|galois|

Scytl IR&D and Audit Work Proposal

Galois, Inc.
421 SW 6th Ave., Ste. 300
Portland, OR 97204

Abstract

Galois offers several options of joint work with, and for, Scytl, focusing on high-assurance case studies and audits. Option one is a small-scale, no-cost, validation and verification (V&V) case study of a single algorithm (SHA-256), upon which Scytl's code base critically depends. Option two is a high-level audit of a single package of Scytl's authorship upon which Scytl's code base critically depends at a cost of $10,000. Option three is a more ambitious, medium-scale, moderate-effort, fixed-cost V&V case study of the same subsystem at a cost of $100,000. Lastly, option four is an ambitious, large-scale, multi-stage, negotiated-scope, milestone-based V&V endeavor whose goal is the full-blown V&V of the entire core Scytl platform. Options are independent and can be pursued in any order, sequentially or in parallel.

1. Goals

The purpose of these activities is to help Galois and Scytl and their key employees in this relationship better understand the capabilities, working methods, and personality of the other. Doing so will help us better understand how we might work more together in the future.

Additionally, as a side-effect, Scytl's code base's quality will improve. It will witness the most rigorous validation and verification possible, ranging from protocol mechanical verification to full functional verification of source code. As such, Scytl's public claims about code quality and audits will be evidence-based as audit artifacts can be broadly shared with potential clients, the media, activists, etc.

2. Galois Validation and Verification

Galois formal validation and verification (V&V) of a system is characterized in the appendix (Appendix A). That document describes the contract on the audit, including what is expected of the client (a precondition) and what we provide as a result of the audit. Strongly fulfilling the precondition (i.e., providing more than the worst-case situation we are often in) appropriately reduces the cost of project start-up.

Option one is a *free low-level audit*, option two is a *low-cost high-level audit*, option three is a *medium-cost medium-level audit*, and option four is any kind of audit, but only changes the focus of the audit, and thus its cost.

3. Motivation for Proposed Projects

The subtext of this proposed work is Galois understands that, for Scytl to earn business and trust in new key large markets like those in North America, Scytl's projects must have greater transparency and (actual and perceived) higher-assurance than that which they have today. By having a respected, high-profile security company perform correctness and security audits, Scytl obtains a very high degree of confidence in the quality, correctness, and security of their systems. Moreover, by publicly publishing and widely sharing those audits, Scytl significantly differentiates themselves from their competitors, opening up new markets, and creating a new dialog with election activists.

4. Proposed Projects

Option 1: V&V of Critical Scytl Dependency: BouncyCastle SHA-256

Option 1 is a small-scale, no-cost V&V case study (a *low-level audit*, see Appendix A) of a single algorithm. We will focus on SHA-256 since it is heavily used in several packages

in the Scytl code base.

This option represents a low-cost beachhead for future work by Galois for Scytl along a path of public high-assurance and greater transparency.

This option requires nothing technical from Scytl.

Galois provides, as a result of this work, a technical report summarizing the result of the audit, including a fragment of a domain model for cryptography, a formal specification of the fragment architecture, static model & dynamic model, and representative V&V artifacts.

Option 2: High-level Audit of Critical Scytl Subsystem

Option two focuses on a *high-level audit* (see Appendix A) of a single critical Java package, namely `com.scytl.evote.protocol.ciphers` (as named in the Norwegian code drop), and the Scytl packages and classes upon which it directly depends, namely, `com.scytl.evote.\-protocol.params`, `com.scytl.evote.protocol.engines`, and a small number of utility classes.

Cost

The cost of a *high-level audit* is based upon an analysis of the code base under audit. The quality and size of the code base are the primary factors that influence cost.

Based upon Galois's analysis of the Scytl Norway code drop, performing a *high-level audit* of the `ciphers` package and dependent classes is $10,000, were we to audit exactly that code. If those subsystems have significantly evolved, Galois can estimate cost of the audit in rapid order by analyzing a snapshot provided by Scytl.

Option 3: V&V of Critical Scytl Subsystem

Option 3 focuses on a *medium-level audit* (see Appendix A) of the same set of code as Option 2.

A dependent class not included in this estimate is `VoteOptionPartialDecryptor`, given its use of concurrency. Verifying its correctness, while possible, requires a different effort estimate than we include here.

While this subsystem is mainly a custom set of facades for JCE classes, its quality is critical to the correctness of the Scytl platform.

This option represents a moderate-cost beachhead for future work by Galois for Scytl along a path of public high-assurance and greater transparency.

Cost

The cost of a *medium-level audit* is based upon an analysis of the code base under audit. The quality, size, and architecture & code complexity of the code base are the primary factors that influence cost.

Based upon Galois's analysis of the Scytl Norway code drop, performing a V&V *medium-level audit* of the `ciphers` package and dependent classes is $100,000. Once again, this estimate is based upon the assumption that the current code base is not significantly different from that code drop.

Note that we have *not* promised V&V of Scytl's `jbasis` packages. The above work would be conducted under the assumptions that those packages represented trusted, correct code. A *medium-level audit* including those dependent packages would cost approximately an *additional* $475,000, given our analysis of the Norwegian code drop. Nor have we made an offer to perform V&V on the `keymanager` subsystem.

Option 4: V&V of the Scytl Platform

Option 4 focuses on an ambitious, large-scale, multi-stage, negotiated-scope, milestone-based deep V&V audit of the entire Scytl platform.

The above cost guidelines give Scytl the ability to estimate the cost of performing a *high-level audit* or a *medium-level audit* of their platform's subsystems. Galois can rapidly provide a more detailed cost estimate for any subsystem on request.

If Scytl is interested in a *low-level audit*, based upon past experience, such an audit costs approximately three times as much as a *medium-level audit*. Consequently, for example, a *low-level audit* of Scytl's `ciphers` package and dependent subsystems would cost approximately $300,000.

5. Provisos

5.1. Non-disclosures

In the area of elections, Galois will not perform audits on any code that is not publicly available, nor will Galois produce audits that are not publicly available.

Publicly available code is unrelated to licensing of said code; e.g., code need not be released under an open source license.

Consequently, Galois refuses to work under non-disclosure for any election-related technology.

The timing of public release is negotiable, so long as it is contractually obligated.

5.2. Intellectual Property

All Scytl intellectual property will remain exclusively Scytl's. The audit report and associated artifacts will be Scytl's exclusive property as well.

Any concepts, tools, or technology that Galois develops in the course of this audit that is generic and independent of the audit is Galois's intellectual property.

5.3. Warranties

Galois makes no warranty with regards to the completeness, consistency, or accuracy of our high-level audit, as we are performing a best-effort audit within the stipulated time frame and are not performing formal verification or validation against formal specifications.

Galois warrants that our services and deliverables described here will not violate any agreements or obligations that Galois may have to any other person or entity and that we will perform the services in a timely, competent, and professional manner and in accordance with the statement of work and the prevailing standards in our industry. However, Galois disclaims all other express and implied warranties, including the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

A. An Overview of Galois Correctness and Security System Assessments

This appendix outlines the process typically used by Galois, Inc. for software system assessments of correctness and security.

Galois offers three levels of assessment: high-level, mid-level, and low-level. Each focuses on the software system rather than the owning organization or its policies and practices. Each level uses progressively deeper inspection and delivers progressively more detail to the client at conclusion of the assessment.

Although each assessment follows closely best practices in the computer security industry, each assessment is bespoke, focused on the client's particular need. Typical steps in our software system assessment process are discussed here. In some cases, other steps not discussed here may be included in an assessment. Such steps might include penetration testing of the running system, or fuzz testing of inputs to detect vulnerabilities.

Often our software system assessments are one valuable part of a broader assessment that includes human resource aspects of security, assessment of an organization's network security, and organizational security policy audits. At other times, for example in security assessment of a software product, our assessment may stand alone as an attestation of the correctness and security of that product.

A.1. Assessment Entry Conditions

Galois typically expects to receive from the client a static snapshot of the system to be assessed. This snapshot typically includes the code and build environment from which Galois can reliably and straightforwardly construct a working system instance for examination and assessment. All source code and configuration data, along with a working set of build and deployment scripts, are typically included. A set of system documentation (user's manual, tutorial) are also typically included. Typically excluded, though valuable if available, are:

- requirements specification for the system
- domain models for the system
- architecture specification of the system
- test suites
- protocol specifications that the system is meant to implement
- security properties specification that the system is meant to fulfill

A.2. The Assessment Process

0. Ensure the the system builds. This typically means using either the provided build or CI system (e.g., Make, Ant, Maven, etc.) or building an entirely new workspace within the appropriate IDE (e.g., Eclipse, Visual Studio, etc.).

1. Perform static analysis of system to characterize its overall size, quality, and nature, capturing information in a lightweight fashion using a table that summarizes the system's top-level properties, a by-module breakdown, and highlighting of subsystems that appear particularly at risk.

2. Extract an informal domain model for the system, expressed in CRC cards, BON informal models, or structured English and scenarios/use cases.

3. Extract an architecture specification for the system, expressed in one of several representations: architecture box-and-line diagrams; BON static diagrams; UML package/component/class diagrams; AADL, Acme, or SDL specifications; or Microsoft layers diagram.

4. Capture system requirements, focusing on top-level implicit critical requirements for operational and functional behavior, and security properties. Such specifications are expressed in BON informal specifications, UML use cases, or Agile user stories.

5. Extract a static model of the system by low-level architecture extraction using a CASE tool. Such specification is expressed in a BON formal specification or a UML static model specification.

6. Capture the system's data model by code review, focusing particularly on persistent data stores, data subsystems, and major data flows within the system.

7. Concurrency properties about the system are discovered by a combination of static analysis and code review. These properties are annotated in the architecture, static, dynamic, and data models via concurrency semantic properties.

8. Security properties are concretized:

- in the architecture model by virtue of characterizing the maximal architectural connectors and communication and
- in the data model by labeling data and relationships with high/low (secret/public) annotations or ACLs, in extraordinary cases.

9. Protocols involving secure data or security properties are extracted from the system and expressed using BON dynamic diagrams; UML collaboration, sequence, and activity diagrams; state charts; or abstract state machines.

10. Security audit goals which are concretized in security requirements are methodically checked against the system's architecture, static & dynamic models, and source code by a combination of static analysis, dynamic analysis, and code review.

11. If a live snapshot of the system is provided, or if we are asked to audit a live system that we spin up, the dynamic outward-facing security requirements on live systems are analyzed using a combination of static & dynamic analysis and automated and manual exercise of user interfaces.

12. A risk analysis of the system and its identified flaws is performed. This analysis is written up in structured natural language.

13. Recommendations for changes in the system architecture, requirements, design, or implementation are made for serious correctness and security flaws.

14. A final report is written summarizing the above and providing guidance to the client about changes to, or introduction of, new processes, methodologies, concepts, tools, and technology to improve the quality of the system, particularly with regards to the correctness and security issues discovered in the audit.

A.3. Post-assessment Deliverables

For a *high-level audit*, the client is provided the following:

- a buildable workspace snapshot,
- a high-level summary of the metrics of the system,
- an informal domain model,
- an architecture specification,
- a risk analysis of the system and its identified flaws,
- a report integrating all of the above in an evidence-based, traceable fashion, including recommendations for correcting the most serious correctness and security flaws uncovered.

For a *medium-level audit*, in addition to the above the client receives:

- a high-level operational (non-functional) requirements specification,
- a high-level functional requirements specification,
- a high-level security requirements specification,
- a static model of the system, and
- a data model of the system,
- a specification of the concurrency properties of the system,
- a security properties concretization,
- high-level protocol descriptions of security-related system protocols (both internal and external), and
- a correctness and security properties checklist with accompanying analysis

For a *low-level audit*, in addition to the above the client receives:

- low-level operational requirements,
- low-level security requirements,
- a static model annotated with behavioral, operational, and security contracts amendable to static and dynamic analysis,
- low-level formal protocol specifications of security-related system protocols amenable to machine-assisted analysis,
- low-level concurrency specifications amenable to machine-assisted analysis, and
- recommendations for all flaws uncovered.

A.4. Summary

Galois offers three levels of correctness and security assessments. Our assessments focus on analyzing system architecture, discovering flaws and vulnerabilities, and recommending mitigations.