

# **Security and The “Zachman Framework”**

**By  
John A. Zachman  
© Copyright 2001 Zachman International**

All of a sudden, there seems to be an increased interest in the subject of “Security Architecture.” I would judge that this interest is being precipitated by the increased interest in Architecture in general, coupled with the Information Age characteristic of the “Powershift,”<sup>1</sup> where everybody has access to the same information at the same time. Security becomes important because, sometimes you simply don’t want to allow anybody or everybody to have access to any or all of your information (or other systems components, for that matter) at any time! Therefore, “Security Architecture!”

## **Assuming the BEST**

Let’s assume the best for a moment. Let’s assume that you took my advice 25 years ago and started working on Architecture, building “primitive models,” “Enterprise Architecture.” Let’s assume that over the last 25 years, that for your Enterprise (however you define “Enterprise”), you had gradually made ALL of those “primitive models” explicit, and had finally gotten to the place where all of the models were Enterprise-wide, all of them horizontally and vertically integrated, and all of them at excruciating level of detail and on top of that, you had gotten rid of all the “legacy.” That is, let’s assume the very best.

The elegance of having such an Enterprise Architecture is that administratively, change becomes simply changing a row in a table. The “Security Administrator” would change the table entries in the database where the Enterprise Architectural representations were stored as an Organization changed, or as a Role changed, or as a User changed, or as a User changed relative to a Role, or as a Role changed relative to a Data Entity, or as a Role changed relative to an Application Function, etc., etc.

“Security” would be a piece of cake! The only thing you would have to do is to positively identify who is sitting at the terminal (or whatever access device) ... (Column 4, Row 5 “Security Architecture”), because you could relate that “Identity” to a specific “User” which you would have already defined in Column 4, Row 4 (“Presentation Architecture”), which you could relate to a specific “Role” which you would have

---

<sup>1</sup> “Powershift” by Alvin Toffler. Bantam Books 1990.

already defined in Column 4, Row 3 (“Human Interface Architecture”), which you could relate to a specific “Organization” which you have already defined in Column 4, Row 2, (“Work Flow Model”), in which you would have already described all of the “Organizations” that are authorized to insert work into your Enterprise ... at excruciating level of detail.

Then across Row 2, The “Owner’s Row,” you would know what “Business Entities” (“Things,” Column 1) each “Organization” was authorized to touch, which “Business Processes” (Column 2) they were authorized to perform, which “Business Nodes” (Column 3) they were authorized to insert work from, which “Business Events” (Column 5) you would accept from them and which “Business Objectives” (Column 6) they were supporting.

Furthermore, you would know which “Roles” in Column 4, Row 3, the “Human Interface Architecture,” were performed by each “Organization” (from Row 2) at excruciating level of detail so across Row 3, the “Designer’s Row” you would know which “Data Entities” (Column 1) each “Role” could access, which “Application Functions” (Column 2) they could execute, which “I/T Components” (Column 3) they could employ, which “System Events” (column 5) they could initiate and which “Business Rules” (Column 6) constrained or guided them.

Then, you would also know which “Users” by name (at excruciating level of detail) in Column 4, Row 4, the “Presentation Architecture,” fulfilled which “Roles” (from Row 3) so therefore, across Row 4, the “Builder’s Row” you would know which “Tables” (databases in Column 1) each User could access, which “Computer Functions” (sub-routines in Column 2) they could execute, which “Hardware/Systems Software” (Column 3) they could use, which “Executes” (enter keys in Column 5) they could insert and which “Rule” (“Conditions/Actions” in Column 6) constrained or guided them.

Therefore, you would also know in Column 4, Row 5, the “Security Architecture,” the “Identity” of each “User” (from Column 4) which is the key to Security and the reason that I named Column 4, Row 5 “Security Architecture.” I probably don’t need to trace the relationships across Row 5 but for completeness, if you knew the “Identity” of the person (from Column 4, Row 5) at the access device, you would also know which “Fields” (Column 1) they had access to, which “Language Statements” (Column 2) they could execute, which “Addresses” (Column 3) they were authorized to access from, which “Interrupts” (Column 5) they could insert and which Rule “Sub-Conditions” (Column 6) constrained or guided them.

Actually, in practice, it probably would be wisdom to control the horizontal integration for Security purposes across the Rows at Row 3, the “logical view” (not Rows 4 or 5). That is, you would control Security by the “Role” as it related to the other Columns because, the Row 4 “physical view” (technology-constrained view,” “Builder’s View”) changes at a higher frequency than the Row 3 “Logical View.” In this fashion, Security would be based on “Role” which is relatively stable rather than on “User” which is relatively dynamic or on the “User’s” “Identity” which is coupled with the “User.” That

is, the “Role” remains the same from a Security perspective, but different “Users” rotate through the same “Role.” Therefore, a substantial amount of change could be accommodated by inserting or deleting “Users” from the “Role” table in the Enterprise Database.

I probably don’t have to observe this, but you would have exhaustive cross- verification because down each Column you would have the relationships defined just like I have illustrated above with Column 4. For example, in Column 1, you would have defined which “Business Entity” (Row 2) was represented by which “Data Entity” (Row 3) which was stored in which “Table” (Row 4) which contained which “Fields” (Row 5). The same would apply to every other Column.

In fact, I just saw a security application prototype demonstration in Canada where the “Screen Format” (Column 4, Row 4) was customized for every specific “User” and the specific “User’s” “Screen” did not even display data that that “User” was not authorized to view, nor did the “Screen” display any application function options that were not authorized for that specific “User.” In fact, the “User” only had access to the “Screens” for which they were authorized. Security was controlled through the “Screen Formats,” built into the implementation and based on the “Role” within the Enterprise. So Security, in that case, was administered simply by moving specific “Users” into or out of “Roles” in the Enterprise Database.

The key to Security lies, once again, in the positive identification of the person accessing the Enterprise, whether they are manually accessing the Enterprise or accessing the Enterprise through automation. With biological means for identifying individuals becoming commercially viable, positive identification no longer rests solely on password mechanisms. With Enterprise Architecture and Positive Identification, all of the Security constraints are directly built into the Enterprise Architecture, by definition.

All of the above discussion about Security applies to Security *within* the Enterprise. If you are going to transmit data *outside* of the Enterprise, then Security would have to include some kind of encryption mechanism associated with the network operating system (Column 3, Row 4 “Technology Architecture”) (which is beyond the scope of this discussion). Ultimately, Security external to the Enterprise is also going to rest on positive identification of the recipients of the transmissions. In fact, if you are going to transmit data outside of, or allow outsiders to have access to the Enterprise, you are going to have to treat them as integral to (that is, part of) your Enterprise Architecture. Furthermore, given the sophistication of the surveillance technology available these days, any kind of transmissions, internal or external, is likely going to have to be encrypted.

### **Assuming the WORST**

Now ... let’s assume the worst for a moment. Let’s assume that over the last 50 years, no one paid any attention to Enterprise Architecture and you had built thousands of applications and thousands of databases and had thousands of users and there was no place to go to find out what the applications actually did, what data was actually in the

databases, who the users actually were, which applications they could run and what data they could access. And now, you want to control which users can run which applications and have access to which data. That is, assuming the very worst.

First, before you do anything else, you are probably going to have to figure out what the applications are actually doing, what the data is that is actually in the databases and who the actual users are. Maybe you can reduce the amount of labor by *assuming some risk* and lopping out wholesale numbers of applications and databases and users that are not (or that are *presumed* not to be) security sensitive. Then, with whatever is left over, you are going to have to get very definitive about what the applications are actually doing, what the data actually is and who the users actually are. The Security Administrator will have to decide which user will be authorized to run which application and/or have access to which data. I might add, no machine is going to do this for you. Only some person will be able to figure out what makes up the actual legacy and who is authorized for what data or what application function. This may be a somewhat daunting task if the users number in the thousands and the applications functions in the tens of thousands and the data (at the field level) in the hundreds of thousands.

However, having done all of that analysis, it occurs to me then that there are two possible approaches to Security; the “Systems-Centric Approach,” and the “User-Centric Approach.” Either you put the point of control at the system or you put it at the user.

First the “Systems-Centric Approach: you could build into every database the ability to store a list of authorized users and the ability to identify every user (probably through some password identification) and to match every access against the list and either allow or deny that access. Of course, to make this easier, you could insist that the vendors build this capability into their database management systems so you would only have as many security mechanisms (“security systems,” or “security architectures”) as you have different database management systems ... as opposed, for example, to having as many different security mechanisms (“security systems,” or “security architectures”) as you have databases!

Then, what remains for the Security Administrator to do is to define the list of authorized users to each specific database and administer the on-going operation by making sure each database has an accurate authorization list and each user is accurately authorized. The problems with this approach are two-fold: a) if you want to change either the user authorization or the database or the user’s password, there may be thousands of changes that have to be made, depending on how many databases, how many users and how many passwords there are and b) there are as many points where security breeches could occur as there are databases.

The same approach could be used for running the applications where the security mechanism could be built into the vendors’ operating systems to constrain the number of security implementations (“security architectures”) to only the number of different operating systems. Then, each application would have to have its list of authorized users and identify each user access (probably through some password identification), match

against the authorized list and either accept or deny the access to the application. Once again, the Security Administrator would have to define the authorized list of users to each application, ensure the list is accurate and make appropriate changes as either the applications change or the user authorization changes or the passwords change. The problems with controlling access to the applications at the application are basically the same as controlling access to the data at the data, namely, the complexities and volume of administration with large numbers of systems and users, plus there are as many points for security breeches as there are applications.

The second possible approach is the “User-Centric Approach.” You could build a user profile that would define all the application functions and all the data (probably at the field level) that any one specific user is authorized to access. Then, the user profile would be assigned to each user, be managed in their own access device and would be communicated to the operating system and the database management system upon each access, defining what application functions and what data were authorized for that access.

The problems with this approach are that the user profiles could become extremely complex and voluminous and the administrative challenge for maintaining and changing them for each user could easily approximate unmanageability. Also, the amount of data to be transmitted with each access could severely impact the performance of the system. Maybe a way around the performance problem would be, for each user’s initial access, to build into the operating system and the database management system a way to hold the profile active locally in the system until the connection is broken. (Clever users would undoubtedly get around their personal performance problems by keeping the access open continuously to the detriment of the overall system performance, and to the advantage of clever hackers who would just have more access points and more time to work.)

An alternative to controlling the user’s profiles at each user’s device would be to control the profiles centrally, which means that every access from every user would have to be channeled through a central control point which would match the user’s identification to their profile and then permit the access to proceed. This would introduce all kinds of other interesting performance issues as well, though maybe not quite as onerous as managing the user’s profiles at their own devices.

The security risk in the “User-Centric Approach” would be on the user side where every user could constitute a potential security breach if they (or somebody) could get access to and manipulate their user profile.

## **Conclusions**

In both the “Enterprise Architecture Approach” and the “Existing Legacy Systems Approach,” it is key to positively identify the person at the access point. In the “Enterprise Architecture Approach,” that would be the only Security dependency outside of building the Enterprise Architecture to begin with.

The magnitude of the analysis work required for the existing, legacy systems and the complexity of the on-going management and administration coupled with the sheer number of potential breach points make the “Existing Legacy Approach” (Systems-Centric or User-Centric) intimidating and discouraging. In fact, it may be an enormous amount of work for a modest amount of Security.

The magnitude of the analysis and engineering work of the “Enterprise Architecture Approach” are also intimidating, but may be *no more intimidating* and may even be *less than* the magnitude of the analytical work required for the “Existing Legacy Approach.” The work is going to be done ... it is only a matter of *when* you want to do the work. If you wait to do the security work system by system, the redundancies and discontinuities will astronomically magnify the amount of work. If you do the security work up-front, you can eliminate the redundancies and discontinuities and astronomically reduce the amount of work.

Clearly, the on-going management and administration work of the “Enterprise Architecture Approach” would be far less complex and far less costly than the “Existing Legacy Approach”(maybe orders-of-magnitude less complex and orders-of-magnitude less costly because everything would be “normalized” that is, there would be no redundancy and no discontinuities). Also, in the “Enterprise Architecture Approach,” the potential points of security breach would be reduced to the number of storage points for the Enterprise Architecture models coupled with positive identification of the “User” (maybe using biological techniques), which would make effective Security *an achievable objective*.

Being pragmatic about the whole thing, probably the only viable strategy is to take the triage approach to the legacy, focusing efforts on only the highest priority security risk data and/or applications using the “Existing Legacy Approach.” Then, at the same time, begin the Enterprise Architecture work, build to a critical mass of “primitive model,” non-redundant, “normalized” implementations; shift the Enterprise’s dependence on to the Architected environment; and then start turning off the legacy and getting rid of the potential Security breach points.

Then, 25 years from now, we won’t still be having the same discussion ... if we can muddle through the security breaches in the meantime.

I probably don’t have to make this observation, but Security is only ONE benefit (and only one rather *incidental* benefit at that) to having Enterprise Architecture.

*John A. Zachman*  
Zachman International  
2222 Foothill Blvd. Suite 337  
La Cañada, Ca. 91011  
818-244-3763 (Phone and Fax)  
*johnzachman@compuserve.com*