ISSN:0975-9646

Analysis and Evaluation of "Reducing the Attack Surfaces" to improve the security of the software at Design Level

Mubashirah Majeed[1], S.M.K Quadri[2]
[1,2] *Department of Computer Sciences*
University of Kashmir, India

Abstract—**After the requirements for a solution has been identified in the early stages of an application's software development lifecycle, the next step is to design and architect a solution that satisfies those identified requirements. Developing trusted applications requires that sound security and privacy decisions be made early in the design phase because decisions made at this stage will highly influence subsequent efforts in the latter stages of the software development lifecycle and the final state of the application. It has been found that by adopting this approach, application development costs (such as those required to address and resolve security and privacy issues) are significantly reduced compared to if security and privacy were considered later in the SDLC or not at all. This is because applications developed against more secure and privacy aware designs tend to be exposed to fewer threats and contain less vulnerabilities. The important practice that should be taken into consideration at the design phase of the software development life cycle is addressing security and privacy concerns. In this paper we analyze and evaluate as to how security of the software will be improved if reducing the attack surfaces at design level are addressed. The Attack Surface describes all of the different points where an attacker could get into a system, and where they could get data out. We explore the attack surface first then we discuss how we can Measure and assess them. We also explore as to how we can manage them and finally we perform the analysis and review of the attack surfaces as part of the research findings.**

Keywords— **vulnerability, attack surfaces, entry points, risks, security**

I. Introduction

The design phase of the software is considered as the most important phase and is a process of weighing important needs for the software in terms of efficiency/speed, future code maintenance, designing to minimize bug possibilities, testing/validation before release. Different industries require much more and every industry emphasizes different parts of design. The hope is that a good design lasts a long time and is stable and easily upgraded in the future. It's hard to get that from classroom assignments that never last longer than a single semester going to school. Industry, however, relies heavily upon design for its future and profits. [1]

In the real world there is a difference between design (doing good program design as you go, based on challenges faced) and design spending a few weeks doing UML diagrams and formal documents based off inaccurate, idealized views of things which are out of date as soon as the first line of code is written).Development teams on the internet' like coding, because design is worthless alone. Design exists to support coding, and in that regard is invaluable. Frankly, many of the important problems in software development aren't in the code, but in program design.

The important practice that should be taken into consideration at the design phase of the software development life cycle is addressing security and privacy concerns .If early thought is given to this it helps minimize the risk of schedule disruptions and reduce a project's expense. Validating all design specifications against a functional specification involves accurate and complete design specifications, including minimal cryptographic design requirements and a specification review. The point of Attack Surface Analysis is to understand the risk areas in an application, to make developers and security specialists aware of what parts of the application are open to attack, to find ways of minimizing this, and to notice when and how the Attack Surface changes and what this means from a risk perspective. [2]

Attack Surface Analysis is usually done by security architects and pen testers. But developers should understand and monitor the Attack Surface as they design and build and change a system. Attack Surface Analysis helps you to:

- identify what functions and what parts of the system you need to review/test for security vulnerabilities
- identify high risk areas of code that require defense-in-depth protection - what parts of the system that you need to defend.
- identify when you have changed the attack surface and need to do some kind of threat assessment

II. Defining the Attack Surface of an Application

An attack is the "means of exploiting vulnerability" [3].The Attack Surface describes all of the different points where an attacker could get into a system, and where they could get data out.

The Attack Surface of an application is:

1. the sum of all paths for data/commands into and out of the application, and
2. the code that protects these paths (including resource connection and authentication, authorization, activity logging, data validation and encoding), and
3. all valuable data used in the application, including secrets and keys, intellectual property, critical business data, personal data and PII, and
4. the code that protects these data (including encryption and checksums, access auditing, and data integrity and operational security controls).

You overlay this model with the different types of users - roles, privilege levels - that can access the system (whether authorized or not). Complexity increases with the number of different types of users. But it is important to focus especially on the two extremes: unauthenticated, anonymous users and highly privileged admin users (e.g. database administrators, system administrators).

Conceptual model of an aggregate attack surface model is given in Figure 1 below. It is an aggregate in a sense because, although it is the system attack surface with which we are most concerned, various pre-system access controls reduce both the opportunities to reach a system and the number of system elements an attacker can actually see or use. The amount of time and effort in ASR activities is system- and data-classification dependent [4]

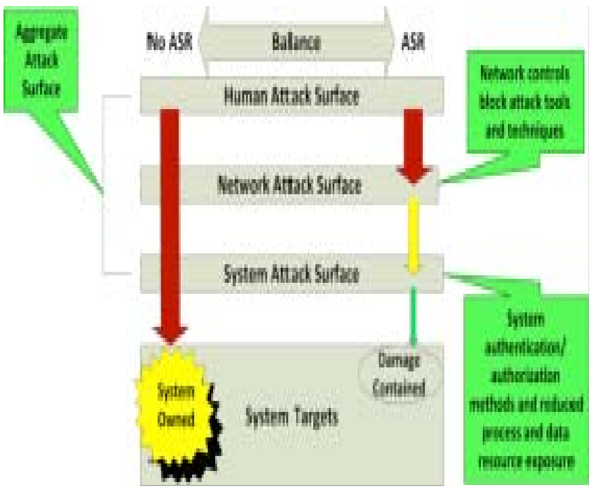

Fig 1: Aggregate Attack Surface Model

With this approach, you don't need to understand every endpoint in order to understand the Attack Surface and the potential risk profile of a system. Instead, you can count the different general type of endpoints and the number of points of each type. With this you can budget what it will take to assess risk at scale, and you can tell when the risk profile of an application has significantly changed. [5]

III. Identifying and Mapping the Attack Surface

System software is less than perfect [6].You can start building a baseline description of the Attack Surface in a picture and notes. Spend a few hours reviewing design and architecture documents from an attacker's perspective. Read through the source code and identify different points of entry/exit:

- User interface (UI) forms and fields
- HTTP headers and cookies
- APIs
- Files
- Databases
- Other local storage
- Email or other kinds of messages
- Run-time arguments
- …. [custom points of entry/exit]

The total number of different attack points can easily add up into the thousands or more. To make this manageable, break the model into different types based on function, design and technology:

- Login/authentication entry points
- Admin interfaces
- Inquiries and search functions
- Data entry (CRUD) forms
- Business workflows
- Transactional interfaces/APIs
- Operational command and monitoring interfaces/APIs
- Interfaces with other applications/systems
- ... [custom types]

You also need to identify the valuable data (e.g. confidential, sensitive, and regulated) in the application, by interviewing developers and users of the system, and again by reviewing the source code. [5]

IV. Measuring and Assessing the Attack Surface

Once you have a map of the Attack Surface, identify the high risk areas. Focus on remote entry points – interfaces with outside systems and to the Internet – and especially where the system allows anonymous, public access.

- Network-facing, especially internet-facing code
- Web forms
- Files from outside of the network
- Backwards compatible interfaces with other systems – old protocols, sometimes old code and libraries, hard to maintain and test multiple versions
- Custom APIs – protocols etc – likely to have mistakes in design and implementation
- Security code: anything to do with cryptography, authentication, authorization (access control) and session management

These are often where you are most exposed to attack. Then understand what compensating controls you have in place, operational controls like network firewalls and application firewalls, and intrusion detection or prevention systems to help protect your application. [5]

V. Managing the Attack Surface

Once you have a baseline understanding of the Attack Surface, you can use it to incrementally identify and manage risks going forward as you make changes to the application. Ask yourself:

- What has changed?
- What are you doing different? (Technology, new approach,)
- What holes could you have opened?

The first web page that you create opens up the system's Attack Surface significantly and introduces all kinds of new risks. If you add another field to that page, or another web page like it, while technically you have made the Attack Surface bigger, you haven't increased the risk profile of the application in a meaningful way. Each of these incremental changes is more of the same, unless you follow a new design or use a new framework.

If you add another web page that follows the same design and using the same technology as existing web pages, it's easy to understand how much security testing and review it needs. If you add a new web services API or file that can be uploaded from the Internet, each of these changes have a different risk profile again - see if the change fits in an existing bucket, see if the existing controls and protections apply. If you're adding something that doesn't fall into an existing bucket, this means that you have to go through a more thorough risk assessment to understand what kind of security holes you may open and what protections you need to put in place.

Changes to session management, authentication and password management directly affect the Attack Surface and need to be reviewed. So do changes to authorization and access control logic, especially adding or changing role definitions, adding admin users or admin functions with high privileges. Similarly for changes to the code that handles encryption and secrets. Fundamental changes to how data validation is done. And major architectural changes to layering and trust relationships, or fundamental changes in technical architecture – swapping out your web server or database platform, or changing the run-time operating system.

As you add new user types or roles or privilege levels, you do the same kind of analysis and risk assessment. Overlay the type of access across the data and functions and look for problems and inconsistencies. It's important to understand the access model for the application, whether it is positive (access is deny by default) or negative (access is allow by default). In a positive access model, any mistakes in defining what data or functions are permitted to a new user type or role are easy to see. In a negative access model, you have to be much more careful to ensure that a user does not get access to data/functions that they should not be permitted to.

This kind of threat or risk assessment can be done periodically, or as a part of design work in serial / phased / spiral / waterfall development projects, or continuously and incrementally in Agile / iterative development. [5]

VI. Results and Research Findings

Flaws in system software create vulnerabilities that enable most1 of the reported system intrusions. Anecdotal evidence supports a hypothesis that poor system administration practices, including the failure to apply available patches in a timely fashion, results in an excessive window of vulnerability for the affected systems. As far as we have been able to determine, no studies exist that would either confirm or refute this conjecture though is is widely believed and often repeated. [6] Building secure software, software that withstands attacks, isn't easy and at the same time reducing the code open to attack by default is again not an easy job. These nasty security issues occur for the following reasons:

- The product had a security flaw.
- The product is popular or is running by default.

For example I tested the security awareness of a development team working on a project with help of the following code snippet. To their understanding there was no flaw/weakness in this code.

```
char *ptr = "Hello, How are you! "
     "This is a just for testing purpose"
     "You should always focus on writing the secure code";
char *buf = new char[sizeof(ptr) + 1];
if (buf)
    strncpy(buf,ptr,strlen(ptr));
delete [] buf;
```

However there is a security vulnerability which is "buffer over run". Buffer size was calculated incorrectly. sizeof(p) is the size of a pointer, which, in this case is only 4 bytes. Sizeof (p) does not return the length of the string, which is what was intended. This security vulnerability is called as "buffer over run". It was there in the code because neither its requirement was identified at requirement engineering level nor was it handled at the design level.

Apart from this an interesting question which might arise: how much more secure is the product that is currently in development than the product that is currently shipping? This is really a difficult question to answer but you could calculate the "Attackability" of a product or its exposure to attack, but not necessarily its vulnerability. That is, how many features are there to attack, not necessarily exploit. Intuition: Reduce the ways attackers can penetrate surface means to increase the system's security [7].

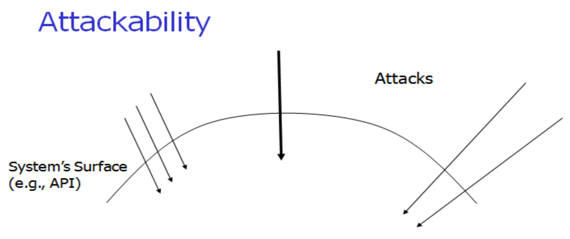

Fig 2: Attackability Scenario

Here's how you do it. First, all products are attacked in certain ways—most products are often attacked through open ports, Windows has its services attacked, weak ACLs are an attack point too. Many Linux and Unix operating systems are attacked through root applications and symbolic links. So, the first step is to look over old vulnerabilities and determine the root attack vector. For Windows, we came up with this list:

- Open sockets
- Open RPC endpoints
- Open named pipes
- Services
- Services running by default
- Services running as SYSTEM
- Active Web handlers (ASP files, HTR files etc)
- Active ISAPI Filters
- Dynamic Web pages (ASP and such)
- Executable virtual directories
- Enabled Accounts

Think of this as the list of features an attacker will attempt to compromise. Each attack point has a weight or a bias. For example, a service that runs by default under the SYSTEM account and opens a socket to the world is a prime candidate for attack. It may be very secure code, but the fact that it is running by default, and is running with such elevated privileges, makes it high on the list of points for an attacker to probe. And if the code is vulnerable, the resulting damage could be disastrous.

On analyzing various versions of Windows, people at Microsoft end up with the following relative attack surface figures as shown in Table 1.

TABLE 1
Relative attack surface for various versions of Microsoft Windows

Version of Microsoft Windows	Relative attack surface figures
Windows NT® 4 SP6a	316.9
Windows NT 4 SP6a with Option Pack	597.9
Windows 2000	348.9
Windows Server 2003	173.9
Windows Server 2003 with Internet Information Services (IIS) 6.0	188.8
Windows XP	189.5
Windows XP with Internet Connection Firewall enabled	124.4

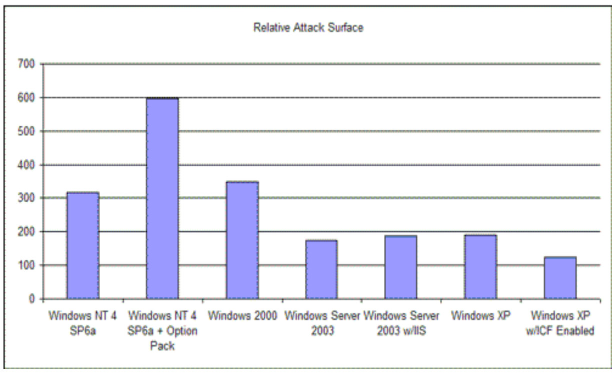

Fig 3. Relative attack surface for various versions of Microsoft Windows

The most telling figures are between Windows 2000 and Windows Server 2003, and between Windows Server 2003 and Windows Server 2003 with IIS 6.0 installed. As you can see, the attack surface of a default Windows Server 2003 computer is approximately half that of a Windows 2000 computer. This speaks loudly to the security work performed by the Windows product group [8]

We have analyzed that to improve the security of the software at design level there are a couple of caveats to this attack surface calculation. First, it does not represent the code quality, but rather determines the relative "Attackability" of the software. It does not mean that there are security flaws in the code.

Second, it is possible to create a product that manipulates these figures. For example, you could multiplex functionality behind single analysis points, which would skew the figures.

Keep the following things in mind while working with applications which are vulnerable to attacks:

- Reduce the amount of running code. Use the 80/20 rule; if 80 percent of the users accessing the system do not need a service or process, do not let it run. If you are the developer, make it the default setting: if the security practitioner, turn it off.
- Restrict access to network endpoints used by your application to the local network segment or an explicit IP address range. Conversely, consider allowing access to system entry points only for subjects in trusted network segments.
- Limit access to network endpoints using authentication. Simply validating a subject reduces your system's attack surface.
- Reduce the privilege under which processes execute. This includes both code written in-house and by third-parties.
- As you review data flow diagrams and attack trees, look for anonymous threat paths: paths for which authentication or authorization are not necessary. Consider controlling them with authentication and assignment of access rights.
- Apply the 80/20 rule to all protocols.
- Define your minimal attack surface early in system or application design and measure it periodically to ensure compliance.
- If you have a large attack surface, you will spend more time managing vulnerabilities and trying to ensure all code and system configurations are perfect. Again, this is an impossible task.[4]

VII. Conclusion

Security at design layer should be considered as most important activity within the phase. Eradicating coding bugs with security implications is not sufficient. Design vulnerabilities can have a substantial detrimental impact on security and are much more difficult to address during the verification phase. This paper analyses and evaluates as to how security of the software will be improved if reducing the attack surfaces at design level are addressed. Reducing the number of vulnerabilities in an application, is the goal of secure coding and not attack surface reduction. The use of security code scanning tools will help developers identify vulnerabilities in code and reduce the overall number of vulnerabilities present in an application, but will not reduce the attack surface. We tested the security awareness of a development team working on a project with help of a code snippet. To their understanding there was no flaw/weakness in the code. However there was a security vulnerability which is "buffer over run". It was there in the code because neither its requirement was identified at requirement engineering level nor was it handled at the design level. So before proceeding to design make sure that the security requirements have been enlisted in the requirement engineering phase and similarly before proceeding to coding again make sure that security requirements have been included in the design which will be finally implemented in the implementation phase of the software development cycle.

REFERENCES

[1] "Why is design so important in building software" http:// www.quora.com.

[2] SDL Process: Design https://www.microsoft.com/en-us/sdl/process/design.aspx.

[3] Howard, M., Pincus, J., & Wing, J. (2002, February 11). *Measuring Relative Attack Surfaces.* Retrieved February 8, 2012, from Carnegie Mellon School of Computer. http://www.cs.cmu.edu/~wing/publications/Howard-Wing03.pdf

[4] "Attack Surface Reduction – Chapter 4" http://resources.infosecinstitute.com/attack-surface-reduction/.

[5] "Attack Surface Analysis Cheat Sheet" https://www.owasp.org/.

[6] Browne, Hilary K., William A. Arbaugh, John McHugh, and William L. Fithen, ``A Trend Analysis of Exploitations," In Proceedings of the 2001 IEEE Security and Privacy Conference, pages 214-229, Oakland, CA, http://www.cs. umd.edu/~waa/pubs/CS-TR-4200.pdf, May 2001

[7] Wing, J., Howard, M., & Pincus, J. (2003, December). *Measuring Relative Attack Surfaces (PowerPoint Presentation).* Retrieved February 11, 2012, from Academia Sinica, Institute of Information Science: http://iis.sinica.edu.tw/wadis03/slides/Wing.ppt

[8] Michael Howard, "Secure Windows Initiative" "Fending off Future Attacks by Reducing Attack Surface" https://msdn.microsoft.com/en-us/library/ms972812.aspx.