

Using Integer Programming to Optimize Investments in Security Countermeasures: A Practical Tool for Fixed Budgets

Jonathan Caulkins (caulkins@andrew.cmu.edu) – Carnegie Mellon University

Eric Hough (ehough@cmu.edu) – Carnegie Mellon University

Nancy Mead (nrm@sei.cmu.edu) – Software Engineering Institute

Hassan Osman (hassan.osman@ey.com) – Ernst & Young LLP

Keywords: software engineering, requirements engineering, risk management, integer programming

Abstract: Software engineers and businesses must make the difficult decision of how much of their budget to spend on software security mitigation for the applications and networks on which they depend. In this article, we introduce a novel method of optimizing, using Integer Programming (IP), the combination of security countermeasures to be implemented in order to maximize system security under fixed resources. We describe the steps involved in our approach, and discuss recent results with a case study client.

Introduction

Software engineers and their customers are continuously faced with a complex and frustrating decision: Given a fixed budget, which combination of vulnerability mitigation actions produces optimal security for a system? In a world with no budgetary or temporal constraints, one could invest in whatever tools or training deemed necessary to safeguard the applications and networks on which customers depend. Engineers could spend arbitrary amounts of time and money patching existing code and take painstaking precaution in writing new software to ensure its security. Of course, the economic reality is that software engineers are pushed to get their product to market as fast as possible, and security is often deemed a distant priority due to budgetary constraints. Fixing the remaining security vulnerabilities post-production can be costly and is wasteful of resources.

In this article we describe a novel methodology for quantitatively optimizing the blend of architectural and policy recommendations, in terms of threat mitigation, that engineers can apply to their products in order to maximize their level of security under a fixed budget. The results of

our optimization are sometimes surprising and even counter-intuitive: bigger budgets do not always produce greater security, and the optimal combination of corrective actions changes non-linearly with increasing expenditures. These findings suggest that some form of formal decision support may usefully augment traditional methods, and the apparent success of our case study demonstrates the feasibility of this integer programming approach.

The problem we are addressing is nontrivial for several reasons. The first challenge is the ability to obtain plausible, reliable, and quantitative estimates of the cost of fixing security vulnerabilities and the corresponding benefits from averting losses due to a breach. There is a large body of academic literature on eliciting such subjective judgments, particularly for risky outcomes, and we used rather than advanced that knowledge base [4, 11, 12]. The innovation here is not how to elicit those parameter values, but rather what methods are used to move from those judgments to an action plan.

The second challenge to prioritizing such corrective actions is their many-to-many relationship with the security problems they solve. If there were a one-to-one relationship between action and vulnerability, one could simply order the security actions in terms of “bang for the buck.” The security mitigation action that solved the most threatening problem would be implemented first, and the remaining actions would then be taken in order of decreasing cost-effectiveness ratio until either time or resource limitations prohibit further steps.

However, no such simple ordering is possible when there is a many-to-many relationship. In fact, the decision of which blend of mitigation steps should be pursued is combinatorial, with the number of combinations growing exponentially with the number of potential actions. Thus, considering every combination is prohibitively time-consuming. Fortunately, the problem can be couched as a mathematical optimization problem (specifically, an integer program or IP [13]) and solved at least to a very good approximation of optimality with a range of software packages, including even ubiquitous spreadsheet programs such as Microsoft Excel [7, 8]. The details of this formulation are given in a technical report [6].

At the Software Engineering Institute (SEI) at Carnegie Mellon University, we have been able to experiment with this new methodology during a case study of the Security Quality Requirements Engineering (SQUARE) methodology [2, 5, 6]. The SQUARE case study client (referred to here as "Acme Company" to be consistent with other reports on the project), has a staff of approximately 1000 employees and provides technical and managerial services to large organizations. Mitigating security concerns of their products and services is a central component of Acme's business mission.

Misuse Cases

The SQUARE process is a comprehensive approach to eliciting, categorizing, and prioritizing the security requirements of a system under design. Here, we focus on a subset of this process in which threats to the client's system are identified by means of *misuse cases*, and their impact quantified via standard risk assessment methodologies. We then apply our IP optimization methodology in terms of choosing the optimal combination of actions that engineers can take in securing the system against these misuse cases under a fixed budget.

Misuse (or abuse) cases, as explained by Hope, McGraw and Antón, are scenarios that describe how an attacker or malicious user would attempt to abuse a system [3]. In studying them, software developers can gain insight on how to design systems to counteract prospective attacks. Successfully preventing a misuse case requires implementing a combination of *architectural recommendations (ARs)* and/or *policy recommendations (PRs)*. An example misuse case, and the corresponding AR and PR to prevent it, are listed below in Table 1.

Misuse Case	User gains access to the system using spoofed identities.
Architectural Recommendations (AR)	Set up firewalls between servers and workstations.
Policy Recommendations (PR)	Patch the firewall software weekly.

Table 1: A sample misuse case and recommended mitigation strategies for Acme Corp

Obviously, not all misuse cases are this simple, and preventing a more complex misuse case may require the implementation of multiple architectural and/or policy recommendations. In addition, a single AR or PR can be part of the solution of multiple misuse cases. In the example above, the

suggested PR also mitigated the effect of another misuse case: “Main server gets infected with a virus or worm”. Hence, there is a many-to-many relationship among misuse cases and recommendations, both policy and architectural.

The Costs of Mitigation

In order to prioritize the implementation of recommendations, it is necessary to assign a cost to each, say, in dollars per year. To implement an AR, costs might include initial hardware or software expenses and their corresponding implementation and maintenance requirements.

In the firewall example above, the AR costs include the price of the physical firewall unit as a fixed hardware cost in addition to the number of hours expected to be spent by an employee to set up the firewall and monitor it (multiplied by that employee’s hourly rate). The number of person-hours spent (again, multiplied by the hourly rate) learning how to patch the firewall and the expense of patching thereafter were included as the PR costs.

All these costs can be summed to reflect the total yearly costs per recommendation. However, this sum is misleading due to the many-to-many relationship between recommendations and misuse cases. The more misuses cases a recommendation addresses, the lower its marginal cost of implementation becomes. Our optimization methodology reduces the complication of which misuse cases should be addressed when budget restrictions preclude addressing them all.

In our sample application, Acme Company decided that misuse case resolution for them was a binary variable, meaning that each misuse case would either be completely resolved or completely unresolved. This implied that for Acme, even if four out of five of a misuse case's recommendations were implemented, the vulnerability would still be considered to be unresolved.

Finally, we can estimate an expected total yearly loss for these unresolved misuse cases. These losses are calculated on a per incident basis and multiplied by an estimated yearly frequency. Other opportunity costs, such as loss of reputation, are also considered in the estimation. Verdon

and McGraw highlighted several common methods for calculating such losses as elaborated in [4].

In the Acme case study, the team discovered twelve misuse cases of their system, an average of about ten recommendations per misuse case, and a single budget constraint (dollar cost). The optimization approach extends readily to handle additional constraint types (e.g., time and/or availability of staff with particular skills). The team then developed a yearly cost per recommendation, yearly cost per resolution of misuse cases, and yearly cost per unresolved misuse case.

Acme was not sure how much money they could budget towards threat mitigation, so the optimization model was solved repeatedly for different budget levels between \$5,000 and \$120,000. For any given budget, the model could be solved to optimality with Excel's built-in "Solver" within a second or two.

Results

The results were instructive on a number of dimensions. Figure 1, which displays optimal spending on remediation as a function of budget, shows that it would never be optimal to spend between \$10,000 and \$40,000 on security countermeasures. The number of misuse cases resolved does not improve with more spending in this range since Acme would not consider a misuse case to be resolved unless every recommendation was implemented. If there were very limited resources, the first \$10,000 could productively be employed to address the "low hanging fruit" such as preventing brute-force password cracking attacks. Not until the budget reaches \$40,000 does it make sense to address other more difficult misuse cases such as buffer overflows, accidental deletion of configuration files, and SQL injection attacks.

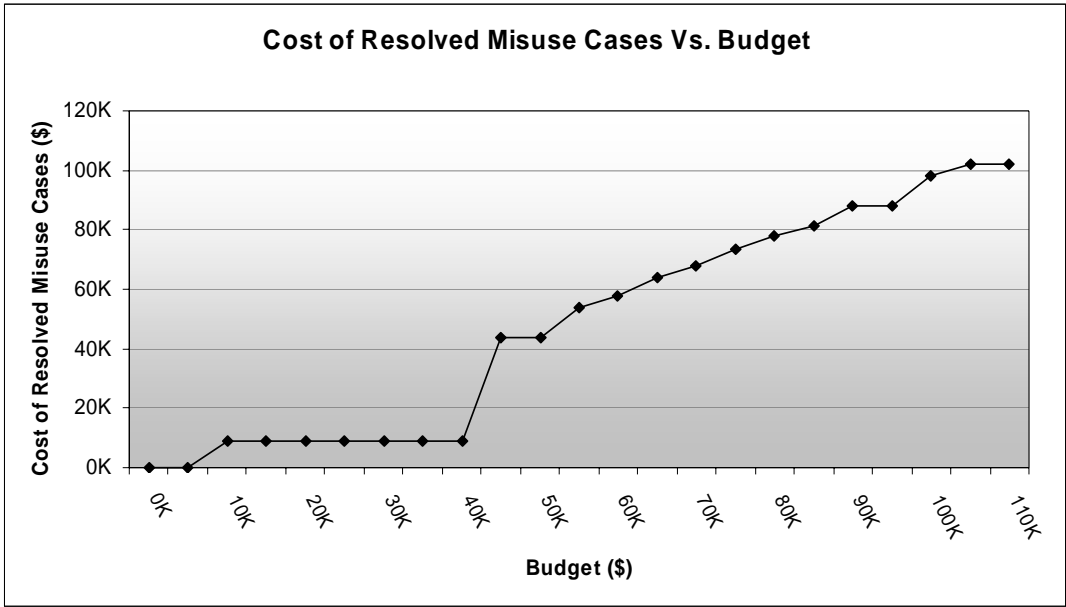


Figure 1: Spending between \$10K and \$40K would never be optimal

Another interesting result was that whether or not a particular AR or PR should be implemented was not a monotonic function of the size of the budget. That is, it was *not* the case that for each recommendation there was a simple decision rule: “If the budget available exceeds X dollars, then implement additional recommendation Y.”

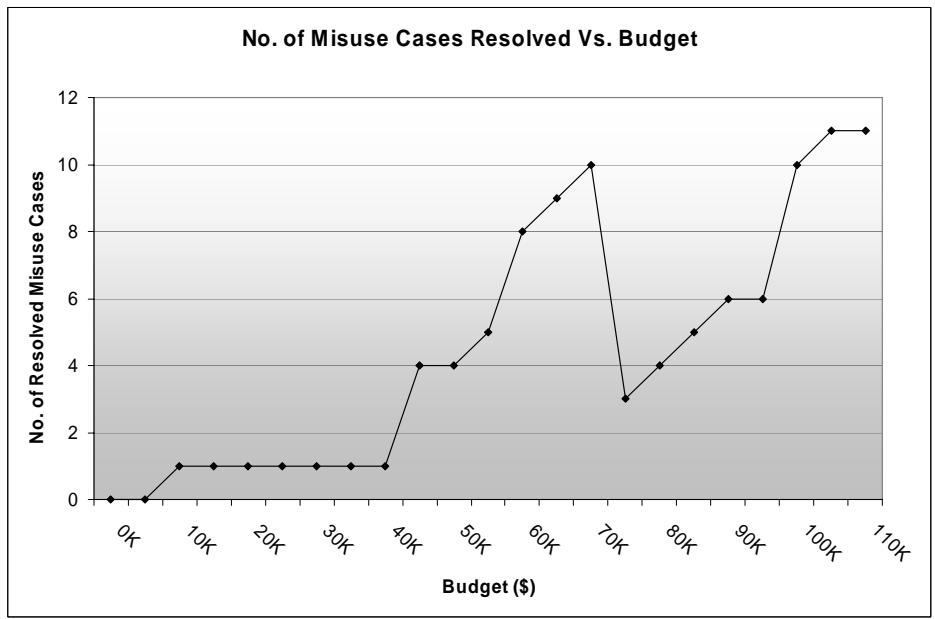


Figure 2: Bigger budgets do not always mean better security

For example, the optimal strategy called for implementing recommendations to prevent "Unauthorized access to the main server" if the budget was between \$60,000 and \$70,000 or if it were greater than \$100,000, but *not* if it were between \$70,000 and \$100,000. In retrospect, the reasoning is clear. When the budget is less than \$60,000, it is not possible to implement all of the required ARs and PRs necessary to address the vulnerability. If the budget grows beyond \$70,000, then it is possible to address an even more pressing misuse case, such as input validation attacks, but not while simultaneously addressing "Unauthorized access to the main server" unless the budget exceeds \$100,000. While sensible in retrospect, these strong interaction effects are hard to anticipate *ex ante* or to work out intuitively. Figure 2 shows exactly how the number of resolved misuse cases grows non-linearly with budget size.

Conclusion

This optimization model indicates which combination of ARs and PRs should be performed in order to maximize the security of a system within a fixed budget of time and resources. The method of resolving misuse cases by implementing discrete recommendations makes it easy to understand which specific actions pay dividends by addressing multiple misuse cases. Without this approach, such instances of "killing two birds with one stone" would tend to be overlooked, leading to misinformed judgments about the actual difficulty of addressing various combinations of misuse cases.

Naturally, there are several limitations regarding this optimization approach. First, as with any quantitative analysis, the results are only as good as the accuracy of the parameters. Although calculating the cost of ARs and PRs could be done reasonably precisely, misuse case losses tend to be merely expert judgments, making sensitivity analysis important. In particular, it may be advisable to have multiple, independent analysts identify the misuse cases and costs, run the optimization with each set of parameters, and observe what results are robust with respect to differences across experts in these judgments.

The software engineering discipline as a whole is still in need of formal tools to assess the accuracy of human estimations. Even in so-called "hard" sciences, experts are known to make consistent, serious errors in judgment and tend to be optimistic about their level of accuracy [10].

This weakness in the model could be mitigated by the collaboration of multiple risk experts performing their analyses independently. As human judgment is prone to be fallible, we can leverage to some extent the "wisdom of crowds" in estimating the costs of misuse cases [9].

The second limitation of this approach is the fact that outlining the entire set of misuse cases is not a trivial task. Given the relatively small size of Acme's application, this wasn't a problem for our case study. However, in larger applications there can be many possibilities by which a system could be infiltrated, and there needs to be an effective method to select and prioritize a set of them [4, 1].

Note, none of these limitations is specific to our optimization-based approach; they would be equally pertinent for any quantitative approach to making decisions about investments in security countermeasures. Furthermore, the application of this methodology to a medium-sized company such as Acme proved to be quite fruitful, so we believe that the methodology holds promise as a tool for further applications deployed in such environments.

References

1. Whittle, Jon and Kruger, Ingolf H. A Methodology for Scenario-Based Requirements Capture.
2. Mead, N., Hough, E., and Stehney II, T. Security Quality Requirements Engineering (CMU/SEI-2005-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
3. P. Hope, G. McGraw and A. I. Antón, "Misuse and Abuse Cases: Getting Past the Positive", IEEE Security & Privacy, vol. 2, no. 3, 2004, pp. 90-92.
4. D. Verdon and G. McGraw, "Risk Analysis in Software Design", IEEE Security & Privacy, vol. 2, no. 4, 2004, pp. 79-84.
5. N.R. Mead, "Requirements Elicitation and Analysis Processes for Safety & Security Requirements", presented at 4th International Workshop on Requirements for High Assurance Systems, Kyoto, Japan, Sep. 2004.
6. H. Osman, P. Chen, M. Dean, L. Lopez, D. Ojoko-Adams, N. Xie and N. R. Mead, "Systems Quality Requirements Engineering (SQUARE) Methodology: Case Study on an Asset Management System", Carnegie Mellon University, Pittsburgh, PA, August 2004.
7. C. Albright, "Premium Solver Platform for Excel", *OR/MS Today*, vol. 28 no. 3, pp. 58-63, 2001.
8. D. Fylstra, L. Lasdon, J. Watson and A. Warren, "Design and Use of the Microsoft Excel Solver", *Interfaces*, vol. 28, no. 5, pp. 29-55, 1998.

9. Surowiecki, James (2004). *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations* Little, Brown ISBN 0-316-86173-1
10. Statistical Software Engineering Commission on Physical Sciences, Mathematics, and Applications. The National Academies Press. 1996. pp 39.
11. Kahneman, D. & Tversky, A. (Eds.) (2000). *Choices, Values and Frames*. New York: Cambridge University Press.
12. Kahneman, Daniel, Paul Slovic, and Amos Tversky (Eds.) (1982) Judgment under uncertainty: Heuristics and Biases. New York: Cambridge University Press
13. L.A. Wolsey. *Integer Programming*. Wiley, New York, 1998.