Computational Evaluation of Software Security Attributes

Gwendolyn H. Walton
Florida Southern College and
Software Engineering Institute/CERT
Carnegie Mellon University

Thomas A. Longstaff
Johns Hopkins University
Applied Physics Laboratory

Richard C. Linger
Software Engineering Institute/CERT
Carnegie Mellon University

Abstract

In the current state of practice, security properties of software systems are typically assessed through subjective, labor-intensive human evaluation. Moreover, much of the quantitative security analysis research to date is characterized by the development of approximate solutions and/or based on assumptions that severely constrain the operational utility of the results. In order to achieve a dramatic increase in maturing the discipline of software security engineering, a fundamentally different approach to analysis and evaluation of security attributes is required. The computational security attributes (CSA) approach to software security analysis provides a new approach for specification of security attributes in terms of data and transformation of data by programs. This paper provides an introduction to the CSA approach, provides behavioral requirements for several security attributes, and discusses possible application of the CSA approach to support analysis of security attributes during software development, acquisition, verification, and operation.

1. Introduction

Dynamic security analyses are necessary if an organization hopes to keep pace with the potential impacts from ongoing program maintenance and system evolution, changing threats in the operational environments, and changes in organizational security strategies. However, in the current state of practice, security properties of software systems are typically assessed through subjective, labor-intensive human evaluation.

The behavior of users and administrators, the operating environment, and data transmission mechanisms are non-deterministic. Researchers have typically addressed this non-determinism by using a variety of analytical tools such as model checking, concurrent sequential process modeling, and rule-based systems to model, analyze, and verify security protocols. In recent years, several research threads have emerged that address security metrics and quantitative and qualitative analysis and evaluation of security attributes. Much of the security attribute analysis research to date is characterized by the development of approximate solutions and/or based on assumptions that severely constrain the operational utility of the results. In order to achieve a dramatic increase in maturing the discipline of software security engineering, a fundamentally different approach to analysis and evaluation of security attributes is required.

The CSA approach to software security analysis provides theory-based foundations for precisely defining and computing security attribute values. The translation of a static security property expressed as an abstraction of external data to dynamic behavior of a program expressed in terms of its data and functions is key to the CSA approach to verification of behavior that meets a specific security property. The ultimate goal of the CSA research is to develop mathematical foundations and corresponding automation to permit both rigorous evaluation and improvement of the security attributes of software during development and real-time evaluation of security performance during operation.

2. The CSA Analysis Approach

The CSA approach to security attribute analysis, illustrated in Figure 1, consists of three steps:

- *Define required security behavior*: Specify security attributes (such as authentication, non-repudiation, etc.) in terms of required behavior during execution expressed in terms of data and transformation of data by programs.
- *Calculate program behavior:* Create a behavior database using the new technology of function extraction (FX) that contains the specification for

978-0-7695-3450-3/09 $25.00 © 2009 IEEE

the complete "as built" functional behavior of the code.

- *Compare program behavior to required security behavior:* Compare the computed behavior database with required security attributes to verify whether the software meets its security requirements.

2.1. Define required security behavior

Requirements for security attribute behavior must explicitly define expected behavior of code in all circumstances of interest. Thus, the requirements for security attribute behavior must include a minimal definition of required behavior for all inputs of interest to the security attributes, including desired inputs (for example, an authenticated user id) and undesired inputs (for example: an unknown user id). Usage environment conditions related to security attributes are specified in the same manner as inputs to the system. For example, availability of the network might be specified by a Boolean value that indicates whether or not the network is currently available. Security successes and failures are also specified in terms of data. For example, system control data can be used to indicate whether the current user has been authenticated using a trusted authentication mechanism.

The level of abstraction at which a security attribute is specified can depend on the specific situation. For example, if all available data transmission mechanisms have previously been certified to be trusted, the security attribute requirements would not need to include details about data transmission. If there is one trusted data transmission mechanism, X, and one or more data transmission mechanisms that may not be trusted, the security attribute requirements could specify that all data transmissions will be performed using X. If none of the data transmission mechanisms have been previously certified as trusted, the security attribute requirements will need to include required control data effects for transmission security.

A "never responded" and "no output" case for each external function call of interest must be considered, including a definition of correct behavior in the case of intentional and unintentional aborts and hangs. In addition, security attribute requirements may specify a specific order in which certain functions can be called. For example: user authentication must occur before any data access functions can be called.

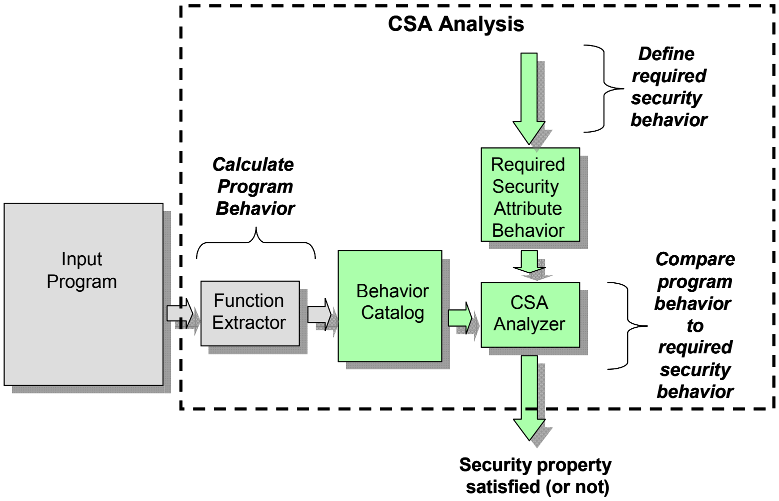

CSA Analysis
Define required security behavior
Calculate Program Behavior
Required Security Attribute Behavior
Input Program
Function Extractor
Behavior Catalog
CSA Analyzer
Compare program behavior to required security behavior
Security property satisfied (or not)

Figure 1. CSA analysis

The requirements may specify that a certain set of data transformations always occur. For example, the control data that indicates that data transmission is secure must always be set by the trusted data transmission mechanism. The requirements may specify that a certain set of data transformation must never occur. For example, the control data that indicates that data transmission is secure must never be set by any code other than the trusted data transmission mechanism. If a user is authorized to only access specific data, the requirements may state that no data transformations other than a specific set of data transmissions can occur.

Any amount of traceability and control can be specified in the requirements for security attribute behavior. For example, the requirements may include specifications of bounded behavior. (i.e., the execution will proceed so long as the behavior is within a specified domain). Specifications for trusted mechanisms can be included in the requirements as a constraint. For example, one might specify that “a call to method XXX that returns a value of y is sufficient to satisfy a requirement that a trusted mechanism must be used to perform authentication.”

A behavioral approach also supports dealing with some uncertainty in the specification of the security attribute requirements. For example, a security requirement might state that the code must guarantee security properties modulo some defined value. Some constraints might be specified using a stochastic component. For example, “The response history of component X must indicate that the component was available at least 94% of the time.”

2.2. Calculate program behavior

In this context, b*ehavior* means the as-built net functional effect of a program (and its constituent control structures and instructions) from entry to exit. *Behavior calculation* means to create a behavior database that specifies the behavior of the code.

Software with unknown behavior cannot be certified as secure. Thus, a behavior database containing all externally observable behaviors is an essential input to security attribute analysis. For software developed in-house, the behavior database might be derived from the specification from which the software is developed and verified. For procured software, a complete behavior database might be part of the acquisition requirements. For existing software for which a behavior database is not available, intensive manual effort to understand the code may be necessary.

The CERT® organization of the Software Engineering Institute is developing Function Extraction (FX) theory and technology for understanding program behavior. [1, 4, 5, 7] The objective of this work is to compute the functional behavior of programs with mathematical precision to the maximum extent possible. The FX approach to program comprehension eliminates the need to study and understand code by manual means. The ultimate goal of the FX technology project is to develop theory and tools to automatically calculate program behavior based on precise semantics of the programming language, producing databases that represent the net behavior of programs in terms of disjoint conditional concurrent assignment statements.

FX technology is based on a function-theoretic view that treats programs and their parts as rules for mathematical functions or relations subject to compositional analysis to derive net effects. CERT® researchers are currently developing an FX system to support function extraction of assembly programs using complete semantics for Intel 32-bit assembly language code.

Behavior databases produced by the FX system provide several advantages for security attribute analysis:

- A behavior database lends itself to query and thus can facilitate risk assessments and other query-driven analyses of security attributes.
- Because the behavior database supports examination of the functional transformations on data and does not require examination of the state space, this approach is more scalable than formal methods approaches. For example, when formal methods are used to examine the integrity attribute, the entire state space must be computed. Integrity exists if the system can never map outside the state space. In contrast, with CSA and the use of behavior databases, only the calculated behavior is of interest. Because only externally observable behavior is relevant to security attribute analysis, only the behavior at the boundary of the system is required for the calculation. Thus, there is no need to calculate the internal behavior of trusted components, and there is no need to expose the entire internal state space.
- If a property can be expressed in code, FX technology can be used to determine if that property holds within a program.
- FX technology can be used to describe the behavior of a function that combines two behaviors. Thus, FX can be used to give the exact description of the composition of the behaviors.
- Corporate policies and “intentions” can be defined in a behavioral format in advance of the design of the architecture and code. Queries to examine the behavior database for the presence or absence of desired properties can be developed in parallel with design of the architecture. If pre- and post-conditions are defined behaviorally, they can be used to evaluate all artifacts (i.e., the behavioral databases, not just the code.)

2.3. Compare program behavior to required security behavior

The translation of a static security property expressed as an abstraction of external data to dynamic behavior of a program expressed in terms of its data and functions is key to the verification of behavior that meets a specific security property. Verification that a security property is satisfied requires verification of both the data at rest (i.e. the control data values) and the data in motion (i.e., the mechanisms used to perform the data transformations). Some common tasks to verify data at rest include checking to make sure that a specific task (for example, an audit task) will always be carried out to check the contents of a specific control data structure.

The CSA approach to security attribute verification includes the use of constraints and boundary conditions to make any assumptions explicit. People and process issues can be handled by the CSA approach by using assumptions and constraints as part of the behavior databases. In addition, behaviors can embody requirements for a given security architecture. Thus, the attribute verification process will expose security vulnerabilities, making it easier to address evolution of code, environment, use, etc.

3. Behavioral requirements of security attributes

The behavioral requirements for security attributes can be completely described in terms of data items and constraints on their processing. The processing can be expressed, for example, as logical or quantified expressions or even conditional concurrent assignments, which can be mechanically checked against the calculated behavior of the software of interest for conformance or non-conformance with security attribute requirements.

Security attributes are inter-related. For example, authentication relies on non-repudiation as a strict subset of the required security behavior. Thus, quantitative reasoning about the attributes requires an integrated set of security attribute definitions.

3.1. Use of a trusted mechanism

Each security attribute requires the use of one or more trusted mechanisms that are implemented in software components. A behavior database is needed for each mechanism to describe all cases of behavior in terms of the mechanism’s data and the transformations it carries out on that data. This behavior database is analyzed in terms of control data and evidence data to ensure the following:

- The trusted mechanism sets the values of control data, which indicates whether the mechanism executed correctly.
- If control data indicates that the mechanism executed correctly, there exists evidence data to show that the data transformation was performed in a manner that satisfies the defined security specification.

Note that the implementation of a security attribute may include a trusted third party to acquire, authenticate, and adjudicate evidence of transactions. However, for the purposes of behavior specification of security attributes, the specific mechanism and actors are not relevant. All that is needed is a precise specification of the data and the transformations of the data and any constraints concerning those transformations.

3.2. Trusted data transmission

Figure 2 illustrates the requirements for trusted data transmission:

- A trusted data transmission mechanism is used for all data transmissions. If the mechanism is not available or the mechanism fails, the requirement fails.
- No mechanism outside this trusted data transmission mechanism sets the value of the control data that indicates whether the data transmission mechanism executed correctly.

Figure 3 illustrates the process for determining whether data transmission security properties are satisfied by the data transmission components of a system. This process consists of the application of the CSA analysis process illustrated in Figure 1, where the input is the data transmission components of the system, and the output is a determination of whether the data transmission security requirements have been satisfied. Similarly, the process for determining whether the remainder of the system can modify the control data set by the data transmission components consists of application of the CSA analysis process of Figure 1, where the input to the process is the software for the entire system, and the output is a determination of whether the control data set by the data transmission components can be modified by any other part of the system.

The process for determining whether the remainder of the system can modify the control data set by the data transmission components, illustrated in Figure 4, consists of the application of the CSA analysis process of Figure 1, where the input to the process is the software for the entire system, and the output is a determination of whether the control data set by the data transmission components can be modified by any other part of the system.

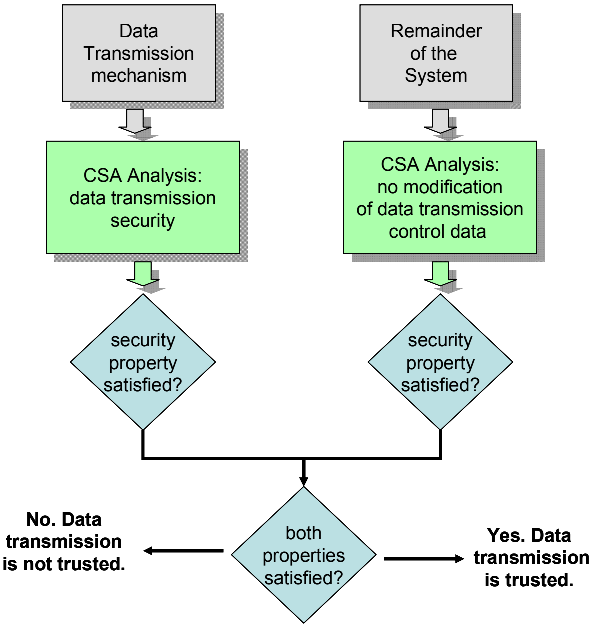

Data Transmission mechanism
Remainder of the System
CSA Analysis: data transmission security
CSA Analysis: no modification of data transmission control data
security property satisfied?
security property satisfied?
both properties satisfied?
No. Data transmission is not trusted.
Yes. Data transmission is trusted.

Figure 2. Requirements for trusted data transmission

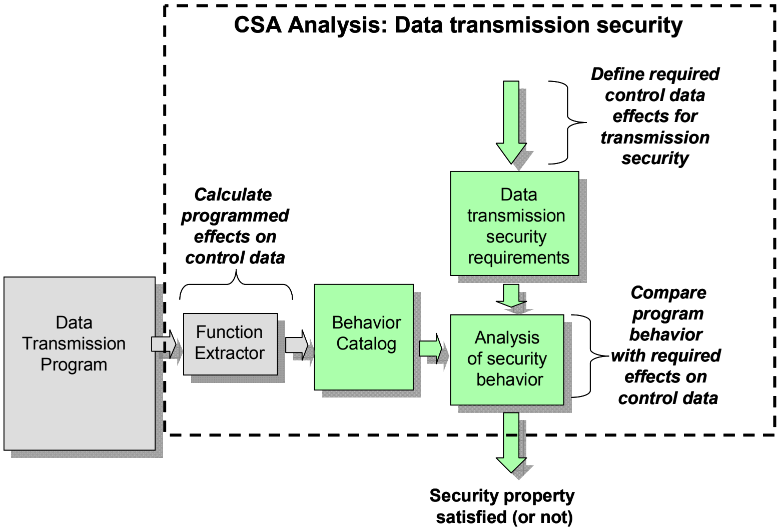

CSA Analysis: Data transmission security
Define required control data effects for transmission security
Data transmission security requirements
Calculate programmed effects on control data
Data Transmission Program
Function Extractor
Behavior Catalog
Analysis of security behavior
Compare program behavior with required effects on control data
Security property satisfied (or not)

Figure 3. CSA analysis of data transmission security

As illustrated in Figure 4, to verify that a data transmission mechanism is trusted, it is necessary to verify the data that provides the evidence related to data transmission. For example, the specification for the data that provides evidence of valid data transmission may describe the mechanism by which each data message output incorporates a shared (between sender and receiver) data item that can be used to verify that the transformation worked correctly. Assignments to this shared data must not be reversible (i.e., guaranteed encryption.) As another example, suppose the behavior databases for all of the code have been examined to verify that all data transmissions in the system occur as a result of calls to function YYY. To verify the necessary security properties for data transmission, it is necessary to examine function YYY’s behavior database to determine the net effect of the data transformations related to any conditions for which invalid data transmission could occur.

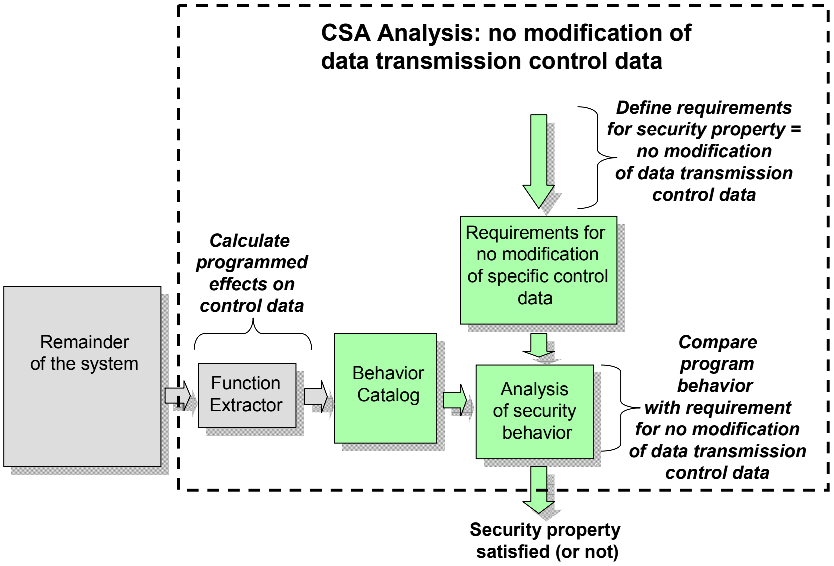

CSA Analysis: no modification of data transmission control data
Define requirements for security property = no modification of data transmission control data
Requirements for no modification of specific control data
Calculate programmed effects on control data
Remainder of the system
Function Extractor
Behavior Catalog
Analysis of security behavior
Compare program behavior with requirement for no modification of data transmission control data
Security property satisfied (or not)

Figure 4. CSA analysis of modification of data transmission control data

3.3. The authentication security attribute

Authentication requires that a trusted user has been bound to the behavior. That is, the system will only allow the requested program to be executed if the user has previously been determined to be a trusted user. To verify authentication, one must examine the net effects on the control data related to authentication: verify the data that provides evidence that the binding took place, and verify that this evidence data was not changed before completion of any operation that required authentication. The requirements for authentication are as follows:

- A trusted data transmission mechanism is always used for every data transmission.
- A trusted identification mechanism is always used to provide proof of a user’s identification. Note that the “user” to be identified may be a person, a process, a program, or other entity.
- A trusted binding mechanism is always used to bind user data (user identification, password, or other information to confirm the identification, and the system information that provides the proof of the identification) to an execution environment.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly and that indicates the status of the bound data.
- If any of the above requirements or mechanisms fails, authentication fails.

The mechanism for using CSA to determine whether the data transmission is trusted was discussed earlier. The application of the CSA approach for analysis of the user identification mechanism and the user binding mechanism proceeds along the same lines as Figures 3 and 4 to determine whether each of these mechanisms is trusted.

3.4. The authorization security attribute

Authorization requires that a user has the right to do the requested process. To verify that an authorized operation took place, one must examine the net effects on the control data to verify that it provides evidence that authorization occurred before the operation, and that the evidence data for the authorization was not changed before that operation completed. The requirements for authorization are as follows:

- A trusted authentication mechanism (subsection 3.3) is always used to authenticate the user. Note that this requirement includes a requirement for trusted data transmission and authentication.
- A trusted lookup mechanism is always used to determine that the user has the right to complete the specified request.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly and that identifies the authorized user and the scope of the authorization.
- If any of the above requirements or mechanisms fails, authentication fails.

Analysis of the authentication mechanism was discussed in the previous subsection. Analysis of the lookup mechanism applies the CSA approach by proceeding along the same lines as Figures 3 and 4 to determine whether this mechanism is trusted.

3.5. The non-repudiation security attribute

Non-repudiation of data transmission requires that neither the sender nor the recipient of the data can later refute their participation in the transaction. Non-repudiation of changes to a dataset requires that the means for authentication of changes cannot later be refuted. For the purposes of this discussion we treat data change as a special case of data transmission, where receipt of the data transmission includes making and logging the requested change to the dataset. To verify non-repudiation, one must examine the net effects on the control data related to non-repudiation. The requirements for every data transmission that is subject to non-repudiation are as follows:

- Trusted authorization (subsection 3.3) is always used for sender, receiver, and the scope of any data changed or transmitted. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users, and trusted authorization of users and processes for the specific data scope.
- Trusted binding is used to bind the sender to the data sent and to bind the receiver to data received.
- The authorization, binding, and data transmission are handled as a single atomic operation within the boundary of the authorized secure process.
- A trusted mechanism is always used to provide traceability and audit. This trusted mechanism ensures data persistence of the audit data so the means of authentication and the data transmission cannot later be refuted.
- Every data transmission is preceded by an absolute definition of the data and identification that binds the data to the sender.
- Every data receipt is preceded by an absolute definition of the data and identification that binds the data to the recipient.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of any of the control data generated by the trusted mechanisms.
- If any of the above requirements or mechanisms fails, non-repudiation fails.

Analysis of the authorization mechanism was discussed in the previous subsection. Analysis of the binding mechanism and the traceability and audit mechanism applies the CSA approach by proceeding along the same lines as Figures 3 and 4 to determine whether each of these mechanisms is trusted.

3.6. The confidentiality security attribute

Confidential data access or confidential data transmission requires that unauthorized disclosure of one or more specific data items will not occur. Confidentiality is often described in terms of a security policy that specifies the required strength of the mechanisms that ensure that the data cannot be accessed outside the system. For example, the security policy may require verification that approved encryption mechanisms are used for the output. To verify confidentiality, one must examine the net effects on the control data related to confidentiality. The requirements for confidentiality are as follows:

- A trusted non-repudiation mechanism (subsection 3.5) is always used to process requests for confidential data access and confidential data transmission. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes, and trusted authorization of users and processes for the particular data scope.
- A trusted mechanism is always used to ensure that the data cannot be read outside the system.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of any of the control data set by the trusted mechanisms.
- If any of the above requirements or mechanisms fails, the request for confidential data access or confidential data transmission fails.

Analysis of the non-repudiation mechanism was discussed in the previous subsection. Analysis of the data access mechanism applies the CSA approach by proceeding along the same lines as Figures 3 and 4 to determine whether this mechanism is trusted.

3.7. The privacy security attribute

Privacy requires that an individual has defined control over how his/her information will be disclosed. To verify privacy, one must examine the net effects on the control data related to privacy. The requirements for privacy are as follows:

- A trusted confidentiality mechanism (subsection 3.6) is always used for all accesses of a user's personal information. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes, trusted authorization of users and processes for the specific data scope, and trusted non-repudiation for access to a user's personal information
- All access to a user's personal information satisfies an existing privacy and confidentiality policy that includes control data that defines the scope of access for each user.
- A trusted non-repudiation mechanism (subsection 3.4) is used for all changes to the control data that defines the scope of access for each user. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes and scope of data
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of any data set by the trusted mechanisms.
- If any of the above requirements or mechanisms fails, the request for confidential data access or confidential data transmission fails.

Analysis of the confidentiality and non-repudiation mechanisms was discussed in previous subsections. Analysis of the access mechanism applies the CSA approach by proceeding along the same lines as Figures 3 and 4 to determine whether this mechanism is trusted.

3.8 The integrity security attribute

Integrity requires that authorized changes are allowed, changes must be detected and tracked, and changes must be limited to a specific scope. Integrity is defined as a property of the object, not of a mission. To verify integrity, one must examine the net effects on the control data related to integrity. That is, one must be able to: isolate the object, isolate all the behaviors that can modify the object, detect any modifications to the data, and ensure that all transformations of the data across the object are within the pre-defined allowable subset. The requirements for integrity are as follows:

- A security policy exists that describes the scope of allowed changes as an invariance function: certain data transformations must hold; others must never hold.
- If the security policy data is changed to remove any element from the allowable subset, integrity of the data fails.
- A trusted non-repudiation mechanism (subsection 3.4) is always used for changes to data and changes to policy to ensure that all changes to the security policy and changes to data are performed using a trusted non-repudiation mechanism. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes, trusted authorization of users and processes for the specific data scope, and trusted non-repudiation for changes to the data. Every authorization for data changes must be restricted to the allowable subset as defined in the security policy.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of the control data set by any of the trusted mechanisms.
- If any of the above requirements or mechanisms fails, integrity of the data fails.

Analysis of the non-repudiation mechanism was discussed in a previous subsection. Analysis of the data change mechanism applies the CSA approach by proceeding along the same lines as Figures 3 and 4 to determine whether this mechanism is trusted.

3.9. The availability security attribute

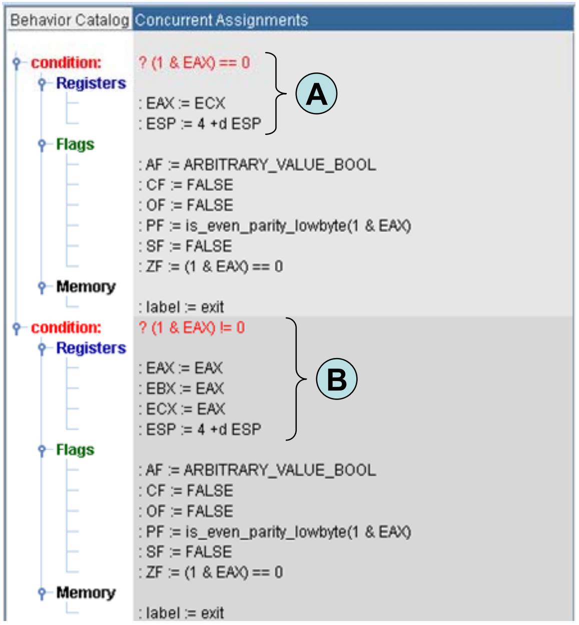
Availability requires that a resource is usable during a given time period, despite attacks or failures. To verify availability, one must examine the net effects on the control data related to availability. To avoid having to consider temporal properties, one can specify non-availability rather than availability. (i.e., specify under what conditions the program's behavior database do not apply.) Liveness properties are translated to the behavioral characteristics that are evident when the system is actually available. Specific behaviors associated with non-availability due to denial of service must also be specified.

4. Miniature illustration of CSA with FX

Consider the screen image of behavior generated by the CERT® Function Extraction for Malicious

Code (FX/MC) prototype, as shown in Figure 5. This system is under development by CERT® to compute the behavior of malicious code expressed in Intel assembly language. FX/MC produces the net behavior of programs in terms of disjoint conditional concurrent assignment statements. That is, each case of behavior is expressed by a condition and a set of non-procedural assignments that define the final values of registers, flags, and memory locations on exit from an assembly language program.

This notional example illustrates application of the CSA approach to determine whether a security requirement is satisfied. Suppose that the specification of requirements for a security attribute includes a constraint that the value of the second argument to each call to function XXX must not be equal to 4. A constraint such as this, expressed in terms of concrete data operations and values, could be part of the requirements specification for any of the security attributes discussed above.



```
condition:        ? (1 & EAX) == 0
   Registers
                  : EAX := ECX
                  : ESP := 4 +d ESP
   Flags
                  : AF := ARBITRARY_VALUE_BOOL
                  : CF := FALSE
                  : OF := FALSE
                  : PF := is_even_parity_lowbyte(1 & EAX)
                  : SF := FALSE
                  : ZF := (1 & EAX) == 0
   Memory
                  : label := exit
condition:        ? (1 & EAX) != 0
   Registers
                  : EAX := EAX
                  : EBX := EAX
                  : ECX := EAX
                  : ESP := 4 +d ESP
   Flags
                  : AF := ARBITRARY_VALUE_BOOL
                  : CF := FALSE
                  : OF := FALSE
                  : PF := is_even_parity_lowbyte(1 & EAX)
                  : SF := FALSE
                  : ZF := (1 & EAX) == 0
   Memory
                  : label := exit
```

Figure 5: Computed behavior

Imagine that the behavior of Figure 5 defines the behavior of a fragment of code that executes immediately before the program calls function XXX with the second argument equal to the value stored in register EAX. The computed behavior highlighted as section A is the behavior with respect to register values for the condition *?(1 & EAX) == 0*). This condition means "if the value of register EAX at the beginning of this fragment of code is even." As shown on the figure, if this condition is true, the value of register EAX after executing this fragment of code is equal to the initial value of register ECX, and therefore the value of the second argument to function XXX will be the initial value of register ECX. In contrast, the computed behavior highlighted as section B is the behavior with respect to register values for the condition *?(1 & EAX) != 0*). This condition means "if the initial value of register EAX is odd." As shown on the figure, if this condition is true, the value of register EAX is unchanged and therefore the value of the second argument to function XXX will be the initial value of register EAX. Thus, the security constraint is not satisfied because, under either of these conditions, it is not possible to certify that the value of the second argument will never be equal to 4.

Note that, using FX technology, this analysis can take place in seconds through computational automation, thereby eliminating the need to study and understand the code by manual means.

5. Implications

The behavioral requirements for security attributes provided in the previous section are consistent with the general descriptions of the eight security dimensions (access control, authentication, non-repudiation, data confidentiality, communication flow security, data integrity, availability, and privacy), as contained in standard ISO/IEDC 18028-2, *Information Technology - Security Techniques - IT Network Security - Part 2: Network Security Architecture* [3] and originally provided in the Bell

Labs Security Framework [6]. We expect that the CSA approach to security attribute analysis will be very supportive of work to address the challenges of security management and approaches to secure an enterprise and regulatory mandates (Discussions of the challenges of security management, are provided in [2, 8].)

Advantages of the CSA approach include:

- A rigorous method is used to specify security attributes in terms of the actual behavior of code and to verify that the code being evaluated is correct with respect to security attributes.
- The specified security behaviors can provide requirements for the security architecture.
- Traceability capabilities can be defined and verified outside of the code being evaluated.
- Vulnerabilities can be well understood, making it easier to address evolution of code, environment, use, and users.

CSA technology can address specification of security attributes of systems before they are built, specification and evaluation of security attributes of acquired software, verification of the as-built security attributes of systems, and real-time evaluation of security attributes during system operation.

It should also be possible to verify that the behavioral characteristics relating to security properties are consistent with a global security policy (e.g., running the code in the context of a system with an expressed security policy). The use of a system such as FX to perform this level of validation would require a behavioral specification of the global security policy in terms of system behavior, which could then be automatically checked against program behavior. As behavioral patterns scale through different layers of abstraction, high-level security behaviors can be expressed as a straight-forward composition of program behavior.

6. Open Questions

Behavioral requirements for concurrency and parallel system issues must be addressed. In addition, because computational security attributes and function extraction are emerging technologies, there has been limited experience in applying the technology to large systems. The computational effort involved in analyzing large, complex components and in analyzing large numbers of component compositions to yield a system security specification remains to be studied. However, it is clear that the CSA approach can be effective in addressing state space explosion while yielding complete, correct answers.

7. Acknowledgement

CERT is a registered trademark and service mark of Carnegie Mellon University.

8. References

[1] R.W. Collins, A.R. Hevner, G. H. Walton, R.C. Linger, “The impacts of function extraction technology on program comprehension: A controlled experiment”, *Information and Software Technology,* 50(2008) 1165-1179. Elsevier, 2008.

[2] A.K. Gupta, U. Chandrashekhar, S. Sabnis, and F.A. Bastry, “Building Secure Products and Solutions”, Bell Labs Technical Journal, 12(3), p 23-38. Wiley Periodicals, Inc. 2007.

[3] International Organization for Standardization and International Electrotechnical Commission, *Information Technology - Security Techniques - IT Network Security - Part 2: Network Security Architecture, ISO/IEC 18028-2 ,* Feb. 2006.

[4] R. Linger, M. Pleszkoch, L. Burns, A. Hevner, G. Walton. “Next generation software engineering: Function extraction for the computation of software behavior. *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS40)*, Hawaii. IEEE Computer Society Press, Los Alamitos, CA 2007.

[5] R. Linger, S. Prowell, R. Bartholomew, L. Burns, T. Daly. “Funcation extraction: automated behavior computation for aerospace software verification and certification” *Proceedings of the American Institute of Aeronautics and Astronautics (AIAA) Infotech & Aerospace 2007 Conference*, California. IEEE Computer Society Press, Los Alamitos, CA. May 2007.

[6] A.R. McGee, S.R. Vasireddy, C. Xie, D.D. Picklesimer, U. Chandrashekhar, and S.H. Richman. “A Framework for Ensuring Network Security”, *Bell Labs Technical Journal,* 8(4), pages 7-27. Wiley Periodicals, Inc. 2004.

[7] M. Pleszkoch and R. Linger, “Improving Network System Security with Function Extraction Technology for Automated Calculation of Program Behavior,” *Proceedings of 37th Hawaii International Conference on System Sciences*, Hawaii, January, 2004, IEEE Computer Society Press, Los Alamitos, CA, 2004

[8] S. Sabnis, U. Chandrashekhar, and F. Bastry. “Challenges of Securing an Enterprise and Meeting Regulatory Mandates”, P*roceedings of the 12th International Telecommunications Network Strategy and Planning Symposium, 2006. NETWORKS 2006.* Nov. 2006, pages 1-6. IEEE Computer Society Press, Los Alamitos, CA, 2002.