JONATHON BOURKE
C00242865

Research Document

CONTENTS

ABSTRACT

Vulnerability Management encompasses the acts that identify, evaluate, remediate, and document security weaknesses in software, hardware, and operating systems. Following good quality vulnerability management practices is vital in reducing organisations attack surface and prioritizing relevant security threats that are constantly discovered.

This document details any research that will have degrees of relevance in fulfilling the proposed project brief. It includes discussion into tackling the difficult challenges in its implementation.

ABBREVIATIONS

CVE - Common vulnerabilities and exposures

NVD - National Vulnerability Database

CVSS - Common vulnerability scoring system

CCE - Common configuration enumeration

CPE - Common platform enumeration

WFN - Well-formed name

DDOS – Distributed Denial of Service

CSRF – Cross-site Request Forgery

URL – Uniform Resource Locator

PVG – Patch and Vulnerability group

WAF – Web Application Firewall

AV – Anti Virus

OO – Object Orientated

ACID - Atomicity, Consistency, Isolation, Durability

CAP - Consistency, Availability and Partition Tolerance

OLAP – Online analytical processing

OLTP – Online transactional processing

VULNERABILITY MANAGEMENT

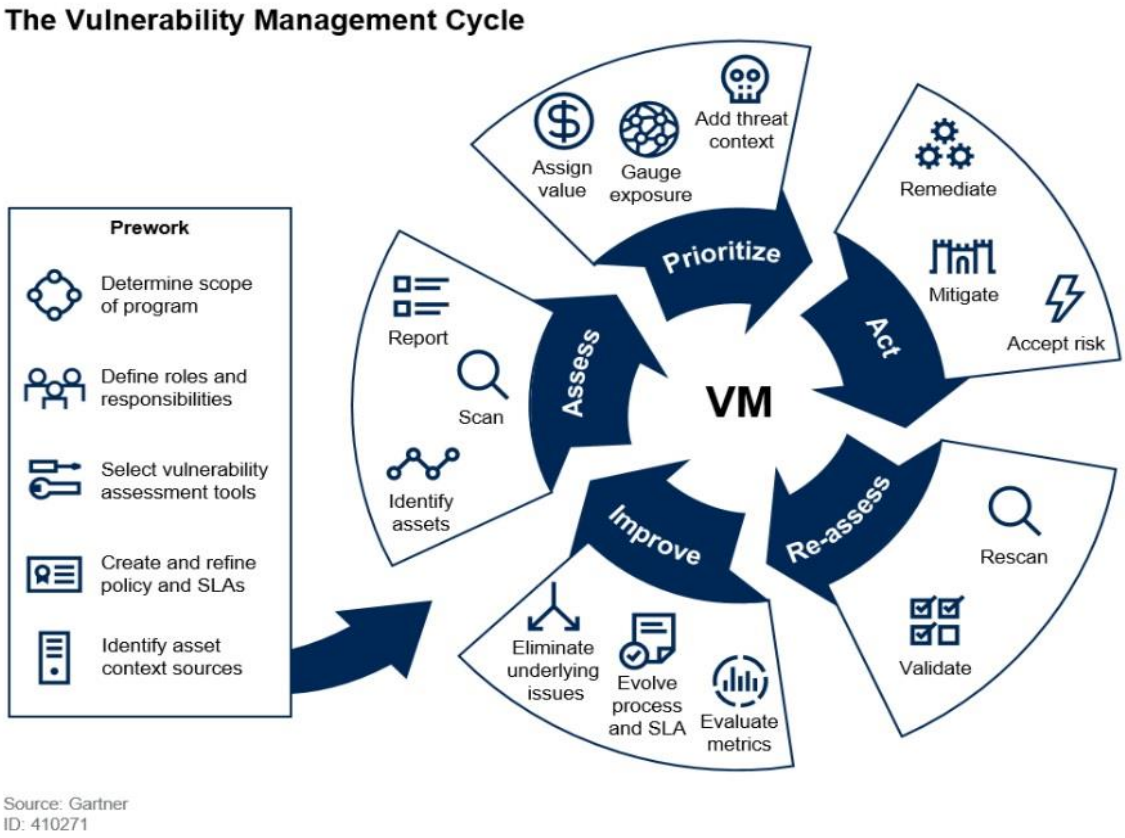
Vulnerability management is the constant cycle of finding, evaluating, and remediating cyber security issues in an environment. This helps an organisation identify and patch issues that arise from changing environments and new vulnerabilities. As a result, the organisations attack surface is minimized. The potential consequence of emerging threats can also be prevented.

A formal Vulnerability management is a critical part of any organisation's security program. It helps to educate the workforce on how threat adversaries gain initial access, bring attentions to existing vulnerabilities in the environment and the mitigation of their risk.

Data breaches and system disruption incidents are continuing to increase. This is augmented by the fact that becoming more damaging on organisations. It is estimated that the cost of a data breach on an organisation is increasing 10% every year. (IBM Security, 2021). These costs are significantly lower for organisations with mature security postures.

(IBM Security, 2021)

THE VULNERABILITY MANAGMENT LIFECYCLE



(Barros, 2019)

PREPARATION

Determine scope of program: To determine the vulnerability programs scope, the concerned organisational assets and services must be documented. Once these have been determined, the operational environment in which the assets are present should be evaluated and recorded. This will help to define the types of exposures the assets may be subject to.

Define roles and responsibilities: NIST recommends that organisations form a group known as the Patch and Vulnerability group (PVG). This should contain individuals who are skilled in the area. Individuals with knowledge of administration, intrusion detection and firewall management are advised to be included in the group. The PVG will monitor for potential vulnerabilities or weaknesses within the scope defined. The size and complexity needed for group roles varies in each organisation. Typical roles in the PVG include:

- Vulnerability monitoring
- Vulnerability priority
- Remediation database maintenance
- Remediation testing
- Remediation distribution
- Remediation verification
- Update Management
- Internal Training

(Mell, Bergeron, & Henning , Creating a Patch and Vulnerability Management Program, 2005)

Select vulnerability assessment tools: Selected suitable vulnerability assessment tools can be difficult due to varying needs and the expansive list of tools out there. Generally paid tools will give better performance than free tools. Most of the paid tools were once free but switched once their development hit a certain level. Below is a list of criteria to assess if the tool is sufficient:

- **Quality and Speed:** When a tool scans for vulnerabilities it happens in real time. This metric defines the reliability and promptness of the tool.
- **Usability:** The tools should be easy to interpret and provide intuitive functions with few to no complications.
- **Compatibility:** The product should be compatible with major operating systems, applications, and components.
- **Support:** Should be able to detect issues with all types of cloud computing in real time.
- **Compliance:** The selected tool should comply with all policies and standards set within the organisation.
- **Prioritization:** Should include manual and automatic prioritisation in its functionality.
- **Remediation assistance:** The tools should offer remediation steps on any vulnerability it finds
- **Vendor Support:** Should be included as part of the client-vendor contract. This should be one of the top prioritizations when selecting a tool.

Relevant personnel should receive training in the chosen tools under their scope.

Create and refine policy and SLAs

Policies should be created on the first lifecycle and further refined on at the start of subsequent policy periods. The scope of the policy should be determined first. Vulnerability scopes can fall under categories such as:

- Network
- Host
- Server
- Virtual Machine
- Operating systems
- Database
- Applications

Upon determining the scope of the vulnerability, the objectives and expectations should be established. The process in achieving the objectives would be documented in a detailed, high-level manner. Policies should follow also appropriate compliance standards that an organisation may be subject to.

The next step is to define roles and responsibilities for the policy. This is to ensure the smooth procedures and there is an appropriate sequence from reporting to remediation and measuring performance.

SLAs are timeframes for vulnerability remediation based on the organisations risk tolerance. These will be further influenced by the risk of the vulnerability and the importance the affected product has to the organisation. From this information, specific SLAs depending on the situation. This allows for resources to be allocated most effectively and to track performance metrics against the allocated SLA.

Identify asset context sources

Find up to date sources of vulnerability information about the types of assets used within the scope. Any new assets must be appropriately update with this information in a repository. Usually this will include any sources of information directly from the vendor and groups who specialise in the asset.

ASSESS

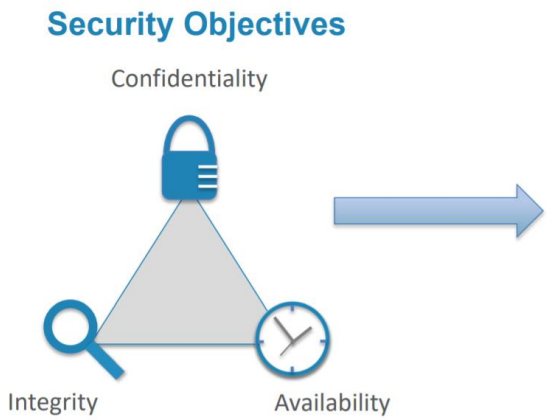
Build an asset inventory

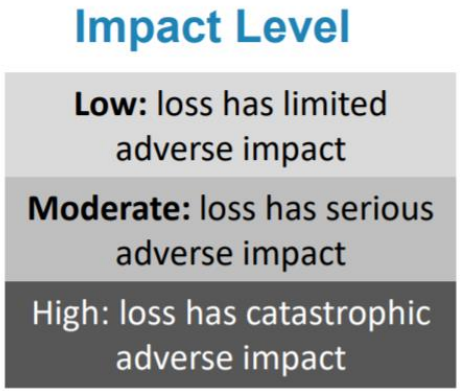
Create an inventory of assets comprised of hardware, Operating Systems, applications used within the organisation. This will define the scope and priority of the program and allow response to be quick and effective. It can be challenging to keep and maintain an accurate asset inventory due to diversity, lack of control over assets.

Identify and categorize assets

Assets should be defined under previous vulnerability policy items. This will help define SLAs when a risk score is determined for the asset. This can be aided by tools which will record all up-to-date assets within the environment. A risk profile will be included for each asset. This

is achieved by assigning values to different threats that are considered against the asset and its value.





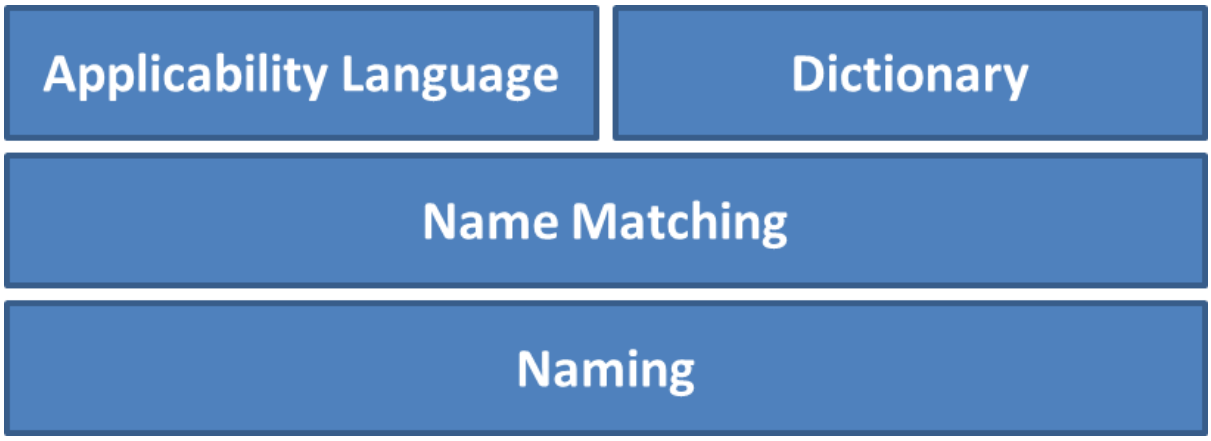
(NIST, 2021)

The value of an asset can be calculated by determining the subsequent loss of confidentiality, integrity, and availability in the event of compromise.

Common platform enumeration (CPE)

CPE is a standardised list of techniques used to find and categorize applications, operating systems and hardware that may be present in an organisations environment. CPE does not document unique variations of the product but instead identifies them abstractly such as Microsoft Word (all versions).

Current CPE 2.3 Stack



(NIST, 2021)

Applicability Language: Defines structure for forming logical expressions out of WFNs. These are used to highlight any checklists, policies, and related documents to the product.

Dictionary: A repository of CPE names and metadata. Names defines the class of the IT product

Name Matching: Defines methods and procedures for identifying similar WFNs to determine if they fall under some or all the same technology.

Naming: Contains four attributes; Well-formed names (WFNs), URI Bindings, formatting string bindings and procedures for converting WFNs to and from bindings

(NIST, 2021)

Vulnerability scan

This is the process of determining weaknesses on systems and applications. Identifying vulnerabilities can be automated using tools or through manual penetration testing.

- **Vulnerability Scanning:** An automated high-level scan performed by dedicated software that finds potential vulnerabilities in the environment.
- **Penetration Testing:** Process of probing for weaknesses on systems and applications performed by dedicated Penetration Testers. This has the advantage of determining the root cause of the weakness over vulnerability scanning.

A combination of the two is often the best approach. Vulnerability scanners can perform the repetitive tasks and find the high-level weaknesses. Penetration testing can dig deeper and expose vulnerabilities not picked up by the scanner. More refined overviews are also provided.

Vulnerability report

Once weaknesses have been determined in the vulnerability scanning phase, the findings must be summarised into a vulnerability report. This will include a description of the security weaknesses found, the assets it was discovered on and the severity.

The typical structure of the vulnerability report will be:

Executive summary: High level overview for readers. Should contain the risk imposed, the issues and how they can be secured. It may also contain any notable findings and if the objectives were achieved.

Assessment Overview: Summary of the activities of the vulnerability scans. This will detail the approach used, confirmation of results, tools utilized and the methodology of the evaluation.

Assessment Findings: Goes into detail on any of the vulnerabilities found, each with their own section noting the following items:

- Name of the vulnerability
- Discovery date
- Severity
- Description
- Steps to replicating
- Impact
- Remediation steps

Severities are normally ranked from highest to lowest. This emphasises the most critical to remediate first.

PRIORITIZE

Assign Value

Assign the vulnerability under the appropriate scope and product. Using risk profiles and scores set on the affected assets, determine the SLA for remediation.

Gauge Exposure

Determine the exposure the organisation has to the vulnerability. This will help with prioritization and resource allocation to the issue. Exposure may fall under categories like:

- Financial
- Competitive
- Reputational
- Regulatory

Add Threat Context

Find intelligence which provides more information on the vulnerability. This can include previous victim incidents, how the technique is employed by threat actors, their capabilities, typical targets and other information on these campaigns that might be beneficial in adding threat context.

ACT

Three types of action path may be taken depending on the characteristics of the found weakness, resources available and associated risk. Any action taken should be tested before rolling it out to the greater environment.

Remediate

Begin the procedures for remediation of the weakness on the affected assets. This will vary depending on the vulnerability. A full remediation means that the weakness will be fully neutralized. This will mean the threat is either patched or blocked on all affected assets.

Mitigate

Vulnerability mitigation involves lessening the impact the weakness could have on the organisation. The threat will not be fully nullified. This involves either reducing the attack surface of the or following mitigation protocols assisted by advisories. A mitigation might be chosen as opposed to a remediation if it could cause availability issues. The associated cost may outweigh the risk. Unavailable updates or patches may be the root cause. In these events, it is advisable to find relevant and accredited advisories to help determine the appropriate mitigation path. It is advised to review for any released patches once available.

Accept risk

Risk acceptance of a vulnerability may occur if the following conditions are met. The cost or negative impact of remediating/mitigation outweighs the damage potential on the organisation. The likelihood of exploitation may be incredibly low with mild negative impacts. Risk acceptance does not need to be indefinite and can be revisited once a more suitable resolution is available.

REASSESS

Validate

Any remediations/mitigations will have to gauged for their effectiveness against preventing the exploitation of the vulnerability. This can be achieved through efforts of a Penetration Testing team to expose any remaining risks after the patch was applied. There may have been issues with the deployment also. The success of the distribution should be confirmed

and remediated as necessary in the event of a problem. Any findings should be documented in the appropriate repositories.

Rescan

Preforming a precautionary vulnerability scan using chosen tools to determine that the asset is now protected against the weakness. Document any findings from the interpreted scan results and determine the effectiveness of previous remediation steps. If it is found remediated, the issue can be resolved. Otherwise, the issue should remain open and be scheduled for further analysis.

IMPROVE

Eliminate underlying issues

Determine any obstacles or challenges faced in this vulnerability management cycle. Areas of improvement should also be focused. The gathered information can be used for refining process and SLAs.

The root cause of the vulnerability's existence should be determined through analysis. Some example areas include:

- Third party vendor issue
- Misconfiguration
- Non-compliance with policies
- Inappropriate training
- Difficulty in operations

Evolve process and SLA

Using the gather information, decide how these underlying problems could be alleviated. SLA times may have been unsatisfactory or miscategorized. Research could be performed into preventing similar incidents in the future. Security training or incident response procedures could be improved depending on the potential faults discovered. Incorrect immediate actions could have been taken or the wrong personnel could have been notified for the vulnerability.

Evaluate metrics

Use the recorded metrics to determine the efficiency of vulnerability management processes**.** The main areas to evaluate are:

- The effectiveness of the remediation process
- The time it took from discovery and patching.
- Asset exposure time

VULNERABILITY MANAGEMENT MATURITY MODEL

The vulnerability management maturity model is a mixture of asset analysis, vulnerability scanning, patch management, policy implementation and its success metrics. These characteristics are used to define an organisation's rank in vulnerability management maturity. Under current models, this is comprised of five stages: Initial, Managed, Defined, Quantitively Managed and Optimised.

LEVEL 1 – INITIAL

This is the initial stages where an organisation starts a vulnerability management program. This is usually out of a need to establish compliance with various cyber security standards. This is normally seen with weak/non-existent policies that may also be intertwined with maintenance

LEVEL 2 – MANAGED

The organisation is beginning to expand its vulnerability management program. This can be due to an audit which requires the organisation to expand on something missing in their program. This will then result in specific policies been written. As a result, the company's program is categorized as Level 2.

LEVEL 3 – DEFINED

After some incidents, an organisation may begin researching on how to create a better vulnerability management program. This is normally followed by research into a better program for dealing with these threats. After some policy creation and internal training, the organisation is becoming more proactive in its approach. This marks the beginning of stage 3.

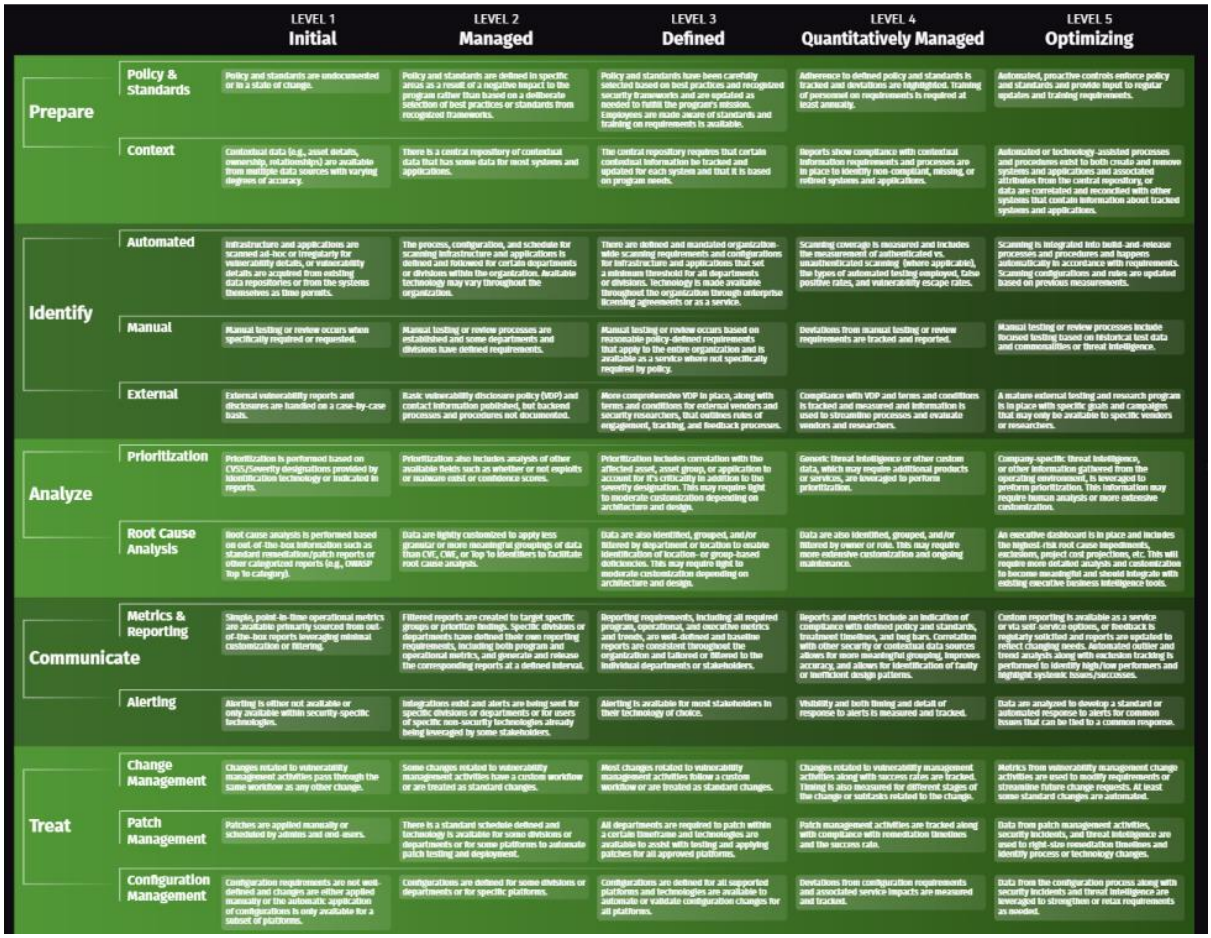
LEVEL 4 – QUANTITATIVELY MANAGED

Stage 4 is when an organisation begins tracking the effectiveness of various vulnerability management policies in place. This includes compliance, incidents and anomalies that may occur in its proceeding. The tracking of these lets the organisation focus on their weaknesses and implement the short comings in internal training.

LEVEL 5 – OPTIMIZED

The vulnerability management program is now driven by automated policies. When any anomalies are found, alerts are automatically generated and are subsequently remediated. These events are used to drive internal training to ensure these deviations do not become an issue.

(Risto, 2020)



		LEVEL 1 Initial	LEVEL 2 Managed	LEVEL 3 Defined	LEVEL 4 Quantitatively Managed	LEVEL 5 Optimizing
Prepare	Policy & Standards	Policy and standards are undocumented or in a state of change.	Policy and standards are defined in specific areas as a result of a negative impact to the program rather than based on a deliberate selection of best practices or standards from recognized frameworks.	Policy and standards have been carefully selected based on best practices and recognized security frameworks and are updated as needed to fulfill the program's mission. Employees are made aware of standards and training on requirements is available.	Adherence to defined policy and standards is tracked and deviations are highlighted. Training of personnel on requirements is required at least annually.	Automated, proactive controls enforce policy and standards and provide input to regular updates and training requirements.
	Context	Contextual data (e.g., asset details, ownership, relationships) are available from multiple data sources with varying degrees of accuracy.	There is a central repository of contextual data that has some data for most systems and applications.	The central repository requires that certain contextual information be tracked and updated for each system and that it is based on program needs.	Reports show compliance with contextual information requirements and processes are in place to identify non-compliant, missing, or retired systems and applications.	Automated or technology-assisted processes and procedures exist to both create and remove systems and applications and associated attributes from the central repository, or data are correlated and reconciled with other systems that contain information about tracked systems and applications.
Identify	Automated	Infrastructure and applications are scanned ad-hoc or irregularly for vulnerability details, or vulnerability details are acquired from existing data repositories or from the systems themselves as time permits.	The process, configuration, and schedule for scanning infrastructure and applications is defined and followed for certain departments or divisions within the organization. Available technology may vary throughout the organization.	There are defined and mandated organization-wide scanning requirements and configurations for infrastructure and applications that set a minimum threshold for all departments or divisions. Technology is made available throughout the organization through enterprise licensing agreements or as a service.	Scanning coverage is measured and includes the measurement of authenticated vs. unauthenticated scanning (where applicable), the types of automated testing employed, false positive rates, and vulnerability escape rates.	Scanning is integrated into build-and-release processes and procedures and happens automatically in accordance with requirements. Scanning configurations and rules are updated based on previous measurements.
	Manual	Manual testing or review occurs when specifically required or requested.	Manual testing or review processes are established and some departments and divisions have defined requirements.	Manual testing or review occurs based on reasonable policy-defined requirements that apply to the entire organization and is available as a service where not specifically required by policy.	Deviations from manual testing or review requirements are tracked and reported.	Manual testing or review processes include focused testing based on historical test data and commonalities or threat intelligence.
	External	External vulnerability reports and disclosures are handled on a case-by-case basis.	Basic vulnerability disclosure policy (VDP) and contact information published, but backend processes and procedures not documented.	More comprehensive VDP in place, along with terms and conditions for external vendors and security researchers, that outlines rules of engagement, tracking, and feedback processes.	Compliance with VDP and terms and conditions is tracked and measured and information is used to streamline processes and evaluate vendors and researchers.	A mature external testing and research program is in place with specific goals and campaigns that may only be available to specific vendors or researchers.
Analyze	Prioritization	Prioritization is performed based on CVSS/Severity designations provided by identification technology or indicated in reports.	Prioritization also includes analysis of other available fields such as whether or not exploits or malware exist or confidence scores.	Prioritization includes correlation with the affected asset, asset group, or application to account for it's criticality in addition to the severity designation. This may require light to moderate customization depending on architecture and design.	Generic threat intelligence or other custom data, which may require additional products or services, are leveraged to perform prioritization.	Company-specific threat intelligence, or other information gathered from the operating environment, is leveraged to perform prioritization. This information may require human analysis or more extensive customization.
	Root Cause Analysis	Root cause analysis is performed based on out-of-the-box information such as standard remediation/patch reports or other categorized reports (e.g., OWASP Top 10 category).	Data are lightly customized to apply less granular or more meaningful groupings of data than CVE, CWE, or Top 10 identifiers to facilitate root cause analysis.	Data are also identified, grouped, and/or filtered by department or location to enable identification of location- or group-based deficiencies. This may require light to moderate customization depending on architecture and design.	Data are also identified, grouped, and/or filtered by owner or role. This may require more extensive customization and ongoing maintenance.	An executive dashboard is in place and includes the highest-risk root cause impediments, exclusions, project cost projections, etc. This will require more detailed analysis and customization to become meaningful and should integrate with existing executive business intelligence tools.
Communicate	Metrics & Reporting	Simple, point-in-time operational metrics are available primarily sourced from out-of-the-box reports leveraging minimal customization or filtering.	Filtered reports are created to target specific groups or prioritize findings. Specific divisions or departments have defined their own reporting requirements, including both program and operational metrics, and generate and release the corresponding reports at a defined interval.	Reporting requirements, including all required program, operational, and executive metrics and trends, are well-defined and baseline reports are consistent throughout the organization and tailored or filtered to the individual departments or stakeholders.	Reports and metrics include an indication of compliance with defined policy and standards, treatment timelines, and bug bars. Correlation with other security or contextual data sources allows for more meaningful grouping, improves accuracy, and allows for identification of faulty or inefficient design patterns.	Custom reporting is available as a service or via self-service options, or feedback is regularly solicited and reports are updated to reflect changing needs. Automated outlier and trend analysis along with exception tracking is performed to identify high/low performers and highlight systemic issues/successes.
	Alerting	Alerting is either not available or only available within security-specific technologies.	Integrations exist and alerts are being sent for specific divisions or departments or for users of specific non-security technologies already being leveraged by some stakeholders.	Alerting is available for most stakeholders in their technology of choice.	Visibility and both timing and detail of response to alerts is measured and tracked.	Data are analyzed to develop a standard or automated response to alerts for common issues that can be tied to a common response.
Treat	Change Management	Changes related to vulnerability management activities pass through the same workflow as any other change.	Some changes related to vulnerability management activities have a custom workflow or are treated as standard changes.	Most changes related to vulnerability management activities follow a custom workflow or are treated as standard changes.	Changes related to vulnerability management activities along with success rates are tracked. Timing is also measured for different stages of the change or subtasks related to the change.	Metrics from vulnerability management change activities are used to modify requirements or streamline future change requests. At least some standard changes are automated.
	Patch Management	Patches are applied manually or scheduled by admins and end-users.	There is a standard schedule defined and technology is available for some divisions or departments or for some platforms to automate patch testing and deployment.	All departments are required to patch within a certain timeframe and technologies are available to assist with testing and applying patches for all approved platforms.	Patch management activities are tracked along with compliance with remediation timelines and the success rate.	Data from patch management activities, security incidents, and threat intelligence are used to right-size remediation timelines and identify process or technology changes.
	Configuration Management	Configuration requirements are not well-defined and changes are either applied manually or the automatic application of configuration is only available for a subset of platforms.	Configurations are defined for some admins or departments or for specific platforms.	Configurations are defined for all supported platforms and technologies are available to automate or validate configuration changes for all platforms.	Deviations from configuration requirements and associated service impacts are measured and tracked.	Data from the configuration process along with security incidents and threat intelligence are leveraged to strengthen or relax requirements as needed.

(SANS, 2021)

PATCH MANAGEMENT

Patch management is the process of acquiring, verifying, testing, and installing patches for products and systems. Patches help fix any security weakness that may have been present in the product. Apply security patches to vulnerable products helps reduces attack surface by closing potential opportunities for threat adversaries.

There are several challenges to developing a safe and efficient Patch Management program. These serve to complicate the process but can be overcome with proper protocol.

- **Acquire:** Patches may be included in automatic updates from vendors or manually retrieved from vendor/accredited sites.
- **Verify:** Verify the source and integrity of the patch. A digital signature is usually provided. Search relevant forums for more information on the patch.
- **Test:** Test the patch in a controlled environment and observe any anomalies or issues.
- **Install:** Dispatch the patch to affected hosts on the organisations network.

GENERAL RECOMMENDATIONS

- In large organisations where large numbers of assets exist, it is best to develop a phased approach to patch management. This involves setting up a test environment to assess the patch before rolling it out to a greater audience.
- Assess the risk of introducing third party patch management software to your environment. These tools can alleviate workload in the process. Some environments may be impossible to manage with aid. Introducing these tools can add extra attack surface by introducing their own potential vulnerabilities or allow adversaries to intercept their communication.
- Installing patches may affect the availability of systems. Organisations should look to balance their security with their needs. Installing a low severity vulnerability patch is not worth taking devices offline for a few hours.

CHALLENGES OF PATCH MANAGEMENT

Timing and Prioritization: In a perfect setting, patches should be rolled out immediately to reduce the window for exploitation to occur. A released patch notifies any attacker of the issue. However, most organisations have limited resources at their disposal. These patches also need to be tested before dispatch to reduce risk. This makes it necessary to assign priority to patches in most cases.

Configuration: There is a disparity across software and systems to how patches are applied. One application may be able to automatically update itself. Another may require the patch to be manually installed. This can lead to conflicts in patch management applications. This also includes any tampering the user may have performed which could affect patch dispatch.

Host architectures: This is normally not an issue where all hosts in an environment are managed centrally and are running identical technology. This is rare and number of problem areas arise from the following categorizations.

- **Unmanaged hosts:** These are not centrally managed.
- **Off-premises hosts:** Not protected by the organisation's outer security controls like firewalls
- **Non-standard hosts:** May not be possible to patch without a dedicated software update
- **Mobile hosts:** Run different operating systems. Normally require a different approach to patching
- **Virtual Hosts:** Requires patch maintenance for each image. Different patching protocols.
- **Firmware:** Differ to standard patching and require special privileges.

Software Inventory Management: This refers to maintaining an inventory of what software is installed on host devices. This is required to identify and acquire the appropriate patches. Not having a maintained software inventory can also lead to issues due caused by different versions.

Configuration changes: Installing a security patch may inadvertently modify the current security settings of a host. This can be avoided by testing the patch before dispatch. (Souppaya & Scarfone, 2013)

PATCH MANAGEMENT TECHNOLOGIES

Enterprise patch management and deployment software achieves its functionality under three possible categorizations: Agent-based, Agentless Scanning and Passive network monitoring

- **Agent-Based:** Requires an agent application to be run on the host to detect applications.
- **Agentless Scanning:** Network scans the environment and can detect applications on hosts if it has privileges.
- **Passive Network Monitoring:** Monitors network traffic for traffic regarding applications and their versions.

Comparison of Categorizations

Characteristic	Agent-Based	Agentless Scanning	Passive Network Monitoring
Admin privileges needed on hosts?	Yes	Yes	No
Supports unmanaged hosts?	No	No	Yes
Supports remote hosts?	Yes	No	No
Supports appliances?	No	No	Yes
Bandwidth needed for scanning?	Minimal	Moderate to excessive	None
Potential range of applications detected?	Comprehensive	Comprehensive	Only those that generate unencrypted network traffic

(Souppaya & Scarfone, 2013)

RESEARCH ON VULNERABILITIES

As documented by SANS Institute, “Vulnerabilities are gateways by which threats are manifested”.

Vulnerabilities occur due to:

Systems can initially come with both known bugs and exploits. They may also come with unknown vulnerabilities that have yet to be uncovered. The default configuration on systems may also be insecure.

Misconfigurations by engineers or administrators may occur.

VULNERABILITY TYPES

Vulnerability types can be categorized based on the technology that they exist. These include but are not limited to:

Software: Flaws or weaknesses in the code of a software application.

Firewall: Firewall vulnerabilities are the result of configuration error, absence of packet inspection or security patches. This can allow attackers to potentially bypass an organisations outer perimeter.

Network: These vulnerabilities originate from misconfigurations and weaknesses on ports, services, and intermediary network devices across a network.

Operating System: Errors or weaknesses in an operating systems code which can allow an attacker to take over, damage or persist on the host.

Web Server: These are categorized under misconfigurations or weaknesses in websites or a web applications code. Exploiting these potentially allows attackers to affect its confidentiality, integrity, and availability.

Database: Vulnerabilities that allow access, damage or theft to a database application and its assets. This could be through the likes of SQL injection though unvalidated variable in an application, weak authentication mechanisms on the database or weaknesses discovered in the database applications code.

CODE EXECUTION

This includes security flaws which allow arbitrary code execution either remotely or locally. Arbitrary code execution is the ability to run any code/commands on a device. This will essentially allow the attacker full control over a system at the same privilege level as the

target process. Attackers are not confined to this level if they can find a weakness that allows privilege escalation.

Mitigations

- Anti-virus solutions offer a layer of protection against files that have unwanted code execution.
- Firewalls and WAFs can prevent executables from reaching out to a command and control if it is a known malicious source.
- Review application code for any potential security issues like unvalidated input from the user.
- Arbitrary code execution normally requires initial access. Maintaining a good security posture will help prevent compromise in the first place.
- Ensure to employ user compromise prevention tactics with least privilege methodologies.

BUFFER OVERFLOW

This occurs when more data is added to a buffer than it holds. This will cause data to overflow and overwrite to adjacent storage. This can allow an attacker to write malicious code to this storage by packaging it within the payload that caused the buffer overflow. Buffer overflows are a dangerous type of attack as this code can be written at a kernel level and be almost impossible to remove. It is the leading software weakness due to several challenges in mitigating it and often requiring understanding for prevention.

Mitigations

- Choosing a strongly typed programming language that does not allow direct access to memory is highly recommended. Languages like C++ and assembly are notorious for buffer overflow vulnerabilities due to these characteristics. It is advised to have experience in terms of correct coding practices involving memory management before choosing to develop in these languages.
- Ensure applications have the latest patches especially if security orientated
- Audit the code in any application before it is allowed for use within the environment to ensure recommended buffer overflow mitigations are applied.
- Note any buffer overflow vulnerabilities that have been picked up though scans or penetration tests and mitigate by either patching or disabling its use if it is greater than accepted risk.

DENIAL OF SERVICE

A Denial-of-Service vulnerability affects the availability of affected hosts or networks. This will make it unavailable to intended users. This may be caused by more data being sent than systems can handle. It also includes exploits which can crash systems.

Mitigations

- Avail of anti DDOS services and providers.
- Increase your bandwidth can make a service more costly to DDoS. Larger providers in the cloud for example provide this.
- Ensure the latest patches and vulnerability mitigations are on application assets within the environment.

SQL INJECTION

SQL injection vulnerability allows an attacker to pass malicious SQL commands to compromise underlying SQL servers. This occurs through unvalidated user input which allows direct SQL command to be entered. This can lead to a full compromise of the contents of an SQL database as the attacker can enter a query they wish. This is a highly dangerous and common vulnerability.

- User input validation and sanitisation.
- Filters to block commands commonly used by attackers
- Ensuring current patches are installed for database infrastructure and its software.

DIRECTORY TRAVERSAL

These are HTTP exploits which can allow unauthorized access to files and folders on a web server. This can be a result of insufficient browser input validation or inadequate security mechanisms.

Mitigations

- User input validation and sanitisation.
- Filters to block commands commonly used by attackers
- Ensuring current patches are installed for the web server and its software.

CROSS SITE SCRIPTING

Cross site scripting vulnerability are web exploits where malicious JavaScript is injected into trusted web sites. These vulnerabilities normally exist due to lack of user input validation on the affected website. The consequences of XXS are vast and can include user compromise and information disclosure of affect assets.

Cross-site scripting can be categorized under

Persistent (Stored) Cross-site scripting: This type of XXS is where a successful payload is stored on the target servers. Depending on the characteristics of the vulnerability, the malicious payload may affect visitors or on an administrator. This is the most dangerous type of XXS due to amount and profile of the users it can affect.

Reflected Cross-site scripting: Reflected Cross-site scripting is where the un-sanitated payload is reflected off the web server onto an unexpecting user though social engineering. This type of attack is more targeted and takes additional steps to perform.

The target must click on the crafted link. This sends a request to the server, and it returns the page to user with the payload included.

DOM Based Cross-site scripting: DOM based Cross-site scripting can be defined by XXS attacks that occur within the DOM environment. This means that this type never reaches the web server and executes entirely in the browser.

Mitigations

- Ensure recommend security practice when coding application User input should be validated and sanitised from known malicious input.
- Only allow expected input. Compromise can be made even more improbable if anything not expected in a field is rejected. An example is an ID should only allow numbers.
- Audit any relevant applications introduced to the environment for appropriate coding practices.

CROSS SITE REQUEST FORGERY

In a successful CSRF attack, the victim is tricked into preforming an action they did not preform by an attacker. The following assumes user is currently authenticated. If the user is deceived into clicking a URL which will either send sensitive parameters in a GET request or a direct to a crafted site, any consequent request sent to a CSRF vulnerable web server will have no way to tell if the action was not performed by the user.

CSRF attacks require the following conditions:

- An action that would be lucrative for an attacker to preform like modifying the user's password.
- The user session is tracked only through a cookie. There is no anti-CSRF mechanisms in place.
- There are no undetermined parameters for the attacker.

Mitigations

- Having parameters that an attacker cannot predict. If your site preforms password resets for users, require their previous password to be entered.
- Attaching an anti-CSRF token to the user's session
- Validate any request that is sent to the server by the user. An example of this would be to track the user with an origin header.

PRIVILEGE ESCALATION

This type of vulnerability occurs when a user can exploit a security hole to either gain unauthorized entry to areas that should be unavailable, or functionality only intended for higher privilege levels. This can allow the attacker to make modifications to data, steal information stored in the application or run administrative functions.

Privilege escalation vulnerabilities are normally exploited in the early stages of an attack when initial access is gained. This could be registered user in your application or a compromised account to stage the attack from.

Mitigations

- Protecting user accounts through secure password policies and social engineering education.
- Ensure latest security patches are installed.
- Use good practice with security in mind when coding. User input should always have appropriate sanitisation.
- Design by least privilege and change any default or hardcoded configuration that may exist.

AUTHENTICATION BYPASS

An authentication bypass vulnerability allows an attacker to bypass any authentication mechanisms that are in place on a vulnerable application. Authentication may be bypassed in several scenarios including:

- Stealing valid Session IDs and cookies.
- Misconfigurations in coding may allow access to areas of the application.
- Weak authentications mechanisms can expose vulnerabilities to the attacker
- Applications may have authentication in place, but certain files or locations may be unprotected.

Mitigations

- Ensure latest patches for application are installed as soon as possible.
- Protect session IDs and tokens through encryption.
- All locations of the application should have adequate security protection.

- Validate user requests at the web server.

SECURITY MISCONFIGURATION

Security misconfigurations are created due to mistakes, misinterpreted requirements and systems that are left unpatched or without recommended settings. In large environments, these are hard to avoid and discover but leave an opening for attackers to exploit.

Mitigations

- Developing a hardening process that is repeatable across the environment makes it more secure and easier to spot anomalies.
- Avoid overburdening your application or environment with unnecessary requirements.
- Regular reviews and updates of current configurations.

COMMON WEAKNESS ENUMERATION (CWE)

Common Weakness Enumeration (CWE) is a list of generalized weaknesses of software/hardware that is actively maintained by the CWE community. The goal of CWE is education for developers to these common weaknesses and how to prevent from being present in their product.

The goal of CWE is to:

Categorize software and hardware flaws.

- Provide a resource for checking for product weaknesses.
- Document tools which target the weaknesses
- Establish standards for vulnerability identification, mitigation and how to prevent them.
- Patch vulnerabilities in the development stage.

(CWE, 2021)

CWE also provides a comprehensive list of the top 25 most dangerous software weaknesses. The ain is to bring awareness and priority to common and dangerous weaknesses.

The Common Weakness Enumeration (CWE) Top 25 (Chaudry, et al., 2021)

Place	ID	Name	Score	Position Change from 2020
1	CWE-787	Out-of-bounds Write	65.93	+1
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1

3	CWE-125	Out-of-bounds Read	24.9	+1
4	CWE-20	Improper Input Validation	20.47	-1
5	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
6	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
7	CWE-416	Use After Free	16.83	+1
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
9	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5
11	CWE-306	Missing Authentication for Critical Function	7.93	+13
12	CWE-190	Integer Overflow or Wraparound	7.12	-1
13	CWE-502	Deserialization of Untrusted Data	6.71	+8
14	CWE-287	Improper Authentication	6.58	0
15	CWE-476	NULL Pointer Dereference	6.54	-2
16	CWE-798	Use of Hard-coded Credentials	6.27	+4
17	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
18	CWE-862	Missing Authorization	5.47	+7
19	CWE-276	Incorrect Default Permissions	5.09	+22
20	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	4.74	-13
21	CWE-522	Insufficiently Protected Credentials	4.21	-3
22	CWE-732	Incorrect Permission Assignment for Critical Resource	4.2	-6
23	CWE-611	Improper Restriction of XML External Entity Reference	4.02	-4
24	CWE-918	Server-Side Request Forgery (SSRF)	3.78	+3
25	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3.58	+6

(Chaudry, et al., 2021)

To create the 2021 list, the CWE Team leveraged Common Vulnerabilities and Exposures (CVE) data found within the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD), as well as the Common Vulnerability Scoring System (CVSS)

scores associated with each CVE record. A formula was applied to the data to score each weakness based on prevalence and severity.

It is interesting to note that that the most common weaknesses can be categorized under buffer overflow and improper input validation/sanitisation.

CATEGORIZATION OF A VULNERABILITY

Common vulnerabilities and exposures (CVE)

CVE records are unique identifiers for publicly known Cyber Security vulnerabilities. Every CVE record contains:

- **CVE ID Number:** This is the number portion of the CVE record.
- **Description:** Written by CVE Numbering Authorities (CNAs) or individuals requesting a CVE ID. Descriptions include relevant details to help find the CVE record of specific vulnerability. They will also have details regarding the affected vendors, products, versions and code components.
- **References:** References are used to identify the source and includes an identifier for searching.

(Mitre, 2021)

Common configuration enumeration (CCE)

CCE components are part of a list of unique identifiers used to identify configuration issues related to systems related to Cyber Security

Each component in the CCE List contains:

- **CCE ID:** Consists of the identifier: CCE, the number of the identifier and a check digit produced from Luhn Check Digit Algorithm.
- **Description:** Gives an easy-to-understand description to identify the configuration issue. This will most likely be a statement on if a configuration should or should not be made.
- **Parameters:** Gives a list of parameters which will be needed to implement a specific configuration. Associates' parameters like a software configuration to a parameter like Off, Manual, On.
- **Associated technical mechanisms:** May contain alternate ways to achieve implementing compliance with the CCE
- **References:** Documents reference material for configuration guidelines.

(MITRE, 2013)

Common vulnerability scoring system (CVSS)

The Common Vulnerability Scoring System (CVSS) provides an open-source framework to categorise vulnerabilities based on characteristics and potential risk involved in the event its exploited. Based on these attributes, a severity rating is calculated ranging from low to critical.

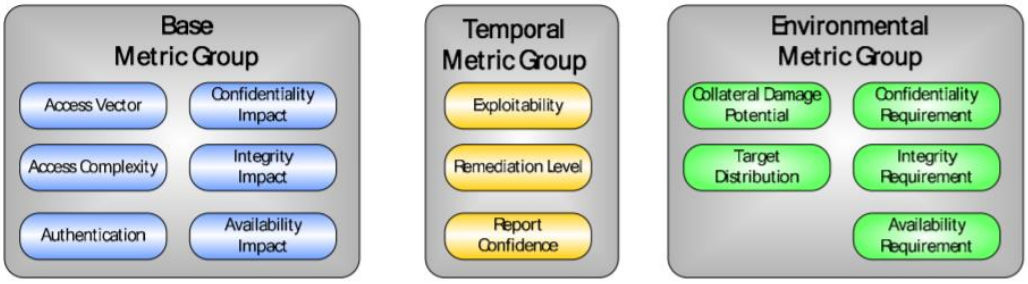


Figure 1: CVSS Metric Groups

(Mell, Scarfone, & Romanosky, A Complete Guide to the Common Vulnerability Scoring System Version 2.0, 2007)

The score of a vulnerability can be calculated through the following formulas.

CVSS Base equations

Exploitability Sub Score	$8.22 \times AttackVector \times AttackComplexity \times PrivilegeRequired \times UserInteraction$
$\boldsymbol{ISC_{Base}}$	$1 - [(1 - ImpactConf) \times (1 - ImpactInteg) \times (1 - ImpactAvail)]$
Impact sub score	$Scope\ Unchanged\ 6.42 \times ISC_{Base}$ $Scope\ Changed\ 7.52 \times [ISC_{Base} - 0.029] - 3.25 \times [ISC_{Base} - 0.02]15$
Base Score	$If\ (Impact\ sub\ score \leq 0)$ $0\ else,$ $Scope\ Unchanged_4$ $Roundup(Minimum[(Impact + Exploitability), 10])$ $Scope\ Changed$ $Roundup(Minimum[1.08 \times (Impact + Exploitability), 10])$

*"Roundup" is defined as the smallest number, specified to one decimal place, that is equal to or higher than its input.

**"Minimum" Insert the smaller of the two arguments

(NIST, 2021)

CVSS Temporal Equation

Temporal Base Score	$Roundup(BaseScore \times ExploitCodeMaturity \times RemediationLevel \times ReportConfidence)$

(NIST, 2021)

CVSS Environmental Equations

Modified Exploitability Sub Score	$8.22 \times M.AttackVector \times M.AttackComplexity \times M.PrivilegeRequired \times M.UserInteraction$
$ISC_{Modified}$	$Minimum\,[[1 - (1 - M.IConf \times CR) \times (1 - M.IInteg \times IR) \times (1 - M.IAvail \times AR)], 0.915]$
Modified Impact Sub Score	$If\ Modified\ Scope\ is\ Changed\ 7.52 \times [ISC_{Modified} - 0.029] - 3.25 \times [ISC_{Modified} \times 0.9731 - 0.02]13$ $If\ Modified\ Scope\ is\ Unchanged\ 6.42 \times [ISC_{Modified}]$
Environmental Score	$If\ (Modified\ Impact\ Sub\ score <= 0)\quad 0\ else,$ $If\ Modified\ Scope\ is\ Unchanged\quad Round\ up(Round\ up\ (Minimum\ [\ (M.Impact + M.Exploitability)\ ,10])\ \times Exploit\ Code\ Maturity \times Remediation\ Level \times Report\ Confidence)$ $If\ Modified\ Scope\ is\ Changed\quad Round\ up(Round\ up\ (Minimum\ [1.08 \times (M.Impact + M.Exploitability)\ ,10])\ \times Exploit\ Code\ Maturity \times Remediation\ Level \times Report\ Confidence)$

(NIST, 2021)

*Modified Metrics have the same base metric value as corresponding base metrics. Example: AttackVector = M.AttackVector

*"Roundup" is defined as the smallest number, specified to one decimal place, that is equal to or higher than its input.

**"Minimum" Insert the smaller of the two arguments

CVSS Metric Values

Metric	Metric Value	Numerical Value
Attack Vector / Modified Attack Vector	Network	0.85
	Adjacent	0.62
	Local	0.55
	Physical	0.2
Attack Complexity / Modified Attack Complexity	Low	0.77
	High	0.44
Privileges Required / Modified Privileges Required	None	0.85
	Low	0.62 (or 0.68 if Scope / Modified Scope is Changed)
	High	0.27 (or 0.5 if Scope / Modified Scope is Changed)
User Interaction / Modified User Interaction	None	0.85
	Required	0.62
Confidentiality / Integrity / Availability / Modified Confidentiality / Modified Integrity / Modified Availability	High	0.56
	Low	0.22
	None	0
Exploit Code Maturity	Not Defined	1
	High	1

	Functional	0.97
	Proof of Concept	0.94
	Unproven	0.91
Remediation Level	Not Defined	1
	Unavailable	1
	Workaround	0.97
	Temporary Fix	0.96
	Official Fix	0.95
Report Confidence	Not Defined	1
	Confirmed	1
	Reasonable	0.96
	Unknown	0.92
Confidentiality Requirement / Integrity Requirement / Availability Requirement	Not Defined	1
	High	1.5
	Medium	1
	Low	0.5

(FIRST, 2021)

Qualitative Severity Rating Scale

Using the described formulas,

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

(FIRST, 2021)

Exploitability Metrics

Attack Vector (AV): This refers to the context in which the vulnerability can be exploited.

- **Network:** This means that the vulnerability is exploited on Layer 3 of the OSI model. This category is exploitable across OSI layer 3 boundaries. As a result, the vulnerability is remotely exploitable.
- **Adjacent Network:** This means that the vulnerability is exploited on Layer 3 of the OSI model. This category is not exploitable across OSI layer 3 boundaries. Limited to the network it exists on.

- **Local:** Not exploitable across layer 3 of the OSI model. Vulnerability exists through read/write/access on a target machine. May need to be logged in locally or require a file to be executed.
- **Physical:** This type requires physical access to a vulnerable component on a device to be exploited.

Attack Complexity (AC):

- **Low:** Does not require specific conditions or circumstances to exploit. The vulnerability is generally successful when attempted.
- **High:** Successful attempts at exploiting these vulnerabilities can require specific conditions which may be out of the attacker's control. Time and effort must be taken by the attacker before its successful exploit can be expected.

Privileges Required (PR):

- **None:** Does not require access to settings or files to exploit.
- **Low:** Requires basic privileges associated with user roles.
- **High:** Requires significant administrative privileges to exploit.

User Interaction (UI):

- **None:** Does not require any user interaction to be exploited.
- **Required:** Requires a user to perform some actions before the exploit can be successful.

Scope (S):

- **Unchanged:** Can only affect resources managed by the same authority.
- **Changed:** Can affect resources outside the same authority of the vulnerably component.

Impact Metrics

Confidentiality Impact (C):

- **None:** No loss of confidentiality within the affect component.
- **Low:** There is a loss of confidentiality. Attacker will only be only able to get specific information. The disclosed information is not of high value.
- **High:** All confidential information within the component is compromised by the attacker. Disclosed information is of high value.

Integrity Impact (I):

- **None:** No loss of integrity within the affect component.

- **Low:** Modification is possible but there is no free control over what data is modified. Affected data is not of high value.
- **High:** Attacker has free reign to modify files protected by the component. Affected data is of high value.

Availability Impact (A):

- **None:** No loss of availability within the affect component.
- **Low:** This will cause performance issues within the affected component. Repeated exploits do not result in a complete Denial of Service. Impact is not of serious consequence.
- **High:** Total loss in availability due to the exploit. This can also include when an attacker can deny some availability, but the consequences are high.

(NIST, 2021)

Common Attack Pattern Enumeration and Classification (CAPEC)

Correlates a comprehensive list of common attack patterns and techniques which are influenced by known vulnerabilities and weaknesses in Information technology.

These patterns and techniques include intelligence on how these attacks are conducted and designed. Advice is included on how to mitigate their effectiveness.

CAPECs are identified by mechanisms of attack, their domain, and an identification number. This helps cyber security personnel and developer pay particular attention to common techniques and their prevention.

(MITRE, 2019)

National Vulnerability Database (NVD)

The NVD is the official United States governments library of standard based vulnerabilities. To compile this extensive list, analysts are tasked with investigating CVEs that have been added to the CVE dictionary. The analysis enumerates CVSS, CWE, CPE as well as other important data like descriptions, advisories, and patches. The NVD record will be updated accordingly if any pertinent information surfaces.

(NIST, 2021)

MITRE ATT&CK

MITRE ATT&CK is a library of tactics and techniques used by threat actors based on real world observations. This is used as a framework to develop specific threat models for organisations. The model provides classification of specific actions preformed in attack

methodologies from an offensive and defensive standpoint. Defensive countermeasures are also provided against the technique.

The behavioural model used in the framework contains three fundamental elements:

- **Tactics:** Short term goals in attack methodologies
- **Techniques:** The means in which the attack goals are commonly accomplished
- **Documentation:** Describes how the techniques were used by attackers historically with other relevant data.

Mitre ATT&CK matrices contained a comprehensive collection of attack techniques used by threat actors to accomplish specific goals. The techniques are all categorized under tactics which are the objectives.

- **Reconnaissance:** The means in which intel is gathered about the target
- **Resource Development:** Acquiring the means to support the attack
- **Initial Access:** Strategies to gain a foothold in the network
- **Execution:** Methods to execute malicious code
- **Persistence:** Keeping access in an environment once gained
- **Privilege Escalation:** Gaining the privileges needed for further methods
- **Defence Evasion:** Tactics to avoid detection in the environment
- **Credential Access:** Techniques to steal account credentials
- **Discovery:** Tactics to gain information of the target infrastructure and networks
- **Lateral Movement:** Methods used to control remote systems
- **Collection:** Techniques to gather data to achieve a goal
- **Command and Control:** Methods to communicate with compromised systems
- **Exfiltration:** Tactics to steal data from target systems
- **Impact:** Methods to manipulate, destroy and deny service on systems

Enterprise Matrix

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command And Control
10 items	31 items	56 items	28 items	59 items	20 items	19 items	17 items	13 items	9 items	21 items
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Automated Exfiltration	Commonly Used Port
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Data Compressed	Communication Through Removable Media
Hardware Additions	Command-Line Interface	AppCert DLLs	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Distributed Component Object Model	Clipboard Data	Data Encrypted	Connection Proxy
Replication Through Removable Media	Control Panel Items	AppInit DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	File and Directory Discovery	Exploitation of Remote Services	Data from Information Repositories	Data Transfer Size Limits	Custom Command and Control Protocol
Spearphishing Attachment	Dynamic Data Exchange	Application Shimming	Application Shimming	Clear Command History	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Local System	Exfiltration Over Alternative Protocol	Custom Cryptographic Protocol
Spearphishing Link	Execution through API	Authentication Package	Bypass User Account Control	CMSTP	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Network Shared Drive	Exfiltration Over Command and Control Channel	Data Encoding
Spearphishing via Service	Execution through Module Load	BITS Jobs	DLL Search Order Hijacking	Code Signing	Exploitation for Credential Access	Password Policy Discovery	Pass the Ticket	Data from Removable Media	Exfiltration Over Other Network Medium	Data Obfuscation
Supply Chain Compromise	Exploitation for Client Execution	Bootkit	Dylib Hijacking	Component Firmware	Forced Authentication	Peripheral Device Discovery	Remote Desktop Protocol	Data Staged	Exfiltration Over Physical Medium	Domain Fronting
Trusted Relationship	Graphical User Interface	Browser Extensions	Exploitation for Privilege Escalation	Component Object Model Hijacking	Hooking	Permission Groups Discovery	Remote File Copy	Email Collection	Scheduled Transfer	Fallback Channels
Valid Accounts	InstallUtil	Change Default File Association	Extra Window Memory Injection	Control Panel Items	Input Capture	Process Discovery	Remote Services	Input Capture		Multi-hop Proxy
	Launchctl	Component Firmware	File System Permissions Weakness	DCShadow	Input Prompt	Query Registry	Replication Through Removable Media	Man in the Browser		Multi-Stage Channels
	Local Job Scheduling	Component Object Model Hijacking	Hooking	Deobfuscate/Decode Files or Information	Kerberoasting	Remote System Discovery	Shared Webroot	Screen Capture		Multiband Communication
	LSASS Driver	Create Account	Image File Execution Options Injection	Disabling Security Tools	Keychain	Security Software Discovery	SSH Hijacking	Video Capture		Multilayer Encryption
	Mshta	DLL Search Order Hijacking	Launch Daemon	DLL Search Order Hijacking	LLMNR/NBT-NS Poisoning	System Information Discovery	Taint Shared Content			Port Knocking
	PowerShell	Dylib Hijacking	New Service	DLL Side-Loading	Network Sniffing	System Network Configuration Discovery	Third-party Software			Remote Access Tools
	Regsvcs/Regasm	External Remote Services	Path Interception	Exploitation for Defense Evasion	Password Filter DLL	System Network Connections Discovery	Windows Admin Shares			Remote File Copy
	Regsvr32	File System Permissions Weakness	Plist Modification	Extra Window Memory Injection	Private Keys	System Owner/User Discovery	Windows Remote Management			Standard Application Layer Protocol
	Rundll32	Hidden Files and Directories	Port Monitors	File Deletion	Replication Through Removable Media	System Service Discovery				Standard Cryptographic Protocol
	Scheduled Task	Hooking	Process Injection	File System Logical Offsets	Securityd Memory					Standard Non-Application Layer Protocol
	Scripting	Hypervisor	Scheduled Task	Gatekeeper Bypass	Two-Factor Authentication Interception					Uncommonly Used Port
	Service Execution	Image File Execution Options Injection	Service Registry Permissions Weakness	Hidden Files and Directories						Web Service
	Signed Binary Proxy Execution	Kernel Modules and Extensions	Setuid and Setgid	Hidden Users						
	Signed Script Proxy Execution	Launch Agent		Hidden Window						
	Source			HISTCONTROL						
	Space after Filename			Image File Execution Options Injection						

Bug Bounty Programs

Many organisations offer monetary reward to the public for reporting vulnerabilities and weaknesses in their externally facing assets. This allows developers to potentially patch vulnerabilities before public disclosure.

The practice is becoming more popular as opposed to condemning white hat hackers. It can be more cost efficient for organisations than bug testing themselves.

Prevention of vulnerabilities and their potential exploitation can occur in the following ways:

- Creating security guidelines following a security standard like ISO 27001
- Ensuring available security patches are installed for relevant systems when vulnerabilities are discovered.
- Regular vulnerability scanning of the environment
- Following and adhering to various Cyber security advisories
- Protecting a networks perimeter through firewalls and router ACLs
- Implementing IDS and AV technology on company endpoints.

CONSEQUENCES OF EXPLOITED VULNERABILITES

Confidentiality: Vulnerabilities can lead to the compromise of confidential data. This would include something like a SQL injection attack which steals user data.
Integrity: Vulnerabilities that target the accuracy or completeness of data. An example would be an exploit which allows an individual to spoof a user.
Availability: Vulnerabilities that affect the availability of systems. The most common vulnerability of this area is that which causes Denial of Service.

MALWARE

Ransomware: This is a type of malware which is designed to blackmail a victim into preforming certain demands by encrypting files on a system. This renders them unusable for the victim until a decryption is performed with the associated decryption key. This will be retrieved either when the attacker's demands are met, or a decryption can be performed through already obtained keys usually from a third party.

Rootkit: Allows remote access to the infected device and is often obfuscated as a backdoor for further actions. Very difficult to remove as AV technology are often unreliable at detecting them and can reside at the kernel level.

RAT (Remote access trojan):

Trojan: Malware that impersonates legitimate software to avoid detection and trick the user to introduce it to a system. Can perform a wide array of actions depending on the intent of the attacker.

Worm: A worm is a type of malware that can propagate over a network, spreading to other hosts. Extremely infectious and is difficult to stop at the root before other hosts are infected.

Spyware: Spyware aims to gather information about the user through illegitimate means. The user's behaviour is tracked through keylogging, web history and cookie theft. Motives can range from monetary gain to espionage.

Adware: Malware designed to inject unwanted advertisements frequently while using a browser. Adware is similar to trojans where it will commonly disguise itself as a legitimate program or come bundled with another software download. Its goal is to generate revenue for the creator through advertisements or gather user information to sell. It can be dangerous as they are likely not confined by the same rules other companies must abide by when gathering personal information. The advertisements also displayed can be dangerous and lead to further compromises.

DATA EXFILLTRATION

Data exfiltration can be defined as the unauthorized exportation of data from a device. The target of this type of attack is financial gain, reputation damage or sabotage. These types of attacks are usually targeted to gain access to specific data.

On obtaining a foothold within an environment, threat actors will normally achieve persistence to be able to stealthily monitor an environment for the next destination to their target. Upon gaining access to this target, exfiltration can begin by uploading data to the internet through the likes of cloud storage applications, emails and even DNS.

Mitigations

- Detect any suspicious connections malware may be using to communicate with a command-and-control server.
- Monitoring DNS records for requests to cloud. Large data transfers can be detected through the total number of bytes sent.
- Compromised accounts can be used to exfiltrate via email.
- Obfuscated techniques like TXT based DNS requests to send data to modified DNS servers exist.
- Follow steps for removing persistent threats.

COMPRIMISED ACCOUNTS

A compromised account is one which has been accessed by an unauthorized actor and remediation steps to secure it have not taken place. This can occur due to vulnerabilities that affect the confidentiality of credentials. It may also be due to a vulnerability to either assume the user's profile or bypass authentication controls.

Gaining access to user's accounts can be lucrative to attackers as an initial access point into an environment, gaining information, compromising linked components to the account and using it as a staging point for further attacks like phishing.

Mitigations

- Enabling multi-factor authentication can make it incredibly difficult for attacks to compromise an account
- Implement lockouts based on conditions like too many failed authentication attempts from an IP address in a short period of time.
- Implementing minimum password strength policies
- Enforce frequent password change requirements on users

PERSISTENCE

Persistence enables bad actors who have compromised an environment to keep a foothold for later use or if initial attack vectors have been neutralized. This ensures they have long term access and are never truly removed. This is normally done through malware which is obfuscated in legitimate folders, registry or scheduled tasks. Through these methods, persistent threats can re-establish themselves on start-up.

Mitigations

- The best way to prevent persistent threats is to not get initially compromised in the first place.
- Anti-virus solutions are relatively ineffective at detecting these. By design, these files are designed to be stealthy through obfuscation.
- It typically requires human analysis to uncover what does not look right to be guaranteed to eliminate persistent threats.
- The best mitigation is a developed security posture through techniques like vulnerability management.

LATERAL MOVEMENT

This is a technique which allows attackers to move deeper into a network after initial access. Lateral movement normal begins on the malicious actor obtaining valid credentials. This is followed by reconnaissance on the local network to determine their goals.

Ultimately the attacker will use techniques like “pass the hash”, “pass the ticket” and keylogging to compromise other accounts as a form of persistence. Their goals also include escalating their privileges in the environment to bypass security controls.

Mitigations

- Detecting lateral movement can be incredibly difficult due to the activities of the attacker looking relatively normal.
- Like persistence, software solutions are relatively ineffective at detecting it due to low footprint on the environment.
- Eliminating laterally moving threats normally requires human intervention through threat hunting.
- Ensuring the latest patches are installed can help prevent these from establishing a foothold like persistence.

DATA AND HARDWARE DESTRUCTION

Adversaries may opt to destroy data that could be stored on systems across the organisations network. The destroyed data would likely be also made unrecoverable via forensic means. This could be done by instead of deleted the data, it is instead overwritten with random values. The decided attack vector would be used to cause both monetary and reputational damage to the organisation. This destructive attack has even extended to targeting hardware in the past. An example would be the famous Stuxnet attack. This was achieved by modifying controls to cause parts to damage themselves through overwork.

Mitigations

- Due to threats like data destruction, having offsite data backups is highly recommended. This can provide data restoration in the event of disaster.
- Avail of AV solutions to detect known malicious executables that that exist in these attack chains.
- Continuous Cyber security monitoring can help limit the attack if the adversary hasn’t managed to laterally move yet. The attacker may also be detected if they managed to spread to other assets.
- Employing strong cyber security practices to prevent initial compromise.

PRIVILEGE ESCALATION

This is comprised of the techniques that adversaries use once gaining initial access to an environment to gain higher permissions. This is typically achieved by testing other vulnerabilities and weaknesses once inside. The techniques normally come with persistence because OS elements that let an attacker persist are executed in an elevated setting.

Mitigations

- Protecting user accounts through secure password policies and social engineering education.
- Ensure latest security patches are installed.
- Use good practice with security in mind when coding. User input should always have appropriate sanitisation.
- Design by least privilege and change any default or hardcoded configuration that may exist.

COMMAND AND CONTROL

The adversary may try to communicate with the infected asset through Command and Control. This is accomplished through a Command-and-Control Server to upload malicious commands to the host. Cloud based providers are normally used to blend in with legitimate network traffic.

Mitigations

- AV solutions can pick up known malicious files that could be used for command and control. Some also benefit from being able to pick up on suspicious connections made by processes.
- Ensure suitable account security is implemented and that users are aware of social engineering techniques.
- Ensure Firewalls and IPS systems are configured to block known malicious connections once detected.

PROJECT BRIEF WITH BRAINSTORM

Description. A tool that captures the attributes of validated vulnerabilities by a penetration tester and manages the process and timeline of each vulnerability within an organisation. Technologies.

(1) Windows or Linux (2) C#, C++, Python (3) GitHub
Will likely develop on Linux due to Kali Linux being the premier distribution used in Penetration testing. I would prefer to use Python for this project due to familiarity with the language and its text parsing features. Most tools or alternatives are available on Windows. Would like to implement on Windows to display vulnerability management interfaces.

(2) Provide a management interface to facilitate all aspects of vulnerability management for an organisation.
Interface to display discovered vulnerabilities that have not yet been remediated. These will be generated through scans preformed through the application and fed to here. The vulnerabilities would be categorized based on severity.

Interface for evaluating these vulnerabilities. This could be an area for remediation steps to be entered by a user. Once completed this will be set to "in progress" in the interface. Once remediation steps have been finished, the status can be set to completed. This would prompt the system to attempt to exploit the vulnerability again. The result of this would either close the issue or reopen it if the remediation was unsuccessful.

Could offer the option to generate remediation steps if applicable or realistic.

Allow users to generate penetration testing reports from the application. This could export the entire list of found vulnerabilities or may add a GUI which allows a custom report to be created.

Add an interface to display items relating to the organisation's security posture

(3) Data Reading: The capture and management of vulnerability data through its lifecycle. Display: A GUI that allows users to store all attributes of vulnerabilities and provide a dashboard displaying the current security status of an organisation.
The application will need to gather information from common penetration testing tools. This can be done through APIs that the tools may offer (probably the better approach especially for compatibility) or by pulling information from the tools through CLI (Most Kali Linux tools offer this).

This data will then need to be parsed for useful information and passed to the application for display and appropriately formatted in the display.

(4) Management: System contains management software to allow for multiple users. Extraction: Relevant and useful information is extracted from the raw data and shown in a user-friendly manner.
This means the application will need to support user authentication and creation. This can be added with GUI components. Will require storage of user credentials, current vulnerability items and security posture. Could store these on an SQL database

Python Requests, Beautiful Soup, Wget – Could be useful to allow the application to make HTTP requests to gather vulnerability information.

Create a script which queries the registry on host machines in environment on start-up. Wipe previous applications for this host and store current list of these on a database. Use Python requests to receive recent CVEs from NVD on applications. Display CVEs with affected hosts if

there is a close match for the affected application in the environment on a dashboard. Could prompt user to auto create a task in application.

TOOLS

Tool	*Operating System*	*Cost*	*Description*
Nmap	*Linux/Windows/Mac OS X*	*Freeware*	*Going to be a key part of this project. Nmap is used in the initial stages of penetration testing for reconnaissance and to determine what infrastructure the client has that can be potentially exploited. Will be key if I want to implement automated scanning with minimal needed input from the user.*
Zap	*Linux/Windows/Mac OS X*	*Freeware*	*Web proxy – Used for detecting web application vulnerabilities. Nmap can potentially detect hosted websites through reverse DNS resolution on any address it scans.*
Burp Suite	*Linux/Windows/Mac OS X*	*Freeware*	*Web Proxy*
Wireshark	*Linux/Windows/Mac OS X*	*Freeware*	*More for observing and confirming vulnerabilities in real time and discovering problem areas. Worth being aware of it but will likely not be used. Location matters in a network when using Wireshark ie you will only see traffic which travels through your switchport if connected to one.*
TCPDump	*Linux*	*Freeware*	*Provides similar functionality to Wireshark by allowing the analysis of captured traffic*
Metasploit Framework	*Linux/Windows*	*Freeware and Commercial*	*Can scan systems to check if they are vulnerable to RCE exploits*
OpenVAS	*Linux/Windows*	*Freeware*	*Vulnerability scanner tool which can scan for exploits in authenticated/unauthenticated situations*
Qualys	*Linux/Windows/Mac OS X*	*Commercial*	*Vulnerability Management Platform used to*
Aircrack-ng	*Linux/Windows/Mac OS X*	*Freeware*	*Wi-Fi security assessment tool, this tool can perform both attacks against Wi-Fi and Wi-Fi cracking.*
John the Ripper	*Linux/Windows/Mac OS X*	*Freeware*	*Could try to implement a brute force attack on any SSH, RDP, FTP etc ports found. Catches weak set passwords.*

Sqlmap	*Designed for Linux. Can run on Windows/OS X*	*Freeware*	*Can be used to validate SQL injection vulnerabilities in applications that use SQL databases in their stack.*
Sqldict	*Linux*	*Freeware*	*Dictionary attack tool for SQL Server*
Nessus	*Linux/Windows/Mac OS X*	*Freeware*	*Vulnerability scanning tool*
STAT	*Windows*	*Commercial*	*Suite of tools:* *STAT Scanner: Vulnerability scanner. Can integrate with Nessus and ISS. Doesn't use client/server model. STAT must exist on the host that is to be scanned.* *Stat Scanner console can correlate results from several hosts.* *Reputed to provide a lot of information on the vulnerability. Can remediate certain vulnerabilities itself.*
TARA	*Linux/Windows*	*Commercial*	*Vulnerability Scanner which highlights found vulnerabilities based on their risk.*
Cybercop	*Linux/Windows*	*Commercial*	*Vulnerability scanning, firewall scanning, OS detection, DNS server tests. The ability to add custom scripts to the application is interesting.*

SIMILAR TECHNOLOGY

Dradis	*Freeware*	*This is an open-source vulnerability management tool which is incredibly similar to what is requested in the brief if not incorporating all of the requirements. Since this is open source, the source code can be viewed directly*
Magic Tree	*Freeware*	*A vulnerability management tool used for consolidating penetration testing data. Data is imported manually via XML and stored in a tree like structure*
Faraday IDE	*Commercial*	*Encompasses the brief by including all wanted features. Unfortunately, is a pay to use tool. Features automated vulnerability response which could be interesting to investigate. Merges identical issues produced by different tools*
Serpico	*Freeware*	*An open-source vulnerability management tool used to alleviate report writing associated with Penetration testing. Data is imported manually via XML. Features Metasploit integration*
Qualys	*Commercial*	*Delivers a Vulnerability management platform. Delivered as SaaS through the cloud. Continuously scans the*

		environment based on pre-set and custom policies. Automatically applies necessary patches to affected hosts. Several tools are provided to find and mitigate vulnerabilities.
Tenable.io	*Commercial*	*Hosted in the cloud and runs on Nessus technology, offers the industry's most complete vulnerability coverage as well as the ability to anticipate which security issues should be addressed first.*
Rapid7 InsightVM	*Commercial*	*Rapid7's Insight cloud, which released in 2015, combines Nexpose's vulnerability research, Metasploit's exploit knowledge, global attacker behaviour, internet-wide scanning data, exposure analytics, and real-time reporting.*

PROGRAMMING LANGUAGES

PYTHON

Python is a high-level Object-Orientated programming language that serves as a general-purpose toolkit that focuses on readability through clear and logical mechanisms. It was first released in 1991 and has seen numerous new features though updates since. Python is consistently one of the most popular languages used in the world. As of August, it is the most popular programming language according to the PYPL index.

Python is a multi-paradigm language that is designed to be highly extensible with a wide range of libraries. Object orientated and structured programming are fully support. The language utilizes dynamic typing where types are verified at runtime. Security is enforced even though dynamic by causing an error at runtime instead of attempting to compute the code.

Advantages

- Python is an easy-to-understand high level language. It is comparable to English in its syntax
- Strong security practices with garbage collection, strong typing and does not allow the use of pointers.
- Python is suitable for virtually any task with its vast array of libraries
- Python is open source which allows developers to release modifications of the language.
- Runs on different platforms without requiring changes to source code

Disadvantages

- Slow due to code being interpreted line by line
- Can require a large amount of memory compared to other languages.
- Because Python can throw runtime errors, it requires a large amount of testing before releasing applications.
- Support for SQL databases is considered lacklustre compared to other languages

JAVA

Java is a high level, class-based and Object-orientated programming language developed by Sun Microsystems. Java source files are compiled into bytecode which is then passed to and executed by a Java Interpreter. This can be done broadly as Java Interpreters and runtime environments exist on most systems.

Java is general purpose and is well suited for application development on the world wide web. It is the primary development platform on Android and has a "write once run anywhere" philosophy where an application could be developed for Windows but also will run on other platforms like Linux. This is achieved by compiling source code to Java bytecode instead of machine code.

Advantages

- Java's syntax is easy to understand
- Java fully supports Object-orientated programming
- While it has had a previous reputation for insecurities, it has secure programming language characteristics. These include garbage collection, no direct access to memory and class security.
- Applications are automatically compatible with other platforms that have the Java runtime environment present.
- Multi-threading is supported allowing more efficient use of the CPU.
- General purpose language that has support to complete virtually any task.

Disadvantages

- Has a high memory consumption
- Relatively slow language compared to the likes of C++
- Java code can be highly verbose

C++

C++ is a general programming language released as an expansion of the C language. Support is provided for a wide range of features including Object-Orientated and functional programming. The intention of C++ was to provide a language that has fast speed and low memory usage for systems with limited resources. This is achieved through source code being converted to machine code in one go and low-level memory manipulation.

C++ boast many powerful features but also comes with the caveat of complexity and obscurity. These features can come with security and performance issues like memory management.

Advantages

- C++ is relatively portable and there is few issues porting applications to other platforms
- Allows low-level manipulation of data
- Total control over memory management
- Efficient memory and execution times

Disadvantages

- Pointers can use a large amount of memory
- No use of built-in threads
- Security issues can be caused by friend functions, global variables, and pointers
- No garbage collection can lead to memory mismanagement
- Compile times can be long due to source code being assembled at once.

PHP

PHP is a general-purpose scripting language that was developed to accommodate web development. Source code is typically executed on a web server by a PHP interpreter. PHP comes with the advantage that it can embed HTML which results in interactive web pages.

PHP is primarily a language used in web development, being used in about 80% of technologies stacks on applications on the internet. It has seen some use in other areas and can be run directly from the command line.

Advantages

- PHP can run on any platform like Windows and Linux
- PHP has straightforward syntax and is easy to understand
- Plenty of library support to accomplish tasks
- Forms part of the LAMP (Linux, Apache, MySQL, PHP) stack which is free and easily accessible to everyone.

Disadvantages

- Weak typing leaves it vulnerable to malicious input if not factored.
- Error handling is considered poor.
- Losing popularity to choices with more support.

JAVASCRIPT

JavaScript is a high-level, multi paradigm language that features just in time compilation and dynamic typing. It along with HTML and CSS forms the core of the World Wide Web. JavaScript's function here is to provide dynamic websites behaviour in the browser. 97 percent of websites use JavaScript for this purpose.

Although JavaScript was primarily a client-side language and required support for server-side operations, recent support has made it an increasingly popular choice that can perform server-side operations with frameworks like Node.js

Advantages

- JavaScript's syntax bears similarities to Java and is relatively easy to pick up for those with Java experience.
- JavaScript is gaining popularity over PHP for backend due to frameworks like Node.js. It is now possible and very viable to write Web application entirely using JavaScript.
- JavaScript has stronger security characteristics over PHP.
- Due to being primarily executed in the browser, it reduces server load.

Disadvantages

- While JavaScript is generally stronger than PHP in security, it still suffers from a weak type system.
- JavaScript is associated with client-side security issues. Weaknesses and vulnerabilities can lead to malicious activity
- JavaScript can behave differently depending on the browser it is used in. This means testing is required for all notable browser types.

RUBY

Ruby is a high-level interpreter language that serves general purpose needs for application development. Its intention is to provide an object orientated scripting language which is achieved. Ruby shares a lot of similarities between Perl and Python. The author states one of the reasons the language was create was due to neither fulfilling his needs.

Ruby on Rails was written in Ruby and is an extensive framework used for web applications. It is used extensively and includes all important web development structures.

Advantages

- Applications written in Ruby can run on multiple platforms.
- Strong security characteristics: Garbage collection, dynamic typing, and restriction on memory access.
- Large standard library and access to more through RubyGems

- Well suited for web development with Ruby on rails framework.

Disadvantages

- Relatively slow due to being an interpreted language
- Considered a niche language outside of web applications due to more extensive support in other languages like Python
- Unique coding mechanisms can make it difficult to learn

PERL

Perl is a group of two general purpose interpreted high level languages that adopts many features of C programming and shell scripting. The Perl family consists of Perl 5 and Perl 6, now known as Raku. Both languages still see development in different directions.

Perl is procedural in nature and denotes variables with different sigils. These do not signify the type of variable rather the type of expression. Perl like C uses automatic typing to denote these. Perl uses references to allow users access to memory. This is a powerful feature but comes with drawbacks like seen in C.

Advantages

- Can share library compatibility with C/C++ through XS or Swig
- Being open sources allows user to release modifications of the language
- Perl is exceptional at text parsing.
- Is faster at execution than most languages even though interpreted.
- Allow control over memory
- Perl is easy to get functioning on other platforms

Disadvantages

- Has weak security practices implemented like access to memory, weak typing and no garbage collection.
- Can be tough to learn for beginners due to expressiveness of the language
- Bug fixing can be difficult with Perl due to the characteristics of error messages.

HASKELL

Haskell is a general purpose that is at its core a functional programming language. Haskell is known for introducing innovative ideas like using type classes that allow type-safe operator overloading. A purely functional language has the characteristics of having no side effects from using functions.

Features include a strong static type system, lazy evaluation, lambda, type classes and type polymorphism. It has many implementations including GHC (Glasgow Haskell Compiler).

Advantages

- Haskell has great security features. These includes strong type safety, garbage collection and no pointers.
- Haskell executables have good performance.
- Syntax is highly meaningful and concise.

Disadvantages

- Lack of widespread implantation means it doesn't have as much support as other languages
- Haskell code can be cryptic and difficult to learn
- Can be difficult to quickly build applications

PROGRAMMING LANGUAGE COMPARISON

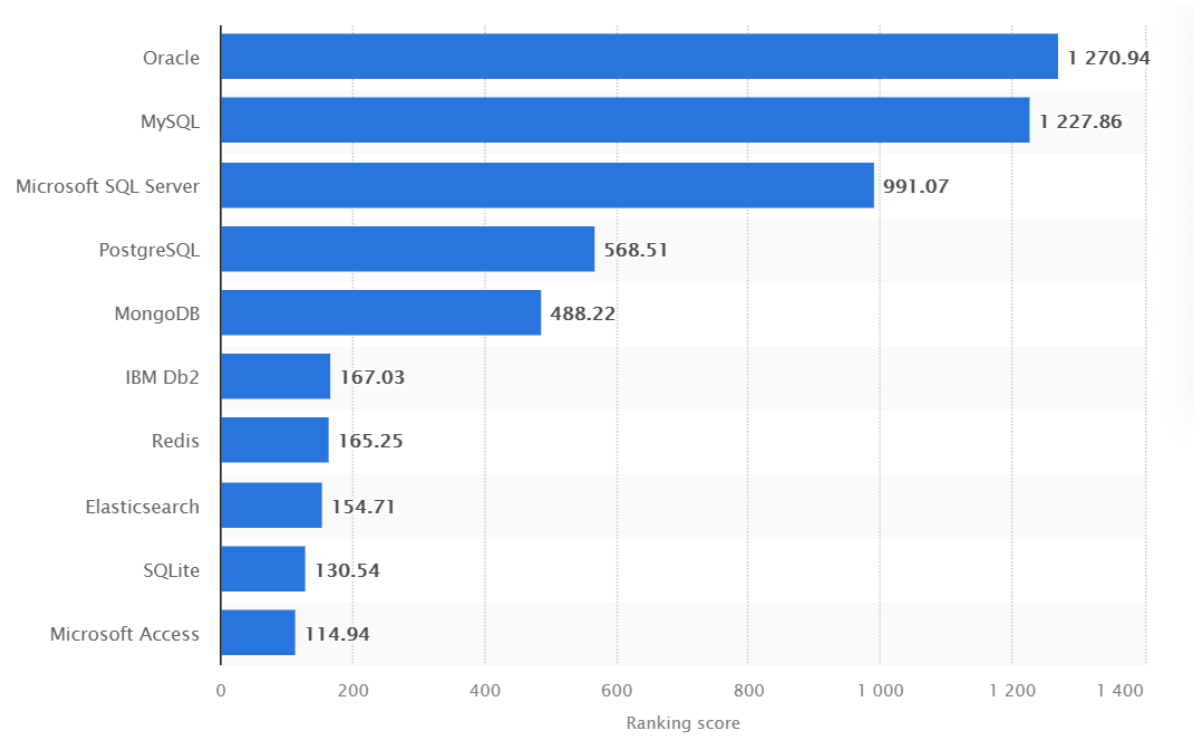
	Java	Python	C++	C#	PHP	JavaScript	Ruby	Perl	Haskell
More secure default programming practices	Features garbage collection, absence of pointers, packages, and threads makes a secure choice.	Has garbage collection, no pointers and exception handling. Uses dynamic typing. Types are verified safely at runtime	No detection on buffer overflow. Pointers can lead to memory mismanagement Generally insecure	Strong type system, garbage collection. Allows pointers	Regarded as insecure with no default input validation. Weak on types. Requires additional frameworks for security.	Dynamic typing, garbage collection and no access to pointers. JavaScript is weakly typed due to implicit casting that can be used.	Garbage collection, no pointers and exception handling. Uses dynamic typing. A secure choice	No default garbage collection. No pointers but uses references instead. No type system implemented	Secure with garbage collection, no pointers and a strong type system.
Web Applications	Plenty of libraries and servlets makes it a popular choice	Python has many frameworks like Django, Flask and Turbogears	Can be complicated and difficult to achieve. Suffers in querying and string manipulation	Well suited for building web applications with frameworks like .NET	One of the top choices due to features. Integrates with HTML coding.	Frameworks like Node.js allow JavaScript to be a stand-alone language in web development	Very flexible and supports many frameworks	Perl is a great choice due to support and features for web development.	Can accomplish the task with frameworks like Yesod, Happstack and Scotty.
Web Services	Portable and large number of APIs for XML makes this a solid choice.	Plenty of supporting framework to complete the task	Supports REST, XML and WS02 framework	.NET framework fully supports the task	Excellent for web services with plenty of support	JavaScript can easily achieve the task with recent support	Supports a wide range of frameworks with good performance.	Has most required features to implement	Supports SOAP, REST, WSDL and USDL
Object Orientated based abstraction	Is an Object Orientated language with plenty of features to support	Fundamentally an object orientated language	Supports OO programming principles	Supports object-orientated programming but is missing features like multiple inheritance	Supported and has good readability but PHP characteristics make it less than ideal	Does not directly support but has prototype-based OOP. This uses constructors that can be reused	Supports OO Programming by default	Supports OO Programming	Requires an extension. Not supported by default
Reflection	Huge number of operations to support reflection and avoids complications	Supports reflection with many features.	Limited reflection functionalities	Reflection is supported with the Reflection API. Is performance heavy	Supported through included Reflection API. Is performance heavy	Supported through the Reflect API. Is performance heavy	Supports reflection	Has support for full reflection	Has libraries to implement some features but does not fully support reflection

Aspect-Oriented Programming	With extensions provide support for seamless integration	Can be achieved through libraries like aspectlib	Can be challenging due to C++ being a static language. Supported by AspectC++	Achievable through RealProxy Class. More suitable than C++ due to stronger typing	Can be achieved through libraries. Not ideal due to weak typing	Can be achieved through libraries	The package AspectRuby provides full support	Built in package called Aspect for full support	Not directly supported but has extensions
Functional Programming	No functions. However, it is easy to mimic with interfaces and inner classes	Offers support by default and has plenty of libraries to supplement	Doesn't fully support. Can be achieved with FC++	Offers some support with first-class functions and closures	Supported but calls of functions are verbose.	Functional programming is one of the paradigms supported in JavaScript	Functional programming can be achieved but is not forced	Some support with references and closures	Haskell is a Functional Programming language by default.
Batch Scripting	Achieved with the runtime and process class.	Python supports scripting by default. A good choice	Libraries allow scripting but performance is poor	Supported but there are better suited languages for the task	Supported but limited security and scripting options	Cannot be done as JavaScript is primarily a client-side language	Scripting is supported	Perl is a scripting language by design	Possible with HSH
UI prototype design	Large set of libraries to easily support the class	Large number of libraries and frameworks to develop UIs	Supported but can be difficult to implement by default	.NET framework makes C# well capable of the task	Not supported by default. Achieved with supporting frameworks	Not supported by default. Achieved with supporting frameworks	GUI applications can be developed easily	Plenty of support for GUI applications	Has Libraries for GUI applications
Compiler Efficiency	One of the fastest interpreter languages due to combining features of compiler languages. Very memory intensive.	Relatively slow due to interpreter languages converting code line by line. Memory efficient.	Compiled language which converts all code to machine code at once. One of the fastest of this type. Memory efficient.	C# is both compiled and interpreted by a VM. It benefits from advantages of both.	PHP is an interpreted language. This suits better for web applications due to compilers being less efficient compiling rarely used code.	JavaScript engines use just-in-time compilers for improved preformance over interpreters.	"Just In Time" Compiler converts code to machine code. Based off C++	Interpreter language converts code line by line	Can be either compiled or interpreted depending on implementation.

(Dwarampudi, Dhillon, Shah, Sebastian, & Kanigicharla, 2010)

STORAGE/DATABASES

To achieve the requirements of the brief, an application storage solution will be needed. At a minimum, user credential and vulnerability records will have to be kept. In order have a basis for research, the following Database management solutions were found based on popularity



(Statista, 2021)

Oracle

Oracle is a multi-model database management system that is developed and maintained by the Oracle corporation. Oracle Database offers on-premises, cloud, and hybrid solutions of its users. It currently is the most used database management system overall. This is due to many innovative features that make it an appealing choice to developers. Oracle Database Express Edition provides a free solution and the ability to upgrade if the free specifications are outgrown.

Advantages

- ACID transactional guarantee
- Instant CAP reliability as a single server.
- Supports Structured, Semi-Structured, Spatial Data and RDF store
- Provides Blockchain tables
- Suitable for both OLTP and OLAP
- Can be used for free with Oracle Database Express Edition and upgraded to commercial versions as necessary
- Provides a good solution when a converged or master database is needed
- Excels with traditional transactional workloads with structured data.

Disadvantages

- Costly if more than the free version is required
- Does not feature multi-Master ACID transactions
- Poor with Semi-structured data
- Does not perform as well with graph-like data as other solutions

MySQL

MySQL is an open-source relational database system. It forms a component of the famous LAMP stack composed of Linux, Apache, MySQL, and PHP. It is a close second in the most chosen database management solution. MySQL is offered under two different versions: the open-source community server and the commercial Enterprise server. These are both built on the same code base, but Enterprise has several commercial extensions which can be used as plugins.

Advantages

- It provides an ACID transactional guarantee
- Instant CAP reliability
- Offers sharding which results in high availability and throughput. This also comes with low latency and linear scaling.
- Provides multi-master ACID transactions
- Excellent choice when both OLTP and OLAP are needed
- Supports both structured and semi-structured data.
- Provides excellent data security

Disadvantages

- Unsuitable for distributed SQL
- Better solutions with graph-like data
- Poor with semi-structured data
- Does not provide Advanced data protection functionalities like masking, obfuscating and throttling

BIBLIOGRAPHY

Souppaya, M., & Scarfone, K. (2013, July 1). *Guide to Enterprise Patch Management Technologies.* Retrieved from nist.gov: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r3.pdf

Barros, A. (2019, October 25). *The New Vulnerability Management Guidance Framework*. Retrieved from Gartner: https://blogs.gartner.com/augusto-barros/2019/10/25/new-vulnerability-management-guidance-framework/

Chaudry, A., Coley, S. C., Crouse, K., Davis, K., Ellis, D., Garrison, P., . . . Vohaska, B. (2021, July 26). *2021 CWE Top 25 Most Dangerous Software Weaknesses*. Retrieved from https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

CWE. (2021, March 13). *About CWE*. Retrieved from Common Weakness Enumeration: https://cwe.mitre.org/about/index.html

Dwarampudi, V., Dhillon, S. S., Shah, J., Sebastian, N. J., & Kanigicharla, N. S. (2010, August 10). *Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby PHP & Scheme.* Retrieved from arxiv.org: https://arxiv.org/pdf/1008.3431.pdf%26embedded%3Dtrue

FIRST. (2021, October 20). *Common Vulnerability Scoring System v3.1: Specification Document*. Retrieved from www.first.org: https://www.first.org/cvss/v3.1/specification-document

Ghasemzadeh, M. (2015, August 1). *The CVSS base score formula*. Retrieved from ResearchGate: https://www.researchgate.net/figure/The-CVSS-base-score-formula-12_fig5_290624055

IBM Security. (2021). *Cost of a Data Breach 2021.* New York: IBM Security.

Kamaruzzaman, M. (2021, January 20). *Top 10 Databases to Use in 2021*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba

Mell, P., Bergeron, T., & Henning , D. (2005, November 1). *Creating a Patch and Vulnerability Management Program.* Retrieved from tim.kehres.com: https://tim.kehres.com/docs/nist/SP800-40v2.pdf

Mell, P., Scarfone, K., & Romanosky, S. (2007, July 1). *A Complete Guide to the Common Vulnerability Scoring System Version 2.0.* Retrieved from nist.gov: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=51198

MITRE. (2013, February 14). *CCE List — Archive*. Retrieved from CCE: https://cce.mitre.org/lists/cce_list.html

MITRE. (2019, April 04). *About CAPEC*. Retrieved from CAPEC: https://capec.mitre.org/about/index.html

Mitre. (2021, March 3). *Frequently Asked Questions*. Retrieved from CVE: https://cve.mitre.org/about/faqs.html#cve_record_descriptions_created

NIST. (2021, October 19). *Common Vulnerability Scoring System Calculator*. Retrieved from NIST: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?name=CVE-2016-0051

NIST. (2021, October 19). *Common Vulnerability Scoring System Calculator*. Retrieved from NVD: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

NIST. (2021, October 22). *CVEs and the NVD Process*. Retrieved from National Vulnerability Database: https://nvd.nist.gov/general/cve-process

NIST. (2021, October 29). *NIST Risk Management Framework Overview.* Retrieved from www.nist.gov: https://www.nist.gov/system/files/documents/2018/03/28/vickie_nist_risk_management_framework_overview-hpc.pdf

NIST. (2021, July 16). *Security Content Automation Protocol* . Retrieved from NIST: https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/cpe

Risto, J. (2020, July 6). *Vulnerability Management Maturity Model Part I.* Retrieved from SANS: https://www.sans.org/blog/vulnerability-management-maturity-model/

SANS. (2021, February 4). *CISO Mind Map and Vulnerability Management Maturity Model.* Retrieved from sans.org: https://sansorg.egnyte.com/dl/p6YbmrhJy6

Shanks, W. (2015, May 21). *Building a Vulnerability Management Program – A project management approach.* Retrieved from www.giac.org: https://www.giac.org/paper/gcpm/344/building-vulnerability-management-program-project-management-approach/117144

Statista. (2021, June 1). *Ranking of the most popular database management systems worldwide, as of June 2021*. Retrieved from statista.com: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/