AFRL-AFOSR-UK-TR-2020-0001

Foundations of Language-Based Provenance Security

James Cheney
UNIVERSITY OF EDINBURGH (THE)

11/27/2018
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ IOE
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
22-04-2020	Final	01 Jan 2013 to 30 Jun 2018

4. TITLE AND SUBTITLE
Foundations of Language-Based Provenance Security

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA8655-13-1-3006

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)
James Cheney

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
UNIVERSITY OF EDINBURGH (THE)
OLD COLLEGE, SOUTH BRIDGE
EDINBURGH, MIDLOTHIAN, EH8 9YL GB

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
EOARD
Unit 4515
APO AE 09421-4515

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/AFOSR IOE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-AFOSR-UK-TR-2020-0001

12. DISTRIBUTION/AVAILABILITY STATEMENT
A DISTRIBUTION UNLIMITED: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this project, the PIs focused on defining and studying provenance and information flow, dependency-tracking and program slicing. They made great progress on understanding the formal requirements of provenance security and on ways to analyze existing systems to understand their behavior. The project ran from 2013 to 2018 and supported three postdoctoral researchers and three PhD students. This grant resulted in four workshop or position papers, nine papers in competitive peer-reviewed conferences, and three journal articles. The project has also contributed synergistically to other funded projects, including participation in ADAPT, part of the DARPA Transparent Computing program, a small grant from the UK Research Institute on Verified Trustworthy Software Systems on formalizing database query languages, and an European Research Council (ERC) grant on programming language design for data curation.

15. SUBJECT TERMS
EOARD, Information provenance, Cyber security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			MAILLOUX, LOGAN
Unclassified	Unclassified	Unclassified	SAR		19b. TELEPHONE NUMBER *(Include area code)* 314 235 6163

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Foundations of Language-Based Provenance Security
Final Report

Period of performance: January 2013–June 2018
Grant number: FA8655-13-1-3006

PI: Dr. James Cheney
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh
jcheney@inf.ed.ac.uk

November 27, 2018

Abstract

Provenance is information about the origin, history or derivation of something, which could be a physical object (such as a work of art), information (such as a Word document containing security-critical information), or a combination of the two (such as a computer system with both physical and informational characteristics). Provenance tracking has been identified as both an opportunity and a challenge for security: it offers the opportunity of increased awareness of actionable information about information quality and trustworthiness, while also introducing new risks such as unintended consequences of pervasive system monitoring. Although provenance has been studied in other settings, such as computational science, foundational research on provenance has not yet addressed key questions needed in security-critical settings.

In this project, a range of foundational and practical aspects of provenance security were studied, organized around the following four themes: (1) models and semantics of provenance, (2) expressing provenance security policies and properties, (3) language integration and efficiency and (4) verification. This project adopted a language-based approach to provenance, seeking to understand provenance and its relationship to other concepts such as program slicing, bidirectional transformations, and concurrency studied by the programming languages community. The main results were new techniques for program slicing, auditing and provenance inspection, benchmarking techniques for provenance-tracking systems, efficient language-based support for provenance and view updates, and the first formally verified results about languages with provenance-tracking.

Keywords provenance, accountability, transparency, explanation, program slicing, audited computation

1 Introduction

Whenever we make decisions, we rely on information that may be of high or low quality: it may be recent, it may be out of date; it may be accurate or approximate; it may have been provided by objective, disinterested sources or by sources with self-interest that incentivizes them to conceal or deceive. Traditionally, people have used a number of heuristics to assess the quality of the information they receive, such as the origin, authorship, or production process: metadata that are often summarized by the term *provenance*. These heuristics have traditionally been supported by gatekeeping processes such as scientific peer review, journalism standards, and trust in government and other authority figures. As information and decision-making increasingly move online, the different characteristics of electronic information systems compared to traditional ones mean that the attributes people have historically used to assess information, and the gatekeeping processes that make them work, are no longer effective. Reaping the benefits of modern information systems, while avoiding pitfalls due to widespread misinformation or even lack of trust in sources of ostensibly objective information, is an emerging critical need in many domains, from science to public policy and governance to national security.

These needs became apparent in the scientific community in the late 1990s, for example in bioinformatics where progress has been driven by large-scale accumulation and analysis of data. The relevance of provenance to security was recognized around 2007 [19, 8] and has since developed into a significant branch of systems security (see e.g. [31, 5, 22, 18]). Unfortunately, it is not well understood how mechanisms such as provenance tracking actually support intended policies of increased trust, accountability, or transparency. In fact, to date we lack clear policies describing the aims we wish to achieve, and without such policies it is difficult to assess proposed provenance mechanisms, understand their strengths and weaknesses, and improve them. Conversely, it has also been observed that a number of proposed approaches to provenance, such as pervasive monitoring of causal activity at the operating system level, may have other compelling applications to security, such as detection and mitigation of Advanced Persistent Threats (APTs), as in DARPA's $60m Transparent Computing program.

I began studying foundations for provenance security, with an emphasis on language-based foundations, in 2010, leading to a CSF 2011 paper [10], and I was subsequently encouraged to apply for AFOSR EOARD funding on this topic. This project is the result.

In this report, I will summarize the activities performed thanks to AFOSR EOARD funding, as well as concurrent activities that have mutually reinforced and benefited from this project. The project has run from 2013 to 2018 and (at different times) supported three postdoctoral researchers and three PhD students, and has resulted in four workshop or position papers [11, 14, 24, 9], nine papers in competitive peer-reviewed conferences [13, 26, 28, 30, 27, 16, 29, 20, 21], and three journal articles [2, 3, 25]. The project has also contributed synergistically to other funded projects, including participation in ADAPT, part of the DARPA Transparent Computing program, a small grant from the UK Research Institute on Verified Trustworthy Software Systems on formalizing database query languages, and an European Research Council (ERC) grant on programming language design for data curation.

2 Personnel

2.1 PhD students

Herry Herry was supervised by a colleague Paul Anderson, and was supported for a few months by this project. This support contributed to his thesis completion and a journal article on semantics of configuration languages [3].

Weili Fu has been jointly supervised with Anderson, and was also supported for a few months by this project. Her research is on semantics and provenance for configuration languages which are widely used to manage critical systems. One paper has been published on the semantics of the Puppet configuration language [16] and her thesis and an additional publication are in progress.

Sheung Chi (Arthur) Chan has been supported by the AFOSR funding throughout his PhD. His research has been on benchmarking provenance-tracking systems in order to assess their completeness, correctness, and suitability for security purposes [9]; his thesis and additional publications are in preparation.

2.2 Postdoctoral researchers

Dr. Roly Perera has worked on surveying provenance security mechanisms [14], and formalizing causal equivalence and slicing for the pi-calculus [24, 26, 25]. He has also contributed to research on Puppet semantics [16], foundations of provenance [2, 11], slicing imperative functional programs [30], and language-based approaches to view update in databases [20]. He has been supported full-time and part-time at different points and has now transitioned to a full-time research position in Glasgow.

Dr. Wilmer Ricciotti has worked on foundations of provenance tracking and auditing [28, 29] including a single-author paper on provenance inspection [27], and led work on slicing for imperative functional programs [30]. He has subsequently been awarded funding (joint with the PI) on formalizing database query languages. He was supported by the project from September 2015 until October 2017.

Dr. James McKinna has been supported part-time for six months in 2018. His research related to the project focused on language extensibility, which has applications to instrumenting programs to track their own provenance, and the resulting paper [21] will appear at POPL 2019, the top conference on foundations of programming languages.

3 Contributions

The project proposal outlined an open-ended program of research to investigate the following research hypothesis: *formal and general language-based techniques can provide a solid foundation for secure, end-to-end provenance in realistic applications*. The plan of research was organized around the following themes:

- How can we design programming languages and systems to facilitate end-to-end support for provenance?
- What languages or logics can be used to specify and analyze provenance-aware security policies?
- How can we provide efficient mechanisms for securing provenance?
- What aspects of provenance and its security can be captured by objectively verifiable criteria, and how can we formalize and verify them?

This plan was intended to allow for flexibility due to the foundational nature of the research and the need to tailor projects to the skills of personnel available to work on the project. It was not expected that all four themes would be fully explored. In the following subsections, I will summarize the original vision and research questions of each theme, briefly describe the main contributions of the corresponding research papers, and highlight how these contributions have advanced knowledge and what questions remain unaddressed or what new questions arise.

3.1 Models and semantics of provenance

The goal of this theme was to investigate the question "How can we design programming languages and systems to facilitate end-to-end support for provenance?" Because most of the other themes depend on answers to this question, much of the project's work focused on this area.

The starting point for work on this theme was initial publications by the PI and colleagues on connections between provenance and already-studied topics such as data dependence, information flow, and slicing [12, 1, 23]. Cheney et al. [12] had identified a form of provenance for database queries inspired by dependency tracking and information flow in programming languages. Subsequently Perera et al. [23] developed a form of *program slicing* (an approach to debugging and explanation) based on a rich form of traces that could also serve as a form of provenance, and Acar et al. [1] further explored the connections between trace-based slicing, security, and provenance.

A key contribution of Perera et al. [23], which plays an important role in later work on this project, was the observation that slicing can be formulated as a pair of functions between partial inputs and partial outputs of a program: a *forward* evaluation function that evaluates "as much as possible" of the result given an input where some parts are marked as "missing", and a *backward* slicing function that identifies "as little as necessary" of the input that suffices to recompute a part of the output. As a very simplistic example, consider a "program" $f(x, y) = (x + 1, x + y)$, applied for example to $(1, 2)$ to yield $(2, 3)$. The forward evaluation of $(1, \Box)$ would yield $(2, \Box)$ where $\Box$ stands for a missing part of the input or output; conversely, if we wanted to backward slice to explain the output $(\Box, 3)$ then the resulting part of the input would be $(1, 2)$, since both x and y are involved in computing the second part of the result.

Moreover, we can identify pairs of forward and backward functions as being particularly well-behaved when they form a *Galois connection*, that is, when $y \leq fwd(bwd(y))$ and $bwd(fwd(x)) \leq x$. Another important insight of Perera et al. was that given a reasonably well-behaved forward evaluation (in a sense made precise in the paper), there is a unique, computable backward slicing function that forms a Galois connection. However, actually evaluating the backward slicing function can be very expensive, and traces and trace-based slicing were introduced as a way to make backward slicing more practical.

Explanation and slicing In a position paper [11], we presented a vision of a theory of "self-explaining computation", that is, programming languages (or other computer systems) that explain their own behavior. We highlighted connections to topics such as information flow in language-based security and slicing in software engineering, as well as prior work on provenance in databases. We also outlined a framework for different kinds of explanation based

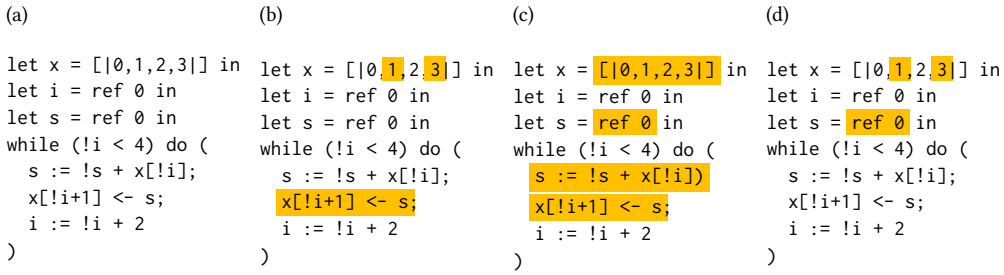

Figure 1: Slicing of imperative programs with arrays. (a) The program we wish to slice, which initializes x to a array, i and s to mutable references, and then does a while-loop that modifies the array elements along with s and i. (b) A slice that explains the value of s at the end of the loop. Yellow highlighting identifies parts of the program that can be ignored if we want to understand how s was computed. (c) A slice that explains the value of i at the end of the loop. Here, updates to s and x, and their initial values, are irrelevant. (d) A slice that explains the value of $x[3]$ at the end of the loop. Here, only the initial values of s and $x[1]$ and $x[3]$ are irrelevant.

on analysis of the semantics of programming languages, and illustrated this framework using a simple imperative language.

In a subsequent paper [13], we investigated self-explaining database queries, building on previous work on slicing for functional programs [23]. In contrast to previous work on provenance in databases, which focus on restricted query languages for which simple forms of explanation suffice, this work emphasized forms of explanation that provide greater specificity about how each part of a query result was constructed from the input, in the sense that we can use the explanation to replay the relevant computational steps exactly. However, such detailed explanations are potentially very expensive to compute. Subsequent research (see below) has begun to investigate how to make this approach realistic.

Perera et al. [26] also extended the slicing framework in a different direction to accommodate concurrency. In this work, one of the central issues is that in a concurrent setting, the appropriate correctness criteria for slices are not as clear. In a sequential setting, we can specify correct slices by saying that rerunning the (sliced) program on a (sliced) input should produce (at least) the desired part of the output. In a concurrent setting, there is no guarantee that rerunning would follow the same execution path, so it is necessary to make the forward and backward functions trace-dependent. This leads to a large number of subtle issues which were resolved by developing the main results of this paper formally in Agda (further discussed below).

Ricciotti et al. [30] revisited the original approach to slicing (originally considered only for pure functional programs [23]), and extended it to a setting with higher-order functions, references (mutable state) and exceptions (nonlocal control flow). In this much more challenging setting we again found it necessary to make the definition of correct slices depend on the execution path taken in the original program, although in contrast to the concurrent setting the programs we considered are still deterministic. As a result, we were able to define slicing and explanation techniques for programs combining imperative features such as references with higher-order functional programming, exceptions, and even arrays. Figure 1 illustrates how our approach to slicing works on some simple array-based functional programs.

Audited computation and provenance inspection The line of work on explanation and slicing discussed above (like most work on provenance generally) adopts the perspective that we have an ordinary program and wish the explanation mechanism to give us some extra information explaining how the program executed (perhaps specialized to some part of the input or output). That is, there is a separation between the (standard) program being investigated and a framing layer that investigates the program. In concurrent work, others such as Bavera and Bonelli [6] had begun to study languages related to *justification logic* that have both a notion of run-time trail (i.e. trace) recording and *trail inspection* allowing run-time access to the trail.

We found it irresistible to try to understand this model and adapt it to handle provenance or slicing but found the details of [6] quite complicated and heavyweight; in addition, they had left some important open questions regarding the termination (strong normalization) of their language in the presence of trail inspection. Ricciotti and Cheney [28] presented a simplified and improved version of their system and settled the open question. We built on this system in

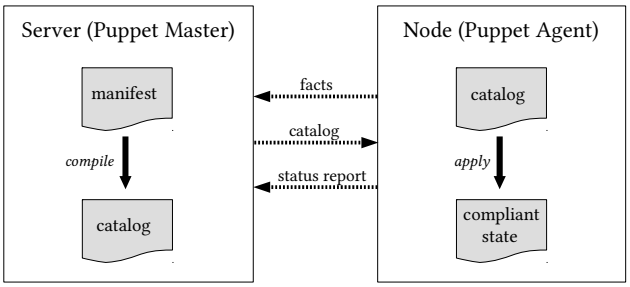

Figure 2: Puppet configuration system architecture. A server ("puppet master") manages a number of nodes ("puppet agents") using a *manifest*, that is, a collection of configuration files specifying what software packages, services and other components should be installed and running on each node. Periodically each node contacts the server and provides *facts* that describe it, such as the hostname or other data. The server *compiles* the manifest using the facts into a *catalog* that declaratively describes the specific configuration state the node should have. The Puppet agent software on the node compares the new catalog with the previous configuration and *applies* it to reach a new (hopefully) compliant state. Finally, the node reports back to the server with a status report, indicating success or any problems.

subsequent work on explicit auditing [29], discussed below.

We subsequently studied the problem of how to extract other forms of provenance from Bavera and Bonelli-style trails, but found that (in contrast to the traces of [2]) it is less clear how to extract familiar forms of provenance from their trails. The key difficulty is dealing with function bodies (code) with variable binding: when we want to find the source of a data item in the return value of a function call, we need to be able to analyze the function call body and this is nontrivial since in most languages the code of a function is not inspectable at run-time. Ricciotti [27] adopted a similar programming language-based approach to inspecting and extracting provenance; like Bavera and Bonelli's system, Ricciotti's system makes provenance inspection part of the language rather than an a extralinguistic feature, but provides the features missing from Bavera and Bonelli's trails for extracting familiar provenance forms.

Configuration languages A prerequisite to studying provenance formally (in any system) is to define the semantics of the system. *Configuration systems* such as SmartFrog and Puppet as important examples of systems where provenance would be beneficial. Configuration systems are used to configure large datacenters, often using high-level *configuration languages*. Figure 2 illustrates the architecture of the Puppet configuration system. Errors in such configurations can lead to expensive failures which are difficult to debug. Anderson and Herry [3] developed a semantics for SmartFrog, a configuration language introduced by HP; to our knowledge this is the first example of a formal semantics for such a language. Fu et al. [16] developed a semantics for Puppet, one of the most commercially successful configuration languages. This semantics is providing a basis for studying provenance-tracking for Puppet, which is intended to provide a rigorous basis for understanding the sources of errors in Puppet configurations after a failure, or even detecting such errors before they cause damage. In addition to Herry and Fu's thesis research, this work has led to subsequent collaboration with colleagues Anderson and Vaniea in Edinburgh on the design and usability of configuration languages (with an emphasis on security).

Summary The contributions listed above have led to significant improvements of our understanding of provenance and related techniques such as information flow and slicing. However, it still appears to be an open problem to define a well-founded provenance tracking system for a general-purpose programming language, including features such as objects/classes or modules/interfaces or concurrency. Additional work could also consider incorporating first-class provenance inspection or reflection on provenance traces. Moreover, the work on semantics and provenance for configuration languages has raised a number of interesting questions which we hope to explore in future work.

3.2 Expressing provenance security policies and properties

The goal of this theme was to investigate the question "What languages or logics can be used to specify and analyze provenance-aware security policies?" This question was intended to build on and consolidate the research direction initiated by my paper [10], which proposed a framework for foundations of provenance security including aspects such as "disclosure" (ensuring that all necessary information is present in a provenance view of the system behavior) and

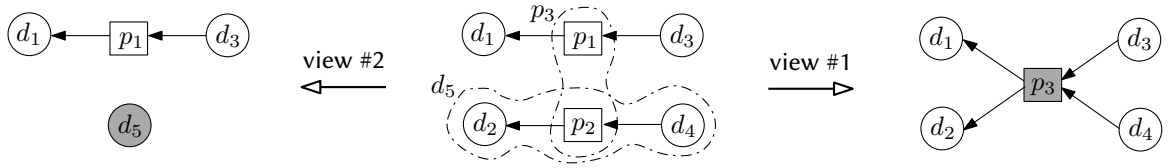

Figure 3: Provenance views. The graph in the middle is a small provenance graph including two concurrent processes p_1, p_2 that independently read from d_1, d_2 and generate d_3, d_4 respectively. The dotted lines indicate two possible ways of abstracting this graph leading to different views. In the first view (to the right), the two processes are merged into one abstract process node p_3, thus losing information about how d_3 and d_4 were derived. Alternatively, in the second view (to the left), the three nodes d_2, p_2, d_4 are collapsed into one abstract object node d_5, hiding the process and state change. Provenance sanitization and abstraction techniques surveyed by Cheney and Perera [14] employ a number of different kinds of provenance views with different expressiveness and guarantees.

"obfuscation" (ensuring that unnecessary and/or sensitive information is not deducible from such a view). Since these topics were not a speciality of staff or students recruited to the project, progress on them was more limited.

Acar et al. [2] built on [10] and [23] to study provenance traces and slicing concepts from the point of view of disclosure and obfuscation. An important contribution was identifying *positive* and *negative* refinements of disclosure and obfuscation, reflecting that the original symmetric versions of the proposed properties were too inflexible. Furthermore, several forms of provenance studied in databases were adapted to a programming language setting and their correctness and security properties were studied.

Cheney and Perera [14] surveyed the available proposals for *provenance sanitization*, that is, the problem of hiding irrelevant information in provenance graphs (for example, to ensure obfuscation of sensitive information). Figure 3 illustrates one of the key concepts discussed in the survey, *provenance views*, which are used in a number of research papers on provenance sanitization in different forms. Some of the available proposals were developed with usability rather than security goals in mind, while others were developed with specific security goals or policies (in some cases, with provable guarantees). Nevertheless, the survey concluded that there were few techniques based on formal semantics or rigorous policies.

Chan et al. [9] proposed *provenance expressiveness benchmarking* for OS-level provenance tracking systems such as those being used in the Transparent Computing program. We had observed (with colleagues at SRI, Cambridge, and elsewhere) that different systems varied widely in terms of the detail, structure, and completeness of the provenance they record. This paper presented the results of a manual analysis of around forty system calls on two different provenance-tracking systems, SPADE [18] and OPUS [4], and the results identified some significant differences between the two systems. In subsequent work, Chan has developed ProvMark[1], a system that automates the manual analysis performed in the workshop paper, reducing the underlying graph matching problems to a logical representation that can be solved by a logic programming-based solver. Figure 4 illustrates the architecture of ProvMark. This work is the core of Chan's PhD thesis and the thesis and a related conference paper are in preparation.

Summary The work envisioned for this theme had a more foundational and logical flavor. However, the mix of skills of people recruited to work on the project turned out not to fit this theme very well. Initial results on provenance security did provide raw material for other efforts, however. The work that fits this theme best has been that on benchmarking the expressiveness of provenance systems, which turns out to hinge on representing provenance graphs using logical formulas and using a logic-based solver to compare such graphs. Many of the questions of interest in this theme, such as identifying security properties for provenance and appropriate query languages, remain open.

3.3 Language integration and efficiency

The goal of this stream was to investigate the question "How can we provide efficient mechanisms for securing provenance?" Progress on this question was somewhat limited, due to the relatively strong dependency on challenging,

[1] `https://github.com/arthurscchan/ProvMark`

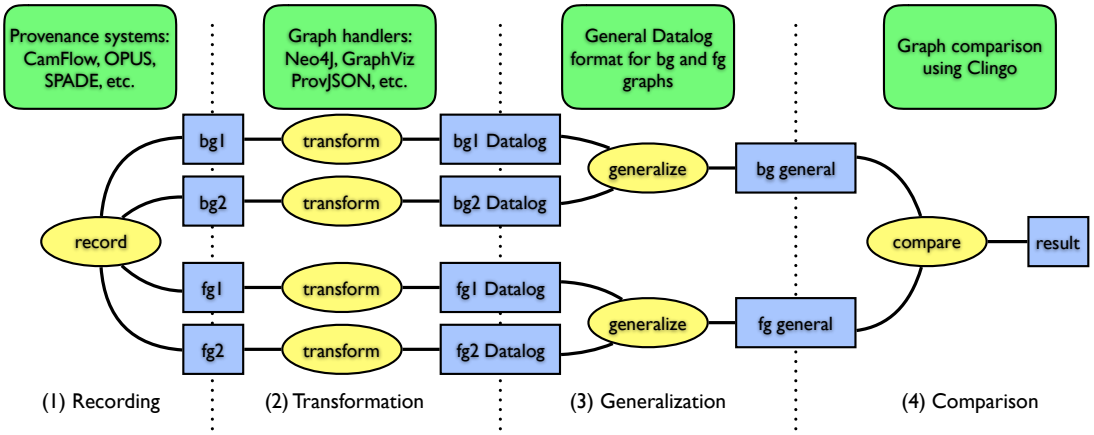

Figure 4: ProvMark architecture. Given a (small) target activity, we first record provenance graphs that include (foreground) or exclude (background) that activity using one of the provenance recording systems considered, such as SPADE, OPUS or CamFlow. We next transform the recorded graph from the native format used by the recording system to a uniform, logic-based representation called Datalog. Next, we compare similar runs that contain (or omit) the target activity to identify transient or volatile attributes such as timestamps or process IDs, and remove them, to abstract these runs into *generalized* graphs. Finally, we compare the foreground and background generalized graphs to compute the *difference* between the graphs, which is the part of the graph corresponding to the target activity. This is nontrivial because the background graph may include 10–20 nodes and edges representing process startup, reading library files and so on, and matching this subgraph to the foreground graph requires solving NP-complete *subgraph isomorphism* problems. For this purpose we use a logic programming-based solver, Clingo [17].

foundational issues mentioned earlier.

Another student, Stefan Fehrenbach (supported by complementary funding to this project), has focused on developing language-integrated support for provenance in Links, a Web and database programming language project in Edinburgh. This work has shown that existing forms of provenance for databases can be implemented efficiently at the language level [15]. His subsequent thesis research (in progress) is focusing on generalizing his initial results using ideas from Cheney et al. [13] to allow for arbitrary forms of provenance for queries through trace analysis (analogous to the trail or provenance inspection of [28, 27]).

Ricciotti and Cheney [29] extended the work on justification logic and trail inspection by introducing an *abstract machine*. To avoid the expensive trail construction operations implicit in Bavera and Bonelli's initial system [6] (and in our system [28]) we introduced a notion of *explicit auditing*, which is analogous to the notion of *explicit substitution* used in efficient abstract machines for functional programming languages. In explicit auditing, we introduce new language constructs that signpost where auditing needs to take place, but we construct trails lazily until a trail inspection takes place. In particular, if a trail inspection does not take place, then we save a large amount of unnecessary effort. The main focus of this paper was to present the design and an initial, proof-of-concept implementation; we believe the same idea would apply to other provenance-tracking or slicing settings based on traces.

Horn et al. [20] implemented an incremental, language-based approach to *view updates* over relational tables, implemented in Links. View updates are related to provenance; in particular, often the goal of provenance tracking is to facilitate correcting wrong source data in a database, and view updates automate this correction process. In this paper, we showed how to implement an existing theoretical model of view updates [7] efficiently by incrementalizing the underlying relational operations and translating small changes ("deltas") to a view to small changes to the source databases. Figure 5 illustrates the high-level idea of relational lenses and how deltas are propagated backwards through a lens from the view to the source.

Finally, Morris and McKinna [21] have developed a new theory of extensible programming based on *rows*. Rows are a programming feature that generalize the idea of key-value pairs in types, such as C `structs` and `unions`, records and datatypes in functional languages, or relational schemas in databases. Extensible row-based programming techniques would be extremely helpful for implementing provenance tracking in a maintainable way, in contrast to the conventional approach of modifying an interpreter or database to support provenance in ad hoc ways.

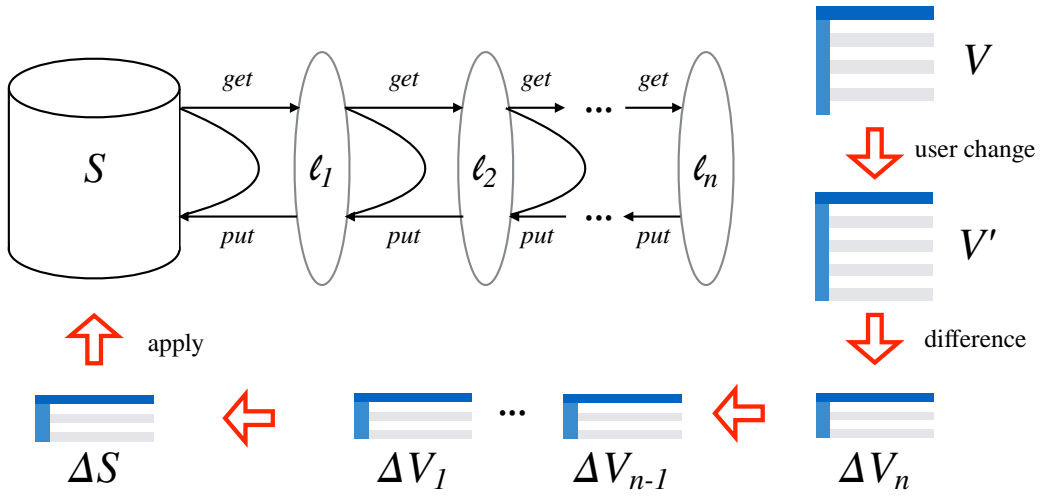

Figure 5: Incremental relational lenses. This figure illustrates a relational lens composed of a series of individual lens combinators, representing selection, projection or join operations on the source database. The lens can be executed in the forward direction to construct a *view* V of the data, typically corresponding to the data a user might edit on a Web page or through some other application. When the user wants to change the data to a new form V', we calculate the difference between V and V' as a *delta* ΔV_n that indicates what rows are added and what rows are removed from V to form V'. Using the incremental versions of the lens primitives we introduced [20], we can translate each ΔV_i to ΔV_{i-1} until we obtain a delta ΔS to the source tables S, which we can then apply to S. In common cases, our experiments show this can be dramatically more efficient (close to linear time in the size of the change compared to linear or quadratic time in the size of the database).

Summary Some progress was made on this theme in the project, but the efficiency and language-integration of provenance techniques still leave much to be desired. The work on language-integrated provenance for databases, and that on support for provenance inspection in efficient abstract machines, suggests a natural next step: developing a single language that supports provenance-tracking in both ordinary code and integrated queries, with provenance tracking for ordinary code supported by the abstract machine (as in [29]) and in queries supported by query translation (as in [15]). However, there are clearly a number of nontrivial problems that would need to be overcome to make this idea practical. Better language support for extensibility and generic programming (as envisaged by Morris and McKinna [21]) could contribute to this problem.

3.4 Verification

The goal of this theme was to investigate the question "What aspects of provenance and its security can be captured by objectively verifiable criteria, and how can we formalize and verify them?" Formalizing security properties of systems, and verifying them in a theorem prover, are recognized as providing the highest possible degree of confidence in these results; however, such formalizations are very challenging. At the time of the initial proposal, there had been a few formalizations of information-flow security techniques, but there were no previous formalizations of provenance tracking, or of related techniques such as slicing.

Perera and Cheney [24] developed a new formalization of the π-calculus, a core calculus for concurrent and mobile computation. Although there have been a number of formalizations of this calculus, our formalization was distinctive in focusing on properties such as *causal equivalence* of sequences of transitions. The classical approach to the semantics of π-calculus breaks down execution into a sequence of discrete steps, but this introduces spurious distinctions between sequences that represent "the same" computation but where causally independent steps are in different orders. We adapted a technique called *permutation of transitions* from previous work on simpler models of concurrency to the π-calculus, and developed and proved key properties of causal equivalence in this model. We believe this is the first work to adapt permutation of transitions to the π-calculus. This work initially appeared in a workshop [24] and was subsequently invited to appear in a special issue of a journal on the best papers from that workshop [25]. The results are formalized in Agda, a dependently-typed programming language that has become a

popular tool for formalizations[2].

Perera et al. [26] built on the aforementioned results to develop a theory of *causally consistent slicing* for the π-calculus. This work has already been discussed above; it was also formalized in Agda [3].

Likewise, the key results about audited computation and trail inspection, including strong normalization, from Ricciotti and Cheney [28] were formalized using a different tool, Isabelle/HOL[4]. Isabelle provides native support for reasoning about abstract syntax with name-binding, which was a considerable burden in our formalizations of π-calculus in Agda, because Agda lacks such support.

Summary The work on this theme has shown that formalization of some aspects of provenance and slicing is possible, but (unsurprisingly) requires a great deal of effort. In some cases, developing the ideas and the formalization in tandem helped to ensure the results were solid from the beginning, but also had a high start-up cost. Subsequently, Ricciotti and Stolarek (another researcher in my group) have worked on formalizations of database query languages (SQL) and of program slicing for imperative programs, respectively. Papers on these topics are under review.

4 Related projects and next steps

Funding complementing this project was obtained from several sources:

- Microsoft Research supported a PhD studentship (Fu) on provenance for configuration language security.
- Google and an EU FP7 project provided support for another PhD student (Fehrenbach).
- A UK EPSRC funded project *A theory of least change for bidirectional transformations* was on topics largely unrelated to this project, but led in part to the work on incremental relational lenses [20].
- A UK Research Institute on Verified Trustworthy Software Systems (VeTSS) small grant awarded to Ricciotti and Cheney on formalizing a realistic semantics of the SQL query language, including null values.
- DARPA funding from the Transparent Computing (2015–2019) program supported the ADAPT project (led by Galois Inc., with Edinburgh and Oregon State University as subcontractors). This work ties in well with the practical focus of Chan's research, and his work involves ongoing collaboration with other participants in Transparent Computing. I am grateful to a previous AFOSR EOARD contact, James Lawton, for pointing us toward this program.
- ERC funding for the project *Skye: A programming language bridging theory and practice for scientific data curation* (2016–2021) supports work on language-integrated query and Web programming, and I anticipate building on some of the results of the AFOSR-funded research in this project.

5 Conclusions

Establishing a basis for trust in information is a critical problem, and requires progress on a number of fronts. In this project, we have focused on defining and studying provenance and closely-related issues such as information flow, dependency-tracking and program slicing. We have made progress on understanding the formal requirements of provenance security and on ways to analyze existing systems to understand their behavior. We have also investigated how to integrate provenance as a programming language feature, either by translating integrated queries to queries that propagate their own provenance, or by augmenting an abstract machine to support trail inspection more efficiently. We have also developed efficient techniques for propagating changes through relational views and foundations for more flexible programming with record and variant types. Finally, in many cases we have also developed complete formalizations of theoretical results in theorem provers, and these have been the first formalizations of comparable results.

As discussed earlier, open questions remain to be investigated in each of the major themes of this project. Some of these are the subject of other projects currently under way. Overall, the AFOSR EOARD funding supporting this

[2] https://github.com/rolyp/proof-relevant-pi

[3] https://github.com/rolyp/concurrent-slicing

[4] http://www.wilmer-ricciotti.net/FORMAL/ccau.tgz

project from 2013–2018 has been instrumental in building my research group and establishing foundations for secure provenance.

References

[1] U. A. Acar, A. Ahmed, J. Cheney, and R. Perera. A core calculus for provenance. In *POST*, volume 7215 of *LNCS*, pages 410–429. Springer-Verlag, 2012.

[2] U. A. Acar, A. Ahmed, J. Cheney, and R. Perera. A core calculus for provenance. *Journal of Computer Security*, 21:919–969, 2013. https://arxiv.org/abs/1310.6299.

[3] P. Anderson and H. Herry. A formal semantics for the SmartFrog configuration language. *J. Netw. Syst. Manage.*, 24(2):309–345, Apr. 2016. https://homepages.inf.ed.ac.uk/dcspaul/homepage/live/pdf/sfsemantics.pdf.

[4] N. Balakrishnan, T. Bytheway, R. Sohan, and A. Hopper. OPUS: A lightweight system for observational provenance in user space. In *TaPP 2013*, 2013.

[5] A. M. Bates, D. Tian, K. R. B. Butler, and T. Moyer. Trustworthy whole-system provenance for the Linux kernel. In *USENIX Security 2015*, pages 319–334, 2015.

[6] F. Bavera and E. Bonelli. Justification logic and audited computation. *J. Log. Comput.*, 28(5):909–934, 2018. Published online, June 19, 2015.

[7] A. Bohannon, B. C. Pierce, and J. A. Vaughan. Relational lenses: a language for updatable views. In *PODS*, pages 338–347. ACM Press, 2006.

[8] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *Proceedings of the 3rd conference on Hot topics in security*, pages 4:1–4:5, 2008.

[9] S. C. Chan, J. Cheney, A. Gehani, R. Sohan, and H. Irshad. Expressiveness benchmarking for system-level provenance. In *Proceedings of the 9th USENIX Workshop on Theory and Practice of Provenance (TaPP 2017)*, 2017. https://www.usenix.org/system/files/conference/tapp2017/tapp17_paper_chan.pdf.

[10] J. Cheney. A formal framework for provenance security. In *CSF*, pages 281–293. IEEE, 2011.

[11] J. Cheney, U. A. Acar, and R. Perera. Toward a theory of self-explaining computation. In *In search of elegance in the theory and practice of computation: a Festschrift in honour of Peter Buneman*, number 8000 in LNCS, pages 193–216. Springer-Verlag, 2013. http://homepages.inf.ed.ac.uk/jcheney/publications/drafts/festschrift.pdf.

[12] J. Cheney, A. Ahmed, and U. A. Acar. Provenance as dependency analysis. 21(6):1301–1337, 2011. Full version of a DBPL 2007 paper.

[13] J. Cheney, A. Ahmed, and U. A. Acar. Database queries that explain their work. In *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 271–282, 2014. https://arxiv.org/abs/1408.1675.

[14] J. Cheney and R. Perera. An analytical survey of provenance sanitization. In *Proceedings of the 2014 International Provenance and Annotation Workshop (IPAW 2014)*, number 8628 in Lecture Notes in Computer Science, pages 113–126. Springer-Verlag, 2015. Pre-proceedings version available at http://arxiv.org/abs/1405.5777.

[15] S. Fehrenbach and J. Cheney. Language-integrated provenance. *Science of Computer Programming*, 155:103–145, 2018.

[16] W. Fu, R. Perera, P. Anderson, and J. Cheney. μPuppet: A declarative subset of the Puppet configuration language. In *Proceedings of the (31st European Conference on Object-Oriented Programming ECOOP 2017)*, pages 12:1–12:27, 2017. https://arxiv.org/abs/1608.04999.

[17] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. T. Schneider. Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[18] A. Gehani and D. Tariq. SPADE: support for provenance auditing in distributed environments. In *Middleware 2012*, pages 101–120, 2012.

[19] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of*

the 2007 ACM workshop on Storage security and survivability (StorageSS 2007), pages 13–18, New York, NY, USA, 2007. ACM.

[20] R. Horn, R. Perera, and J. Cheney. Incremental relational lenses. *Proceedings of the ACM on Programming Languages*, 2(ICFP):74:1–74:30, 2018. https://arxiv.org/abs/1807.01948.

[21] J. Morris and J. McKinna. Abstracting extensible data types; or, rows by any other name. *PACM:PL*, POPL, 2019. To appear.

[22] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. M. Eyers, M. Seltzer, and J. Bacon. Practical whole-system provenance capture. In *SoCC 2017*, 2017.

[23] R. Perera, U. A. Acar, J. Cheney, and P. B. Levy. Functional programs that explain their work. In *ICFP*, 2012.

[24] R. Perera and J. Cheney. Proof-relevant pi-calculus. In *Proceedings Tenth International Workshop on Logical Frameworks and Meta Languages: Theory and Practice (LFMTP 2015)*, volume 185 of *EPTCS*, pages 46–70, 2015. https://arxiv.org/abs/1507.08054.

[25] R. Perera and J. Cheney. Proof-relevant π-calculus: a constructive approach to concurrency and causality. *Mathematical Structures in Computer Science*, 28(9):1541–1577, 2018. https://arxiv.org/abs/1604.04575.

[26] R. Perera, D. Garg, and J. Cheney. Causally consistent dynamic slicing. In *27th International Conference on Concurrency Theory (CONCUR 2016)*, pages 18:1–18:15, 2016. https://arxiv.org/abs/1610.02327.

[27] W. Ricciotti. A core calculus for provenance inspection. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, Namur, Belgium, October 09 - 11, 2017*, pages 187–198, 2017. www.wilmer-ricciotti.net/PAPERS/insp_corecalculus.pdf.

[28] W. Ricciotti and J. Cheney. Strongly normalizing audited computation. In *Proceedings of the 26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, pages 36:1–36:21, 2017. https://arxiv.org/abs/1706.03711.

[29] W. Ricciotti and J. Cheney. Explicit auditing. In *15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018)*, number 11187 in LNCS, pages 376–395, 2018. https://arxiv.org/abs/1808.00486.

[30] W. Ricciotti, J. Stolarek, R. Perera, and J. Cheney. Imperative functional programs that explain their work. *Proceedings of the ACM on Programming Languages*, 1(ICFP):14:1–14:28, 2017. https://arxiv.org/abs/1705.07678.

[31] J. Thalheim, P. Bhatotia, and C. Fetzer. Inspector: Data Provenance using Intel Processor Trace (PT). In *proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2016.