*************************Secure Software Engineering **************************

Data Flow Analysis called tainted analysis --meged 347,354

Taint analysis is a process used in information security to identify the flow of user input through a system to understand the security implications of the system design. This analysis aims to mitigate risks such as SQL injection, which generally arise when other parts of the program use user input without proper sanitization

Tracking how private information flows through the program and if it is leaked

to public observers

Security analyser use data flow analysis primarily to reduce false positive and false negatives eg: many buffer overflow

in real codes are not xploitable because the attacker can not control the data that overflow the buffer

The data flow analysis that is often used in security related application is taint analysis

--A variable is tainted if its value is influenced by potential hacker

--If tainted variable is used to compute the value of second variable then the second variable also tainted.

****

SCA - Software Composition Analysis ---merged 319

Generally the term is used for managing open source component, Involves scan of of an application code base to identify all open source component for: license compliance data and security vulnerability

The scan cover identifying : direct dependency and transitive dependency

Software composition analysis (SCA) is an automated process that identifies the open source software in a codebase. This analysis is performed to evaluate security, license compliance, and code quality.

Companies need to be aware of open source license limitations and obligations. Tracking these obligations manually became too arduous of a task—and it often overlooked code and its accompanying vulnerabilities. An automated solution, SCA, was developed, and from this initial use case, it expanded to analyze code security and quality.

In a modern DevOps or DevSecOps environment, SCA has galvanized the “shift left” paradigm. Earlier and continuous SCA testing has enabled developers and security teams to drive productivity without compromising security and quality.

****

Threat modeling methods create these artifacts:

An abstraction of the system

Profiles of potential attackers, including their goals and methods

A catalog of threats that could arise\

Threat modelling book

STRIDE mitigation ---page 14

DFD of a 3 tier architecture ----page 45

Data flow diagram of the Acme/SQL database ----page 35

Trust boundaries in a web server ---- page 51

STRIDE Threat examples ---page 65

Addressing Threats According to Who Handles Them ---page 77

Managing and Addressing Threats --- page 176

*********Book Reference ***************************

Example of attack pattern -111 T1

Attack pattern Look Like

page -109

****

Types of Atatck

--merge slides 29

****

Non repudiation in privacy context ---slide 189

****

Attack pattern help to identify positive an negative requirements

page -118

****

Requirement suffer from the problems.

pag -147

Think as atatcjer prespective

merge slide --211

Security is not set of features ---212 -215

****

Security is emergent property

page-163

slide 68 --merged

slide 213 --merged

Elicitation methods ---merged 237

Sometimes building security in at the beginning of the SDLC means making explicit tradeoffs when specifying system requirements. For example, ease of use might be paramount in a medical system designed for clerical personnel in doctors' offices, but complex authentication procedures, such as obtaining and using a cryptographic identity, can be hard to use [Whitten 1999]. Furthermore, regulatory pressures from

Every time a new requirement, feature, or use case is created, the developer or security specialist should spend some time thinking about how that feature might be unintentionally misused or intentionally abused. Professionals who know how features are attacked and how to protect software should play active roles in this kind of analysis

****

Design for security concerned with ---merged 70

Sofwtare Qulaity tradeoff --- merged 75

SDL roles --merged 107

Who are attacker ---merged 136

****

Missue Case/Abuse Case

page-168

--slide 90, 91, 92, 217, ,234 ,235

SDL roles ---merged 107

Abuse case question ----218

Elicitation Method ---merged 237

****

SQUARE Process

page -170 T1

****

Output of SQUARE

pg-176 T1

****

Software is not 100% secure Page 125 T1

****

SAM Security recuirement ---merged 242

SDL security requirement ---merged 249

SDL Agile --merged 254

Architecture fundamental Design concept ---merged 280

Virtual desktop infrastructure (VDI) can reduce the administrative and management headaches introduced by user-owned devices in the enterprise. VDI enables sensitive applications and data to live in the data center where they can be centrally protected and managed

****

*********Slide Reference ***************************

Asymetric property of security

No matter how much effort we expand , we will never get code 100 % coreect

Thuis is asymetric problem

we must be 100 percent correct 100 % of the time , on schedule , with limited resources only knowing that we nknow today

Product has to be reliable , supportable , compatible , manegable , affordable ,accessible , usable , global , doable , deployablae

Hacker can spend as long as like to find the bug with the benifit of futrure resarch and unlimited resources

They do not find every bug in the , tehy need to find only one bug to exploit.Many tools avaialble that make life of hacker easy.

Reverse engineering tools/Penteration tools available which hacker can use , metaspolit tool.

Large number of tools available

Cost of attacker to build an attack is very low

built for organization the cost of attack is high and to reacting is higher than defending the attack.

****

Software Security emergent properties

slide 82 --merged

slide 213 --merged

Method of defence ---merged 138

****

Design for security is conecrned with:

slide no --68

Make sure meet CIA

Resistance and resilience.

How to prevent unauthorized disclosure,creation,change, deletion or deneial of access to information and other resources

It is also concerned with how to tolerate with scecurity related attack or voilations by limiting the damage,continuing service, sppeding repair and recovery and failing recovering securely/.

****

Software quality

slide no 74

Software engineering economics --slide no 80

****

Goal of security engineering

The goal of software security engineering is to build better, defect-free software. Software-intensive systems that are constructed using more securely developed software are better able to do the following: Continue operating correctly in the presence of most attacks by either resisting the exploitation ofweaknesses in the software by attackers or tolerating the failures that result from such exploits Limit the damage resulting from any failures caused by attack-triggered faults that the software was unable to resist or tolerate and recover as quickly as possible from those failures

The objective of software security is to field software-based systems that satisfy the following criteria: The system is as vulnerability and defect free as possible. The system limits the damage resulting from any failures caused by attack-triggered faults, ensuring that the effects of any attack are not propagated, and it recovers as quickly as possible from those failures. The system continues operating correctly in the presence of most attacks by either resisting the exploitation of weaknesses in the software by the attacker or tolerating the failures that result from such

exploits.

****

Software that has been developed with security in mind generally reflects the following properties throughout its

development life cycle:

Predictable execution. There is justifiable confidence that the software, when executed, functions as

intended. The ability of malicious input to alter the execution or outcome in a way favorable to the attacker is significantly reduced or eliminated.

Trustworthiness. The number of exploitable vulnerabilities is intentionally minimized to the greatest

extent possible. The goal is no exploitable vulnerabilities.

Conformance. Planned, systematic, and multidisciplinary activities ensure that software components, products, and systems conform to requirements and applicable standards and procedures for specified uses.

In addition to predictable execution, trustworthiness, and

conformance, secure software and systems should be as attack resistant, attack tolerant, and attack resilient as possible. To ensure that these criteria are satisfied, software engineers should design software components and systems to recognize both legitimate inputs and known attack patterns in the data or signals they receive from external entities (humans or processes) and reflect this recognition in the developed software to the extent possible and practical.

To achieve attack resilience, a software system should be able to recover from failures that result from successful attacks by resuming operation at or above some predefined minimum acceptable level of service in a timely manner. The system must eventually recover full service at the pecified level of performance.

****

The Role of Processes and Practices in Software Security]

A number of factors influence how likely software is to be secure. For instance, software vulnerabilities can originate in the processes and practices used in its creation. These sources include the decisions made by software engineers, the flaws they introduce in specification and design, and the faults and otherdefects they include in developed code, in advertently or intentionally. Other factors may include them choice of programming languages and development tools

used to develop the software, and the configuration and behavior of software components in their development and

operational environments. It is increasingly observed, however, that the most critical difference between secure software and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software

****

Threats to Software Security

Threats during development (mainly insider threats). A software engineer can sabotage the software at any point in its development life cycle through intentional exclusions from, inclusions in, or modifications of the requirements specification, the threat models, the

design documents, the source code, the assembly and integration framework, the test cases and test results, or the installation and configuration instructions and tools. The secure development practices described in this book are, in part, designed to help reduce the exposure of software to insider threats during its development process

Threats during operation (both insider and external threats). Any software system that runs on a network connected platform is likely to have its vulnerabilities exposed to attackers during its operation. Attacks may take advantage of publicly known but unpatched vulnerabilities, leading to memory corruption, execution of arbitrary exploit scripts, remote code execution, and buffer overflows. Software flaws can be exploited to install spyware, adware, and other malware on users' systems that can lie dormant until it is triggered to execute.[

****

Software—especially networked, application-level software —is most often compromised by exploiting weaknesses that

result from the following sources:

Complexities, inadequacies, and/or changes in the software's processing model (e.g., a Web- or serviceoriented architecture model).

Incorrect assumptions by the engineer, including assumptions about the capabilities, outputs, and

behavioral states of the software's execution environment or about expected inputs from external

entities (users, software processes).

Flawed specification or design, or defective

implementation of

- The software's interfaces with external entities.

Development mistakes of this type include inadequate (or nonexistent) input validation, error

handling, and exception handling.

- The components of the software's execution

environment (from middleware-level and operatingsystem-

level to firmware- and hardware-level

components).

Unintended interactions between software components,

including those provided by a third party.

****
Prevent and remove defect early...!!

If an organization can prevent defects or detect and remove them early, it can realize significant cost and schedule benefits. Studies have found that reworking defective requirements, design, and code typically accounts for 40 to 50 percent of the total cost of software development [Jones 1986b]. As a rule of thumb, every hour an organization spends on defect prevention reduces repair time for a system in production by three to ten hours. In the worst case, reworking a software requirements problem once the software is in operation typically costs 50 to 200 times whatit would take to rework the same problem during the requirements phase

The savings potential from early defect detection is significant: Approximately 60 percent of all defects usually exist by design time [Gilb 1988]. A decision early in a project to exclude defect detection amounts to a decision to postpone defect detection and correction until later in the project, when defects become much more expensive and time-consuming to address

Since nearly three-quarters of security-related defects are design issues that could be resolved inexpensively during the early stages, a significant opportunity for cost savings exists when secure software engineering principles are applied during design.

****

Security breach eg:

Stack Overflow exploit

If an attacker submits a very long string of input data that includes both malicious code and a return

address pointer to that code, because the program does not do bounds checking, the input will be accepted by the program and will overflow the stack buffer that receives it. This outcome will allow the malicious code to be loaded onto the program's execution stack and overwrite the subroutine return address so that it points to that malicious code. When the subroutine terminates, the program will jump to the malicious code, which will be executed, operating with root privilege. This particular malicious code is written to call the system shell, enabling the attacker to take control of the system. (Even if the original program had not operated with root privileges, the malicious code may have contained a privilege escalation exploit to gain those privileges.)

****

Attack resistance is the ability of the software to prevent the capability of an attacker to execute an

attack against it. The most critical of the three characteristics, it is nevertheless often the most difficult to achieve, as it involves minimizing exploitable weaknesses at all levels of abstraction, fromarchitecture through detailed implementation and deployment. Indeed, sometimes attack resistance is impossible to fully achieve.

Attack tolerance is the ability of the software to "tolerate" the errors and failure that result from successful attacks and, in effect, to continue to operate as if the attacks had not occurred.

Attack resilience is the ability of the software to isolate, contain, and limit the damage resulting from any failures caused by attack-triggered faults that the software was unable to resist or tolerate and to recover as quickly as possible from those failures.[7]

Attack tolerance and attack resilience are often a result of effective architectural and design decisions rather than implementation wizardry. Software that can achieve attack resistance, attack tolerance, and attack resilience is implicitly more capable of maintaining its core security properties.

****
*

Failure to properly

check an array bound, for example, might permit an attacker to execute arbitrary code on the target host, whereas failure to perform proper input validation might enable an attacker to destroy an entire database.

****

Attack Surface

Put simply, your attack surface is the sum of your security risk exposure. Put another way, it is the aggregate of all known, unknown and potential vulnerabilities and controls across all software,

hardware, firmware and networks. A smaller attack surface can help make your organization less exploitable, reducing risk.

A typical attack surface has complex inter relationships among three main areas of exposure: software attack surface, network attack surface and the often-overlooked human attack surface

Software Attack Surface

The software attack surface is comprised of the software environment and its interfaces. These are the applications and tools available to authorized (andunauthorized) users. The software attack surface is calculated across a lot of different kinds of code, including applications, email services, configurations, compliance policy, databases, executables, DLLs, web pages, mobile device OS, etc

Network Attack Surface

The network attack surface presents exposure related to ports, protocols, channels, devices (from routers and firewalls to laptops and smart phones), services, network applications (SaaS) and even firmware interfaces.Depending on your infrastructure, may need to include cloud servers, data, systems and processes to your network attack surface.

Human Attack Surface

Humans have range of complex vulnerabilities that are frequently exploited. One of the great strengths of highly secure organizations is their emphasi on communicating security awareness and safety principles to their employees, partners, supply chain and even their customers.

A sampling of the existing definitions define the attack surface as follows:

- "...union of code, interfaces, services, protocols, and practices available

to all users, with a strong focus on what is accessible to unauthenticated 25 users." [3]

- "...the system's actions that are externally visible to its users and the

system's resources that each action increases or modifies." [4]

- "...a list of attack features: Open sockets, Open RPC endpoints, Open

named pipes, Services, etc

****

Attack surface reduction and managenment

Defining the Attack Surface of an Application¶

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

Difference between attack vector and attack surfacce

In an IT environment, an attack surface is referred to as the sum of all potential points or attack vectors from which an unauthorized user/attacker can gain unauthorized access to a system and extract data from within.

In other words, an attack surface consists of all endpoints and vulnerabilities an attacker could exploit to carry out a security breach. As such, it is a security best practice to keep the attack surface as small as possible to reduce the risk of unauthorized access or data theft.

As previously mentioned, an attack surface represents all the touchpoints on your network through which a perpetrator can attempt to gain unauthorized access to your software, hardware, network and cloud components.

On the other hand, an attack vector is the actual method the perpetrator employs to infiltrate or breach a system or network. Some common examples of attack vectors include compromised credentials, ransomware, malicious insiders, man-in-the-middle attacks, and poor or missing encryption.

****

Advance persistent threat

https://www.crowdstrike.com/cybersecurity-101/advanced-persistent-threat-apt/

An advanced persistent threat (APT) is a sophisticated, sustained cyberattack in which an intruder establishes an undetected presence in a network in order to steal sensitive data over a prolonged period of time. An APT attack is carefully planned and designed to infiltrate a specific organization, evade existing security measures and fly under the radar.

Executing an APT attack requires a higher degree of customization and sophistication than a traditional attack. Adversaries are typically well-funded, experienced teams of cybercriminals that target high-value organizations. They've spent significant time and resources researching and identifying vulnerabilities within the organization.

****

Supply chain attack

merge slides --- 50
https://www.cisa.gov/sites/default/files/publications/defending_against_software_supply_chain_attacks_508_1.pdf

****

Source Code Analysis Tool --merged 340

Source Code Analysis Tools (security analyzers) are automated tools for helping analysts find security-related problems in software

use data- and control-flow analysis to find subtler bugs and to reduce false alarms

Some vulnerabilities (e.g. use of strcpy()) can be detected with high accuracy, others are harder to detect, and, in fact, one can always devise vulnerabilities that are undetectable altogether.

Tools have tradeoff between false alarms (also known as false positives) and missed vulnerabilities (also known as false negatives)

can be configured to make a tool more sensitive (decreasing false negatives while increasing false positives) or make it less sensitive (increasing false negatives while decreasing false positives)

Capability of Security ANalyzer

Examining Calls to Potentially Insecure Library Functions

Detecting Bounds-Checking Errors and Scalar Type Confusion

Detecting Type Confusion Among References or Pointers

Detecting Memory Allocation Errors

Detecting Vulnerabilities that Involve Sequences of Operations (Control-Flow Analysis)

Data-Flow Analysis

Pointer-Aliasing Analysis

****

Characterstics of Good Cypher.

According to Claude Shannon

The amount of secrecy needed should determine the amount of labor appropriate for the encryption and decryption.

The set of keys and the enciphering algorithm should be free from complexity.

The implementation of the process should be as simple as possible.

Errors in ciphering should not propagate and cause corruption of further information in the message. The size of the enciphered text should be no larger than the text of the original message.

****

Principle of easiest Penteration:

An intruder must be expected to use any available means of penteration.The penteration may not necessarily by the most obvious means , nor is it necessarily the one against which teh most solid defence has been isntalled.And it certainely does not have to be the way we want to attacker to behave

****

L(s)>L(O) Allow to read the object of lower level

L(S)<L(O) To prevent Information leak higher level can not allow to write at lower level

****

Security issue in Java & C++

C is prone to buffer over flow problem

C++ has an increased use of pointers and global variables, which makes it easier for memory corruption to occur. Other than buffer overflows, C++ is also prone to injection vulnerabilities. As previously mentioned, injection vulnerabilities can be exploited by attackers in multiple ways such as hijacking control over the victim's device or leaking sensitive information, bringing a severe security risk to improper C++ implementations.

Due to its lack of exception handling and reliance on string functions, C is predisposed to multiple buffer overflow and string manipulation security vulnerabilities. As a buffer overflow vulnerability can result in the corruption or overwriting of the data found in an adjacent memory space, it gives attackers a wide range of control over a program's execution or a system's administrative permissions. Buffer overflow was classified as the most dangerous vulnerability in the CWE top 25 list in 2019 and it currently still poses a critical risk to a lot of systems.

Java is compelx and depend on large numenr of 3rd party libaraies Hence due to this the hacker able to exploit the vulnerabilities present in other dependent lib and this is the reason jaav ais also not ver secure

Java is still vulnerable to different types of malware injection. If this type of vulnerability is discovered and exploited by attackers, it can enable them to execute malicious commands on the behalf of the victim or gain access to sensitive information. The OWASP foundation classifies malware injection and its diverse forms at number 3 in their top 10 security vulnerability list. Recent developments have also brought to life a critical security threat coming from a vulnerability found in Log4J, a Java based logging utility. The newly discovered vulnerability allows attackers to install cryptocurrency-mining malware on affected systems and currently has a risk factor of 10 out of 10. Developers are scrambling to patch the issues and are urging the potential users of affected platforms to install the latest version patches on their devices. The vulnerability currently came to surface on various network surfaces as well as most notably Minecraft servers, and experts are warning that multiple other frameworks and systems present the potential for attack.

Java Security weakness

Fine grained control with lot of complexity.Learning curve for developers

Relies on user to secure their own environment via compelx Policy tool

Multiple JVM implementaion each have their own unique vulnerabilities

Reverse engineering of of class file to source code (software watermarking, code obfuscation can not be securely)

Several flaw have been addressed over the evolution of Java and JVM

****

Threat: Any circumstances or event with the potential to adversly impact organization operations (including mission, functions images or reputation) organizational assets, individual other organization or the Nations through an information system vai

unauthorized access , destruction, disclosure, modification of information or denial of service.Any eevnt that will cause undesirable impact or loss to an organization if it occurs.

A user profile that may reveal to public

--merged 132,133

Threat to Assets are --hardware, software , data

Vulnerability: Weakness in an information system, system security procedure internal controls, or implementation taht could be exploited by threat source. The absence or weakness of a safeguard.It can also be described as weakness in an asset or the method of ensuring that the asset is survivable.

eg: File system that does not authenticate the system

Risk: The measure of the extent to which and entity is threatened by a potential circumstances or event and typically a function

1)adverse impact that would arises if the circumstances or events occurs

2)likelihood of occurrence

Countermeasure-

Action, devices, procedures or technique that meet or oppose(ie counters) a threat or vulnerability or attack by eliminating

or preventing it, minimizing the harm it can cause or by discovering and discovering and reporting it so that corecctive action

can be taken, Action devices procedure technique or measure that reduce the vulnerability of information system

****

Attacker Prespective

Assuming the attacker's perspective involves looking at the software from the outside in. It requires thinking like attackers think, and analyzing and understanding the software the way they would to attack it. Through better understanding of how the software is likely to be attacked the software development team can better harden and secure it against attack.

The attackers' advantage is further strengthened by the fact that attackers have been learning how to exploit software for several decades, but the general software development community has not kept up-to-date with the knowledge that attackers have gained. This knowledge gap is also evident

****

CVE and CWE

Security Measurable program sponsored by the Department of Homeland Security and led by MITRE Corporation produces Common Vulnerabilities and Exposures (CVE), a dictionary of publicly known information security vulnerabilities and exposures, and Common Weakness Enumeration (CWE), a dictionary of software weakness types. Links to all three can be found on MITRE's Making Security Measurable site, along with links to other information security enumerations, languages, and repositories

****

Attack Pattern

Attack patterns describe the techniques that attackers might use to break software

Attack patterns help to categorize attacks in a meaningful way so that problems and solutions can be discussed effectively. They can identify the types of known attacks to which an application could be exposed so that mitigations can be built into the application. Another benefit of attack patterns is that they contain sufficient detail about how attacks are carried out to enable developers to help prevent them. Owing to the omission of information about software security

****

Pattern name and classification: Denial of Service – Decompression Bomb

Attack Prerequisites: The application must decompress compressed data. For the attack to be maximally effective, the decompression should happen automatically without any user intervention.

Description: The attacker generates a small amount of compressed data that will decompress to an extremely large amount of data. The compressed data may only be a few kilobytes in size, whereas the decompressed data may be several hundred gigabytes in size. The target is running software that automatically attempts to decompress the data in memory to analyze it (such as with antivirus software) or to display it (such as with web browsers). When the target software attempts to decompress the malicious data in memory, it runs out of memory and causes the target software and/or target host to crash.

Related Vulnerabilities or Weaknesses: CWE-Data Amplification, CVE-2005-1260

Method of Attack: By maliciously crafting compressed data and sending it to the target over any protocol (e.g., e-mail, HTTP, FTP).

Attack Motivation-Consequences: The attacker wants to deny the target access to certain resources.

Attacker Skill or Knowledge Required: Creating the exploit requires a considerable amount of skill. However, once such a file is available, an unskilled attacker can find vulnerable software and attack it.

Resources Required: No special or extensive resources are required for this attack.

Solutions and Mitigations: Restrict the size of output files when decompressing to a reasonable value. Especially, handle decompression of files with a large compression ratio with care. Builders of decompressors could specify a maximum size for decompressed content and then cease decompression and throw an exception if this limit is ever reached.

Context Description: Any application that performs decompression of compressed data in any format (e.g., image, archive, sound, gzip-ed HTML)

References: Decompression bomb vulnerabilities

** **************************

Pattern Name and Classification: A unique, descriptive identifier for the pattern.

Attack Prerequisites: What conditions must exist or what functionality and what characteristics must the target software have, or what behavior must it exhibit, for this attack to succeed?

Description: A description of the attack including the chain of actions taken.

Related Vulnerabilities or Weaknesses: What specific vulnerabilities or weaknesses (see the glossary for definitions) does this attack leverage? Specific vulnerabilities should reference industry-standard identifiers such as Common Vulnerabilities and Exposures (CVE) number, US-CERT number, etc. Specific weaknesses (underlying issues that may cause vulnerabilities) should reference industry-standard identifiers such as the Common Weakness Enumeration (CWE).

Method of Attack: What is the vector of attack used (e.g., malicious data entry, maliciously crafted file, protocol corruption)?

Attack Motivation-Consequences: What is the attacker trying to achieve by using this attack? This is not the end business/mission goal of the attack within the target context but rather the specific technical result desired that could be leveraged to achieve the end business/mission objective. This information is useful for aligning attack patterns to threat models and for determining which attack patterns from the broader set available are relevant for a given context.

Attacker Skill or Knowledge Required: What level of skill or specific knowledge must the attacker have to execute such an attack? This should be communicated on a rough scale (e.g., low, moderate, high) as well as in contextual detail of what type of skills or knowledge are required.

Resources Required: What resources (e.g., CPU cycles, IP addresses, tools, time) are required to execute the attack?

Solutions and Mitigations: What actions or approaches are recommended to mitigate this attack, either through resistance or through resiliency?

Context Description: In what technical contexts (e.g., platform, OS, language, architectural paradigm) is this pattern relevant? This information is useful for selecting a set of attack patterns that are appropriate for a given context.

References: What further sources of information are available to describe this attack?

** **************************

Pattern name and classification: Make the Client Invisible

Attack Prerequisites: The application must have a multi-tiered architecture with a division between client and server.

Description: This attack pattern exploits client-side trust issues that are apparent in the software architecture. The attacker removes the client from the communication loop by communicating directly with the server. This could be done by bypassing the client or by creating a malicious impersonation of the client.

Related Vulnerabilities or Weaknesses: CWE–Man-in-the-Middle (MITM), CWE- Origin Validation Error, CWE- Authentication Bypass by Spoofing, CWE- No Authentication for Critical Function, CWE-Reflection Attack in an Authentication Protocol

Method of Attack: Direct protocol communication with the server.

Attack Motivation-Consequences: Potentially information leak, data modification, arbitrary code execution, etc. These can all be achieved by bypassing authentication and filtering accomplished with this attack pattern.

Attacker Skill or Knowledge Required: Finding and initially executing this attack requires a moderate skill level and knowledge of the client-server communications protocol. Once the vulnerability is found, the attack can be easily automated for execution by far less skilled attackers. Skill level for leveraging follow-on attacks can vary widely depending on the nature of the attack.

Resources Required: None, although protocol analysis tools and client impersonation tools such as netcat can greatly increase the ease and effectiveness of the attack.

Solutions and Mitigations:

Increase Resistance to Attack: Utilize strong two-way authentication for all communication between client and server. This option could have significant performance implications.

Increase Resilience to Attack: Minimize the amount of logic and filtering present on the client; place it on the server instead. Use white lists on server to filter and validate client input.

Context Description: "Any raw data that exist outside the server software cannot and should not be trusted. Client-side security is an oxymoron. Simply put, all clients will be hacked. Of course the real problem is one of client-side trust. Accepting anything blindly from the client and trusting it through and through is a bad idea, and yet this is often the case in server-side design" [Hoglund 04].

References: Exploiting Software: How to Break Code, p.150 [Hoglund