# Introduction

Within information security, entire classes of application vulnerabilities arise due to problematic user input handling (Christey and Martin, 2007). This covers cross-site scripting (XSS), injection (SQL, LDAP, etc), insecure deserialisation and file inclusion vulnerabilities – all of which are regularly discovered in major software products.

There are many existing products that aim to detect problems within an application. Sadowski et al. describe the benefits of static analysis tooling deployed within Google which allow checks for common issues to be performed as part of the compilation process (2018). The authors explain one of the main challenges with developer adoption as *trustworthiness* "users do not trust to results due to, say, false positives" and highlight the importance of reporting issues early: "survey participants deemed 74% of the issues flagged at compile time as real problems, compared to 21% of those found in checked-in code".

In terms of application security, tooling can be broadly categorised into the two broad areas of DAST (dynamic application security testing) and SAST (static application security testing) tooling. SAST tooling can analyse a codebase at rest and lends itself to generating much more immediate results that can take the entire codebase into account. DAST tooling allows for assessment from a black-box perspective but is much more limited. Research into the effectiveness of DAST-based scanning tools by Doupé et al. drew the conclusion that commonly available tools of this kind often failed to crawl more complex parts of an application which resulted in decreased coverage from a security perspective (2010).

The main objective of this project is to explore the use of a type system which uses regular-expression based refinement types applied to user input. By considering data flow of this user input, it will be possible to make inferences about potential vulnerabilities present in a codebase. Use of refinement types provides for a more informed evaluation of a particular risk than base types alone and should therefore enable reporting of fewer false positive issues.

# Objectives

ℹ Primary objectives are expected to be completed during the lifetime of the project. Additional objectives are identified as potential goals to pursue beyond the original scope of the project, if time permits.

## Primary Objectives

- Formalise a type system that supports types predicated with a regular expression pattern that elements of the refined type will satisfy (be matched by).
  - Explore the consequences of typical string operations (e.g. concatenation) and define the type of their return value when applied to elements of the regular expression type.
  - At minimum, this should allow for simple functions to be declared that can safely accept/return a particular regular expression input[1].
  - Evaluate the rate of false positives when compared to existing static analysis
- Implement such a type system that can guarantee type safety, built against a simplified proof-of-concept language.
  - Test the implementation against a variety of test cases. The testing strategy should make use of automated unit tests, and manual system testing considering both general expected input as well as any relevant "edge-cases" that need to be handled.

[1] As a simplified example, an `unsafe_shell_exec` function might safely be able to accept any input that matches `^[^`]$`

### Additional Objectives

- Apply the theory explored in the primary phase of the project to produce a type analysis tool which works against type annotations applied to a commonly-used language such as C# or Scala. This tooling could be integrated into an IDE or CI pipeline.

# Schedule

Table 1 provides a breakdown of the project time into periods for each fortnight, along with the expected work to be completed.

A meeting will be scheduled for each week to discuss progress and any road-blocks that arise with the project supervisor.

| Time Window | Work |
|---|---|
| October 1st – October 14th | Specification completion, research into prior related works. Study of elementary programming language and type system theory (e.g. simply typed $\lambda$-calculus, SLam). |
| October 15th – October 28th | Begin writing background for report, work on formalisation of regular expression refinement type. |
| October 29th – November 11th | Explore and document properties of type system. Begin implementation of ideas to produce a concrete proof-of-concept. |
| November 12th – November 25th | Completion of progress report, continued implementation work. |
| November 26th – December 9th | Testing of implemented proof-of-concept. |
| January 7th – January 20th | Finalise testing of implementation, write-up test cases. Evaluate false positive rates against existing systems based exclusively on taint tracking. |
| January 21st – February 3rd | Investigate IDE and/or CI integration. |
| February 4th – February 17th | Report work, project presentation preparation |
| February 18th – March 3rd | Project presentation preparation, report work |
| March 4th – March 17th | Project presentation delivery, report finalisation. |

Table 1: Breakdown of projected work by time period.

# Legal, social, ethical and professional issues

As a project with some security motivation, it is possible that legal, social, ethical and professional issues will arise. In particular, evaluation of existing static analysis tooling must be performed with care to ensure that use of any particular external test cases is permitted by the *Copyright, Designs and Patents Act 1988* within the UK.

Additionally, in order to comply with the *Computer Misuse Act 1988*, any static analysis of external code should only be conducted with permission. Discovered issues should be disclosed responsibly.

# References

Christey, S. and Martin, R. A. (2007), 'Vulnerability type distributions in CVE'.

Doupé, A., Cova, M. and Vigna, G. (2010), Why johnny can't pentest: An analysis of black-box web vulnerability scanners, *in* 'International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment', Springer, pp. 111–131.

Sadowski, C., Aftandilian, E., Eagle, A., Miller-Cushon, L. and Jaspan, C. (2018), 'Lessons from building static analysis tools at google', *Commun. ACM* **61**(4), 58–66.
**URL:** *http://doi.acm.org/10.1145/3188720*