How to spend it: optimal investment for cyber security Position paper

Fabrizio Smeraldi
School of Electronic Engineering and Computer Science
Queen Mary University of London, UK
f.smeraldi@qmul.ac.uk

Pasquale Malacaria
School of Electronic Engineering and Computer Science
Queen Mary University of London, UK
p.malacaria@qmul.ac.uk

ABSTRACT

A basic cyber security problem is how to optimally spend a security budget. We cast this problem in the framework of combinatorial optimization and explore its relationship with the classical knapsack problem. As in the latter, given a budget, we wish to optimally select a set of resources, each having a cost and a benefit. We propose optimisation algorithms that can deal with resources that depend non linearly on each other, and an optimal budget allocation algorithm for the case of several targets covered by target-specific resources. The general case of resources each of which benefits multiple targets leads to the multiple objective knapsack problem. Also in this case, we extend the standard dynamic programming solution to deal with non-linear dependencies between resources.

1. INTRODUCTION

Given enough money most cyber security attacks could be prevented. Employing the best security administrators, up to date patching, proper crypto storage and transmission of data, monitoring of network activity etc would provide a strong security for any organisation. Unfortunately budget limitations make strong security beyond the reach of most organisations. Hence one of the most important security questions is: what is the best defensive investment that can be afforded given a particular budget? Should patching be prioritised over strong access control policies? Would hiring an additional security administrator part-time be a better investment than a top notch intrusion detection system?

In this paper we frame this investment problem in the following terms: we consider a set of possible targets for a cyber attacker; these targets would be assets of an organisation like web servers, file servers, workstations, critical data etc. We then consider a set of "defensive" resources, each with its associated cost and benefit: for example for the resource "patching workstations" we can consider the cost of patching software over one year period (software cost and associated security administrator time, say 1000$) and the benefits of having software patched (e.g. the percentage improvement in unsuccessful attacks, say 10%).

Each resource may benefit multiple possible targets, for example patching workstations improves their security, but it also enhances the security of web servers by decreasing the probability that a successful root kit attack on a workstation may escalate to a web server hijack. Hence patching may cost 1000$ per year and provide an improvement of 10% on workstation security and of 5% on web server security.

On the other hand, resources can interact non-linearly with each other. For instance, authentication and encryption do provide a much higher benefit when combined than the sum of the individual benefits. Likewise, software packages may have mutual dependencies or be incompatible with each other.

In considering an optimal solution to the problem we take the "minimum-guarantee" approach i.e. a solution should be better than the other if it guarantees a better defence. This is consistent with the idea that an attacker will try to exploit the weakest link in the chain. We can hence formulate the optimal solution to this investment problem as: find a subset S of the set of all possible resources such that S satisfies the following properties:

1. the cost of S is within budget,
2. any other subset of resources within budget provides some target with less security benefits than S does

This problem is reminiscent of the classical knapsack problem: given a budget and a set of resources each with a cost and benefit, determine the subset of resources within the budget with the highest benefit. In fact in the case of a single target our problem reduces to the knapsack problem because the second requirement above reduces to "any other subset of resources within budget provides a lower benefit on the target". As we show in Section 5, if resources are target-specific and cannot benefit multiple targets the problem can be solved by combining the solutions of classical knapsacks on the individual targets.

In Section 4 we introduce an efficient technique for modelling interaction between resources, that generalises to the case of multiple targets.

However the fact that in cyber security individual resources may benefit multiple targets complicates the problem fur-

ACySe'14 May 05 2014, Paris, France
ACM 978-1-4503-2728-2/14/05$15.00.
http://dx.doi.org/10.1145/2602945.2602952

ther. Consider the example in Table 1 where we have two targets and three resources each costing 1. Assume our budget is 1. Then a solution based on the knapsack solutions of each target would return either r_1 or r_2. However as both those solutions leave one target completely open to an attacker they are both worse than r_3.

	target 1	target 2
r_1	3	0
r_2	0	3
r_3	1	1

Table 1: shared resources

Hence the solution for shared resources is not a combination of the classical knapsack solutions. In the literature, this general problem is known as the multiple objective knapsack problem [2]. We review in Section 6 the dynamic programming solution of this problem and extend it to deal with interacting resources generalising the approach proposed in Section 4.

The vast literature on the knapsack problem includes works on risk mitigation in cybersecurity (eg [4]). However to the best of our knowledge this is the first work on cyber-security investment that uses multiple knapsacks or the multiple objective knapsack in a maximin framework.

2. PROBLEM STATEMENT

In the basic instance of the problem, we have a target t to protect and a budget B that allows us to buy resources from a set $\{r_j\}$, with $1 \leq j \leq n$. Each resource provides a benefit b_j and has a cost c_j. Assuming that B is not sufficient to purchase all resources, we are looking for an optimal allocation of the available budget, that is a solution to the following problem:

$$\max \sum_j b_j r_j \tag{1}$$

$$\text{subject to} : \sum_j c_j r_j \leq B \,\text{and}\, r_j \in \{0,1\} \forall j$$

where the value of r_j indicates whether the resource should be purchased or not. This is an instance of the classical (0/1) Knapsack problem (KP).

Obvious generalisations for the cyber-security problem include situations where the budget is to be split across multiple targets (Sections 6) and where the resources interact between them, so that their benefit is not purely additive (Section 4). However, some significant insight can be gained from investigating the simple problem.

For starters, it is worth noting that the intuitive "greedy solution" that aims at selecting the resources that provide the most "bang for the buck" (see Table 2) does not provide a reliable approximation to the optimal solution. It is in fact possible to show that the greedy solution can be arbitrarily inefficient compared to the optimal one.

3. DYNAMIC PROGRAMMING SOLUTION

The optimal solution can be determined using dynamic programming as detailed below [1, 3]. In fact, this solves for all budgets up to the maximum available B and for all subset of resources of the form $\{r_0, \ldots, r_j\}$ where $0 \leq j \leq J$ (here r_0 is a ficticious resource that stands for no resource at all).

We assume that we have a directed graph (abusing notation we call it B) corresponding to all possible expenses up to the maximum budget:

$$0 \longrightarrow 1 \longrightarrow 2 \rightarrow \cdots \rightarrow B$$

and a directed graph R corresponding to all resources listed in an arbitrary order:

$$r_0 \rightarrow r_1 \rightarrow r_2 \rightarrow \cdots \rightarrow r_J$$

We construct the graph K such that its set of vertices is $V(K) = V(R) \times V(B)$. Each vertex (r_j, d) corresponds to the optimal solution $z_j(d)$ of the KP with resources r_0 to r_j and budget d. The edges of K are of the form $((r_{j-1}, d'), (r_j, d))$ where either $d' = d$ or $d' = d - c_j$. Edges of the form $((r_{j-1}, d), (r_j, d))$ have utility zero and correspond to r_j not been included in the solution, while $((r_{j-1}, d - c_j), (r_j, d))$ has utility b_j and corresponds to resource r_j being deployed (which reduces the available budget accordingly).

Figure 1 shows the example of a graph for $B = 5$ and three resources $(r_1, b_1 = 2, c_1 = 1)$, $(r_2, b_2 = 5, c_2 = 3)$, $(r_3, b_3 = 3, c_3 = 2)$. To obtain the solution, we start by setting the utility of the subset of resources containing only the ficticious "null" resource $\{r_0\}$ to 0 whatever the budget, ie $z_0(d) = 0$ for $\{1 \leq d \leq B\}$. Likewise, the utility of a zero-budget solution is zero irrespective of the number of resources: $z_j(0) = 0$ for $0 \leq j \leq J$. We then calculate the utility of each node row by row by maximising over the edges that lead to it, thus implementing the recursion relation

$$z_j(d) = \begin{cases} z_{j-1}(d) & \text{if } d < c_j \\ \max\{z_{j-1}(d), z_{j-1}(d - c_j) + b_j\} & \text{if } d \geq c_j \end{cases} \tag{2}$$

Note that r_{j-1} is the parent of r_j in the resource graph R. The node in the bottom right corner represents the utility of the optimal solution (here $z_3(5) = 8$), and the path followed to reach it reflects the resources that it includes (in this case, r_2 and r_3). Notice that the greedy solution would pick the two resources that have the highest benefit "per dollar spent", that is r_1 and r_2, with a total utility of 5.

The complexity of the algorithm is of order $O(JB)$.

- Construct the list $U = (b_1/c_1, b_2/c_2, \ldots, b_n/c_n)$ that gives the utility of each resource "per dollar spent"
- Sort U in descending order
- Scan the resources r_j in the order they appear in U. If $c_j \leq B$ for some j, set $r_j = 1$ ("buy it") and reduce B by c_j; otherwise $r_j = 0$.

Table 2: The Greedy solution to the Knapsack problem.

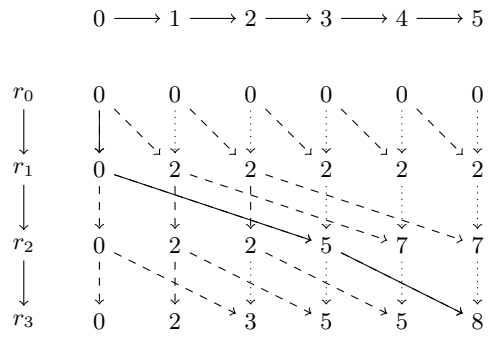


Figure 1: Dynamic programming with 3 resources, 1 target (see text). Dashed edges indicate optimal ways to reach a node; dotted edges are dominated. Solid edges identify the optimal solution, here $\{r_2, r_3\}$ (the resources reached by slanting edges, backtracking from the highest utility).

4. INTERACTING RESOURCES

In a realistic cybersecurity scenario, resources will in general interact with each other. Patching, education, strong cryptography all benefit assets and processes of an organisation.

In the Knapsack literature the standard model for interaction is known as the Quadratic Knapsack Problem. In it, each resource r_j carries its own benefit b_{jj} as well as additional benefit b_{ij} that materialises only if it is deployed together with resource r_i.

This leads to the following problem:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} r_i r_j \tag{3}$$

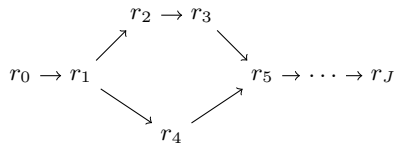
$$\text{subject to} : \sum_{j=1}^{n} c_j r_j \leq B \,\text{and}\, r_j \in \{0,1\} \forall j$$

However, this model has certain shortcomings that make it ill-suited for cybersecurity applications. In particular, it is NP hard in the strong sense (it generalises the problem of finding cliques), and no pseudo-polynomial algorithms can be found [5]. It is also limited to pairwise interactions between resources.

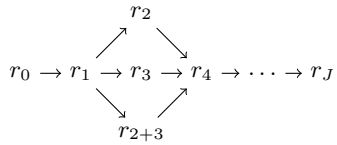
Another more tractable model described in the literature is the Multiple Choice Knapsack problem, that however only describes alternatives between several equivalent resources of which at least one must be present in the final solution [3, 4].

As a more flexible tool for our applications, we propose an extension of the dynamic programming approach presented in Section 3 that efficiently allows optimisation over a limited set of alternatives involving arbitrary groups of resources that may be incompatible, boost each other or otherwise interact in different ways.

With reference to Section 3, we model the dependencies in terms in the structure of the R graph:



for example the above graph models a situation in which resource r_4 is a potential all-in-one alternative for the combination of resources r_2 and r_3, that can be used independently or jointly. The same scheme can be used to express, for instance, a three way alternative between resources r_2 and r_3 (taken individually or not at all) and a ficticious resource r_{2+3} that represents a combination of r_2 and r_3, with benefit $b_{2+3} \gtrless b_2 + b_3$. Nested branching can be used to model more complex alternatives.



Construction of the dynamic programming graph K directly generalises the one presented in Section 3 above, with is $V(K) = V(R) \times V(B)$. The edges of K are of the form $((r_i, d'), (r_j, d))$ where either $d' = d$ or $d' = d - c_j$ and (r_i, r_j) is an edge in R. In this way, the recursion relation 2 is generalised to

$$z_j(d) = \begin{cases} \max_{i \in P(r_j)} \{z_i(d)\} & \text{if } d < c_j \\ \max_{i \in P(r_j)} \{z_i(d), z_i(d - c_j) + b_j\} & \text{if } d \geq c_j \end{cases} \tag{4}$$

where $P(r_j)$ is the set of the parent nodes of resource r_j in the resource graph R.

Similarly to the classical KP the optimal solution can be found in pseudo polynomial time.

Note that the same technique can be used to deal with non-additive interactions among resources in the more general case of shared resources, described in Section 6 below.

5. MULTIPLE TARGETS, SEPARATE RESOURCES

The algorithms presented above allow us to perform optimal resource allocation for a series of targets each of which can be covered by a different set of resources (each potentially with its non-additive benefits), once a budget for each target has been decided. However, in general such a budget allocation is not a given and should be a product of the optimisation. We can obtain the budget allocation by noticing that the dynamic programming algorithm above actually finds an optimal allocation of resources for all budgets up to B.

For each target t, let R_t be the set of resources available to cover target t, and denote by $U_t(B_t)$ the utility of the solution of the Knapsack problem with resources R_t and budget B_t.

In general we have $U_t(B) \leq U_t(B')$ if $B < B'$, while the relation between $U_t(B)$ and $U_s(B)$ is unknown if s and t

1. Let $B_t = 0$ for all t
2. Let $t_{min} = \arg\min_t U_t(B_t)$
3. Let B' be the minimum budget for which $U_{t_{min}}(B') > U_{i_{min}}(B_{t_{min}})$. If this cannot be found goto (5)
4. if $\sum_{t \neq t_{min}} B_t + B' < B$: set $B_{t_{min}} = B'$; goto (2)
5. return (B_t)

Table 3: Budget allocation for targets with separate (target-specific) resources

are two different targets. We can efficiently determine the optimal allocation of budgets B_t for each target (such that $\sum B_t = B$) using the algorithm described in Table 3. Note that this applies both to the standard scenario described in 3 and to the case of interacting resources, in which case $U_t(B)$ is obtained for each target as described in 4.

6. MULTIPLE TARGETS, SHARED RESOURCES

A more general case presents itself when some of the resources are effective in protecting more than one target. Formally, resource r_j brings benefit b_{tj}, to target t. Optimisation thus involves one objective function per target, while the constraint is given by the common total budget B. A natural approach is to look for the set of resources that maximises the minimum benefit across all targets (this is consistent with the idea that an attacker will try to exploit the weaker link in the chain). This naturally leads to the Max-Min Knapsack Problem (MMKP):

$$\max \min_t \sum_j b_{tj} r_j \tag{5}$$

$$\text{subject to} : \sum_j c_j r_j \leq B \,\text{and}\, r_j \in \{0,1\} \forall j$$

For the t part of the input, Yu [6] showed that MMKP is strongly NP-hard even if all costs are equal to one. However, if the number of targets is a constant MMKS can be solved in pseudopolynomial time by dynamic programming [3]. Specifically, for T targets, we define $z_j(d, v_1, \ldots, v_T)$ to be the optimal solution of the following problem:

$$\max \min_t \sum_{i=1}^{j} (b_{ti} r_i + v_t) \tag{6}$$

$$\text{subject to} : \sum_{i=1}^{j} c_i r_i \leq d \text{ and } r_i \in \{0,1\} \forall i$$

Thus $z_j(d, v_1, \ldots, v_T)$ is the optimal solution for a knapsack with capacity d, a reduced resource set $\{r_1, \ldots, r_j\}$ and with the utility of target t increased by the value v_t, with $1 \leq t \leq T$. Note that for J resources, $z_J(B, 0, \ldots, 0)$ corresponds to the optimal solution of MMKP.

The recursion relations for calculating $z_j(d, v_1, \ldots, v_T)$ are defined as follows:

$$z_j(d, v_1, \ldots, v_T) = \begin{cases} z_{j-1}(d, v_1, \ldots, v_T) & \text{if } d < c_j \\ M & \text{if } d \geq c_j \end{cases} \tag{7}$$

where

$$M = \max \begin{cases} z_{j-1}(d, v_1, \ldots, v_T) \\ z_{j-1}(d - c_j, v_1 + b_{1j}, \ldots, v_T + b_{Tj}) \end{cases} \tag{8}$$

Starting with the initialisation $z_0(d, v_1, \ldots, v_T) = \min_t v_t$ all values $z_j(d, v_1, \ldots, v_T)$ with $0 \leq v_t \leq \sum_{j=1}^{J} b_{tj}$ for targets $1 \leq t \leq T$ and budget $0 \leq d \leq B$ can be calculated in $O(JB\beta^T)$ where $\beta = \max_t \sum_{j=1}^{J} b_{tj}$.

Finally, we note that according the technique we introduced in Section 4 for dealing with non-additive interaction between resources can be extended to deal with shared interacting resources. It is enough to describe the interactions in a resource graph R as described in Section 4. The recursion relations in Equations 7 and 8 can then be generalised to maximise over the set of parents $P(r_j)$ of resource r_j instead than simply over the utility of r_{j-1}, in analogy to what is done in Equation 4. Note that if the number of interactions is limited, $P(r_j)$ will in the majority of cases contain only one element.

7. CONCLUSIONS

Optimal budget allocation in cybersecurity can usefully be cast in the framework of combinatorial optimisation. The ample literature on the knapsack problem provides tools to address basic investment questions. However, specific challenges such as the need to protect multiple targets, sharing of resources between targets and complex interactions between the resources themselves require adapting these tools to this context. In this work we reviewed classical knapsack algorithms, showed their relevance to cybersecurity applications and introduced original techniques for addressing the challenges listed above. We see this as the first step in the application of the much wider family of knapsack problems to cybersecurity investment strategies.

Acknolwedgements

We thank Lorenzo Cavallaro for the initial discussion on the Knapsack. This work was supported by EPSRC grant EP/K005820/1.

8. REFERENCES

[1] R. E. Bellman, Dynamic Programming, Princeton University Press, 1957

[2] M. Ehrgott, X. Gandibleux, Multiple Criteria Optimisation: State of the art annotated bibliographic surveys, Springer 02

[3] H. Kellerer and U. Pferschy and D. Pisinger, Knapsack problems, Springer, 2004

[4] A. Lenstra, T. Voss, Information security risk assessment, aggregation, and mitigation, LNCS3108, pp 391–401, Springer 2004

[5] D. Pisinger, The quadratic knapsack problem — a survey, Discrete applied mathematics, vol. 155, no. 5, pp 623–648, March 2007

[6] G. Yu, On the max-min 0-1 knapsack problem with robust optimisation applications, Operations Research, vol. 44, pp 407–415, 1996