CERIAS Tech Report 2018-2
Assumption-Driven Design
by Peter Loscocco, Machon Gregory, Robert Meushaw
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Assumption-Driven Design

A Strategy for Critical Thinking in Trusted Systems Design

Peter Loscocco, Machon Gregory, Robert Meushaw
National Security Agency
Fort G. Meade, Maryland

Abstract—More than ever, information system designers must provide security protection against a wide variety of threats. While numerous sources of guidance are available to inform the design process, system architects often improvise their own design methods. This paper aims to distil the experience gained by NSA trusted system analysts over decades so that it that can be practically applied by others. The general approach is to identify and reduce the number of assumptions on which the security of the system depends. Simply making these assumptions explicit and showing their interdependence has significant, albeit difficult to quantify, benefits for system security. Our hope is that this design methodology will serve as the starting point for the development of a more formal and robust engineering methodology for trusted system design.

Keywords—Secure System Design, Assumption Analysis, Refinement Goal, Trust, Trusted System, Privacy, Design Methodology

I. INTRODUCTION

No experienced system designer expects to develop a system whose security depends only on elements entirely under the designer's control. The designer of a cryptographic algorithm, for example, may assume there is a reliable source of random numbers available for keying. The designer of an operating system with security requirements may assume that the hardware on which the system runs correctly executes its specified instruction set.

Conversely, a system attacker looks for the security-related assumptions made, perhaps implicitly, by the system designers, developers, and implementers and seeks ways to invalidate one or more of those assumptions. This approach has historically been applied to break operational cryptographic systems, where the users of the system violate assumptions made by the designer, by, for example, re-using a one-time-pad and thereby enabling a careful eavesdropper to derive the key stream [BENS00].

In the realm of cybersecurity, systematic methods for breaking into systems began with the Flaw Hypothesis Methodology, developed in the 1970s to organize penetration testing [WEIS73, WEIS94]. The essence of this method is to hypothesize security flaws – places where the designers, developers, or implementers might have made unwarranted assumptions about system security properties – and then to test whether the hypothesized vulnerability is real.

In the course of reviewing the security designs of many commercial and government systems over a period of decades, we have used our method of searching for implicit (and unsupported) assumptions to expose and remediate security flaws in systems before they are fielded. An early result of this approach exposed how the assumption that an application could enforce its own security constraints depended on (typically unsatisfied) assumptions about security properties of the underlying operating system. [LOSC98].

Designing systems to meet security requirements remains more art than science, but we believe that the lessons we have learned over the years offer a means to systematize and improve conventional system design processes. This paper describes an assumption-driven design methodology that has resulted in successful secure system design and analysis.. No process alone can replace knowledge and experience, but using one that helps designs converge to better security solutions can be the deciding factor in producing better systems.

In our experience, information assurance curricula have generally stressed many critically important concepts that prepare future security professionals with necessary security knowledge but have not provided adequate training in applying that knowledge toward secure system design. We offer our Assumption-Driven Design methodology as a

potential starting point for the development of a design process that can augment existing information assurance education programs.

II. An Approach to Secure System Design

There are of course many approaches to system design and implementation: top-down, bottom-up, functional decomposition, stepwise-refinement, waterfall, spiral systems development and more. We do not propose to replace any of these. We describe an approach that can be used in conjunction with any of these approaches to reveal and track security assumptions. Nevertheless, to simplify the presentation, we describe our approach in the context of functional decomposition and stepwise refinement.

The core notion embodied in our approach is that trustworthy system design should be assumption-driven, meaning that all assumptions must be explicitly identified and tracked throughout the entire design process. System design typically involves an iterative set of steps including functional decomposition and replacement of generalized functions with specific mechanisms. Each step invariably leads to the creation of new assumptions or the transformation, often subtle, of existing assumptions. Assumption-Driven Design focuses the designer's attention on systematically identifying, tracking, and validating assumptions throughout the design process.

A. *What is an assumption?*

An assumption in the context of this methodology is an assertion about the system being designed; it is a statement that may be either valid or invalid. In general, it will be motivated by a desire that the system have some particular security property, but it may not refer to any such property. For example, in the context of an access-checking module, an assumption might be:

A1. Access to a controlled resource can only be gained by first passing through the access checking module.

In the context of a cloud system relying on virtualization, perhaps:

A2. An application running within a virtual machine cannot gain access to memory allocated to a different virtual machine.

Or more fundamentally:

A3. The memory-mapping hardware in the cloud server functions as intended.

Though it might sometimes be possible to provide rigorous proofs that a design enforces specific assumptions, absolute rigor is not the goal of this approach. Assumptions provide a way for the designer to organize his or her thinking about the design and to identify (and possibly reorganize) trust dependencies within the system. The approach is formal in the sense that it provides a specific structure in which assumptions can be exposed, documented, and checked, even though the checking may require human evaluation.

Although an assumption may only be either valid or invalid, in fact it may be difficult to determine its validity. Consequently an assumption may also have a level of confidence associated with it (e.g. low, medium, or high), and the designer may also specify a threshold for the desired confidence level using the same scale.

In the context of functional decomposition and stepwise refinement, each time a refinement is introduced, an assumption analysis must be performed, annotating the current design with an updated understanding of identified assumptions. Previously identified assumptions, even those already deemed valid, will have their confidence values reassessed. Newly identified assumptions will be given an initial confidence value and, if necessary, assigned a satisfaction threshold. Maintaining an updated list of assumptions enables the designer to track outstanding issues in the design and helps focus refinement efforts on their elimination.

B. *Can a designer capture all security assumptions?*

At a given level of description, it should be feasible to enumerate the security properties desired of a system and the assumptions on which those properties depend. Yet the nature of security is that it requires humility on the part of the designer. Adversaries are inventive and will seek out assumptions that designers may have made implicitly. For example, recent experiments displayed how acoustic signals might cause a device's accelerometer to deliver invalid inputs that could adversely impact security decisions [TRIP17].

Knowing when all the relevant assumptions have been identified is a human, not mechanical, task, and one that may need to be revisited. Consequently, the effectiveness of a security design is necessarily limited by the designers' ability to identify security assumptions and address them in the final design.

C. *Dimensions of Assumption-Driven Design*

There are two significant and complementary dimensions of our approach to system design. The first is a general approach to design that could be applied independently to any design domain, including but not limited to secure systems. Priority in this approach is placed on identifying and tracking assumptions made throughout the design process. This general strategy is described in the first of the following sections.

The second dimension, described next, can be viewed as a security-specific overlay to the general Assumption-Driven Design process. It focuses on security specific analysis and design techniques. It includes a strategy for identifying hidden assumptions, ideas about approaches to design refinement, and how our methodology supports traditional security activities such as threat modeling, vulnerability analysis, and the assessment of security design tradeoffs.

D. *Challenges and Limitations*

This assumption-driven design process is not intended to replace existing secure system design practices. Such things as requirements engineering, security model specification, or any of the various assurance activities remain valuable. Our methodical approach can yield improved results, not only from the rigor imposed on the design process but also from the creation of design artifacts that capture the rationale for decisions made during its execution.

Preserving information about identified assumptions, how they were addressed in the design, and the rationale for determining confidence levels can prove extremely useful during system evaluation, when determining suitability for different operating environments, and when revisiting designs because of new requirements. In each of these cases, the explicit tracking of assumptions and the means by which they have been addressed facilitate the necessary arguments that designs achieve their stated security goals. However, in the absence of automated tool support, fully tracking all assumptions can be burdensome and the focus should be on identifying those assumptions most relevant to security.

III. ASSUMPTION-DRIVEN DESIGN

A. *Example: Assumption-Driven Design for a Wireless Client*

To illustrate this method, we first introduce a simple example of how a design might proceed and then describe it more abstractly. Suppose an enterprise with an existing, closed wired network infrastructure wishes to add a wireless capability for enterprise client computers without significantly increasing the risk that clients or their communications will be compromised (Fig. 1)

Key assumptions for the wired system might be[1]:

1. System supports only wired connectivity to the enterprise.
2. The security risks related to connectivity are acceptable:
    a. Illegitimate clients cannot access the enterprise servers
    b. Only communication between enterprise systems is possible (no external connectivity possible)
    c. All intranet traffic originates from systems within the enterprise
    d. Intranet traffic is visible only to enterprise clients
    e. All client connectivity to the enterprise is via the installed network device.
    f. Client systems are sufficiently protected from external (non-enterprise) attacks.
    g. Client systems are sufficiently protected from network-based attacks.
    h. Client systems are sufficiently protected from server-based attacks.

[1] This list is to illustrate the approach and not intended to be complete.

Changing to a system that supports wireless connectivity means these assumptions must be reconsidered. The system must now support wireless connectivity as well as wired, so assumption 1 is no longer valid; it can be replaced by

1'. System supports wired and wireless connectivity to the enterprise

Because the connectivity mode has changed, Assumption 2, which depends on 2.a-2.h must be re-validated as well. In the wired-only network, an outsider could not easily monitor or inject traffic into the intranet, yielding a non-zero but acceptable security risk. In the wireless network, it becomes much easier for the outsider to monitor traffic and potentially to introduce traffic.

The need to re-establish the validity of Assumption 2, motivates additional requirements on the system. In particular, assumptions 2.a and 2.d are no longer valid because the wireless intranet traffic can be intercepted, and outside transmitters may be able to inject traffic much more easily than before.

One of many alternative wireless technologies might be used, for example WiFi, cellular, WIMAX, Bluetooth, Zigbee. Each of these represents a design alternative that could validate Assumption 1 and each in turn may generate different, more detailed requirements that would be needed to support the validation of Assumption 2. For example, Assumption 2.d will now motivate the use of encryption, which will generate a rich and design-dependent set of assumptions to ensure the cryptographic mechanisms are employed securely.

Assessing the validity of each assumption completes the assumption analysis for this iteration. Only when all assumptions have been satisfactorily validated, with all confidence values exceeding their thresholds, is the design process complete.

B. *Iterative Approach to Design*

To better understand how the Assumption-Driven Design process encourages better design outcomes, consider the general design process. Designs are descriptions, or blueprints of a sort, of a thing at some level of abstraction. The design process is iterative, creating candidate designs to meet some set of requirements, evaluating them against some decision criteria, and then selecting one or more candidates for refinement.

The designer's task is to make a series of suitable choices from the available design options, guiding the process toward a suitable final design. The choices that a designer makes throughout the process are often ad hoc and once codified in a final design, difficult to trace back to the conditions that motivated them. Adding a degree of rigor to the design process can help guide the designer to better choices and capture the rationale behind decisions.

Every design is created from some set of requirements that drives the choices a designer must make. Before the design process begins, an initial set of requirements must be identified. Ultimately, it is the satisfaction of these requirements that determines the suitability of a design solution. But where do these requirements originate?

A primary source of requirements is the domain of the object being designed. A designer sets out to design a specific instance of a class. Membership in that class implies certain things about the requirements that must be met. Some of the requirements are functional or pertain to specific properties of the target domain. Some may specify materials or processes that must be adhered to during realization of the design. In general, a designer must understand the domain to which the design applies and identify all domain-specific requirements. Additional requirements are specific to the instance of the class being designed. A particular use case, target environment, or customer concern, such as cost or energy consumption, may cause the domain requirements to be augmented or in some cases relaxed.

Given a set of requirements, the designer begins the iterative process of creating a design that can satisfy all requirements. Starting from an initial design, each iteration results in one or more candidate refinements, the best of which becomes the next in a sequence that should eventually converge on a solution. Driving these refinements is the selection of one or more refinement goals for the current design. This goal-driven approach to design allows the designer to focus all modifications to the current design on specific improvements toward the solution.

Refinement goals are generated to meet one or more as yet unsatisfied requirements. Each candidate refinement created in response to a refinement goal is an alternative approach to satisfying the corresponding requirements. Each alternative may add function, remove deficiencies, or increase detail, the best being selected for further refinement.

Each design refinement is intended to advance the design in some way. It would be nice if all refinements resulted in satisfaction of some existing requirement, reducing the set of remaining requirements needing attention in subsequent refinements. Unfortunately, this is not always the case. Design refinements often increase the set of remaining requirements.

Considering the case where design refinements add new components makes this apparent. New components may result in new requirements specific to that component or to its use in the current design domain. These new requirements are only necessary as a consequence of the refinement, but still they must be met.

Consider the space of possible designs as a tree (Fig. 2). Each node represents a candidate design refinement of its parent that could be explored during the design process. The root node represents the blank slate from which initial candidate designs are created to satisfy the top-level requirements. Interior nodes represent partial designs that do not meet all requirements. The leaves of the tree represent alternative complete designs or in some cases, candidate designs that have been abandoned for some reason or have yet to be fully explored.

For a given refinement goal in the context of a given node, refinement will identify all of the children that could be explored on the way to discovering a solution path. When no additional refinement goals can be generated, leaf nodes will have been reached and a solution, if it exists along the current path, will have been found. If not, the path must be backtracked to another node in the tree representing an alternate candidate refinement. From that point, the refinement process can continue in search of a path to a solution.

Care must be taken during backtracking to ensure that requirements that were satisfied at some node are noted to be unsatisfied when backtracking beyond that node. In addition, any derived requirements introduced with a refinement must be removed when backtracking progresses beyond the point where they were introduced. Introducing, a well-defined bookkeeping discipline into the process facilitates the proper tracking of which requirements are active and yet to be satisfied at each stage throughout the process.

C. Using Assumptions to Drive the Design Process

The preceding two sections illustrate how assumption analysis motivates both functional (adding a wireless capability) and security (avoiding increased risk) modifications to an existing design and describe and place it in the context of iterative design. This approach has informed our system design, analysis, and consultation work for many years. We now describe in more detail how assumptions are used to drive the design process.

Whenever a design refinement is made in response to a refinement goal, the designer uses experience-based judgment to modify the existing design in some appropriate way. Modifications are intended to better satisfy existing requirements without negatively affecting any aspect of the design that was previously introduced as a design refinement. Ideally, all design refinements would advance the design without negatively impacting the work of prior refinements. In practice, this is rarely the case.

In order for a refinement to strictly advance a design against the requirements, a number of conditions or factors relating to the specific nature of the design and the proposed refinement need to hold. In making the refinement, the designer in effect asserts that they are indeed valid. In some cases, they have been overtly included in the designer's reasoning about the refinement. In all too many cases however, they have not.

The appropriateness of a refinement will often depend on implicit conditions that may or may not be true, and it is often the case that defects in the final design can be traced back to refinements that were made without regard for unstated assumptions that were never addressed in the design. They may have resulted from explicit assumptions that were erroneously treated as valid, but in most cases this can be traced back to other implicit assumptions that were never considered. Identifying all implicit

assumptions will not necessarily eliminate all design defects, but it should increase the chance that all relevant issues are at least considered.

Recognizing how important assumptions are to refinement yields an important improvement to the overall design process, the need for designers to explicitly track all design assumptions. At each refinement stage, while satisfying unmet requirements, they should also seek the reduction of unsupportable assumptions. They should analyze the new design to identify newly introduced assumptions. In addition, they should revisit previously addressed assumptions to identify adverse impacts. Thorough assumption analysis throughout the process increases the likelihood that implicit assumptions are exposed and that no important details are overlooked.

Recognizing the relationship between requirements and assumptions offers an opportunity to more tightly define the design process, one driven by design assumptions. It is important to understand that all requirements, whether original or derived, can be restated as assumptions, namely an assumption that the requirement is met. By definition, these can only hold true when the requirement is met.

By converting all requirements to assumptions, satisfaction of any type of requirement in the process can be driven by activities that attempt to substantiate assumption validity. Outstanding assumptions can imply refinement goals. These in turn focus the generation and selection of candidate design refinements. If confidence values relating to validity can be associated with outstanding assumptions, it becomes possible to create a more objective costing function to aid in selecting the best candidate refinement. By seeking to eliminate all outstanding assumptions with confidence values below some acceptable threshold, incomplete designs evolve, converging toward an acceptable final design solution.

Figure 3 provides a snapshot of the design refinement process. On the left is the current design state, including some assumptions below the desired confidence threshold that therefore motivate refinement goals. Those goals lead to candidate designs (satisfying the guiding design principles). Each alternative design may have validated, revised, and introduced different sets of assumptions and may have different costs.

A good bookkeeping process, one including confidence values that assumptions are valid, aids tracking assumptions in the context of design refinements and limits flaws in the final design. All assumptions, now explicitly stated, can be treated as derived requirements. As different parts of the design space are searched, only those assumptions associated with refinements in the current solution path will be considered. Assumptions, valid or only partially so, will not be lost when left for future refinements. When solutions are reached, all factors with respect to requirements, limited only by designer knowledge, will have been considered.

Figure 4 provides a flowchart for the entire system design process; exiting the chart at "FAIL" means that no satisfactory design could be identified while exiting at "SUCCESS" corresponds to the identification of a leaf node of a design tree like the one in Figure 2 that provides a satisfactory system design.

IV. Security Overlay

A. *General*

Designing for security has proven to be extremely tricky. Clever attackers have repeatedly demonstrated how the unanticipated use of designed features/privileges, unfortunate design choices seemingly unrelated to security, or reliance on unstated assumptions that had little chance of being valid have resulted in security failures. The assumption-driven design process maximizes the designer's chances of anticipating problems on which attackers thrive and proactively addresses them during design. The structure it imposes focuses the designer's existing knowledge and skill on effective refinements that address current shortcomings while minimizing the chance that important security-related implications of those refinements are overlooked. Several features of the design process help ensure that following the process methodology will yield better designs.

B. *Design Principles/Designer's Toolbox*

Security designers must adopt a core set of design principles that guide them through any design process. These should be explicitly stated and revisited during design, evaluating choices

being made throughout the process against those principles. Enumerating a universal set of security design principles is beyond the scope of this paper. However, much has been written on this topic [SHF01, BISH12, LBBN05]. Two good examples are the least privilege principle and the principle of separating policy from enforcement. It is easy to see how applying these principles during critical process steps like candidate refinement generation or assumption analysis would impact the final design.

Along with design principles, an experienced designer brings a toolbox of mechanisms, tools and techniques that have proven effective against certain security problems. Different designers will have different toolboxes. When choosing refinement goals, the designer can anticipate which tools will have the most benefit. Likewise, when generating candidate refinements against some refinement goal, designers can look to the toolbox with confidence that selected tools will indeed increase confidence values for unmet assumptions. Repeated use of tools facilitates analysis, as experience will indicate what types of assumptions can be addressed, how well, and what, if any, residual assumptions might remain.

A designer's principles and toolbox are the greatest factors contributing to successful design. They drive key aspects of the process, including candidate design refinement generation, assumption analysis, refinement goal generation and selection.

C. *Candidate Refinement Generation*

Creating candidate refinements is more art than science, but it can be taught. The design principles and toolbox concepts are aids that help designers hone their craft. They help the designer recognize classes of problems and how to employ proven solutions. Similarly, looking to other successful designs for ideas is useful.

Other heuristics exist that can also help. Recognizing relationships between different kinds of assumptions or the repetition of certain assumptions across many different components may signal common problems that can be addressed with more centralized mechanisms rather than individually addressing them throughout the design. As an example, consider a system containing many communicating entities. Using a common, secure messaging system rather than attempting to address the concerns individually with each entity might best address assumptions identified throughout the design about message confidentiality of authenticity.

Another powerful concept to help designers generate more effective candidate refinements is termed trust relocation. Whenever security mechanisms are employed in a system, their trustworthiness will depend on the validity of assumptions made about how that mechanism has been integrated and will be used, or in other words, how trustworthy are the mechanisms validating those assumptions. The idea behind trust relocation is that these trust assumptions will always exist somewhere in the design, making the object of design refinement to select candidates where the validity of assumptions rests on mechanisms most worthy of trust. Trust relocation leads to better solutions by encouraging designers to recognize that trust assumptions are shifted and not eliminated during design, and to employ security mechanisms that maximize the number of trust-related assumptions that can be addressed in a trustworthy way while minimizing the number being introduced.

Employing security mechanisms for data isolation within a running system offers a good example for the trust relocation idea. Processes could protect access to data with a combination of cryptography and discretionary access controls, leading to a variety of trust-related assumptions for each process requiring such protections. Alternatively, a strong central mandatory access control system could provide the requisite security guarantees while only requiring new assumptions to ensure the trustworthiness of the MAC mechanism. Here, the need for trust in mechanisms has been relocated from each process to just the one mechanism in the system.

D. *Refinement Selection*

There are several points in the process where designers must make selections impacting designs. The first is in the selection of refinement goals. There is no single right way to make selections, but again heuristics play an important role in allowing efficient convergence toward a successful design.

A straightforward approach is to simply choose the goal that eliminates the most residual

assumptions. This could work but could also easily introduce inefficiencies, as deferring just one critical assumption in favor of many less critical ones might impact the path down the design tree by requiring backtracking when finally addressing the critical assumption.

A better approach seems to be to select refinement goals with an eye toward the likely candidate designs that will result. Looking across the full set of refinement goals, considering how tools might best be applied, how candidate generation heuristics like trust relocation might be invoked, and the likely outcome of assumption analysis will lead to much better selections. In any event, the designer should attempt to make selections that lead to the quickest convergence.

The second selection point is the identification of the most promising candidate refinement on which to grow the design tree. This is always going to be a subjective call, but designers should endeavor to make it as objective as possible. Costing functions can help with this. The idea is to define a number of dimensions on which each candidate would be evaluated and the relative importance of each. The resulting cost vectors can then be compared to focus the selection to the best candidates.

There is no one best set of dimensions for the cost vectors. Many might not even be related to security but instead relate to more practical things such as monetary cost, performance, availability, or constraints either externally imposed or resulting from other design choices earlier in the process. The assumption analysis, however, provides important input to the cost function. The number of residual assumptions, the confidence values of each as compared to assigned thresholds, and ideas about the difficulty of increasing those values above the thresholds are perhaps the most useful dimensions for cost and selections that will lead to quick convergence to success.

E. Assumption Analysis

Assumption analysis for derived requirement generation is the key innovation of this design process. This is what gives designers confidence that designs will fully meet requirements. Issues concerning strength or appropriateness of mechanism are limited because assumptions identified for candidate designs with less optimal choices will highlight inadequate aspects of the design or issues problematic to address.

Assumption analysis is a skill that greatly depends on the insight a designer has into potential problems. The value of experience with a full toolbox becomes evident during this phase, as designers will already be familiar with assumptions associated with the use of each tool. But beyond understanding the use of tools, the designer's perspective when questioning a design is most important to effective assumption analysis.

One effective technique is to reason about each system component individually from three separate perspectives. The first is to consider what must be true about itself in order for the component to correctly perform its function. Recursively repeating this question for each sub-component will help tease out hidden dependencies and design fallacies. The second is to consider what must be true about the component for all others to safely depend on it. And the last is to consider what must be true about each component on which it depends in order to safely depend on them.

Another useful technique for assumption analysis is related to vulnerability analysis. If designers take an adversary's perspective and theorize about vulnerabilities, assumptions will fall out. Imagining what is possible from each component if adversaries have total control over it provides valuable insight. This is best done without regard for any specific threat model, enabling designers to understand the total threat and clearly see all assumptions. Whether or not those assumptions are ever to be directly addressed is where threat models are needed. In this way, after the design process is complete, system implementers can make engineering decisions consistent with the intent of the design, and eventual users will be able to determine if the design is appropriate to their intended threat model.

Regardless of how designers approach assumption analysis, revisiting all assumptions during each iteration is important. This is the only way to ensure that a refinement has not negatively impacted an assumption that a previous refinement addressed. Consider the introduction of a new function to a component where some prior assumptions about an existing function being isolated were present. The addition might invalidate

past satisfaction of that assumption. It is still fine to consider such a refinement, but the old assumption would need to be marked as unsatisfied and readdressed in a future refinement.

V. Related Work

The present work has developed over a period of many years in parallel with development of fault trees, first used in the safety domain [ECK63], and with more recent work on assurance cases [PMMSF02], [GLW14]. These approaches generally aim to argue that a system is safe or secure by identifying potential sources of failure and then creating arguments as to why those particular failures are impossible or unlikely. Tools have been developed to organize the logic of these arguments; potentially such tools could be applied in the context of Assumption-Driven Design.

The importance of recognizing implicit assumptions in security designs has been highlighted along with techniques that might be taught to designers to better prepare them for the task [BIAR05]. Another approach to assuring the security of designs that targets assumptions is the Information Design Assurance Red Team [SAND09]. It aims to identify design flaws by positing attacks, which may target assumptions made by designers, not dissimilar from Weissman's original Flaw Hypothesis Methodology [WEIS73].

Approaches to designing systems to meet both functional and security constraints have a long history, starting with the reference monitor approach [ANDE72]. The design principles developed by Saltzer and Schroeder for MULTICS in 1975 [SASC75] can be fruitfully applied in the context of Assumption-Based Design. An approach for developing application-based security models in 1984 [LHM84] included explicit security assumptions and assertions to be met by the implemented system, but did not provide a refinement structure for them. NIST has recently published a System Security Engineering report [RMO16] that addresses assurance cases as a means of building trust in systems but generally avoids detailed methodologies at the level addressed here.

VI. Conclusions and Future Work

This paper has described how a structured design process can improve secure system design. The assumption-driven design process facilitates the discovery of implicit assumptions that would normally be left unstated by traditional design methods. These unstated assumptions, now made explicit, may or may not be satisfied by the final design, but those making decisions about a design will have gained a more complete understanding about the design's suitability.

The assumption-driven design process adds rigor to the design process and can help move secure system design from art towards science. The structure it brings should increase the likelihood that designs will not only meet stated security requirements but also address intended security goals. It is a straightforward process that is a close approximation to one that has been internally used and informally taught by our organization.

Although some parts of the process may seem obvious, especially to experienced designers, the point of this paper is to describe a *process* that leads to better designs and can be taught and should be taught. When coupled with a strong foundation in the principles of systems security and a good toolbox of security mechanisms, teaching it should lead to system designers and security practitioners obtaining the necessary skills for effective secure system design.

The artifacts created through the bookkeeping process are an additional benefit to this process. They not only help keep the design process converging toward success, but they can add value in other significant ways. Evaluation of designs can be reduced to assessing the set of assumptions that have been addressed and the arguments that were created for validity. Such evaluations are more meaningful than addressing checklists of requirements, as they reflect the actual satisfaction of security objectives.

The artifacts offer a different perspective when considering threat models. Rather than creating threat models for specific use cases, a more comprehensive threat model can be expressed in terms of assumptions and tailored for specific use cases by adjusting acceptable confidence value thresholds. Assessing suitability against various threat models is reduced to evaluating each of the explicit assumptions against the thresholds appropriate to desired use cases. As an added

benefit, divining designer's intent would no longer be necessary, as it would be directly reflected in the bookkeeping.

Bookkeeping artifacts also facilitate changes to designs. When new requirements are identified it is possible to restart the design process, and if necessary, backtrack to previously visited portions of the design tree. Even if the original designers are not involved in the redesign, insight that they gained using the process is reflected in the artifacts and is available to the new design team.

More work is needed if the benefits of assumption-driven design are to be realized. If it is going to advance beyond the practice of a few designers to a process broadly taught and used, expanded curricula for existing security courses must be developed and socialized. System integration issues and mechanism-specific assumption analysis must be included when teaching about security mechanisms. More detail about the process itself and worked examples of designs created with it are needed. The examples must demonstrate key portions of the process such as identifying assumptions, selecting refinement goals, and generating candidate refinements. These examples would also need to include the bookkeeping artifacts, showing how they are useful during the design for performing backtracking and ensuring the proper set of derived requirements, and post design, for such analyses as evaluation and suitability of use. Automated tools to support the entire Assumption-Driven Design process will be needed to handle any large, complex design.

Security by design is the object of the process described here, but the process could be applied to privacy by design as well. In this case, the assumptions would need to reflect appropriately tailored privacy properties.

Capturing the thinking behind this design methodology has proved challenging, reinforcing our intuition about the complexity and subtlety of security designs and the need for an effective design process. Although originally intended to support training junior analysts, the effort resulted in documenting a design process that we hope will be useful to a much broader community. Adherence to the assumption-driven design process is difficult, but we believe it can serve as a practical and effective framework for trusted system design, encouraging critical thinking by focusing designers on security issues most pertinent to identified security goals.

VII. ACKNOWLEDGEMENTS

This paper reflects many years of experience in reviewing designs submitted to the authors' organization for review. The authors of those designs, whom we cannot list, and their interactions with the authors of this paper substantially influenced the ideas documented here. Discussions with Perry Alexander, Dylan McNamee, , Sami Saydjari were also helpful. Carl Landwehr assisted with drafting and editing. Responsibility for any remaining errors or omissions remains with the authors.

REFERENCES

[ANDE72] Anderson, J P. "Computer security techno[ogy planning study," ESD-TR- 73-51, vol 1, ESD/AFSC, Hanscom AFB, Bedford, Mass., Oct. 1972 (NTIS AD-758 206) .

[BIAR05] M. Bishop and H. Armstrong, "Uncovering Assumptions in Information Security," Proceedings of the Fourth World Conference on Information Security Education pp. 223-231, May 2005.

[BENS00] Benson, Robert L. The Venona Story. Center for Cryptologic History, National Security Agency, Ft. Meade, MD. pp 26-27, undated but prior to 2000. Available at: https://www.nsa.gov/about/cryptologic-heritage/historical-figures-publications/publications/coldwar/assets/files/venona_story.pdf

[BISH12] M. Bishop, "Computer Security:Art and Science," Addison-Wesley, 2012, ch.13.

[ECK63] Eckberg, C. R. (1964). WS-133B Fault Tree Analysis Program Plan. Seattle, WA: The Boeing Company. D2-30207-1. Available at: http://www.dtic.mil/get-tr-doc/pdf?AD=AD0299561

[GLW14] Goodenough, J.H.F Lipson and C.B. Weinstock, Arguing Security – Creating Security Assurance Cases, US-CERT, 2001, updated 2014. Available at https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/arguing-security-creating-security-assurance-cases

[LHM84] Landwehr, C.E., C.L. Heitmeyer, and J. D. McLean. A Security Model for Military Message Systems. *ACM Trans. on Computer Systems*, Vol. 2, No. 3, August, 1984, pp. 198-222.

[LBBN05] T. Levin, T. Benzel, G. Bhaskare, T. Nguyen, P. Clark, and C. Irivne, "Design principles for security," 2005.

[LOSC98] Loscocco, P. A., S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, and J. F. Farrell. The

inevitability of failure: The flawed assumption of security in modern computing environments. In Proc. Nat'l Info. Sys. Sec. Conf., pages 303--314, October 1998. Available at https://www.nsa.gov/resources/everyone/digital-media-center/publications/research-papers/assets/files/the-inevitability-of-failure-paper.pdf

[PMMSF02] Park, J., A. Moore, B. Montrose, B. Strohmayer, J. Froscher. *A Methodology, A Language, and a Tool to Privde Information Security Assurance Arguments*. NRL Memorandum Report NRL/MR/5540—02-8600, Feb. 2002.

[RMO16] Ross, R., J. McEvilly, J.C. Oren. System Security Engineering. NIST Special Publication 800-160, November 2016. Available at: https://doi.org/10.6028/NIST.SP.800-160

[SAND09] Sandia National Laboratories. The Information Design Assurance Red Team. Available at http://www.idart.sandia.gov/methodology/IDART.html

[SASC75] J.H. Saltzer, M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 1975

[SHF01] G. Stoneburner, C. Hayden, and A. Feringa, NIST Special Publication 800-27, "Engineering Principles for Information Technology Security (A Baseline for Achieving Security) Revision A," Booz-Allen-Hamilton Inc., McLean, VA, 2001.

[TRIP17] Trippel, Timothy, Ofir Weisse, Wenyuan Xu, Peter Honeyman, Kevin Fu, "WALNUT: Waging Doubt on the Integrity of MEMS Accelerometers with Acoustic Injection Attacks," ESORICS 2017, Paris, France.

[WEIS73] Weissman, C., "System Security Analysis/Certification Methodology and Results," SP-3728, System Development Corp., Santa Monica, Calif., Oct. 1973.

[WEIS94] Weissman, C., "Penetration Testing," Information Securitv Essavs, Abrams, M.D., S. Jajodia, H. Podell, eds., IEEE Computer Society Press,1994. Available at: https://www.acsac.org/secshelf/book001/11.pdf

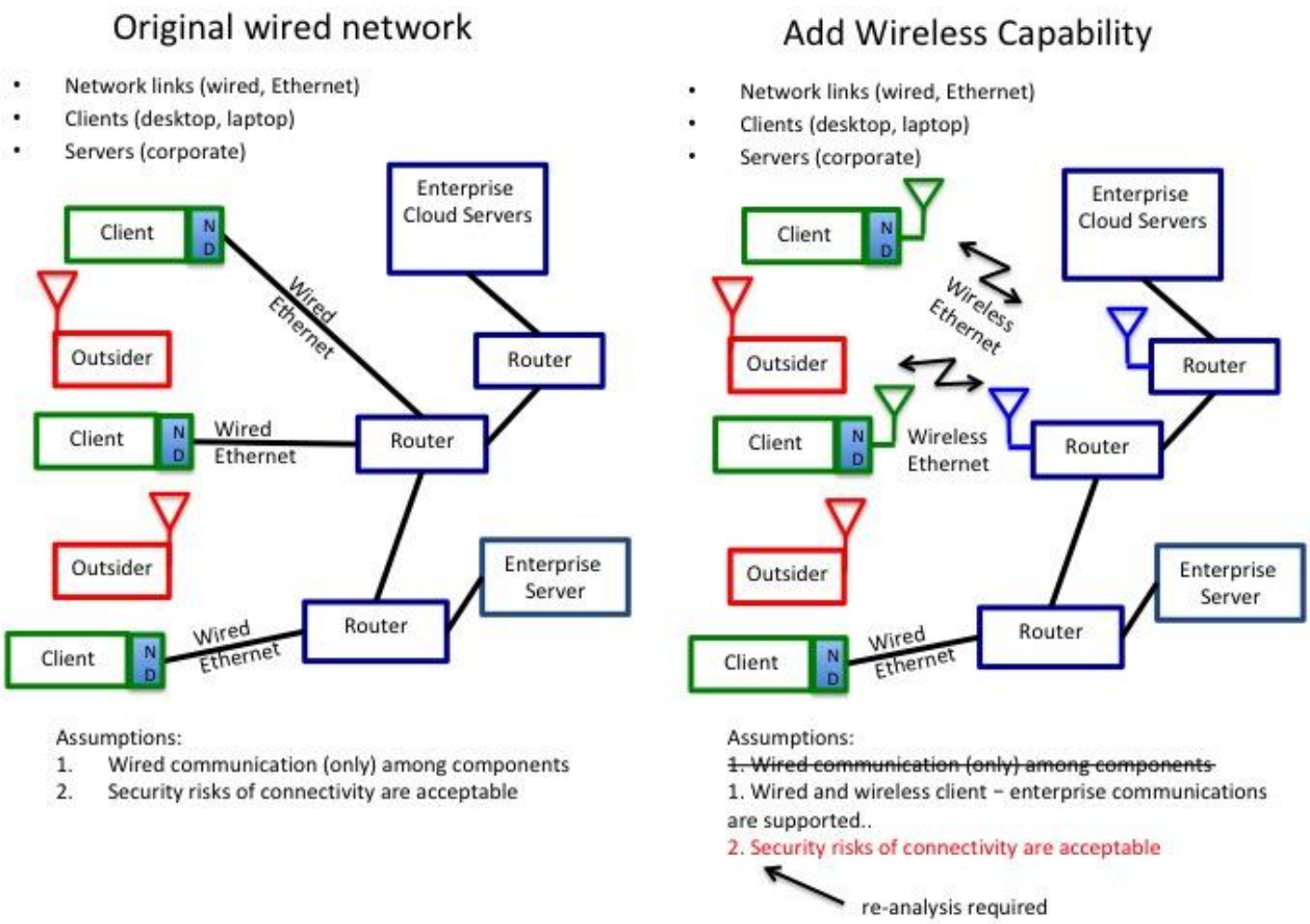

Figure 1. Example System: Addition of wireless connectivity to existing wired network.

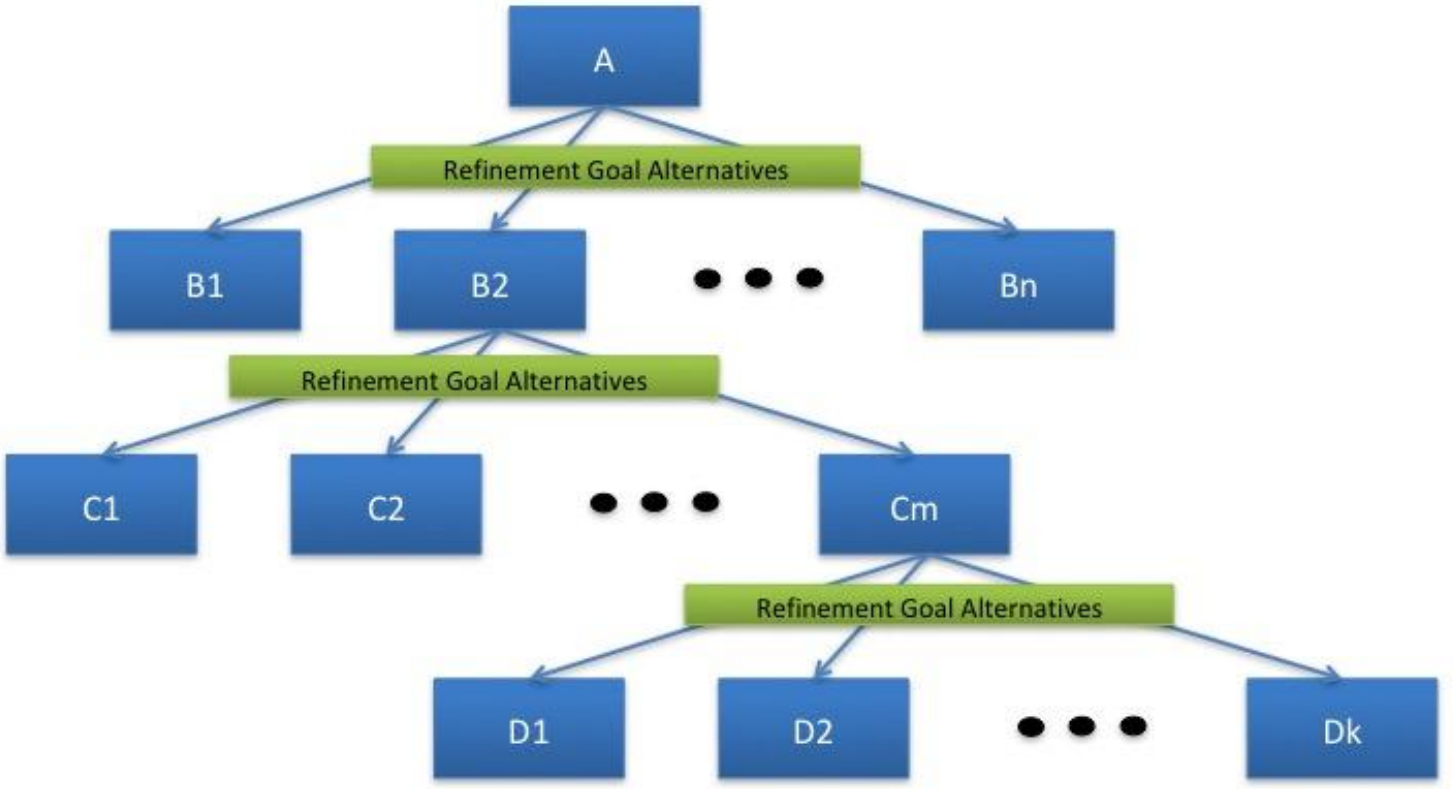

Figure 2. Example design tree created from following design process.

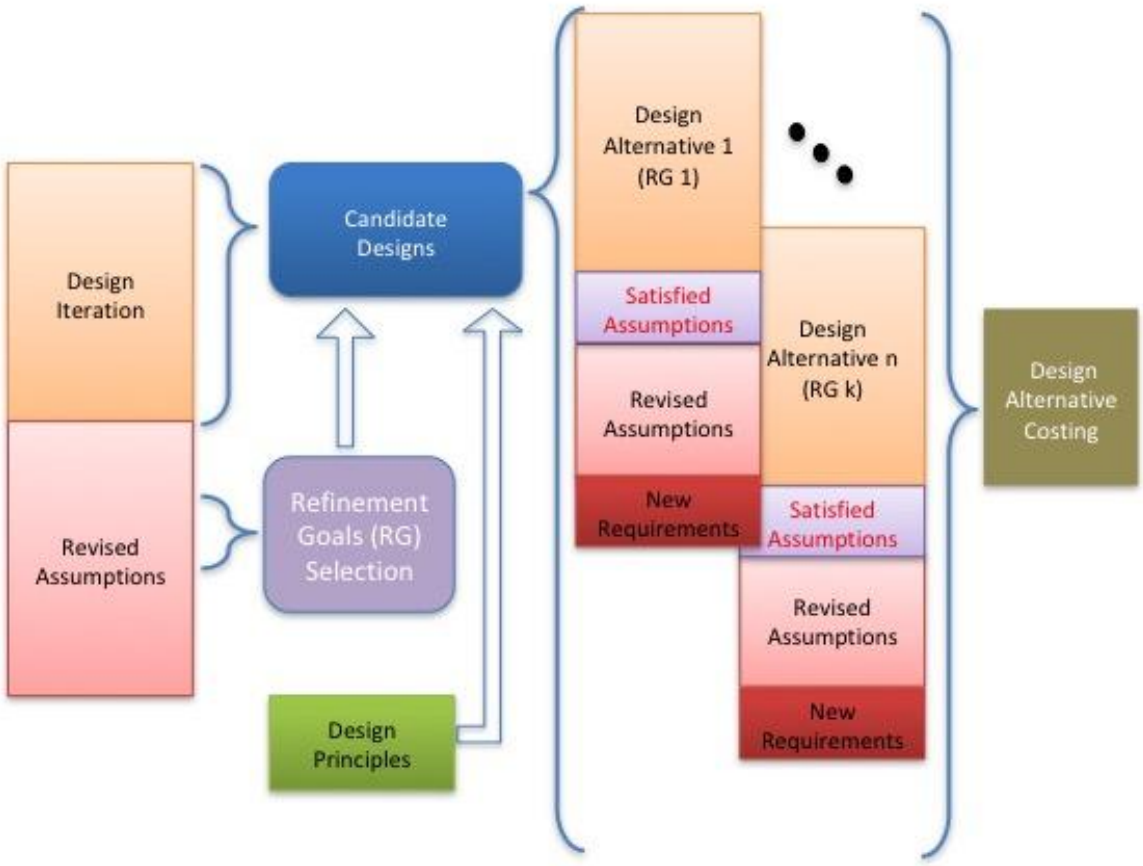

Figure 3. An example node from a design tree showing children nodes resulting from an iteration of the design process. The nodes are annotated to show how identified assumptions drive the process and are revised with each new candidate design.

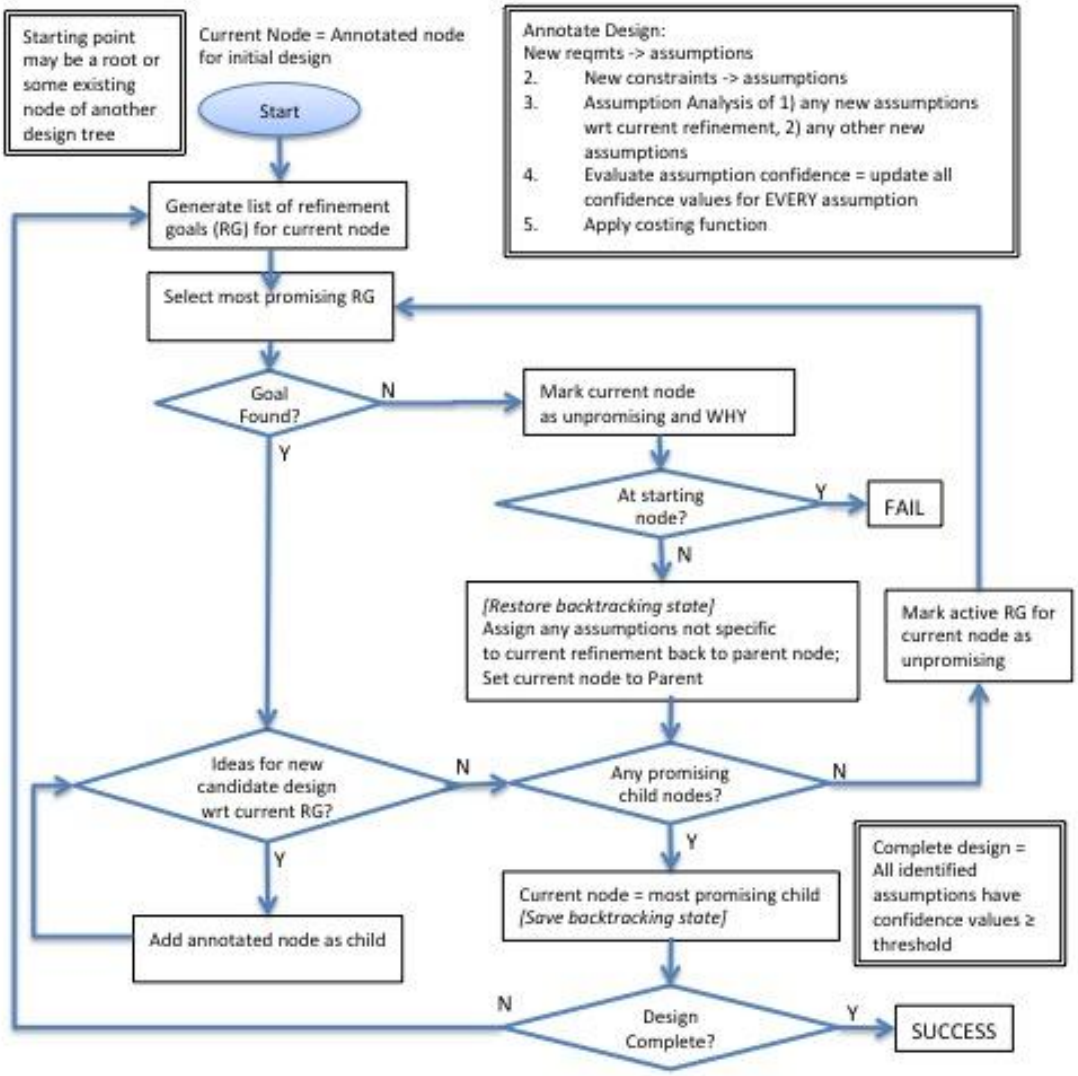

Figure 4. Flow chart describing the entire process.