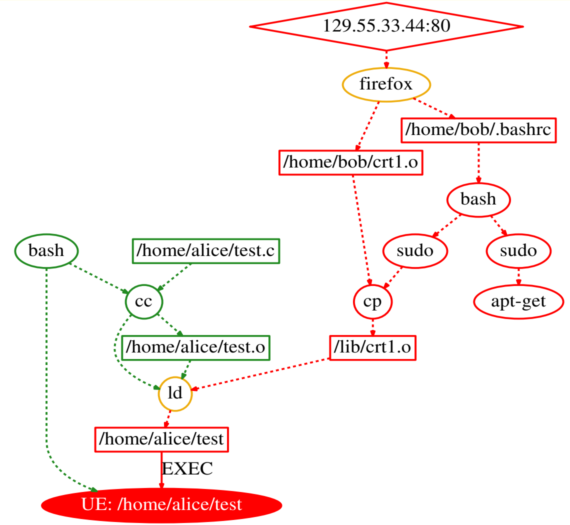
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



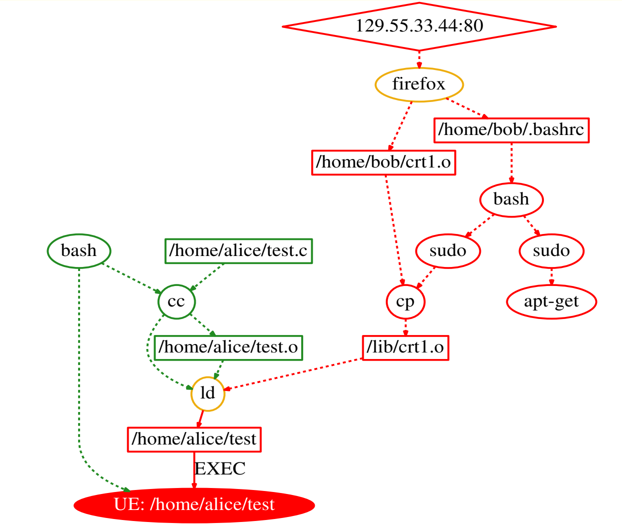
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



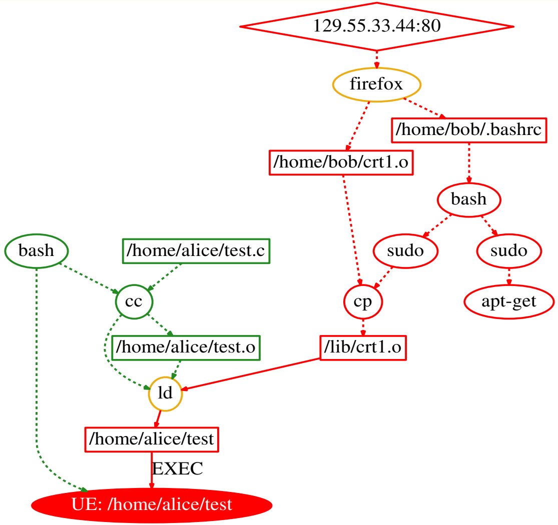
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



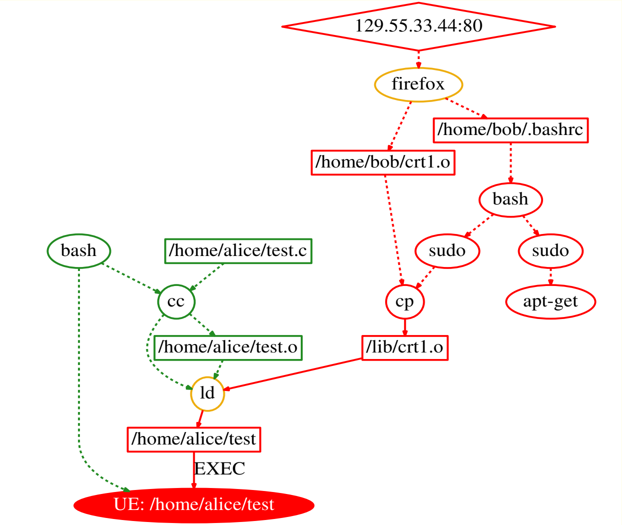
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



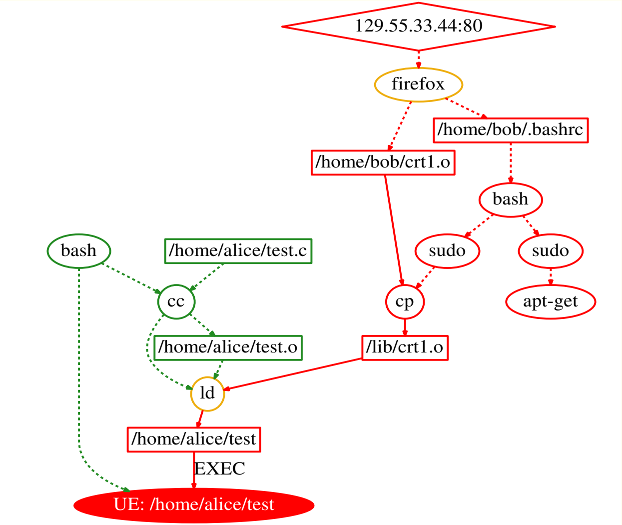
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



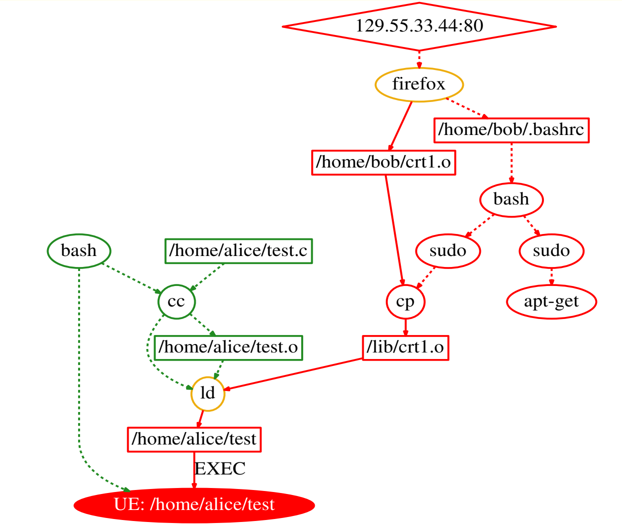
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



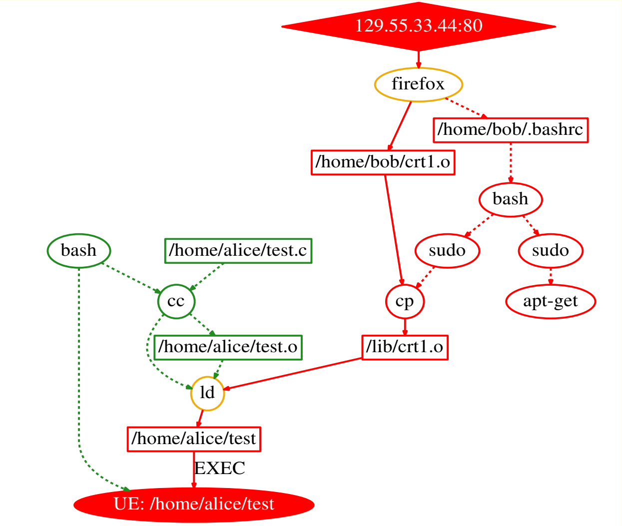
Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



Backward Analysis: Key Ideas

- Prefer shorter paths over longer ones
  - Favor paths that avoid redundant edges
- Prefer edges corresponding to flow of untrusted code
  - and, to a lesser extent, untrusted data
- Preference encoded using a custom edge-weight function to Dijkstra's shortest path algorithm



Forward Analysis

Goal: Identify attack impact, in terms of all objects/subjects affected by the attack.

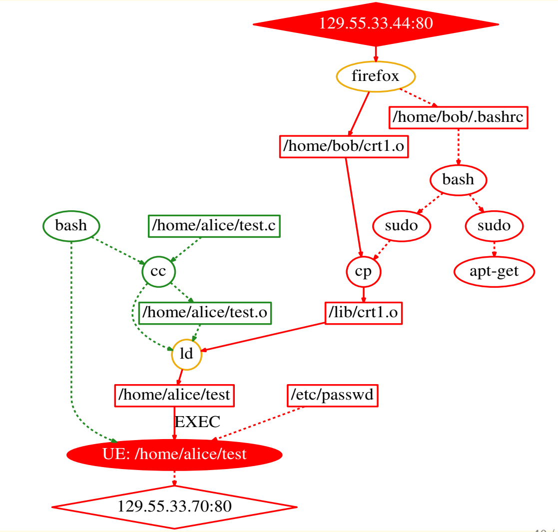
- Generate a subgraph of provenance graph that only includes objects and subjects affected by the attack.

Starting point: Sources identified by backward analysis

Challenge: Straight-forward dependence analysis may yield a graph with hundreds of thousands (if not millions) of edges.

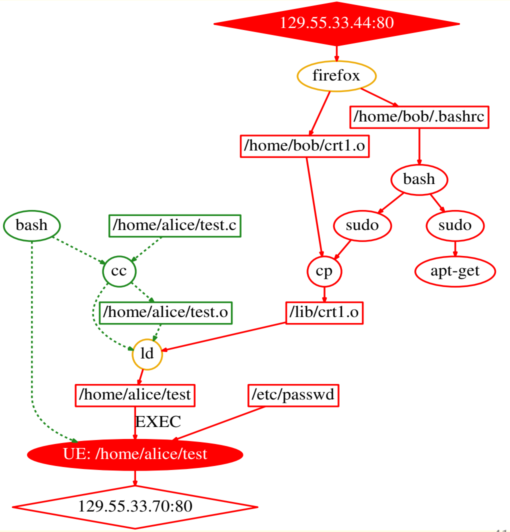
Forward Analysis: Key Ideas

- Use cost metric to prune off distant nodes, i.e., nodes at a distance $\geq d_{th}$



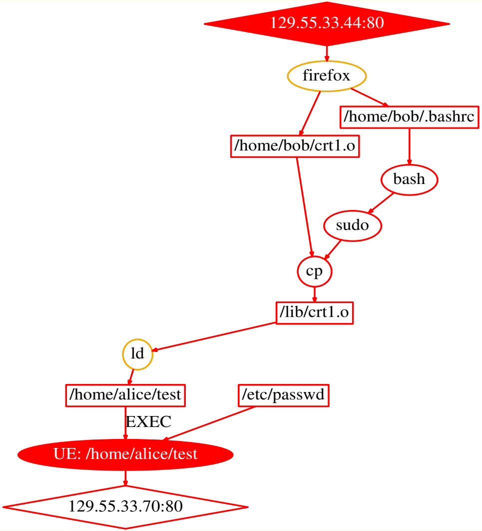
Forward Analysis: Key Ideas

- Use cost metric to prune off distant nodes, i.e., nodes at a distance $\geq d_{th}$
- Cost metric favors
  - edges with untrusted *code* trustworthiness (cost=0);
  - and, to a lesser extent, edges with untrusted *data* trustworthiness (cost=1)



Forward Analysis: Key Ideas

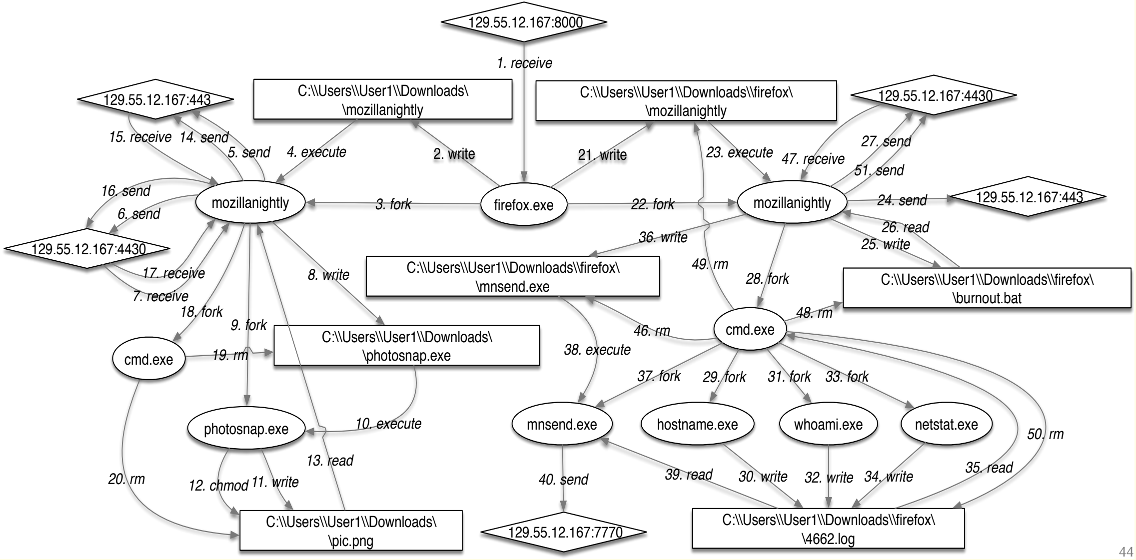
- Use cost metric to prune off distant nodes, i.e., nodes at a distance $\geq d_{th}$
- Cost metric favors
  - edges with untrusted *code* trustworthiness (cost=0);
  - and, to a lesser extent, edges with untrusted *data* trustworthiness (cost=1)
- Define simplifications on output
  - Prune nodes lacking "interesting" descendants
  - Merge "similar" entities
  - Remove repetitions



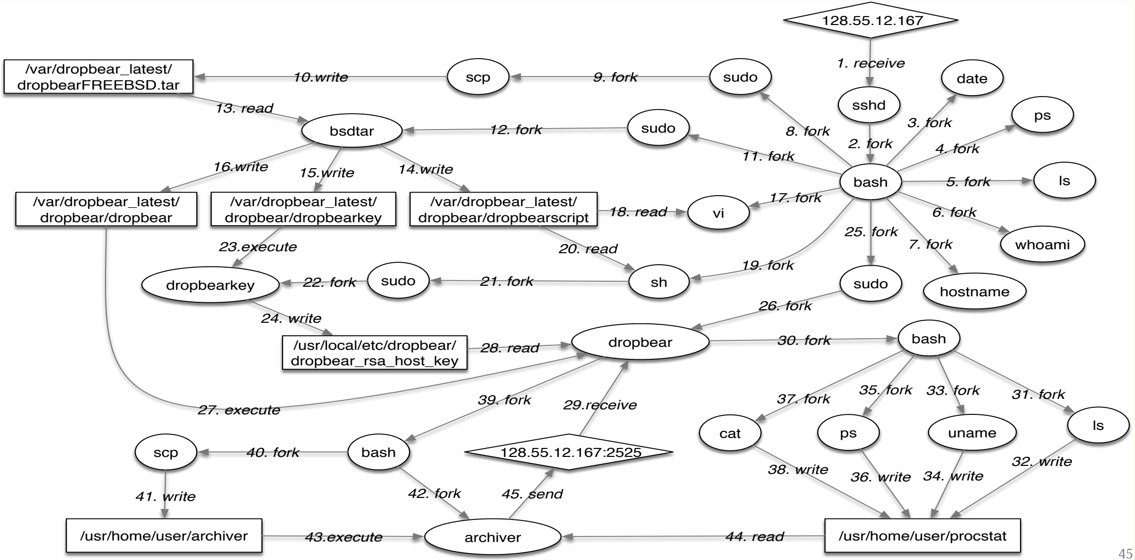
Campaign Reconstruction Summary

Campaign	Entry Points	Programs Executed	Key Files Involved	Exit Points	Correctly Identified	False Positives	Missed Entities
Win-1	2	8	7	3	20	0	0
Win-2	2	8	4	4	18	0	0
Lin-1	2	10	6	2	20	0	0
Lin-2	2	20	11	4	37	0	0
Lin-3	1	6	6	5	18	0	0
Bsd-1	4	13	9	2	13	15	1
Bsd-2	2	10	7	3	22	0	0
Bsd-3	4	14	7	1	26	0	0
Total	**19**	**89**	**57**	**24**	**174**	**15**	**1**

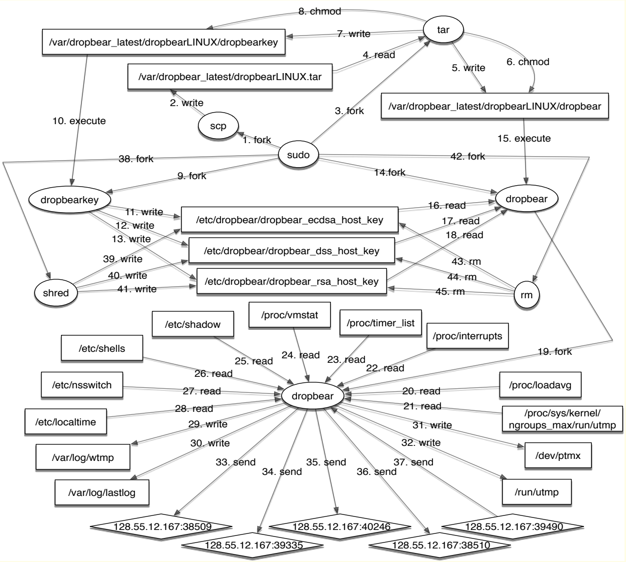
Generated Graph for Scenario Win-1



Generated Graph for Scenario Bsd-3



Generated Graph for Scenario Lin-2



Forward Analysis Selectivity

Campaign	Initial # of Events	Final # of Events	Reduction Factor			
			Single t-tag	Split t-tag	SLEUTH Simplif.	Total
Win-1	100 K	51	4.4x	1394x	1.4x	1951x
Win-2	401 K	28	3.6x	552x	26x	14352x
Lin-2	38.5 M	130	7.3x	2971x	100x	297100x
Lin-3	19.3 M	45	7.6x	1208x	356x	430048x
Bsd-2	5.86 M	39	1.9x	689x	218x	150202x
Bsd-3	5.68 M	45	6.7x	740x	170x	125800x
Average Reduction			**4.68x**	**1305x**	**41.8x**	**54517x**

Memory Use and Runtime Performance

Campaign	Events	Memory Usage	Bytes/ event	Duration (hh:mm:ss)	Runtime	
					Time	Speed-up
Win-1	100K	3 MB	30	06:22:42	1.19 s	19.3 K
Win-2	400K	10 MB	25	19:43:46	2.13 s	33.3 K
Win-Mean			**28**			**26.3 K**
Lin-1	2.7M	26 MB	10	07:59:26	8.71 s	3.3 K
Lin-2	38.5M	329 MB	9	79:06:39	114.14s	2.5 K
Lin-3	19.3M	175 MB	9	79:05:13	74.14 s	3.9 K
Lin-Mean			**9**			**3.2 K**

Related Work

Intrusion detection: Numerous anomaly & misuse detection techniques developed since 80s.

- **Sleuth** advances: novel use of provenance and policies to obtain *application-independent, training-free* detection with very low false positive rate.

Alert correlation: Link alarms using statistical [Qin03], graph-based clustering [Wang08, Pei16], attack specifications [Ning03], and so on

- In contrast, **Sleuth** uses provenance tags and policies to obtain accurate, analyst-friendly scenario descriptions

"Backtracking Intrusions:" Backtracker, Taser, Forensix, ...

- Target forensic analysis, assisted by external detectors.
- **Sleuth** targets fully automated, real-time scenario construction with built-in detectors

Tackling dependence explosion: Orthogonal to (and can benefit) **Sleuth**.

- Fine-grained taint-tracking
- Forensics-targeted: BEEP, ProTracer, ...

Information flow control: [Biba, Bell-LaPadula, PIP, SPIF, ...]

- Goal is to block illegal flow, while minimizing failures.
- In contrast, **Sleuth** needs to distinguish attacks from benign policy violations.

Summary

- Presented techniques that a security analyst can use to understand an ongoing attack campaign, and respond in real-time.

> Automatically generated visual representation that compactly summarizes an ongoing campaign

- Experiments show high accuracy and performance for **Sleuth**
- Effectiveness evaluated using realistic adversarial engagements.

> Key point: Given millions of events in an unknown environment, **Sleuth** consistently managed to be spot-on, zooming in on the 0.01% or less of the events actually involved in attacks.