**Threat Engine Rule Generation Logic**

The following rules are primarily derived from vulnerabilities documented in OWASP archives and chosen in their application to eHealth systems. References for each rule are referenced in the final report.

### eHealth Specific Threats

#### 1.1 Classification of Medical Device

|  |  |
| --- | --- |
| **Title** | Proper Classification of Medical Device |
| **Rule Activation Logic** | el.element.attributes.type == "tm.SmartWatch" || el.element.attributes.type == "tm.Pacemaker" || el.element.attributes.type == "tm.Electrocardiogram" || el.element.attributes.type == "tm.MobilePhone" || el.element.attributes.type == "tm.Laptop" || el.element.attributes.type == "tm.Tablet" |
| **Rule ID** | 1.1 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Federal Governments provide different classifications and requirements for medical devices.  When building a medical device, it is important to classify the device you are building to ensure the system meets standards defined by the main regulatory body in your operating regions.  The threat Your system/device will fail certification. |
| **Mitigation(s)** | Classify your eHealth device before development. Consult the most recent guidance documents provided by the Government of Canada to aid in classifying your device and understanding the systemic requirements. |

#### 1.2 Collection and Storage of Electronic Health Records

|  |  |
| --- | --- |
| **Title** | Compliance in the Collection and Storage of Electronic Health Records |
| **Rule Activation Logic** | el.element.attributes.type == "tm.SmartWatch" || el.element.attributes.type == "tm.Pacemaker" || el.element.attributes.type == "tm.Electrocardiogram" || el.element.attributes.type == "tm.MobilePhone" || el.element.attributes.type == "tm.Laptop" || el.element.attributes.type == "tm.Tablet" |
| **Rule ID** | 1.2 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Depending on the nation and region of subject data collection and storage (local or remote), specific operating rules may apply.  For example: In the storage of electronic health records in Canada, specific rules and legislation are put into place varying by province/territory and continuously change over time. The legislation is written through discussion of principles of consent to collection, limited use, security safeguards, and patient participation. |
| **Mitigation(s)** | Legal council is required when defining User Agreements and when engineering specific rules of collection or storage to ensure all defined standards and criterion are met for the region(s) of operation. |

### Network Specific

#### 2.1 CDP Manipulation

|  |  |
| --- | --- |
| **Title** | CDP Manipulation |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Process" && isTrue(el.element.isANetworkSwitch) |
| **Rule ID** | 2.1 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | CDP Manipulation: CDP packets are enabled on all interfaces by default on Cisco switches and they are transmitted in clear text which allows an attacker to analyze the packets and gain a wealth of information about the network device then the attacker can use this information to execute a known vulnerability against the device platform. |
| **Mitigation(s)** | Solution is to disable CDP on non-management interfaces. |

#### 2.2 MAC Flooding

|  |  |
| --- | --- |
| **Title** | MAC Flooding |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Process" && isTrue(el.element.isANetworkSwitch) |
| **Rule ID** | 2.2 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | MAC Flooding: Here the attacker floods the CAM table with MAC addresses more than the switch can store which leads to the switch operating as hub giving the attacker the opportunity to sniff all traffic on the segment. |
| **Mitigation(s)** | Configuring Port Security: It involves limiting the NO. of MACs allowed through a port and can also specify what is the MAC/MACs are., the switch port have to be in access mode, when a violation occurs one of 3 actions is taken based on your configuration (shutdown, protect and restrict). the default action is to shut down the port and a log message will appear, protect means ignore the violated MAC but there is no way to tell us that a violation had occurred, restrict is the same as protect but it adds a counter to the violation counter and a log message will appear also. if a port is shut down due to violation it has to be manually re opened using the shutdown and no shutdown commands in the same sequence or using the (config)#errdisable recovery cause security-violation then to set the recover interval (config)#errdisable recovery interval {time in sec} and to verify the error disable recovery state #sh errdisable recovery.  ii- Port Base Authentication or 802.1x and also called Identity Based Network Services (IBNS): Requires a PC to be authenticated before joining the LAN, can be combined with port security to allow only authenticated PCs with a specific MAC address to join the LAN. |

#### 2.3 VLAN Based Attacks

|  |  |
| --- | --- |
| **Title** | VLAN Based Attacks |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Process" && isTrue(el.element.isANetworkSwitch) |
| **Rule ID** | 2.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | VLAN hopping: Is when a station is able to access VLAN other than its own. This can be done through one of the following: A- Switch spoofing: A PC will claim to establish a trunk link between itself and the switch and gain all the VLAN information trying to get benefit of the switch default interfaces state (dynamic auto/desirable).  802.1q Double tagging: Here the attacker computer double tags the frame with the native VLAN on its trunk link and the second tag is for the destined victim VLAN, when the frame reaches the first switch it\'s rips off the first tag and forward it to all the trunk links configured for the native VLAN and when it reaches the second switch it will see the second tag and forward the fame to the victim VLAN. |
| **Mitigation(s)** | VLAN Hopping: I- Disable the DTP messages on trunk ports (using no negotiate) and avoid the switch defaults (dynamic auto/desirable) regarding trunk links as possible, better is to hardcode the ports. ii- Configure all the ports that should connect to end stations as access, assign them to an unused VLAN and shut them down.  Double Tagging:  I- The same steps as the switch spoofing. ii- Configuring VACL (VLAN Access Control List). iii- Private VLAN, PVLANs allows you to divide a VLAN into secondary VLANs, letting you isolate a set of ports from other ports within the same VLAN, we create a primary VLAN and a secondary VLANs as desired, we can have one isolated per primary but we can have as many ports in the isolated as desired, private VLAN can only be configured on switches in transparent VTP mode, ports within private VLAN can be one of three: - Community: communicates with other community ports and promiscuous ports. - Isolated: communicates with promiscuous only.  - Promiscuous: communicates with all ports. |

### Authentication and Authorization

#### 3.1 Empty String Password

|  |  |
| --- | --- |
| **Title** | Empty String Password |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Store" && isTrue(el.element.storesCredentials) && isTrue(el.element.remoteMedicalRecordStorage) |
| **Rule ID** | 3.1 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Using an empty string as a password is insecure.  It is never appropriate to use an empty string as a password. It is too easy to guess. An empty string password makes the authentication as weak with the user names which are normally public or guessable. This makes a brute-force attack against the login interface much easier. |
| **Mitigation(s)** | To counter this threat, a password that is not an empty string should be used. Users are suggested to have passwords with at least eight characters long. It is not appropriate to have an empty string as a password (Common Weakness Enumeration, 2018). |

#### 3.2 Password in Configuration File

|  |  |
| --- | --- |
| **Title** | Password in Configuration File |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Store" && isTrue(el.element.storesCredentials) && isTrue(el.element.checkboxRemoteMedicalRecordStorage) |
| **Rule ID** | 3.2 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Storing a password in a configuration file allows anyone who can read the file access to the password-protected resource. Developers sometimes believe that they cannot defend the application from someone who has access to the configuration, but this attitude makes an attacker\'s job easier. |
| **Mitigation(s)** | To mitigate this threat, 2 mitigations are required.  The configuration file needs to employ a form of Access Control to ensure only those who have the privilege to access that file, are the only ones allowed to access that (Han, Wong, & Chao).  To control the information contained in the configuration file, the passwords should be stored in encrypted text which will combine the use of hash functions and the use of salts to take any password of any size and produce a unique hash value of the password and combine it with the original password, that way the password cannot be determined from the file (Han, Wong, & Chao). |

#### 3.3 Hardcoded Password

|  |  |
| --- | --- |
| **Title** | Hardcoded Password |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Store" && isTrue(el.element.storesCredentials) && isTrue(el.element.checkboxRemoteMedicalRecordStorage) |
| **Rule ID** | 3.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Hardcoded passwords may compromise system security in a way that cannot be easily remedied.  It is never a good idea to hardcode a password. Not only does hardcoding a password allow all the project\'s developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability. |
| **Mitigation(s)** | To counter this threat of hardcoding passwords, there are several mitigations/countermeasures that can be implemented:  Ask user for the password. The program should not know the password of a user. The user should be presented with a challenge to enter their password for the program to not be compromised easily (CERN Comouter Security Team, 2019).  If an existing password is stored on an Authentication distributed server such as an AFS (Andrew Filesystem (Faraone, 2012)) or Kerberos, obtain the passwords from the server (CERN Comouter Security Team, 2019).  Have the password stored in a separate configuration file, where that file is strictly read access only and has a level of access control that only certain individuals and processes who have the right privilege can read the file (CERN Comouter Security Team, 2019). |

#### 3.4 Password Plaintext Storage

|  |  |
| --- | --- |
| **Title** | Password Plaintext Storage |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Store" && isTrue(el.element.storesCredentialsStore) |
| **Rule ID** | 3.4 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Storing a password in plaintext may result in a system compromise.  Password management issues occur when a password is stored in plaintext in an application\'s properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password. |
| **Mitigation(s)** | Passwords should never be stored in plain text. Rather these passwords should be stored in encrypted text which will combine the use of hash functions and the use of salts to take any password of any size and produce a unique hash value of the password and combine it with the original password, that way the password cannot be determined from the file (CERN Comouter Security Team, 2019). |

#### 3.5 Least Privilege Violation

|  |  |
| --- | --- |
| **Title** | Least Privilege Violation |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 3.5 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.  When a program calls a privileged function, such as chroot(), it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user. |
| **Mitigation(s)** | There are several ways to mitigate the least privilege violation:  Split an individual component into several components and assign lower privilege levels to those components (Common Weakness Enumeration, 2019).  Identify areas in the system which have that elevated privilege and use those components instead to accomplish the task (Common Weakness Enumeration, 2019).  Create a separate environment within the system/program where only within that area or environment has an elevated privilege (Common Weakness Enumeration, 2019). |

#### 3.6 Code Permission (Development Lifecycle)

|  |  |
| --- | --- |
| **Title** | Code Permission |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isFalse(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isFalse(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isFalse (el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isFalse(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isFalse(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isFalse(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 3.6 |
| **STRIDE Type** | Elevation of privilege, Information Disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | An active developer with access to unrelated module code may tamper or disclose sensitive project information (Interproject Code Access). |
| **Mitigation(s)** | Throughout the development lifecycle, there are several mitigations that can be used:  Within the Implementation phase, if a critical resource is being used, there should be a check to see if a resource has permissions/behavior which are not secure (such as a regular user being able to modify that resource). If there are such behaviors or permissions that exist, the program should create an error or exit the program (Common Weakness Enumeration, 2019).  Within the Architecture and Design phase, one should split up the software components based on privilege level and if possible, control what data, functions and resources each component uses based the privilege level (Common Weakness Enumeration, 2019). Another option in this phase is to create a separate environment within the system/program where only within that area or environment has an elevated privilege (Common Weakness Enumeration, 2019).  In the installation phase, default or most restrictive permissions should be set to avoid any code which doesn\'t have the permissions to be run. Also, the assumption that a system administrator will change the settings based on a manual is incorrect (Common Weakness Enumeration, 2019).  In the System Configuration phase, the configurable, executable files and libraries should be only have read and write access by the system administrator (Common Weakness Enumeration, 2019).  In the Documentation phase, within any documentation, any configurations that are suggested must be secure, and do not affect the operation of the computer or program (Common Weakness Enumeration, 2019). |

### Code Quality

#### 4.1 Double Free Error

|  |  |
| --- | --- |
| **Title** | Double Free Error |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess", "c, c++, assembly"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone", "c, c++, assembly"))) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageSmartWatch", "c, c++, assembly"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageLaptop", "c, c++, assembly"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageTablet", "c, c++, assembly"))) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageElectrocardiogram", "c, c++, assembly"))) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguagePacemaker", "c, c++, assembly"))) |
| **Rule ID** | 4.1 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | Double free errors occur when free() is called more than once with the same memory address as an argument.  Calling free() twice on the same value can lead to memory leak. When a program calls free() twice with the same argument, the program\'s memory management data structures become corrupted and could allow a malicious user to write values in arbitrary memory spaces. This corruption can cause the program to crash or, in some circumstances, alter the execution flow. By overwriting registers or memory spaces, an attacker can trick the program into executing code of his/her own choosing, often resulting in an interactive shell with elevated permissions.  When a buffer is free(), a linked list of free buffers is read to rearrange and combine the chunks of free memory (to be able to allocate larger buffers in the future). These chunks are laid out in a double linked list which points to previous and next chunks. Unlinking an unused buffer (which is what happens when free() is called) could allow an attacker to write arbitrary values in memory; essentially overwriting valuable registers, calling shellcode from its own buffer. |
| **Mitigation(s)** | To mitigate this threat, each allocation should only be freed once. Once the memory has been allocated, the pointer should be set to NULL to ensure the pointer cannot be freed again. In complicated error conditions, ensure that clean-up routines represent the state of allocation. If the language is object oriented, that object destructors delete each allocation of memory one time only (OWASP, 2018). |

#### 4.2 Leftover Debug Code

|  |  |
| --- | --- |
| **Title** | Leftover Debug Code |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.Electrocardiogram") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") |
| **Rule ID** | 4.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Debug code can create unintended entry points in a deployed web application.  A common development practice is to add \"back door\" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. When this sort of debug code is accidentally left in the application, the application is open to unintended modes of interaction. These back-door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application. |
| **Mitigation(s)** | To mitigate this threat, all debug code should be removed prior to delivery of code (Common Weakness Enumeration, 2019). |

#### 4.3 Memory Leak

|  |  |
| --- | --- |
| **Title** | Memory Leak |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess", "c, c++"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone", "c, c++"))) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforSmartWatch", "c, c++"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforLaptop", "c, c++"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforTablet", "c, c++"))) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforElectrocardiogram", "c, c++"))) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforPacemaker", "c, c++"))) |
| **Rule ID** | 4.3 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | High |
| **Description** | A memory leak is an unintentional form of memory consumption whereby the developer fails to free an allocated block of memory when no longer needed. The consequences of such an issue depend on the application itself. Consider the following general three cases:    Short Lived User-land Application: Little if any noticeable effect. Modern operating system recollects lost memory after program termination.    Long Lived User-land Application: Potentially dangerous. These applications continue to waste memory over time, eventually consuming all RAM resources. Leads to abnormal system behavior.    Kernel-land Process: Memory leaks in the kernel level lead to serious system stability issues. Kernel memory is very limited compared to user land memory and should be handled cautiously.    Memory is allocated but never freed. Memory leaks have two common and sometimes overlapping causes:    Error conditions and other exceptional circumstances.  Confusion over which part of the program is responsible for freeing the memory.    Most memory leaks result in general software reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing the program) or take advantage of other unexpected program behavior resulting from a low memory condition. |
| **Mitigation(s)** | To mitigate that threat, 3rd party tools/software are required to see if this vulnerability exists in the code. One such tool that can be used in a Unix/Linux environment is a program called Valgrind. This program will run the desired software program to be checked to check all memory allocation and de-allocation methods are working as intended (OWASP, 2016). |

#### 4.4 Null Dereference

|  |  |
| --- | --- |
| **Title** | Null Dereference |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.PaceMaker") || (el.element.attributes.type == "tm.Electrocardiogram") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") |
| **Rule ID** | 4.4 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The program can potentially dereference a null pointer, thereby raising a NullPointerException. Null pointer errors are usually the result of one or more programmer assumptions being violated. Most null pointer issues result in general software reliability problems, but if an attacker can intentionally trigger a null pointer dereference, the attacker might be able to use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks.    A null-pointer dereference takes place when a pointer with a value of NULL is used as though it pointed to a valid memory area.    Null-pointer dereferences, while common, can generally be found and corrected in a simple way. They will always result in the crash of the process, unless exception handling (on some platforms) is invoked, and even then, little can be done to salvage the process. |
| **Mitigation(s)** | To mitigate this threat, if possible, this vulnerability would be prevented, if the programming language that was used to program the software did not use pointers. Another mitigation suggestion is to check to see if the pointers are referenced correctly prior to their use (OWASP, 2017). |

#### 4.5 Logging Practices

|  |  |
| --- | --- |
| **Title** | Logging Practices |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Store" && isTrue(el.element.isALogStore)) || (el.element.attributes.type == "tm.Process" && isTrue(el.element.isALog)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isALogMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.isALogSmartPhone)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.isALogLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.isALogTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element. isALogElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.isALogPacemaker)) |
| **Rule ID** | 4.5 |
| **STRIDE Type** | Repudiation |
| **Status** | Open |
| **Severity** | Low |
| **Description** | Declare Logger Object as Static and Final:  It is good programming practice to share a single logger object between all of the instances of a particular class and to use the same logger for the duration of the program.  Don\'t Use Multiple Loggers:  It is a poor logging practice to use multiple loggers rather than logging levels in a single class.  Good logging practice dictates the use of a single logger that supports different logging levels for each class.  Don\'t Use System Output Stream:  Using System.out or System.err rather than a dedicated logging facility makes it difficult to monitor the behavior of the program. It can also cause log messages accidentally returned to the end users, revealing internal information to attackers. While most programmers go on to learn many nuances and subtleties about Java, a surprising number hang on to this first lesson and never give up on writing messages to standard output using System.out.println().  The problem is that writing directly to standard output or standard error is often used as an unstructured form of logging. Structured logging facilities provide features like logging levels, uniform formatting, a logger identifier, timestamps, and, perhaps most critically, the ability to direct the log messages to the right place. When the use of system output streams is jumbled together with the code that uses loggers properly, the result is often a well-kept log that is missing critical information. In addition, using system output streams can also cause log messages accidentally returned to end users, revealing application internal information to attackers.  Developers widely accept the need for structured logging, but many continue to use system output streams in their \"pre-production\" development. If the code you are reviewing is past the initial phases of development, use of System.out or System.err may indicate an oversight in the move to a structured logging system. |
| **Mitigation(s)** | To mitigate this threat the logging system should be centralized to the program and give different levels of detail, and log/display all security successes or failures. (OWASP, 2017) |

#### 4.6 Unreleased Resource

|  |  |
| --- | --- |
| **Title** | Unreleased Resource |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.usesExternalResourcesProcess) && isTrue(el.element.usesResourcesDirectlyProcess)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.usesExternalResourcesStore) && isTrue(el.element.usesResourcesDirectlyStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.usesExternalResourcesMobilePhone) && isTrue(el.element.usesResourcesDirectlyMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.usesExternalResourcesSmartWatch) && isTrue(el.element.usesResourcesDirectlySmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.usesExternalResourcesLaptop) && isTrue(el.element.usesResourcesDirectlyLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.usesExternalResourcesTablet) && isTrue(el.element.usesResourcesDirectlyTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element. usesExternalResourcesElectrcoardiogram) && isTrue(el.element.usesResourcesDirectlyElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. usesExternalResourcesPacemaker) && isTrue(el.element.usesResourcesDirectlyPacemaker)) |
| **Rule ID** | 4.6 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.  Resource leaks have at least two common causes:  Error conditions and other exceptional circumstances.  Confusion over which part of the program is responsible for releasing the resource. |
| **Mitigation(s)** | To mitigate this threat, the programming language used to program the desired program, should not allow this threat to occur. Another suggestion is to free all resources that have been allocated and be consistent in terms of how memory is allocated and de-allocated. To furthermore mitigate this threat, a suggestion is to release all the member components of a given object (Common Weakness Enumeration, 2019). |

#### 4.7 Use of Obsolete Methods

|  |  |
| --- | --- |
| **Title** | Use of Obsolete Methods |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Store" || el.element.attributes.type == "tm.Process" || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.Electrocardiogram") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") |
| **Rule ID** | 4.7 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The use of deprecated or obsolete functions may indicate neglected code.    As programming languages evolve, functions occasionally become obsolete due to:    Advances in the language  Improved understanding of how operations should be performed effectively and securely  Changes in the conventions that govern certain operations  Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way.    Refer to the documentation for this function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality. The remainder of this text discusses general problems that stem from the use of deprecated or obsolete functions. |
| **Mitigation(s)** | To mitigate this threat, the documentation for the program should be referred to, to determine the reason it is deprecated and to determine alternatives to using those methods, which may pose not only a function concern, but also a security concern (Common Weakness Enumeration, 2019). |

### Cryptography

#### 5.1 Parameters in URL

|  |  |
| --- | --- |
| **Title** | Sensitive Parameters in URL |
| **Rule Activation Logic** | el.element.attributes.type == "tm.Process" && isTrue(el.element.isAWebApplication) |
| **Rule ID** | 5.1 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Information exposure through query strings in URL is when sensitive data is passed to parameters in the URL. This allows attackers to obtain sensitive data such as usernames, passwords, tokens (authX), database details, and any other potentially sensitive data. Simply using HTTPS does not resolve this vulnerability. A very common example is in GET requests. |
| **Mitigation(s)** | To mitigate this threat, it is recommended to use a POST method, as those parameters that are passed in through the URL are not saved, and therefore cannot be exposed (Common Weakness Enumeration, 2018). |

#### 5.2 Improper Certificate Validation

|  |  |
| --- | --- |
| **Title** | Improper Certificate Validation |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.validatesCertProcess)) || (el.element.attributes.type == "tm.Store" && isFalse(el.element.validatesCertStore)) |
| **Rule ID** | 5.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software does not validate, or incorrectly validates, a certificate.  (Common Weakness Enumeration, 2018) |
| **Mitigation(s)** | Certificates should be carefully managed and check to assure that data are encrypted with the intended owner\'s public key.  If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.  (Common Weakness Enumeration, 2018) |

#### 5.3 Testing for Weak SSL/TLS Ciphers, Insufficient TLS Protection

|  |  |
| --- | --- |
| **Title** | Insufficient TLS Protection |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Flow" && isFalse(el.element.usesTLS)) |
| **Rule ID** | 5.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Sensitive data must be protected when it is transmitted through the network. Such data can include user credentials and credit cards. As a rule of thumb, if data must be protected when it is stored, it must be protected also during transmission.  HTTP is a clear-text protocol and it is normally secured via an SSL/TLS tunnel, resulting in HTTPS traffic. The use of this protocol ensures not only confidentiality, but also authentication. Servers are authenticated using digital certificates and it is also possible to use client certificate for mutual authentication.  Even if high grade ciphers are today supported and normally used, some misconfiguration in the server can be used to force the use of a weak cipher - or at worst no encryption - permitting to an attacker to gain access to the supposed secure communication channel. Other misconfiguration can be used for a Denial of Service attack.    See: <https://www.owasp.org/index.php/Testing_for_Weak_SSL/TLS_Ciphers,_Insufficient_Transport_Layer_Protection_(OTG-CRYPST-001)>  for more information |
| **Mitigation(s)** | To mitigate this threat, web servers that provide https services should have their configuration checked. As well, the validity of an SSL certificate should be checked from a client and server point of view. These would be checked using a variety of tools which are found on the following website:  <https://www.owasp.org/index.php/Testing_for_Weak_SSL/TLS_Ciphers,_Insufficient_Transport_Layer_Protection_(OTG-CRYPST-001)> (OWASP, 2019) |

#### 5.4 Use of hard-coded cryptographic key

|  |  |
| --- | --- |
| **Title** | Hard-coded Cryptographic Key |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isEncryptedMobilePhone)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.isEncryptedActor)) || (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm. SmartWatch" && isTrue(el.element.isEncryptedSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.isEncryptedLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.isEncryptedTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.isEncryptedElectrocardiogram)) ||(el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.isEncryptedPacemaker)) |
| **Rule ID** | 5.4 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | The use of a hard-coded cryptographic key tremendously increases the possibility that encrypted data may be recovered.  If hard-coded cryptographic keys are used, it is almost certain that malicious users will gain access through the account in question. |
| **Mitigation(s)** | To mitigate against this threat, this practice of hard coding the cryptographic key should be avoided to avoid exposing the cryptographic key to a potential adversary for exploitation (Common Weakness Enumeration, 2019) |

#### 5.5 Using faulty cryptographic algorithm

|  |  |
| --- | --- |
| **Title** | Faulty Cryptographic Algorithm |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isEncryptedMobilePhone) && isTrue(dropDownOptionsCheck("encryptionTypeForMobilePhone", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Process" && isTrue(el.element.isEncryptedProcess) && isTrue(dropDownOptionsCheck("encryptionTypeForProcess", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Actor" && isTrue(el.element.isEncryptedActor) && isTrue(dropDownOptionsCheck("encryptionTypeForActor", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Store" && isTrue(el.element.isEncryptedStore) && isTrue(dropDownOptionsCheck("encryptionTypeForStore", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Flow" && isTrue(el.element.isEncryptedFlow) && isTrue(dropDownOptionsCheck("encryptionTypeForFlow", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.SmartWatch" && isTrue(el.element.isEncryptedSmartWatch) && isTrue(dropDownOptionsCheck("encryptionTypeForSmartWatch", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Laptop" && isTrue(el.element.isEncryptedLaptop) && isTrue(dropDownOptionsCheck("encryptionTypeForLaptop", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Tablet" && isTrue(el.element.isEncryptedTablet) && isTrue(dropDownOptionsCheck("encryptionTypeForTablet", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Electrocardiogram" && isTrue(el.element.isEncryptedElectrocardiogram) && isTrue(dropDownOptionsCheck("encryptionTypeForElectrocardiogram", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) || (el.element.attributes.type == " tm.Pacemaker" && isTrue(el.element.isEncryptedPacemaker) && isTrue(dropDownOptionsCheck("encryptionTypeForStore", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) |
| **Rule ID** | 5.5 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Attempting to create non-standard and non-tested algorithms, using weak algorithms, or applying algorithms incorrectly will pose a high weakness to data that is meant to be secure. |
| **Mitigation(s)** | To mitigate this threat, a stronger cryptographic algorithm that is widely known to be secure should be used. Currently, AES is one of the most secure encryption algorithms and is recommended to be used (OWASP, 2019), (Singh & Supriya, 2013). |

### Environment (Platform Vulnerabilities)

#### 6.1 Insecure Transport

|  |  |
| --- | --- |
| **Title** | Insecure Transport |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Flow" && isFalse(el.element.usesTLS)) |
| **Rule ID** | 6.1 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | The application configuration should ensure that SSL is used for all access-controlled pages.  If an application uses SSL to guarantee confidential communication with client browsers, the application configuration should make it impossible to view any access-controlled page without SSL. However, it is not an uncommon problem that the configuration of the application fails to enforce the use of SSL on pages that contain sensitive data.  There are three common ways for SSL to be bypassed:  A user manually enters the URL and types \"HTTP\" rather than \"HTTPS\".  Attackers intentionally send a user to an insecure URL.  A programmer erroneously creates a relative link to a page in the application, failing to switch from HTTP to HTTPS. (This is particularly easy to do when the link moves between public and secured areas on a web site.) |
| **Mitigation(s)** | The first and foremost control that needs to be applied is to check for a lack of transport encryption. This can be done by:  Reviewing network traffic of the device, its mobile application and any cloud connections to determine if any information is passed in clear text  Reviewing the use of SSL or TLS to ensure it is up to date and properly implemented  Reviewing the use of any encryption protocols to ensure they are recommended and accepted  In order to ensure enough transport encryption:  Ensuring data is encrypted using protocols such as SSL and TLS while transiting networks.  Ensuring other industry standard encryption techniques are utilized to protect data during transport if SSL or TLS are not available.  Ensuring only accepted encryption standards are used and avoid using proprietary encryption protocols.  Ensuring the message payload encryption  Ensuring the secure encryption key handshaking.  Ensuring received data integrity verification.  (Micro Focus, 2019) |

#### 6.2 Path Traversal

|  |  |
| --- | --- |
| **Title** | Path Traversal |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.localAccessProcess) && isFalse(el.element.validatesInputProcess)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.localAccessStore) && isFalse(el.element.validatesInputStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.localAccessMobilePhone) && isFalse(el.element.validatesInputMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.localAccessSmartWatch) && isFalse(el.element.validatesInputSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.localAccessLaptop) && isFalse(el.element.validatesInputLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.localAccessTablet) && isFalse(el.element.validatesInputTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.localAccessElectrocardiogram) && isFalse(el.element.validatesInputElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.localAccessPacemaker) && isFalse(el.element.validatesInputPacemaker)) |
| **Rule ID** | 6.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | Allows attackers to access files that are not intended to be accessed. The software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory. By using special elements such as \"..\" and \"/\" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system. One of the most common special elements is the \"../\" sequence, which in most modern operating systems is interpreted as the parent directory of the current location. This is referred to as relative path traversal. Path traversal also covers the use of absolute pathnames such as \"/usr/local/bin\", which may also be useful in accessing unexpected files. This is referred to as absolute path traversal (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Input Validation: Assume all input is malicious. Use an \"accept known good\" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, \"boat\" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as \"red\" or \"blue.\" Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code\'s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.  Run your code using the lowest privileges that are required to accomplish the necessary tasks. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.  Run the code in a \"jail\" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a directory or which command can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.  Attack Surface Reduction: Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server\'s access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.  Ensure that error messages only contain minimal details that are useful to the intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can be used to refine the original attack to increase the chances of success. In the context of path traversal, error messages which disclose path information can help attackers craft the appropriate attack strings to move through the file system hierarchy (Han, Wong, & Chao). |

#### 6.3 Exposure of Private Information (Privacy Violation)

|  |  |
| --- | --- |
| **Title** | Exposure of Private Information (Privacy Violation) |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isFalse(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isFalse(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isFalse(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isFalse(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isFalse(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isFalse(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 6.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software does not properly prevent private data (such as credit card numbers) from being accessed by actors who either (1) are not explicitly authorized to access the data or (2) do not have the implicit consent of the people to which the data is related. Mishandling private information, such as customer passwords or Social Security numbers, can compromise user privacy and is often illegal. An exposure of private information does not necessarily prevent the software from working properly, and in fact it might be intended by the developer, but it can still be undesirable (or explicitly prohibited by law) for the people who are associated with this private information. Some examples of private information include: social security numbers, web surfing history, credit card numbers, bank accounts, personal health records such as medical conditions, insurance information, prescription records, medical histories, test and laboratory results (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Separation of Privilege by compartmentalizing the system to have \"safe\" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges. |

### Error Handling

#### 7.1 Catch Null Pointer Exception

|  |  |
| --- | --- |
| **Title** | Catch NullPointerException |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforSmartWatch", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforLaptop", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforTablet", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforElectrocardiogram", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforPacemaker", "objectivec, c#, java, python"))) |
| **Rule ID** | 7.1 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | It is generally a bad practice to catch NullPointerException. Programmers typically catch NullPointerException under three circumstances:  The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.  The program explicitly throws a NullPointerException to signal an error condition.  The code is part of a test harness that supplies unexpected input to the classes under test. This is the only acceptable scenario (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application (Common Weakness Enumeration, 2019). |

#### 7.2 Empty Catch Block

|  |  |
| --- | --- |
| **Title** | Empty Catch Block |
| **Rule Activation Logic** | {scope: {dropDownOptionsCheck: dropDownOptionsCheck}},[[Element, 'el','(el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess", "objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone","objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforSmartWatch","objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforLaptop","objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforTablet","objectivec, c#, java, python"))) || (el.element.attributes.type =="tm.Electrocardiogram" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforElectrocardiogram","objectivec, c#, java, python"))) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageforPacemaker","objectivec, c#, java, python")))'] |
| **Rule ID** | 7.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software detects a specific error but takes no actions to handle the error (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Properly handle each exception. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.  If a function returns an error, it is important to either fix the problem and try again, alert the user that an error has happened and let the program continue, or alert the user and close and cleanup the program.  When testing subject, the software to extensive testing to discover some of the possible instances of where/how errors or return values are not handled. Consider testing techniques such as ad hoc, equivalence partitioning, robustness and fault tolerance, mutation, and fuzzing (Common Weakness Enumeration, 2019). |

#### 7.3 Missing Error Handling

|  |  |
| --- | --- |
| **Title** | Missing Error Handling |
| **Rule Activation Logic** | {scope: {dropDownOptionsCheck: dropDownOptionsCheck}},[[Element, 'el','(el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess","python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.SmartWatch"&& isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageSmartWatch", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageLaptop","python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageTablet", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Electrocardiogram"&& isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageElectrocardiogram", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type =="tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguagePacemaker", "python, java, objectivec, c#, c++, c")))'] |
| **Rule ID** | 7.3 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | A web application must define a default error page for 404 errors, 500 errors, and to catch java.lang.Throwable exceptions prevent attackers from mining information from the application container\'s built-in error response. When an attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks. If the application shows the attacker a stack trace, it gives information that makes the attacker\'s job significantly easier. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components (OWASP, 2017). |
| **Mitigation(s)** | The application configuration should specify a default error page in order to guarantee that the application will never leak error messages to an attacker. Handling standard HTTP error codes is useful and user-friendly in addition to being a good security practice, and a good configuration will also define a last-chance error handler that catches any exception that could possibly be thrown by the application.  A specific policy for how to handle errors should be documented, including the types of errors to be handled and for each, what information is going to be reported back to the user, and what information is going to be logged. All developers need to understand the policy and ensure that their code follows it.  When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws in the site and/or hacking attempts. Very few sites have any intrusion detection capabilities in their web application, but it is certainly conceivable that a web application could track repeated failed attempts and generate alerts (OWASP, 2015). |

#### 7.4 Return Inside Finally Block

|  |  |
| --- | --- |
| **Title** | Return Inside Finally Block |
| **Rule Activation Logic** | {scope: {dropDownOptionsCheck: dropDownOptionsCheck}},[[Element, 'el','(el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess","python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.SmartWatch"&& isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageSmartWatch", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageLaptop","python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageTablet", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Electrocardiogram"&& isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageElectrocardiogram", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type =="tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguagePacemaker", "python, java, objectivec, c#, c++, c")))'], | |
| **Rule ID** | 7.4 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Low |
| **Description** | The code has a return statement inside a finally block, which will cause any thrown exception in the try block to be discarded (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Do not use a return statement inside the finally block. The finally block should have \"cleanup\" code (Common Weakness Enumeration, 2019). |

#### 7.5 Unchecked Error Condition

|  |  |
| --- | --- |
| **Title** | Unchecked Error Condition |
| **Rule Activation Logic** | {scope: {dropDownOptionsCheck: dropDownOptionsCheck}},[[Element, 'el','(el.element.attributes.type == "tm.Process" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageProcess","python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageMobilePhone", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.SmartWatch"&& isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageSmartWatch", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Laptop" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageLaptop","python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Tablet" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageTablet", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type == "tm.Electrocardiogram"&& isTrue(dropDownOptionsCheck("checkboxProgrammingLanguageElectrocardiogram", "python, java, objectivec, c#, c++, c"))) || (el.element.attributes.type =="tm.Pacemaker" && isTrue(dropDownOptionsCheck("checkboxProgrammingLanguagePacemaker", "python, java, objectivec, c#, c++, c")))'], |
| **Rule ID** | 7.5 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | The choice between a language which has named, or unnamed exceptions needs to be done. While unnamed exceptions exacerbate the chance of not properly dealing with an exception, named exceptions suffer from the up-call version of the weak base class problem.  A language can be used which requires, at compile time, to catch all serious exceptions. However, one must make sure to use the most current version of the API as new exceptions could be added.  Catch all relevant exceptions. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment (Common Weakness Enumeration, 2019). |

### Input Validation

#### 8.1 Deserialization of Untrusted Data

|  |  |
| --- | --- |
| **Title** | Deserialization of Untrusted Data |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.isEncryptedProcess) && isFalse(el.element.validatesInputProcess)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore) && isFalse(el.element.validatesInputStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isEncryptedMobilePhone) && isFalse(el.element.validatesInputMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.isEncryptedSmartWatch) && isFalse(el.element.validatesInputSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.isEncryptedLaptop) && isFalse(el.element.validatesInputLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.isEncryptedTablet) && isFalse(el.element.validatesInputTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.isEncryptedElectrocardiogram) && isFalse(el.element.validatesInputElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.isEncryptedPacemaker) && isFalse(el.element.validatesInputPacemaker)) |
| **Rule ID** | 8.1 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid. It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.  When deserializing data, populate a new object rather than just deserializing. The result is that the data flows through safe input validation and that the functions are safe.  Explicitly define a final object() to prevent deserialization.  Make fields transient to protect them from deserialization.  An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be.  Avoid having unnecessary types or gadgets available that can be leveraged for malicious ends. This limits the potential for unintended or unauthorized types and gadgets to be leveraged by the attacker. Whitelist acceptable classes. NOTE: This is alone is not a sufficient mitigation (Common Weakness Enumeration, 2019). |

#### 8.2 Expression Language Injection

|  |  |
| --- | --- |
| **Title** | Expression Language Injection |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 8.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | Server-side code injection vulnerabilities arise when an application incorporates user-controllable data into a string that is dynamically evaluated by a code interpreter. If the user data is not strictly validated, an attacker can use crafted input to modify the code to be executed and inject arbitrary code that will be executed by the server. Server-side code injection vulnerabilities are usually very serious and lead to complete compromise of the application\'s data and functionality, and often of the server that is hosting the application. It may also be possible to use the server as a platform for further attacks against other systems (Portswigger, 2019). |
| **Mitigation(s)** | Whenever possible, applications should avoid incorporating user-controllable data into dynamically evaluated code. In almost every situation, there are safer alternative methods of implementing application functions, which cannot be manipulated to inject arbitrary code into the server\'s processing.  If it is considered unavoidable to incorporate user-supplied data into dynamically evaluated code, then the data should be strictly validated. Ideally, a whitelist of specific accepted values should be used. Otherwise, only short alphanumeric strings should be accepted. Input containing any other data, including any conceivable code metacharacters, should be rejected (Portswigger, 2019). |

#### 8.3 Form Action Hijacking

|  |  |
| --- | --- |
| **Title** | Form Action Hijacking |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 8.3 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Form action hijacking vulnerabilities arise when an application places user-supplied input into the action URL of an HTML form. An attacker can use this vulnerability to construct a URL that, if visited by another application user, will modify the action URL of a form to point to the attacker\'s server. If a user submits the form then its contents, including any input from the victim user, will be delivered directly to the attacker. Even if the user doesn\'t enter any sensitive information, the form may still deliver a valid CSRF token to the attacker, enabling them to perform CSRF attacks. In some cases, web browsers may help exacerbate this issue by autocompleting forms with previously entered user input (Portswigger, 2019). |
| **Mitigation(s)** | Consider hard-coding the form action URL or implementing a whitelist of allowed values (Portswigger, 2019). |

#### 8.4 Improper Input Validation

|  |  |
| --- | --- |
| **Title** | Improper Input Validation |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.validatesInputProcess)) || (el.element.attributes.type == "tm.Store" && isFalse(el.element.validatesInputStore)) || (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.validatesInputMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isFalse(el.element.validatesInputSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isFalse(el.element.validatesInputLaptop)) || (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.validatesInputTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isFalse(el.element.validatesInputElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isFalse(el.element.validatesInputPacemaker)) |
| **Rule ID** | 8.4 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of Struts Validation Problems  Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.  Assume all input is malicious. Use an \"accept known good\" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, \"boat\" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as \"red\" or \"blue.\" Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code\'s environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid client-side enforcement of server-side security  Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.  Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection.  Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session (Common Weakness Enumeration, 2019). |

#### Missing XML Validation

|  |  |
| --- | --- |
| **Title** | Missing XML Validation |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.isEncryptedProcess) && isTrue(el.element.validateXML)) |
| **Rule ID** | 8.5 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software accepts XML from an untrusted source but does not validate the XML against the proper schema. Most successful attacks begin with a violation of the programmer\'s assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Always validate XML input against a known XML Schema or DTD.  It is not possible for an XML parser to validate all aspects of a document\'s content because a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document\'s structure and therefore guarantee to the code that processes the document that the content is well-formed.  An XML validator should be used to check to check the schema of the XML file. A suggested validator that can be used is found at this website : <https://www.freeformatter.com/xml-validator-xsd.html> (Common Weakness Enumeration, 2019) |

#### 8.6 Overly Permissive Regular Expression

|  |  |
| --- | --- |
| **Title** | Overly Permissive Regular Expression |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.userInputProcess)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.userInputStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.userInputMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.userInputSmartWatch)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.userInputTablet)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.userInputLaptop)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.userInputElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.userInputPacemaker)) |
| **Rule ID** | 8.6 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | High |
| **Description** | The product uses a regular expression that does not sufficiently restrict the set of allowed values (Common Weakness Enumeration, 2019) |
| **Mitigation(s)** | To mitigate this threat, where possible, ensure that the regular expressions does a check to see where the start and end string patterns are. As well there should be a restriction to limit the number of characters in a given string that the regular expression will check (Common Weakness Enumeration, 2019) (OWASP, 2013) |

#### 8.7 Process Control

|  |  |
| --- | --- |
| **Title** | Process Control |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.thirdPartyProcess)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.thirdPartyStore)) |
| **Rule ID** | 8.7 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker. Process control vulnerabilities take two forms: An attacker can change the command that the program executes by explicitly controlling what the command is. An attacker can change the environment in which the command executes by implicitly controlling what the command means. Process control vulnerabilities of the first type occur when either data enters the application from an untrusted source and the data is used as part of a string representing a command that is executed by the application. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | Libraries that are loaded should be well understood and come from a trusted source. The application can execute code contained in the native libraries, which often contain calls that are susceptible to other security problems, such as buffer overflows or command injection. All native libraries should be validated to determine if the application requires the use of the library. It is very difficult to determine what these native libraries do, and the potential for malicious code is high. In addition, the potential for an inadvertent mistake in these native libraries is also high, as many are written in C or C++ and may be susceptible to buffer overflow or race condition problems. To help prevent buffer overflow attacks, validate all input to native calls for content and length. If the native library does not come from a trusted source, review the source code of the library. The library should be built from the reviewed source before using it (Common Weakness Enumeration, 2018). |

#### 8.8 String Termination Error

|  |  |
| --- | --- |
| **Title** | String Termination Error |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 8.8 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Relying on proper string termination may result in a buffer overflow.  String termination errors occur when:  Data enters a program via a function that does not null terminate its output.  The data is passed to a function that requires its input to be null terminated (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Use a language that is not susceptible to these issues. However, be careful of null byte interaction errors with lower-level constructs that may be written in a language that is susceptible.  Ensure that all string functions used are understood fully as to how they append null characters. Also, be wary of off-by-one errors when appending nulls to the end of strings.  If performance constraints permit, special code can be added that validates null-termination of string buffers, this is a rather naive and error-prone solution.  Switch to bounded string manipulation functions. Inspect buffer lengths involved in the buffer overrun trace reported with the defect.  Add code that fills buffers with nulls (however, the length of buffers still needs to be inspected, to ensure that the non-null-terminated string is not written at the physical end of the buffer).  Visit the following pages for more information for mitigation strategies for strings in C and C++:  <http://www.informit.com/articles/article.aspx?p=2036582&seqNum=4>  <https://www.synopsys.com/blogs/software-security/detect-prevent-and-mitigate-buffer-overflow-attacks/> (Common Weakness Enumeration, 2019). |

#### 8.9 Unchecked Return Value

|  |  |
| --- | --- |
| **Title** | Unchecked Return Value |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 8.9 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Low |
| **Description** | The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions. Two common programmer assumptions are \"this function call can never fail\" and \"it doesn\'t matter if this function call fails\". If an attacker can force the function to fail or otherwise return a value that is not expected, then the subsequent program logic could lead to a vulnerability, because the software is not in a state that the programmer assumes. For example, if the program calls a function to drop privileges but does not check the return code to ensure that privileges were successfully dropped, then the program will continue to operate with the higher privileges (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, three techniques must be applied to all functions in the given program that is being evaluated:  Ensure all of the functions that return a value, actually return a value and confirm that the value is expected.  Ensure within each function, that the possible of return values are covered  Within each function, ensure that there is a check/default value when there is an error (Common Weakness Enumeration, 2019). |

#### 8.10 Unsafe JNI

|  |  |
| --- | --- |
| **Title** | Unsafe JNI |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 8.10 |
| **STRIDE Type** | Denial of service |
| **Status** | Open |
| **Severity** | Low |
| **Description** | When a Java application uses the Java Native Interface (JNI) to call code written in another programming language, it can expose the application to weaknesses in that code, even if those weaknesses cannot occur in Java. Many safety features that programmers may take for granted simply do not apply for native code, so you must carefully review all such code for potential problems. The languages used to implement native code may be more susceptible to buffer overflows and other attacks. Native code is unprotected by the security features enforced by the runtime environment, such as strong typing and array bounds checking (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, three techniques must be applied in the given program that is being evaluated:  Implement a form of error handling within each JNI call.  Avoid using any JNI calls if the native library is untrusted.  Seek an alternative to a JNI call such as using a Java API. |

#### 8.11 Unsafe use of reflection

|  |  |
| --- | --- |
| **Title** | Unsafe use of reflection |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.userInput)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.userInput))  || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.userInput)) |
| **Rule ID** | 8.11 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | This vulnerability is caused by unsafe use of the reflection mechanisms in programming languages like Java, C#, or Ruby, etc. An attacker may be able to create unexpected control flow paths through the application, potentially bypassing security checks. Exploitation of this weakness can result in a limited form of code injection. If an attacker can supply values that the application then uses to determine which class to instantiate or which method to invoke, the potential exists for the attacker to create control flow paths through the application that were not intended by the application developers. This attack vector may allow the attacker to bypass authentication or access control checks or otherwise cause the application to behave in an unexpected manner. This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the application\'s classpath or add new entries to the application\'s classpath. Under either of these conditions, the attacker can use reflection to introduce new, presumably malicious, behavior into the application (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | Refactor your code to avoid using reflection.  Do not use user-controlled inputs to select and load classes or code.  Apply strict input validation by using whitelists or indirect selection to ensure that the user is only selecting allowable classes or code (Common Weakness Enumeration, 2019). |

### Mobile Risk

#### 9.1 Insecure Data Storage

|  |  |
| --- | --- |
| **Title** | Insecure Data Storage |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Store" && isFalse(el.element.isEncryptedStore)) |
| **Rule ID** | 9.1 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | An adversary that has attained a lost/stolen mobile device; malware or another repackaged app acting on the adversary\'s behalf that executes on the mobile device. If an adversary physically attains the mobile device, the adversary hooks up the mobile device to a computer with freely available software. These tools allow the adversary to see all third-party application directories that often contain stored personally identifiable information (PII), or personal health records (PHR). An adversary may construct malware or modify a legitimate app to steal such information assets (OWASP, 2018) |
| **Mitigation(s)** | It is important to threat model your mobile app, OS, platforms and frameworks to understand the information assets the app processes and how the APIs handle those assets. Determine how your application or software handles the following information:  URL caching (both request and response);  Keyboard press caching;  Copy/Paste buffer caching;  Application backgrounding;  Intermediate data  Logging;  HTML5 data storage;  Browser cookie objects;  Analytics data sent to 3rd parties (OWASP, 2018). |

#### Improper Platform Usage

|  |  |
| --- | --- |
| **Title** | Improper Platform Usage |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Tablet") |
| **Rule ID** | 9.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. The defining characteristic of risks in this category is that the platform (iOS, Android) provides a feature or a capability that is documented and well understood. The app fails to use that capability or uses it incorrectly. This differs from other mobile top ten risks because the design and implementation are not strictly the app developer\'s issue.  There are several ways that mobile apps can experience this risk.  Violation of published guidelines. All platforms have development guidelines for security (((Android)), ((iOS))). If an app contradicts the best practices recommended by the manufacturer, it will be exposed to this risk. For example, there are guidelines on how to use the iOS Keychain or how to secure exported services on Android. Apps that do not follow these guidelines will experience this risk.  Violation of convention or common practice: Not all best practices are codified in manufacturer guidance. In some instances, there are de facto best practices that are common in mobile apps.  Unintentional Misuse: Some apps intend to do the right thing but get some part of the implementation wrong. This could be a simple bug, like setting the wrong flag on an API call, or it could be a misunderstanding of how the protections work (OWASP, 2017). |
| **Mitigation(s)** | To mitigate this threat, secure coding and proper configurations must be used on the server side of the mobile application (OWASP, 2017). |

#### 9.3 Insecure Communication

|  |  |
| --- | --- |
| **Title** | Insecure Communication |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow) && isFalse(el.element.isPublicNetwork)) || (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow) && isFalse(el.element.wifiInterface)) || (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow) && isFalse(el.element.bluetoothInterface)) || (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow) && isFalse(el.element.cellularInterface)) |
| **Rule ID** | 9.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | When designing a mobile application, data is commonly exchanged in a client-server fashion. When the solution transmits its data, it must traverse the mobile device\'s carrier network and the internet. Attackers may exploit these vulnerabilities to intercept sensitive data such as: social security numbers, web surfing history, credit card numbers, bank accounts, personal health records such as medical conditions, insurance information, prescription records, medical histories, test and laboratory result while travelling across the wire (OWASP, 2017). |
| **Mitigation(s)** | Assume that the network layer is not secure and is susceptible to eavesdropping.  Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to a backend API or web service.  Account for outside entities like third-party analytics companies, social networks, etc. by using their SSL versions when an application runs a routine via the browser/webkit. Avoid mixed SSL sessions as they may expose the user\'s session ID.  Use strong, industry standard cipher suites with appropriate key lengths.  Use certificates signed by a trusted CA provider.  Never allow self-signed certificates and consider certificate pinning for security conscious applications.  Always require SSL chain verification.  Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain.  Alert users through the UI if the mobile app detects an invalid certificate.  Do not send sensitive data over alternate channels (e.g. SMS, MMS, or notifications).  If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. If future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation (OWASP, 2017). |

#### 9.4 Insecure Authentication

|  |  |
| --- | --- |
| **Title** | Insecure Authentication |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element. providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.providesAuthenticationPacemaker)) |
| **Rule ID** | 9.4 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | Authentication vulnerabilities are exploited through automated attacks that use available or custom-built tools. Once the adversary understands how the authentication scheme is vulnerable, they fake or bypass authentication by submitting service requests to the mobile app\'s backend server and bypass any direct interaction with the mobile app. This submission process is typically done via mobile malware within the device or botnets owned by the attacker (OWASP, 2017). |
| **Mitigation(s)** | Avoid weak authentication patterns:  If you are porting a web application to its mobile equivalent, authentication requirements of mobile applications should match that of the web application component. Therefore, it should not be possible to authenticate with less authentication factors than the web browser.  Authenticating a user locally can lead to client-side bypass vulnerabilities. If the application stores data locally, the authentication routine can be bypassed on jailbroken devices through run-time manipulation or modification of the binary. If there is a compelling business requirement for offline authentication, see M10 for additional guidance on preventing binary attacks against the mobile app;  Where possible, ensure that all authentication requests are performed server-side. Upon successful authentication, application data will be loaded onto the mobile device. This will ensure that application data will only be available after successful authentication;  If client-side storage of data is required, the data will need to be encrypted using an encryption key that is securely derived from the user\'s login credentials. This will ensure that the stored application data will only be accessible upon successfully entering the correct credentials. There are additional risks that the data will be decrypted via binary attacks. See M9 for additional guidance on preventing binary attacks that lead to local data theft;  Persistent authentication (Remember Me) functionality implemented within mobile applications should never store a user\'s password on the device;  Ideally, mobile applications should utilize a device-specific authentication token that can be revoked within the mobile application by the user. This will ensure that the app can mitigate unauthorized access from a stolen/lost device;  Do not use any spoof-able values for authenticating a user. This includes device identifiers or geo-location;  Persistent authentication within mobile applications should be implemented as opt-in and not be enabled by default;  If possible, do not allow users to provide 4-digit PIN numbers for authentication passwords.  Reinforce Authentication:  Developers should assume all client-side authorization and authentication controls can be bypassed by malicious users. Authorization and authentication controls must be re-enforced on the server-side whenever possible.  Due to offline usage requirements, mobile apps may be required to perform local authentication or authorization checks within the mobile app\'s code. If this is the case, developers should instrument local integrity checks within their code to detect any unauthorized code changes. See M9 for more information about detecting and reacting to binary attacks (OWASP, 2017). |

#### 9.5 Insufficient Transport Layer Protection

|  |  |
| --- | --- |
| **Title** | Insufficient Transport Layer Protection |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isFalse(el.element.isEncryptedFlow) && isFalse(el.element.usesTLS)) |
| **Rule ID** | 9.5 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | When designing a mobile application, data is commonly exchanged in a client-server fashion. When the solution transmits its data, it must traverse the mobile device\'s carrier network and the internet. Threat agents might exploit vulnerabilities to intercept sensitive data while it\'s traveling across the wire. The following ways are possible threat agents that exist:  An adversary that shares your local network (compromised or monitored Wi-Fi);  Carrier or network devices (routers, cell towers, proxy\'s, etc.); or  Malware on your mobile device (OWASP, 2014). |
| **Mitigation(s)** | General Best Practices:  Assume that the network layer is not secure and is susceptible to eavesdropping.  Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to a backend API or web service.  Account for outside entities like third-party analytics companies, social networks, etc. by using their SSL versions when an application runs a routine via the browser\'s webkit. Avoid mixed SSL sessions as they may expose the user\'s session ID.  Use strong, industry standard cipher suites with appropriate key lengths.  Use certificates signed by a trusted CA provider.  Never allow self-signed certificates and consider certificate pinning for security conscious applications.  Always require SSL chain verification.  Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain.  Alert users through the UI if the mobile app detects an invalid certificate.  Do not send sensitive data over alternate channels (e.g., SMS, MMS, or notifications).  If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. In the event that future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation.  iOS Specific Best Practices:  Default classes in the latest version of iOS handle SSL cipher strength negotiation very well. Trouble comes when developers temporarily add code to bypass these defaults to accommodate development hurdles. In addition to the above general practices:  Ensure that certificates are valid and fail closed.  When using CFNetwork, consider using the Secure Transport API to designate trusted client certificates. In almost all situations, NSStreamSocketSecurityLevelTLSv1 should be used for higher standard cipher strength.  After development, ensure all NSURL calls (or wrappers of NSURL) do not allow self-signed or invalid certificates such as the NSURL class method setAllowsAnyHTTPSCertificate.  Consider using certificate pinning by doing the following: export your certificate, include it in your app bundle, and anchor it to your trust object. Using the NSURL method connection:willSendRequestForAuthenticationChallenge: will now accept your cert.  Android Specific Best Practices:  Remove all code after the development cycle that may allow the application to accept all certificates such as org.apache.http.conn.ssl.AllowAllHostnameVerifier or SSLSocketFactory.ALLOW\_ALL\_HOSTNAME\_VERIFIER. These are equivalent to trusting all certificates.  If using a class which extends SSLSocketFactory, make sure checkServerTrusted method is properly implemented so that server certificate is correctly checked (OWASP, 2014). |

#### 9.6 Unintended Data Leakage

|  |  |
| --- | --- |
| **Title** | Unintended Data Leakage |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Store") |
| **Rule ID** | 9.6 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | Unintended data leakage occurs when a developer inadvertently places sensitive information or data in a location on the mobile device that is easily accessible by other apps on the device. This vulnerability is exploited by mobile malware, modified versions of legitimate apps, or an adversary that has physical access to the victim\'s mobile device. In case the attacker has physical access to the device, then the attacker can use freely available forensic tools to conduct an attack. Another possible attack vector would be if an attacker has access to the device via malicious code, so they will use fully permissible and documented API calls to conduct an attack (OWASP, 2014). |
| **Mitigation(s)** | Threat model your OS, platforms, and frameworks to determine how they handle the following features:  URL Caching (Both request and response)  Keyboard Press Caching  Copy/Paste buffer Caching  Application backgrounding  Logging  HTML5 data storage  Browser cookie objects  Analytics data sent to 3rd parties  Also identify what a given OS or framework does by default, by doing this and applying mitigating controls, unintended data leakage can be avoided (OWASP, 2014). |

#### 9.7 Broken/Insecure Cryptography

|  |  |
| --- | --- |
| **Title** | Broken/Insecure Cryptography |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isEncryptedMobilePhone) && isTrue(dropDownOptionsCheck("encryptionTypeForMobilePhone", "des, rsa, tripleDes, tripleDes3Key, rc2, rc4, 128rc4, desx"))) |
| **Rule ID** | 7.7 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | This threat is cause when an adversary has physical access to data that has been encrypted improperly, or mobile malware acting on an adversary\'s behalf. This can be done in several ways such as decryption access to the device or network traffic capture, or malicious apps on the device with access to the encrypted data Hello. |
| **Mitigation(s)** | To mitigate this threat, avoid using algorithms or protocols that are unsecure such as \'RC2\', \'MD4\', \'MD5\' and \'SHA1\'. A stronger cryptographic algorithm that is widely known to be secure should be used. Currently, AES is one of the most secure encryption algorithms and is recommended to be used. (OWASP, 2019) (Singh & Supriya, 2013) (OWASP, 2014) |

#### 9.8 Client-Side Injection

|  |  |
| --- | --- |
| **Title** | Client-Side Injection |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.userInputMobilePhone) && isTrue(el.element.validatesInputMobilePhone)) |
| **Rule ID** | 9.8 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Client-side injection results in the execution of malicious code on the mobile device via the mobile app. Consider anyone who can send untrusted data to the mobile app, including external users, internal users, the application itself or other malicious apps on the mobile device. A possible attack vector could be an adversary loads simple text-based attacks that exploit the syntax of the targeted interpreter within the mobile app. It is important to understand that almost any source of data can be an injection vector, including resource files or the application itself (OWASP, 2014). |
| **Mitigation(s)** | IOS Specific Best Practices:  SQLite Injection: When designing queries for SQLite be sure that user supplied data is being passed to a parameterized query. This can be spotted by looking for the format specifier used. In general, dangerous user supplied data will be inserted by a %@ instead of a proper parameterized query specifier.  JavaScript Injection (XSS, etc.): Ensure that all UIWebView calls do not execute without proper input validation. Apply filters for dangerous JavaScript characters if possible, using a whitelist over blacklist character policy before rendering. If possible, call mobile Safari instead of rending inside of UIWebkit which has access to your application.  Local File Inclusion: Use input validation for NSFileManager calls.  XML Injection: use libXML2 over NSXMLParser  Format String Injection: Several Objective C methods are vulnerable to format string attacks:  NSLog, [NSString stringWithFormat:], [NSString initWithFormat:], [NSMutableString appendFormat:], [NSAlert informativeTextWithFormat:], [NSPredicate predicateWithFormat:], [NSException format:], NSRunAlertPanel.  Do not let sources outside of your control, such as user data and messages from other applications or web services, control any part of your format strings.  Classic C Attacks: Objective C is a superset of C, avoid using old C functions vulnerable to injection such as: strcat, strcpy, strncat, strncpy, sprint, vsprintf, gets, etc.  Android Specific Best Practices:  SQL Injection: When dealing with dynamic queries or Content-Providers ensure you are using parameterized queries.  JavaScript Injection (XSS): Verify that JavaScript and Plugin support is disabled for any WebViews (usually the default).  Local File Inclusion: Verify that File System Access is disabled for any WebViews (webview.getSettings().setAllowFileAccess(false);).  Intent Injection\/Fuzzing: Verify actions and data are validated via an Intent Filter for all Activities.  Binary Injection\/Modification Prevention for Android and iOS:  Follow security coding techniques for jailbreak detection, checksum, certificate pinning, and debugger detection controls  The organization building the app must adequately prevent an adversary from analyzing and reverse engineering the app using static or dynamic analysis techniques.  The mobile app must be able to detect at runtime that code has been added or changed from what it knows about its integrity at compile time. The app must be able to react appropriately at runtime to a code integrity violation (OWASP, 2014). |

#### 9.9 Poor Client Code Quality

|  |  |
| --- | --- |
| **Title** | Poor Client Code Quality |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 9.9 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | This threat involves entities that can pass untrusted inputs to method calls made within mobile code. These types of issues are not necessarily security issues in and of themselves but lead to security vulnerabilities. For example, buffer overflows within older versions of Safari (a poor code quality vulnerability) led to high risk drive-by Jailbreak attacks. Poor code-quality issues are typically exploited via malware or phishing scams. An attacker will typically exploit vulnerabilities in this category by supplying carefully crafted inputs to the victim. These inputs are passed onto code that resides within the mobile device where exploitation takes place. Typical types of attacks will exploit memory leaks and buffer overflows (OWASP, 2017). |
| **Mitigation(s)** | To mitigate this threat, the following countermeasures should be considered:  Consistent coding patterns, standards in an organization  Write code that is legible and documented  Any code that requires a buffer, the length of the input should be checked, and the length should be restricted.  Use third party tools to find buffer overflows and memory leaks.  Prioritize to fix any buffer overflows and memory leaks that are present in the code before moving on to other issues. (OWASP, 2017) |

#### 9.10 Security Decisions Via Untrusted Inputs

|  |  |
| --- | --- |
| **Title** | Security Decisions Via Untrusted Inputs |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 9.10 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | This threat involves entities that can pass untrusted inputs to the sensitive method calls. Examples of such entities include, but are not limited to, users, malware and vulnerable apps An attacker with access to app can intercept intermediate calls and manipulate results via parameter tampering (OWASP, 2015). |
| **Mitigation(s)** | To mitigate this threat, avoid using depreciated/unsupported methods for each platform that the application is being used. As an example, for iOS, avoid using the handleOpenURL method to process URL scheme calls. Find an alternative method that is supported by the platform (OWASP, 2015). |

#### 9.11 Improper Session Handling

|  |  |
| --- | --- |
| **Title** | Improper Session Handling |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.isAWebApplication)) |
| **Rule ID** | 9.11 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | Anyone or any mobile app with access to HTTP/S traffic, cookie data, etc. Possible attack vectors include physical access to the device, and network traffic capture, or malware on the mobile device. Essentially an adversary that has access to the session tokens can impersonate the user by submitting the token to the backend server for any sensitive transactions such as credit card payments or health information like EKG results sent to a doctor (OWASP, 2014). |
| **Mitigation(s)** | Validate sessions on the backend by ensuring all session invalidation events are executed on the server side and not just on the mobile app.  Add adequate timeout protection to prevent the malicious potential for an unauthorized user to gain access to an existing session and assume the role of that user. Timeout periods vary accordingly based on the application, but some good guidelines are: 15 minutes for high security apps, 30 minutes for medium security apps, and 1 hour for low security apps.  Properly reset cookies during authentication state changes, by destroying sessions on the server side and making sure that the cookies presented as a part of the previous sessions are no longer accepted  In addition to properly invalidating tokens on the server side during key application events, make sure tokens are generated properly by using well-established and industry standard methods of creating tokens. Visit the following websites for more details: <https://www.pcisecuritystandards.org/documents/Tokenization_Product_Security_Guidelines.pdf> and <https://tools.ietf.org/html/rfc7519> for JSON Web Token (JWT) and <https://www.ietf.org/rfc/rfc6750.txt> for Bearer Token Usage (OWASP, 2014). |

#### 9.12 Lack of Binary Protections

|  |  |
| --- | --- |
| **Title** | Lack of Binary Protections |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.SmartWatch") |
| **Rule ID** | 9.12 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | High |
| **Description** | This threat involves an adversary who will analyze and reverse engineer a mobile app\'s code, then modify it to perform some hidden functionality. The majority of mobile apps do not prevent an adversary from successfully analyzing, reverse engineering or modifying the app\'s binary code (OWASP, 2015). |
| **Mitigation(s)** | To mitigate this threat from an adversary from analysis and reversing engineering the code, or unauthorized code modification, an application must follow very secure guidelines to activate the following mechanisms in a platform:  Jailbreak Detection Controls;  Checksum Controls;  Certificate Pinning Controls;  Debugger Detection Controls  These controls also require that the application have two more additional requirements. Firstly, the organization that is making the app must attempt to deny the adversary to analyze and reverse engineer the app using analysis techniques that can be static or dynamic. Lastly, the app must be able to determine at runtime if it\'s application code has been modified or added and react accordingly (OWASP, 2015). |

### Auditing

#### 10.1 Improper Output Neutralization for Logs

|  |  |
| --- | --- |
| **Title** | Improper Output Neutralization for Logs |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Store" && isTrue(el.element.isALogStore)) || (el.element.attributes.type == "tm.Process" && isTrue(el.element.isALog)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isALogMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.isALogSmartPhone)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.isALogLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.isALogTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element. isALogElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.isALogPacemaker)) |
| **Rule ID** | 10.1 |
| **STRIDE Type** | Repudiation |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software does not neutralize or incorrectly neutralizes output that is written to logs (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, there are 2 countermeasures that can be implemented. Firstly, any input should be assumed to be malicious. All input should be validated, where a whitelist should be used to accept input based on specific requirements. Properties that should be considered include length, type, full range of accepted values, missing or extra input, syntax, consistency and conforming to business logic. Another countermeasure is to have the output encoded in a particular format that a downstream consumer can use such as ISO-8859-1, UTF-7 and UTF-8 (Common Weakness Enumeration, 2019). |

#### 10.2 Insufficient Logging

|  |  |
| --- | --- |
| **Title** | Insufficient Logging |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 10.2 |
| **STRIDE Type** | Repudiation |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | When a security-critical event occurs, the software either does not record the event or omits important details about the event when logging it (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there are 2 countermeasures that can be implemented. Firstly, logging should be centralized with different levels of details. However, in a production environment there should be sufficient logging to allow system administrators to see attacks, diagnose and recover from errors (Common Weakness Enumeration, 2018). |

#### 10.3 Information Exposure Through Log Files

|  |  |
| --- | --- |
| **Title** | Information Exposure Through Log Files |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Store" && isTrue(el.element.isALogStore)) || (el.element.attributes.type == "tm.Process" && isTrue(el.element.isALog)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isALogMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.isALogSmartPhone)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.isALogLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.isALogTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element. isALogElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.isALogPacemaker)) |
| **Rule ID** | 10.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, there are a few mitigations that can be implemented. Firstly, any sensitive/secret information should not be written into any log files. Any debug log files should be removed prior to code being deployed in a production environment. Log files should be protected from unauthorized read/write access. Configurations should be changed when an application is transitioning to a production environment (Common Weakness Enumeration, 2019). |

#### 10.4 Logging of Excessive Data

|  |  |
| --- | --- |
| **Title** | Logging of Excessive Data |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Store" && isTrue(el.element.isALogStore)) || (el.element.attributes.type == "tm.Process" && isTrue(el.element.isALog)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.isALogMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.isALogSmartPhone)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.isALogLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.isALogTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element. isALogElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.isALogPacemaker)) |
| **Rule ID** | 10.4 |
| **STRIDE Type** | Repudiation |
| **Status** | Open |
| **Severity** | Low |
| **Description** | The software logs too much information, making log files hard to process and possibly hindering recovery efforts or forensic analysis after an attack (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there are a few mitigations that can be implemented. Firstly, large log files should be replaced with regularly commissioned summaries. Lastly, the log file\’s size should be restricted and controlled by a system administrator (Common Weakness Enumeration, 2018). |

### Authenticate Users

#### 11.1 Not using password aging

|  |  |
| --- | --- |
| **Title** | Not using password aging |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 11.1 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Low |
| **Description** | If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner (Common Weakness Enumeration, 2018) |
| **Mitigation(s)** | To mitigate this threat, an algorithm that would check how old a particular password is, should be implemented and used regularly. This algorithm must notify the user when their password is old and to change the password while not allowing the user to reuse old passwords as their new password (Common Weakness Enumeration, 2018). |

#### 11.2 Password Aging with Long Expiration

|  |  |
| --- | --- |
| **Title** | Password Aging with Long Expiration |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 11.2 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Low |
| **Description** | Allowing password aging to occur unchecked can result in the possibility of diminished password integrity (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there should be a maximum age that a password can be valid for (ex: 4 months) before the user has to change it. An algorithm should be implemented to check the password\’s age and notify users prior to expiration of that password (Common Weakness Enumeration, 2018). |

#### 11.3 Authentication Bypass Using an Alternate Path or Channel

|  |  |
| --- | --- |
| **Title** | Authentication Bypass Using an Alternate Path or Channel |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.3 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | A product requires authentication, but the product has an alternate path or channel that does not require authentication (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, all access is suggested to go through a centralized point of access, where each access of a resource requires a check to see if the user has permission to access that resource (Common Weakness Enumeration, 2018). |

#### 11.4 Authentication Bypass by Alternate Name

|  |  |
| --- | --- |
| **Title** | Authentication Bypass by Alternate Name |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.4 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, avoid hardcoding names of resources that are being accessed, if they can have alternate names (Common Weakness Enumeration, 2019). |

#### 11.5 Authentication Bypass by Capture-Replay

|  |  |
| --- | --- |
| **Title** | Authentication Bypass by Capture-Replay |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.5 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | A capture-replay flaw exists when the design of the software makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes) (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, a timestamp and or checksum with each response and check to see if it\’s an old request to stop a replay of the same authentication process (Common Weakness Enumeration, 2018). |

#### 11.6 Reflection Attack in an Authentication Protocol

|  |  |
| --- | --- |
| **Title** | Reflection Attack in an Authentication Protocol |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.6 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there a few mitigations that can be used. Firstly, it is recommended to have different keys for the requestor and responder of a challenge. Another suggestion is to provide different challenges for the requestor and responder. Prior to the challenge, it is recommended to have the requestor prove it\’s identity (Common Weakness Enumeration, 2018). |

#### 11.7 Authentication Bypass by Assumed-Immutable Data

|  |  |
| --- | --- |
| **Title** | Authentication Bypass by Assumed-Immutable Data |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.7 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The authentication scheme or implementation uses key data elements that are assumed to be immutable but can be controlled or modified by the attacker (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, any immutable data fields should be properly protected such as environment variables, and form fields to ensure that those fields are not tempered with (Common Weakness Enumeration, 2019). |

#### 11.8 Incorrect Implementation of Authentication Algorithm

|  |  |
| --- | --- |
| **Title** | Incorrect Implementation of Authentication Algorithm |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.8 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Low |
| **Description** | The requirements for the software dictate the use of an established authentication algorithm, but the implementation of the algorithm is incorrect (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, the algorithm should be fully tested, from endpoint to endpoint in a pre-production environment prior to being deployed in a production environment (Common Weakness Enumeration, 2018). |

#### 11.9 Missing Authentication for Critical Function

|  |  |
| --- | --- |
| **Title** | Missing Authentication for Critical Function |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.providesAuthenticationProcess))|| (el.element.attributes.type == "tm.Store" && isFalse(el.element.providesAuthenticationStore))|| (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type =="tm.SmartWatch" && isFalse(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop"&& isFalse(el.element.providesAuthenticationLaptop))|| (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.providesAuthenticationTablet))|| (el.element.attributes.type =="tm.Electrocardiogram" && isFalse(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type =="tm.Pacemaker" && isFalse(el.element.providesAuthenticationPacemaker)) |
| **Rule ID** | 11.9 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat there are several counter measures that can be implemented. Firstly, the application should be split up based on privilege levels where it\’s maintained by a centralized authentication mechanism. Secondly, any security check that was implemented on the client side of an application should also be on the server side. Another migration technique is to avoid designing and implementing authentication function that are custom-tailed to the application. Lastly, any library or framework which is known to have counter measures that will have the authentication function (Common Weakness Enumeration, 2019). |

#### 11.10 Improper Restriction of Excessive Authentication Attempts

|  |  |
| --- | --- |
| **Title** | Improper Restriction of Excessive Authentication Attempts |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.10 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there are multiple techniques can be used such as disconnecting the user after a certain number of failed attempts, having a timeout after a certain number of attempts or locking out a targeted account (Common Weakness Enumeration, 2018). |

**11.11 Use of Single-factor Authentication**

|  |  |
| --- | --- |
| **Title** | Use of Single-Factor Authentication |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.11 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, the system or application should use an extra method of authentication (multi-factor authentication). This ensures if one method is compromised, the system or application is still safe (Common Weakness Enumeration, 2018). |

#### 11.12 Key Exchange with Entity Authentication

|  |  |
| --- | --- |
| **Title** | Key Exchange with Entity Authentication |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.12 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software performs a key exchange with an actor without verifying the identity of that actor (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | There are two ways to mitigate this threat. Firstly, ensure when designing the system there is authentication involved. Lastly, validate that the checks that are actually verifying the identity of the user when communicating between identities (Common Weakness Enumeration, 2018). |

#### 11.13 Weak Password Requirements

|  |  |
| --- | --- |
| **Title** | Weak Password Requirements |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.providesAuthenticationProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.providesAuthenticationActor)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.providesAuthenticationStore)) || (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.providesAuthenticationMobilePhone)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.providesAuthenticationSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.providesAuthenticationLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.providesAuthenticationTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.providesAuthenticationElectrocardiogram)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element. providesAuthenticationPacemaker)) |
| **Rule ID** | 11.13 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, a password policy must be in place to have strong passwords (Common Weakness Enumeration, 2018).  A password policy (rules to make strong passwords) should in place to make a password much harder to guess for an attacker.  Such an example of a password policy is as follows:  All passwords should be reasonably complex and difficult for unauthorized people to guess. Employees and pupils should choose passwords that are at least eight characters long and contain a combination of upper- and lower-case letters, numbers, and punctuation marks and other special characters. These requirements will be enforced with software when possible (Gloucestershire Hospital Education Service - Password Policy, 2017). |

#### 11.14 Use of Client-Side Authentication

|  |  |
| --- | --- |
| **Title** | Use of Client-Side Authentication |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 11.14 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | A client/server product performs authentication within client code but not in server code, allowing server-side authentication to be bypassed via a modified client that omits the authentication check (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, authentication must also be performed on the server side of the application or system (Common Weakness Enumeration, 2018). |

#### 11.15 Unverified Password Change

|  |  |
| --- | --- |
| **Title** | Unverified Password Change |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 11.15 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | When setting a new password for a user, the product does not require knowledge of the original password or using another form of authentication (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, two techniques can be implemented. Firstly, when there is a password change, the user must provide the original password. Lastly, a “Forget Password” option can be used, but ensure that the user is requesting a change through a challenge (ex: enter email to receive an email which contains a link to change their password) and not actually changing the user\’s properties until they\’ve clicked that link (Common Weakness Enumeration, 2018). |

#### 11.16 Weak Password Recovery Mechanism for Forgotten Password

|  |  |
| --- | --- |
| **Title** | Weak Password Recovery Mechanism for Forgotten Password |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Flow" && isTrue(el.element.isEncryptedFlow)) || (el.element.attributes.type == "tm.Store" && isTrue(el.element.isEncryptedStore)) |
| **Rule ID** | 11.16 |
| **STRIDE Type** | Spoofing |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there are several counter measures that can be implemented. Ensure that all the input that goes through the mechanism is validated. If security questions are used, ensure that the questions are not simple and there are multiple questions. There should be a limit as to how many attempts one has to answer a question. The user must also answer the question before the password is reset. Do not allow the user to choose which email the password is sent to. As well, the original password should not be given, instead a new temporary password should be provided (Common Weakness Enumeration, 2018). |

### Authorize Users

#### 12.1 External Control of System or Configuration Setting

|  |  |
| --- | --- |
| **Title** | External Control of System or Configuration Setting |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isFalse(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isFalse(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isFalse(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isFalse(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isFalse(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isFalse(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.1 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | One or more system settings or configuration elements can be externally controlled by a user (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, the system can be split up by privilege level, so the settings/control are only changed by authorized users (Common Weakness Enumeration, 2019). |

#### 12.2 Process Control

|  |  |
| --- | --- |
| **Title** | Process Control |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 12.2 |
| **STRIDE Type** | Tampering |
| **Status** | Open |
| **Severity** | High |
| **Description** | Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, libraries and frameworks that are used must be from a trusted source, where these libraries can be relied upon and not be maliciously used by an adversary (Common Weakness Enumeration, 2018). |

#### 12.3 Sensitive Data Under Web Root

|  |  |
| --- | --- |
| **Title** | Sensitive Data Under Web Root |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.isAWebApplication)) |
| **Rule ID** | 12.3 |
| **STRIDE Type** | Information disclosure |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The application stores sensitive data under the web document root with insufficient access control, which might make it accessible to untrusted parties (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, avoid storing information under the web root directory, and access controls should be implemented to not allow these files to be read or written to (Common Weakness Enumeration, 2018). |

#### 12.4 Incorrect Privilege Assignment

|  |  |
| --- | --- |
| **Title** | Incorrect Privilege Assignment |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.4 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | High |
| **Description** | A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, the settings, managements and handling of privileges must be managed carefully. There should be accounts with limited privileges if there is a task that needs to be done, with very specific privilege levels (Common Weakness Enumeration, 2019). |

#### 12.5 Privilege Defined With Unsafe Actions

|  |  |
| --- | --- |
| **Title** | Privilege Defined With Unsafe Actions |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 12.5 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | High |
| **Description** | A particular privilege, role, capability, or right can be used to perform unsafe actions that were not intended, even when it is assigned to the correct entity (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, the settings, managements and handling of privileges must be managed carefully. There should be accounts with limited privileges if there is a task that needs to be done, with very specific privilege levels (Common Weakness Enumeration, 2019). |

#### 12.6 Privilege Chaining

|  |  |
| --- | --- |
| **Title** | Privilege Chaining |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process") || (el.element.attributes.type == "tm.Store") || (el.element.attributes.type == "tm.MobilePhone") || (el.element.attributes.type == "tm.Pacemaker") || (el.element.attributes.type == "tm.SmartWatch") || (el.element.attributes.type == "tm.Laptop") || (el.element.attributes.type == "tm.Tablet") || (el.element.attributes.type == "tm.Electrocardiogram") |
| **Rule ID** | 12.6 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | High |
| **Description** | Two distinct privileges, roles, capabilities, or rights can be combined in a way that allows an entity to perform unsafe actions that would not be allowed without that combination (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, the settings, managements and handling of privileges must be managed carefully. There should be accounts with limited privileges if there is a task that needs to be done, with very specific privilege levels. In addition to those techniques, privileges should be separated where multiple conditions need to be met to access (Common Weakness Enumeration, 2018). |

#### 12.7 Improper Privilege Management

|  |  |
| --- | --- |
| **Title** | Improper Privilege Management |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isFalse(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isFalse(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isFalse(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isFalse(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isFalse(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isFalse(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.7 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software does not properly assign, modify, track, or check privileges for an actor, creating an unintended sphere of control for that actor (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat three techniques are possible counter measures to properly manage privileges. There should be specific trust zones in the system, the least privilege principle should be in effect where the access rights of each user are given the minimum privilege level to do their task as well, privileges should be separated where multiple conditions need to be met to access (Common Weakness Enumeration, 2018). |

#### 12.8 Privilege Context Switching Error

|  |  |
| --- | --- |
| **Title** | Privilege Context Switching Error |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.8 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software does not properly manage privileges while it is switching between different contexts that have different privileges or spheres of control (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, three techniques are possible counter measures to properly manage privileges in different contexts. There should be specific trust zones in the system, the least privilege principle should be in effect where the access rights of each user are given the minimum privilege level to do their task as well, privileges should be separated where multiple conditions need to be met to access (Common Weakness Enumeration, 2018). |

#### 12.9 Privilege Dropping or Lowering Errors

|  |  |
| --- | --- |
| **Title** | Privilege Dropping or Lowering Errors |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.9 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | High |
| **Description** | The software does not drop privileges before passing control of a resource to an actor that does not have those privileges (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, three techniques are possible counter measures to properly manage privileges in different contexts. There should be specific trust zones in the system, the least privilege principle should be in effect where the access rights of each user are given the minimum privilege level to do their task as well, privileges should be separated where multiple conditions need to be met to access (Common Weakness Enumeration, 2018). |

#### 12.10 Improper Check for Dropped Privileges

|  |  |
| --- | --- |
| **Title** | Improper Check for Dropped Privileges |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.10 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software attempts to drop privileges but does not check or incorrectly checks to see if the drop succeeded (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, there are two techniques that can counter against an improper check for dropped privileges. Firstly, the system should be designed from the point of view of privilege level, where there are entry points and trust boundaries to interface components of different privilege levels. Ensure that all functions return a value and verify that the result is expected (Common Weakness Enumeration, 2019). |

#### 12.11 Incorrect Default Permissions

|  |  |
| --- | --- |
| **Title** | Incorrect Default Permissions |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.11 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software, upon installation, sets incorrect permissions for an object that exposes it to an unintended actor (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate the threat of default permissions the settings, management and handling of privileges should be carefully managed (Common Weakness Enumeration, 2019). |

#### 12.12 Insecure Inherited Permissions

|  |  |
| --- | --- |
| **Title** | Insecure Inherited Permissions |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.12 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Low |
| **Description** | A product defines a set of insecure permissions that are inherited by objects that are created by the program (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, the settings, management and handling of privileges need to be managed properly (Common Weakness Enumeration, 2018). |

#### 12.13 Incorrect Execution-Assigned Permissions

|  |  |
| --- | --- |
| **Title** | Incorrect Execution-Assigned Permissions |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.13 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | While it is executing, the software sets the permissions of an object in a way that violates the intended permissions that have been specified by the user (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate this threat, the settings, management and handling of privileges need to be managed properly (Common Weakness Enumeration, 2019). |

#### 12.14 Improper Handling of Insufficient Permissions or Privileges

|  |  |
| --- | --- |
| **Title** | Improper Handling of Insufficient Permissions or Privileges |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.14 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The application does not handle or incorrectly handles when it has insufficient privileges to access resources or functionality as specified by their permissions. This may cause it to follow unexpected code paths that may leave the application in an invalid state (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, there should be areas where there are specific permission levels. In addition, verify that if an access to a resource or system functionality is successful or not in all privilege levels (Common Weakness Enumeration, 2018). |

#### 12.15 Improper Ownership Management

|  |  |
| --- | --- |
| **Title** | Improper Ownership Management |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isTrue(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isTrue(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isTrue(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isTrue(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isTrue(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isTrue(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isTrue(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isTrue(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isTrue(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.15 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Medium |
| **Description** | The software assigns the wrong ownership, or does not properly verify the ownership, of an object or resource (Common Weakness Enumeration, 2018). |
| **Mitigation(s)** | To mitigate this threat, the settings, management and handling of privilege needs to be managed carefully (Common Weakness Enumeration, 2018). |

#### 12.16 Unverified Ownership

|  |  |
| --- | --- |
| **Title** | Unverified Ownership |
| **Rule Activation Logic** | (el.element.attributes.type == "tm.Process" && isFalse(el.element.privilegeLevelForProcess)) || (el.element.attributes.type == "tm.Actor" && isFalse(el.element.privilegeLevelForActor))|| (el.element.attributes.type == "tm.Store" && isFalse(el.element.privilegeLevelForStore))||  (el.element.attributes.type == "tm.MobilePhone" && isFalse(el.element.privilegeLevelForMobilePhone)) || (el.element.attributes.type == "tm.Pacemaker" && isFalse(el.element.privilegeLevelForPacemaker)) || (el.element.attributes.type == "tm.SmartWatch" && isFalse(el.element.privilegeLevelForSmartWatch)) || (el.element.attributes.type == "tm.Laptop" && isFalse(el.element.privilegeLevelForLaptop)) || (el.element.attributes.type == "tm.Tablet" && isFalse(el.element.privilegeLevelForTablet)) || (el.element.attributes.type == "tm.Electrocardiogram" && isFalse(el.element.privilegeLevelForElectrocardiogram)) |
| **Rule ID** | 12.16 |
| **STRIDE Type** | Elevation of privilege |
| **Status** | Open |
| **Severity** | Low |
| **Description** | The software does not properly verify that a critical resource is owned by the proper entity (Common Weakness Enumeration, 2019). |
| **Mitigation(s)** | To mitigate the threat of unverified ownership, the settings, management and handling of privilege needs to be managed carefully and the application needs to be designed from a separation of privilege point of view, which will require multiple conditions to access a resource (Common Weakness Enumeration, 2019). |

To do...