Last week I was creating design and implementation task lists as well as scheduling those tasks for our Attack Surface Explorer. I referred to the PRD, User Scenarios, Requirements List and Features List that we had created previously. I faced considerable difficulty when creating these tasks and scheduling them because in doing so, several questions arose which I did not have immediate answers to. I checked my questions to see if I could postpone them to the design phase, but not all could be postponed. Therefore, I took three days last week to try and find answers to these questions. My main concerns were as follows:

1. What really is an attack surface? Many definitions exist, but it may be worthwhile for us to have a solid definition which we follow through the development of the product.
   * Are all resources used by an application part of its attack surface?
   * What resources actually contribute to the attack surface? How do we detect it?
   * What metrics can we use to compare two attack surfaces?
     + How useful is RASQ as a metric for our product?
2. Threat Qualification Tests:
   * Simply blocking and fuzzing resources will not help – what other tests can we perform?
   * In addition to making these tests context sensitive with regards to the type of resource can we also take application and user permissions into account?
   * Do we want to use Fault Injection in Threat Qualification Tests? A govt. study says that “a vulnerability discovered through fault injection may not have a direct effect on program security” .
3. What type of applications do we want to target?
   * Are we targeting Desktop applications? Server applications? Web applications?
   * Web application analysis may require additional work and a dive into web technologies such as Perl, Python, ASP.NET, JSP etc.

In addition to search for answers to these questions, I spent a short amount of time looking for other software products which allow black-box Attack Surface analysis and found none. However, there are many techniques proposed by research groups and software companies which are used to understand the attack surface of the application. One example of such a technique is the *Commercial Application Threat Analysis* used internally by HP for analyzing and keeping track of the attack surface of their products. CATA uses a technique where users are asked security questions about their components and data flow. Another technique which I found very interesting has been proposed by a group of researchers at Carnegie Mellon University in their paper on *An Approach to Measuring A System's Attack Surface* – I will provide some details of their research later on in the paper since I intend to use their technique extensively in ASE.

I also found some encouraging data which justifies the need for ASE. Michael Howard has said:

|  |
| --- |
| “As each week in the development cycle passes by, you should measure the attack surface of your product. Start with a baseline, and then each week count all the items identified in the previous section using various scanning tools. In some cases, you may need to write your own tools if you have entry points specific to your application. If the attack surface count goes up, determine why it went up, and see if you can drive it back down. When engineers know you are measuring the attack surface, they will try not to stand out by increasing the attack surface in the first place.”(Howard, 2004) |

How exciting would be it if he said “Start with a baseline and then each week count all the items identified in the previous section using ~~various scanning tools~~ Attack Surface Explorer.” instead ☺

In the sections below, I will present brief answers to some of the questions I mentioned above.

# What is an Attack Surface?

“A system’s attack surface is the subset of the system’s resources that an attacker can use to cause damage to the system” .

Based on this, we now have the additional requirement of understanding how to identify resources which affect a system’s attack surface. Thankfully, the research group from CMU has already done this work for us. They have proposed an ***entry point and exit point framework*** to identify these relevant resources. In addition to this, they also propose that “not all resources contribute equally to the measure of a system’s attack surface” and provide the notion of a ***damage potential-effort ratio*** to estimate a resource’s contribution to the attack surface

This is good news because it relates directly to two requirements from our PRD:

* **Quantitative**: Help discover the total attack surface of a product.
* **Qualitative**: Help rank each resource (threat) in the attack surface.

Their work is based on the RASQ metric proposed by Michael Howard; in fact one of the members of the research team worked closely with Mike Howard on his RASQ paper. They have studied the RASQ methodology and proposed a more generalized approach for measuring the attack surface of a product. They have proposed a general method which does not require a very deep understanding of the product being analyzed – which is very important for a tool like ASE.

## How do we visualize the Attack Surface?

Using the *Resource View* panel from Holodeck as a model for presenting the attack surface of a product to the users of ASE was also a big challenge for me. Even after we narrow down the resources which contribute to the attack surface of the product, we will be left with many resources which will need to be organized in a meaningful manner for the user. I found some answers from the CMU research group again. They propose using the following meta-categories for all identified resources:

1. Channels: Sockets, TCP/IP, Pipes, SSL etc.
2. Methods: Local libraries and their methods.
3. Data Items: Files, Registry Keys, Databases etc.

# Threat Qualification Tests

I brainstormed with Rahul about what type of testing tools we want to provide to our users for them to rank resources in the threat map. Depending upon the type of resource and maybe even its permission settings we came up with a few scenarios. Though this work is not exhaustive, we now have a better idea of the direction we want to go in. Here’s a sample of tests we thought of for *Channel* resources:

1. **Fuzz Channel**: We began with thinking that Fuzzing is only for Data Item resources; but then decided that being able to fuzz an entire channel (similar to HD’s Network Corruption tests) will also be useful. Imagine selecting a Pipe channel and being able to Fuzz everything in that Pipe.
2. Detect open channels and try to connect to it – check if you get disconnected after timeout period.
3. Track how long channels are open for in idle state. All idle channels should be closed.
4. Keep sending garbage data – does app keep accepting it.
5. Replay attack on channel to see if connection is made after authentication data is replayed.
6. Track data coming in through a channel to see if this data makes it to a data store without much modification. The test is we allow the user to send arbitrary data and then check if this data ended up in a data store.

In addition to Fuzzing and blocking tests for Data Item resources we should provide tests to check for:

1. SymLink vulnerabilities
2. Database specific attacks

# Target Applications

We should target Server and Desktop applications only. Targeting Web Applications will require us to dive too deep into web technologies and may increase development time considerably.

# Bibliography

Howard, M. (2004). *Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users*. Retrieved from MSDN: http://msdn2.microsoft.com/en-us/magazine/cc163882.aspx

Manadhata, P. K. (2007, 8 1). *An Approach to Measuring A System's Attack Surface.* Retrieved 4 8, 2008, from CMU: http://reports-archive.adm.cs.cmu.edu/anon/2007/CMU-CS-07-146.pdf

Voas, J. M., McGraw, G., Ghosh, A., Charron, F., & Schatz, M. (2000, 11). *Quantifying Minimum-Time-To-Intrusion Based on Dynamic Software Safety Assessment.* Retrieved from http://handle.dtic.mil/100.2/ADA386611