A LAYERED SECURITY APPROACH FOR APPLICATION LAYER WITH AN INTRINSIC APPROACH

[1]D Navya,Assistant Professor,
[2]Sarikonda Sree Hari Raju,Associate Professor,
[3]Ashwini Nangunoori,Assistant Professor,
Department of CSE Engineering,
Nagole Institute of Technology and Science Hyderabad
Kuntloor(V),Hayathnagar(M),Hyderabad,R.R.Dist.-501505.

Abstract - Application Protection's present condition represents the reality that security has been an afterthought. The primary problem was the security of data in transit and storage, and cryptography successfully resolved this question. The challenges to systems have, however, grown beyond those addressable to the device itself through protocols and cryptography. This lack of cyber foresight has cost billions in missed sales and is now disrupting the infrastructure of information technology that the global economic engine is dependent on. The defense of an application against security attacks, Application Security, is a challenging challenge. To integrate the need for Software Safety, Application Security must now expand beyond conventional network and data security. A consistent and detailed view of the possible threats at each stage in the device or network must also guide the approach to Application Protection.
Keywords-application security, threat models, software principles.

I. INTRODUCTION

Protecting an application from security risks is Application Security. This is a daunting challenge, since the program builder or corporate protection planner must have protections to any threat possible, while in order to survive, an intruder must only locate one flaw or point of attack. Past device security strategies have definitely been minimal; however modern technology has been introduced to overcome this difficult issue [1].

- Network Security, Data Security and Device Safety consist of Program Security:
- Network Protection typically tackles foreign threats against infrastructure inside a firewall that delivers a network-wide utility. Using firewalls, intrusion prevention devices and malware scanners, network security has historically been dealt with.
- The preservation of data used locally by an application or transferred between users and servers is Data Security. The key approach here is cryptography, since it is incredibly successful in preserving data during transmission and storage by maintaining its privacy and secrecy.
- Software Security is the protection from assaults on the software or resources offered by the software, thereby avoiding misuse of proprietary property and approved material and ensuring that the software continues to work as expected. These attacks usually involve reverse engineering, tampering, copying, and automatic types of attacks that can be launched by comparatively unsophisticated attackers around the network or on a desktop.

II. THREAT MODELS Network threat model

Network protection professionals have historically seen the hardware and the operating system as trustworthy. This is a Network Vulnerability Paradigm, where the intruder is distant and external. The application is attacked through network ports, so the first and most prevalent method of perimeter security was firewalls that filter external packets from the untrusted environment. The downloaded code still posed a hazard, so to guarantee the security of this code, code signing was invented. Other kinds of threats were malware and worms, so reactive protections such as virus scanners and intrusion detection systems [2] were added. The bugs that occur in application software enabling attacks such as viruses and worms, however, remain a top concern.

B. Model of Untrusted host hazard

At the other extreme of the hazard model continuum is Software Security. In this situation, the data and software must be secured from a legal yet possibly malicious attacker who has full access over the programming platform and may then use a broad variety of resources to find bugs and carry out an assault against the program, such as disassemblers, debuggers and emulators. This is considered the Hazard Paradigm of the Untrusted Host which is the field of copy security and material protection strategies for PC games.

The first perimeter-type defenses developed to secure data and software under the Untrusted Host Threat Paradigm were focused on cryptography. Dynamic memory tracing is an attack strategy that has culminated in reactive protections, such as anti-debug and self-modifying code, being introduced.

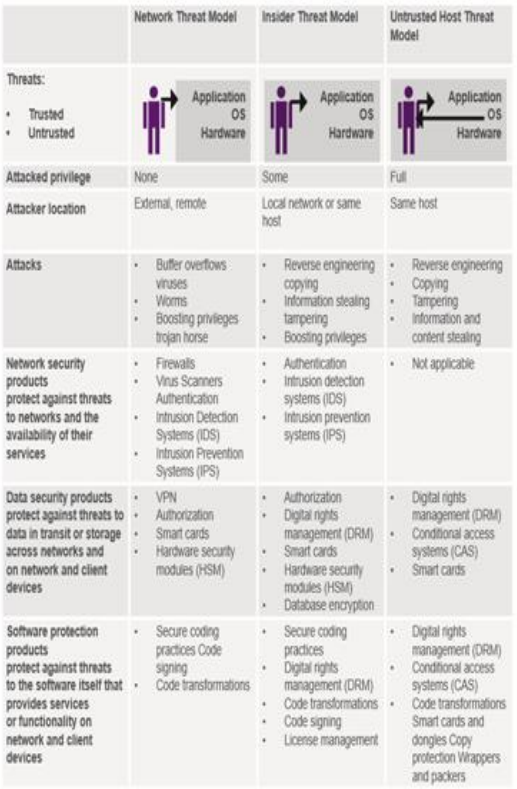

Fig 1. Threat Model

The severity of the threat model implies that in order to deter attacks against such systems [3], new strategies are required.

B. Model of insider hazard

The Insider Vulnerability Paradigm resides in between. The consumer can have restricted device rights but is local to the attacked goal program. Buffer overflows may be leveraged to improve rights, normally network protection style assaults. Threats to intellectual property are often the cloning of apps and tampering or reverse engineering them off-site. Interestingly, it is quite easily probable for the hazard paradigm to alter. For example, a worm or a Trojan horse can gain full control over a computing platform and the software running on it if a successful network intrusion attack is performed. In this scenario, from a Network Vulnerability to an Untrusted Host Threat Model, the threat model switches very rapidly. Corporations also cope with this by cleaning the corrupted hard drive on the device. Ultimately, you have to conclude that the program and devices are untrustworthy for high security applications. Although the techniques to address different threat models are unique, there are many commonalities. The majority of attacks these days are dynamic attacks against the software. They are performed when the application is running and data is decrypted and in the clear.

The Threats:
• Trusted
• Untrusted

OS Hardware Application

Attacked Privilege None Any location of Complete Attacker External, remote Local Network or same host. So, what are some of the problems with application security? Below, ten principals are listed.

I Main one - Define the weakest connection and protect it.

Identify and rate the threats. Although you need to identify whether to secure, from whom and for how long, conduct a cost/risk analysis of protecting the risk.

(ii) Principal Two - In-depth practice defense. Ensuring the access protection standards from device access to access to classified information is multi-level;

After a time of inactivity, lock out the application; have extra protection for private information; and enforce the least privilege principal to guarantee that consumers only have access and features to finish their work.

(iii) Principle three - Unwillingness to trust Identify the trust relationship between and component; identify the risks resulting from the data flow; exercise the least privileged Concept; and procure an impartial security risk management resource to execute a high-level risk assessment (not the functionality).

(iv) Principal four - Note that it's impossible to keep secrets Defense by obscurity isn't working.

(v) Principal five - Obey Least Privilege Theory Giving users just the access they need to play their function. A module that divides the roles and rights may need to be installed.

(vi) Principal six - Failure to recover and recover safely Ensure that the program is connected or has its own identification and sufficient degree of invalid access alert mechanism (internal or external); Review the logs; Maintain track of the investigation, as this is essential documentation if the matter has to be brought to court; and Provide a proper backup copy of the request and the details that might be necessary for total recovery.

(vii) Principal seven: Compartmentalize the restriction of invalid (internal and external) access to assets; guarantee that the IT support team is not the same team that maintains the system; and duty analysis segregation is a must.

(viii) Principal 8 - Make it clear Prevent secret conclusions and ambiguity; and Keep the coding consistent

(ix) Principal nine - maintain trust in yourself (reverse of social engineering)

(x) Key ten - Be suspicious

III. INTRINSIC APPLICATION SECURITY

Intrinsic protection entails systems and approaches that are integrated throughout the phase of design and development. These involve programming methodologies and processes of verification of manual code, as well as applications and tools for development [4]. It is necessary to use intrinsic protections in combination with other conventional defenses. Usually, intrinsic protection exists at the stage of source code and can require any or more of the following techniques:

Transformations in power movement. Control flow applies to the execution direction taken while programs are executed and control is passed to separate statement blocks. Control flow transition secures a program by randomizing the target source code block bodies [9]. This results in code that is incredibly challenging to track and thus greatly raises the expense of the intruder seeking to reverse engineer the program traffic.

Branch safe. In software, commands are referred to as "branches" to theoretically pass power to another instruction. A conditional branch is a branch that has an input value(s) that contains IF statements, Turn statements, and conditional operators. In order to sidestep security checks or in an effort to change the original flow of the software, attackers usually aim to jam or circumvent major branches in the code. Through inserting code that allows the software to act inappropriately if the branch is jammed, branch security eliminates branch jamming.

In-line routine. In this process, before transformations are implemented, different logical portions of code inside a file are combined. (This technique varies from the in-line compiler alternatives that are performed during pre-processing.) The aim is to merge operations and mask the program logic.

Flattening flow regulation. Today's compilers use a defined range of hop and conditional branch instructions from the goal instruction-set to execute the control flow of procedural languages. The flow-of-control is usually accomplished utilizing a formal or rule-driven technique in machine code. Constructs of regulation are converted into canned and repetitive sequences of guidance. Consequently, the control flow of the initial software may also be effectively replicated by reverse-engineering techniques such as decompiles and program slicers. Control Flow Flattening turns the control flow into a Transition statement, eliminating the study of static control flow.

- White-box encryption. Where there is a concern that an intruder will be able to track the application and obtain one or more cryptographic keys embedded or created by the application [5], white-box cryptography functions are used. Black-box attacks in classical cryptography define the condition where the intruder attempts to acquire the key by learning the algorithm and controlling the inputs and outputs, but without being apparent to the execution. The far more extreme threat paradigm of content management schemes where the user will monitor anything is solved by White-box cryptography. It also needs encryption and decryption, but without revealing the cryptographic key. Applications should hold confidential data either encrypted or converted with careful design, or both, so that the original data is never revealed. In a transformed state, all data operations occur.
- Verification of honesty. Integrity Authentication is a more reliable code signing variant that guarantees interest in an untrusted host [6]. It provides a safe method of validating an application's integrity and can also ensure the integrity of external modules, including operating system components that communicate with that application. Integrity Verification assures that, without detection, applications will not be abused, either statically or dynamically. This increases the tamper resistance bar considerably so an intruder would not only reverse-engineer a program, render binary changes, but even defeat integrity testing.
- Anti-debug. Any monitoring or diagnostic feature that operates in the context of an application allows end-users whose purpose is to reverse engineer or subvert the standard functionality of the application deployed. Anti-Debug techniques enable debuggers to be identified operating in the same setting as the application [7]. The program will take steps to either de-activate the debugger or avoid running if it is found.
- Secure/loader packager. In this technique, a Safe Packager/Loader mini-application intercepts user or application calls to a target file during runtime [10]. The Stable Packager/Loader must first verify the trigger event before the so-called aim file [8] is unpacked and executed. It is therefore impossible for an intruder to statically evaluate the file in storage whether the target executable or DLL is protected.

The above approaches establish encryption that is inherent to the code and cannot be isolated or excluded from the program data or functionality as implemented during the creation phase. In addition, the methods allow program diversity, whereby random adjustments are made to the techniques each time they are implemented. This method produces

multiple machine instances, which decreases the efficacy of automatic attacks and masks gradual software changes to deter attacks from differential review.

As a basic and vital aspect of Device Security, Software Defense has been mostly ignored to date. In transport and storage, cryptography effectively safeguards data, but leaves the application subject to attack, literally any device in the communication chain.

IV. CONCLUSION

Since Application Security has traditionally become an afterthought in the creation and implementation of web systems, perimeter and reactive protections has been the subject of conventional network security and software safety strategies.

Those are not enough anymore. Safety professionals and companies recognize the need to keep software automatically protected by investing up front in Device Security. Trends point to automation tools and the use of program diversity and renewability as solutions to ensure application protection and avoid scalable attacks that can disrupt the cyberspace infrastructure, though good application architecture remains relevant. It is possible to provide a simplistic approach that is intuitive, fast to implement, stable, and scalable by integrating multiple essential protection building blocks.

REFERENCES

1. ***www.rdeto.com/documents/Collateral/wp_application_security_en.pdf***
2. ***Exodus communication: Application code security***
3. ***www.exodus.net/seccurity/application/code_review.html***
4. ***Application program security: handbook of information security management***
5. ***Application development: www.atstake.com/services /enterprise /applications.html.***
6. ***Applying the OSI Seven Layer Network Model to Information Security- SANS institute InfoSec reading room***
7. ***Communicationsecurity.html- communication security at application layer.***
8. ***Application layer security by John Ronda, July, 25, 2006.***