All.Net Analyst Report and Newsletter

Welcome to our Analyst Report and Newsletter

API Security vs AUI security

How is an application program interface different from an other application interface? At a basic level, they are the same – in the sense that anything that can be exploited from one can be exploited from another – or at least it should be that way from a security perspective. But unfo9rtunately, that's not how programmers have done or thought about it in most cases.

Application User Interfaces (AUIs)

The interfaces generally run through a user interface program. The program might be a Web server with a browser at the user end, or they might be any other sort of mechanism for interaction, such as a terminal interface, a keyboard, sensor and actuator, and so forth. These user interfaces are typically complex mechanisms designed to take input through designed mechanisms intended for user activities. They tend to operate at limited bandwidth, have some level of physical or logical connectivity, and operate at human interaction speeds.

Abuse of AUIs

In many historical cases, AUIs have been abused for bypassing controls. A typical example is a Web interface which is assumed to connect to a Web browser and a human user, and of course, when you assume, attackers violate your assumptions.[1] So attackers replace the Web browser with a program that automates functions and does them at higher volume, and bypasses any controls at the client side of the connection, leaving the server exposed to the external replacements.

That's why we check the inputs

Any sensibly security approach will check all inputs as a function of their actual internal (and assumed external) state to verify syntax and content in context. If the input is no good, it will be treated as a violation of something and managed through the fail safe mechanisms of the server and it's interface. Of course lots of folks fail to do this well, and the results are lots of bypasses, such as unlimited database access (e.g., so-called SQL injection or massive accesses to records when only a single record was intended), denial of services (e.g., too many requests that would otherwise be controlled by the user interface), and so forth.

Of course APIs are different

NO!!! They are not! And there lies the rub. APIs are typically designed in the mind of the programmer for non-user interfaces. They are treated essentially as internal calls to program functions, and not checked for inputs because the other end is assumed to be valid. So when bad actors exploit them by not following the interface rules, just like AUIs, they get abused.

Belt and suspenders programmers

I remember from long ago being told by someone discussing programming that so-called defensive programming led to inefficiencies and were bad programming practice. The idea was to do things right in the first place and you wouldn't have to worry about such things.

1 2021-03C - How to defeat any system

But I always thought we should check things repeatedly to make sure they are right and remain right, even within programs. Any time you are doing something potentially risky (a substantial range of consequences associated with what could happen if), you should defend against the unanticipated consequences by controlling carefully the thing you were depending on (usually the inputs and outputs).

This is particularly important when dealing with code written by multiple parties and over which you do not have complete control over changes. But also very important when you are an imperfect programmer like I am. I make mistakes, so I cover them up with checks that prevent those mistakes (called faults) from being reflected in failures. I learned those words from my days in fault tolerant computing, and figuring I was likely the biggest source of faults in my programs, I figured I should protect against myself.

So update the standards of practice?

Whenever I think through things like this I recon it might be a good time to update the Standards of Practice at all.net to reflect my new thoughts. So I checked it out. Frankly I was hoping I would announce a few new questions and answers for the SoPs so I could get the publicity for the new decisions. I did a really simple search for “input” and verified relevancy. I found this approach to input checking reflected in; the basis for protection failures, attack mechanisms considered, ISO-12.2.1 (input data validation), SP-SI-9 and 10, workflow controls, lifecycle controls (under content validation and elsewhere), firewalls (high risk zone separation and limited function interfaces), control of harmful and useless content (syntax checking and elsewhere), and logical protection placement (application firewalls and query limits).

I checked the same thing for things like denial of services, etc. and found pretty much the same thing (look for availability in the SoPs and you will find it under attack mechanisms, risk aggregation controls, COBIT elements, control architecture (under availability and use control), authentication, systems lifecycles, zoning trust and separation mechanisms and placement, zone separation verification, data in use protection, version control protections, and business continuity and disaster recovery resource availability.

Generally speaking, I don’t just add willy-nilly new items to the Standards of Practice when they are already covered elsewhere, and thus for API input controls I kind of figured that no new additions were called for.

So nothing new right?

Indeed. Nothing is really new in the security arena. The more we look, the more we find more of the same. And the same is that we know how to do it well as a field of study, but we don’t do it well as practitioners. What we fail to do is systematically learn from our mistakes and track all we need to do against requirements and well-defined criteria.

Conclusions

Airplanes used to crash a lot until we started systematically working through all the things that cause them to fail. Society learned how to make air travel very safe by keeping track and eliminating one after another of the things that went wrong so they didn’t go wrong any more. This includes the people, process, technology, and decision-making. It involved checklists and training and education and a culture. We need the same thing for cyber security.

