PHOENIX SECURITY

# SLA ARE DEAD LONG LIVE SLA DATA DRIVEN APPROACH ON VULNERABILITIES

SLA are dead long live SLA - a white-paper on vulnerabilities management and modern DevSecOps for operational security and software supply chain

ask@phoenix.security
www.phoenix.security
+442031953879

# SLA are dead - A Data-Driven Approach on Vulnerabilities Across Cloud and Software

# Index

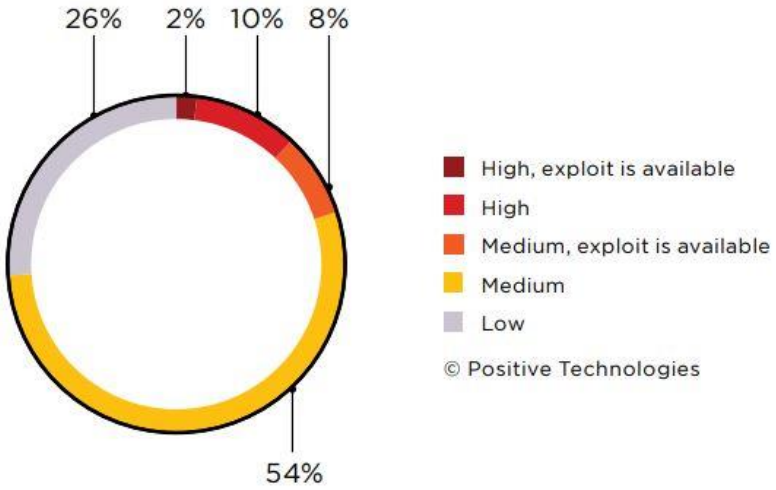
# Purpose

This whitepaper is oriented towards security professionals, CISO and executives that work in the wider field of vulnerability management and software supply chain. The broader category of vulnerability is extended to software (referred from now on as application security), operational vulnerabilities and cloud security.

Application security, Cloud Security, Container Security, operating system and vulnerability management are truly only practices. Still, due to the challenging and different nature of each area, they tend to have their own talent, expertise, and methodologies.

Vulnerability management as field has changed dramatically in the last 3 years, there are vulnerabilities coming from every area of a technology stack.


The US government's National Vulnerability Database (NVD) which is fed by the Common Vulnerabilities and Exposures (CVE) list currently has over 176,000 entries. On a recent study from positive technology only 10% of the total number of vulnerabilities in a sample were found actually exploitable, that number goes further down when considering how many systems are exposed to internet and more easily reachable by attacker with an exploit.




Number of High vulnerabilities with exploitable vulnerabilities in CVE

With only 10% of those vulnerabilities are actually exploitable a Mean time to resolution varying from 57 - 150 Days we put together this whitepaper to provide more clarity in the vulnerability management methodology.

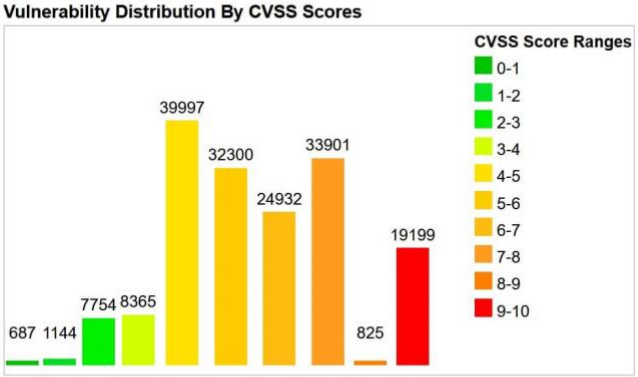
**Current CVSS Score Distribution For All Vulnerabilities**

**Distribution of all vulnerabilities by CVSS Scores**

| CVSS Score | Number Of Vulnerabilities | Percentage |
|---|---|---|
| 0-1 | 687 | 0.40 |
| 1-2 | 1144 | 0.70 |
| 2-3 | 7754 | 4.60 |
| 3-4 | 8365 | 4.90 |
| 4-5 | 39997 | 23.70 |
| 5-6 | 32300 | 19.10 |
| 6-7 | 24932 | 14.70 |
| 7-8 | 33901 | 20.00 |
| 8-9 | 825 | 0.50 |
| 9-10 | 19199 | 11.40 |
| Total | 169104 | |

Weighted Average CVSS Score: **6.5**

**Vulnerability Distribution By CVSS Scores**




Number of CVSS Per category October 22

For this reason, the security community, put together this whitepaper to join the traditionally disjointed activities under one domain. We collected thoughts on evolving and bringing together the activities that happen in application security, cloud security and infrastructure security as simply vulnerability management.

Part 1 is focused on describing the problem landscape and standard methodologies and techniques used so far in both application security, cloud security and infrastructure security/vulnerability management. This describes when SLA shall be used and how to evolve them.

Part 2 presents the evolution and the vulnerability management framework with guidelines on how to evolve and measure success in each phase.

Part 3 focuses more on the metrics and the advanced evolution of the security teams/practices. This part focuses on making a case for OKr vs SLA and when they can live together.

The whitepaper is a collaborative approach by the community and will be made available to the community freely. The support of the whitepaper is with Phoenix Security amongst other platform enables you to automate a lot of the manual work, metrics described in this paper.

# Intro

It's time to start talking about DevSecOps/Appsec and Vulnerability Management/Security Operations approach. This whitepaper intent is to analyze the methodologies and proposing approaches on both Appsec and Sec Ops …

For the purpose of this white paper security operation covers:

- Endpoint like laptops
- Servers that run software built or purchased
- Containers, kubernetes and docker
- Cloud and misconfigurations

For the purpose of this white paper application security is the practices of securing software both in operation and in development encompassing DevSecOps Methodologies. We cover under application security:

- Web and API security for websites and applications
- Code and open-source libraries
- Built software tested (commonly referred to as DAST)
- Preflight cloud and containers (infrastructure as code, container code analysis)

The evolution of application both in term of complexity and in terms of methodologies of deployment have posed new challenges to teams on the ground.

Application security has evolved and become more cloud oriented. Cloud Security teams have become more aware of code and programmatic deployment with automation's new challenges. Infrastructure security teams and vulnerability management teams have faced new challenges from modern

deployment tactics and how to stay on top of ephemeral assets like a container or autoscaling.

Security operation teams have become more aware of the code and challenges coming from software. Software security and artefact exploitation have become the new methodologies attackers are penetrating organizations. As cloud security and software security is generally less mature practice than traditional operational security, we have seen recent attacks like log4j, spring4shell, etc. targeting software.

In this new world, a new figure is arising, especially for modern software companies: the role of product security is taking center stage in modern organizations.

This white paper looks to debate what methodologies work for both (e.g., Security SLA) and what methodologies could work for both (OKRs), and when to use one or another at a different stage of maturity.

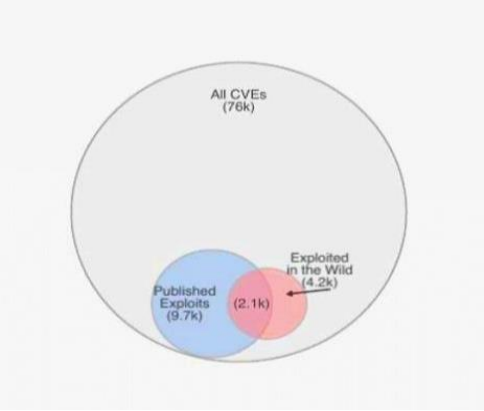
# Problem definition

Why are we debating this approach, and why are we here? We'll explore in this section the history and how we came to the pain points felt by the vulnerability management industry.

## The pain point and vulnerability

Too many vulnerabilities, too little time to solve them, and too few people, and some of them don't even matter.

In recent research, only a small number of CVEs were exploitable (10-15%), and the number of vulnerabilities reported year on year is increasing 35% per year



Improving Vulnerability remediation with EPSS [1]

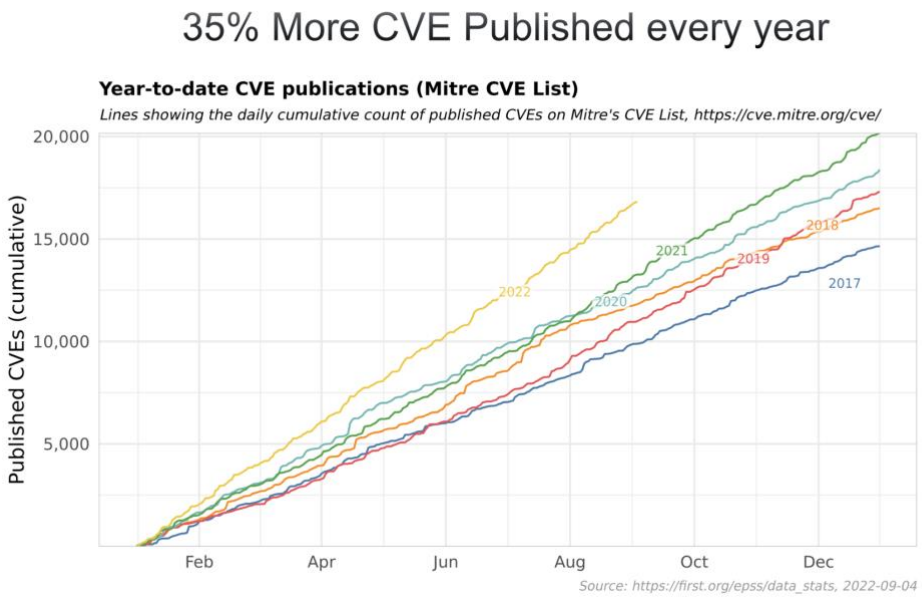



Image 1 - CVE Evolution over time

On top of the transformational challenges, we saw an acceleration in the number of CVE declared every year with 2022 having 34% more vulnerabilities declared.

## Skill shortage

Too few people, too many vulnerabilities, too little time. This sentence has been going round and round in cybersecurity. Together with the shortage described below and the number of new vulnerabilities discovered above, we have a growing problem in our industry. Add the manual triage as well that usually takes a team roughly 9h for new vulnerabilities; in a medium complexity application, we have a recipe for disaster.

[1] Improving vulnerability Remediation

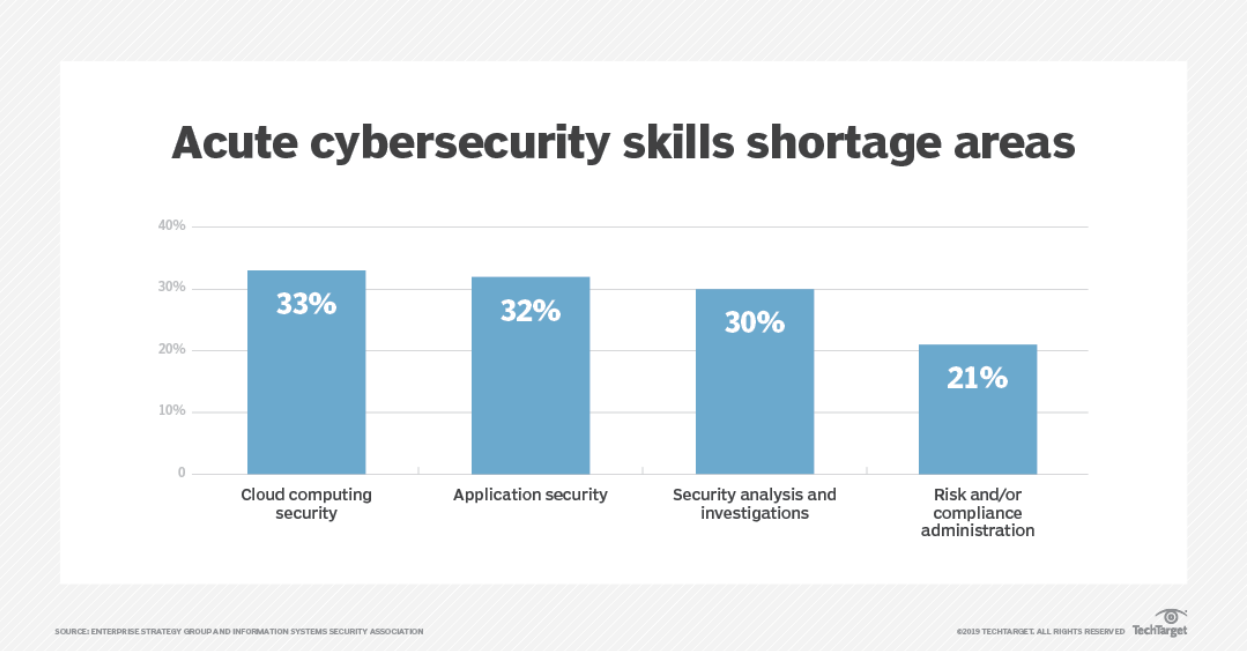



Image 2 - Cybersecurity Shortage on Cloud security & Application security [2]

Security teams are trying to resolve the increased number of vulnerabilities with pure professionals, nonetheless this technique is leading to more burnout and less professionals in the critical areas of application security, cloud security.

With more vulnerabilities being discovered and resolution time pushed lower and lower by attackers, it is challenging for organizations to keep on top of what to fix and when. For example, log4j vulnerabilities were exploited by malicious attackers in 7-14 days from initial discovery and weaponized by malware within one month.

This evolving landscape makes hiring and retaining seasoned teams more difficult. IT complexity is compounded by a lack of cybersecurity automation, poor cross-team collaboration, and a need for workflow coordination.

**Organizational issues:**

Security executives are feeling the pressure to give direction on securing all facets within their organizations. Securing all the vulnerabilities is almost impossible nor practical approach. The number of vulnerabilities is an overwhelming amount and vulnerability sources are only increasing. According to Gartner's 2020 CISO Effectiveness Survey, the average enterprise has more than 16 security tools, with 12% of CISOs reporting 46 or more. "We're swamped by tools and data".

[2] https://www.techtarget.com/searchsecurity/news/252463186/Effects-of-cybersecurity-skills-shortage-worsening-new-study-says

The problems organizations are currently facing are:

- C-Level Executives are disconnected from the business about how effective their security program is as their teams are focusing on fires and volumetrics
- Security teams operate separately from development and operations teams while mandating ineffective vulnerability management policies
- IT infrastructure is increasingly heterogeneous and ephemeral – from on-premises to virtual to serverless, public cloud to private to hybrid, therefore "manual processes can't keep up."
- Directives to remediate application security vs cloud security vulnerabilities are confusing due to different resolution techniques

The challenges outlined above are typically the result of a poor security strategy, resulting in critical gaps in coverage, lack of alignment, and an enormous backlog of tech debt across the organization. Prioritizing vulnerabilities across the application, cloud, and containers differ in every organization. Either security teams are manually triaging findings without context (business logic and locality), or development teams are being tasked with finding and fixing all vulnerabilities under a specific SLA. Both of these common scenarios create friction and frustration between developers and security teams.

Securing applications before they are deployed to production and monitoring them during runtime each has its pain points. Pre-production and post-production vulnerability management are typically handled separately depending on the team responsible for them and where the issue has been identified. For example, what happens when a vulnerability is found within a container? Does the application team fix it? Does the infrastructure team fix it? Again, it depends on the context of the finding. In this scenario, the vulnerability could be source code or third-party specific, or it could be an operating system issue. The process to remediate in this example depends on the context of the finding.

How do you best measure and facilitate the resolution of vulnerabilities without crushing the spirit and soul of development teams with the dreaded sentence "fix vulnerabilities in X number of days"? How do you avoid the pitfalls of all the vulnerabilities that need to always be fixed in X number of days? Are SLAs still relevant for security, or is there a better way?

The answer? It depends. There are several methodologies and no one-size fits all; it depends on the organization and the maturity of both engineering and security teams. The magic words that help in these situations are collaboration, cooperation, and measurements. These are three words, but in reality, there is just one: culture, or the more popular phrase: "Security is everybody's responsibility".

To embed security across the business, security teams must collaborate with everyone who is involved in delivering code. This includes partnering with relevant stakeholders from the business, engineering, and operations teams. When creating the backlog of work, security findings need to be risk-based and contextualized to be adequately prioritized so that teams focus on what matters most.

Developers often lack the time and context that helps them understand why a specific issue is critical to fix. Looking at a problem atomically does not communicate how bad a specific bug/problem is in the wider context (locality) of where that problem materializes.

So how can this be fixed? For a long time, security has been mandated, fixing problems or hitting specific SLA deadlines without any specific context. On other occasions, security fixes have been deemed a problem for security teams that traditionally can't scale.

The resolution times, often called SLAs, are usually not agreed upon in collaboration with the CTO and the development team. The requirement to fix specific vulnerabilities in X amount of time and days, more often than not, cause friction.

When conversing with a range of professionals and experts in the field about Service Level Agreements (SLAs), Service Level Objectives (SLOs), and Objectives Key Results (OKR) in Vulnerability Management, Application, and cloud security, the discussions have been argumentative, to say the least.

This white paper covers the complexity of the landscape and the significant difference between various "updating" strategies and elements to consider when setting SLAs or SLOs.

We will explore how those metrics, with a feedback loop between security and development teams, can facilitate a conversation and turn the tide in what is usually a frustrating conversation.

# Part 1- the Vulnerability landscape

## Definitions

Let's start with definitions of the various metrics like SLA and SLO and what they are:

| | Security SLO | Security SLA | Security OKR |
|---|---|---|---|
| **Definition** | A target reliability level objective | A legal contract or agreement that, if breached, will have penalties[3] | The team will meet Objective agreed internally (e.g., fix rate of critical below 14 days) |
| **In our context (Security)** | Integral part of Security SLA, timers and objectives required to fix a specific vulnerability.<br><br>E.g., SLO between security and developers to review vulnerability exceptions in 3 days | In our specific case, an example of SLA: "A critical level vulnerability must be fixed within X number of days". | The OKR[4] is a collection of objectives and supporting metrics within timeframes. A quick example of an objective for the team could be the number of vulnerabilities resolved per sprint or a balance between user stories and security/bug fixes. |
| **Example** | Critical Vulnerabilities will be resolved in 28 days 95% of the time | Publicly available products will have 0 critical vulnerabilities upon critical release vulnerabilities disclosed will be solved in 10 days | The application will maintain a fix rate defined in SLO of 95% resolution within 15 days |
| **Who Sets it** | Product Owner in partnership with security teams | Business Development, Legal teams, IT and Devsecops | Product owners in collaboration with security team<br><br>Supported by SLO or Key Indicators (the K in O**K**R) |

Table 1 - Vulnerability and SLA/SLO/SLI Descriptions

SLAs are often set as company guideline (internally) nonetheless there are more and more regulations guiding resolution timeframe for identified vulnerabilities (e.g.

- PCI DSS requires security patches to be applied in critical systems and regular vulnerability scan/ penetration test
- NIST SP 800-40 Rev 3[5]

[3] Contractual SLA have penalties while internal policy SLA have (OLA in the common definitions) act more as a guidance for different teams SLAs have been a risk management/mitigation metric that many regulatory bodies use to grant and renew certifications as an indicator for the health of a cybersecurity program

[4] Helpful books on the subject are Measure what matters most or How to measure everything in cybersecurity.

[5] https://csrc.nist.gov/publications/detail/sp/800-40/rev-4/final

- Overall industry is driving towards a 14-day resolution time for vulnerabilities that are critical and exposed to web nonetheless this is a generic aim, not a regulation in itself

Note. For an SLA, an SLO is typically a contractual agreement while for OLA is guidance. SLAs are often misused and agreed upon/mandated internally within teams instead of using OLA. In summary, if there are contractual fines, the OLA are objective to meeting contractually, while internally, they are often captured in policies and agreed upon amongst teams without contractual consequences. An example of OLA is team X must maintain uptime of 99.99. An example of SLA is contractually agreed between a client and a vendor: if availability is below 99%, a service credit/fine will be issued.

# The Landscape

## The Vulnerability Context and Landscape

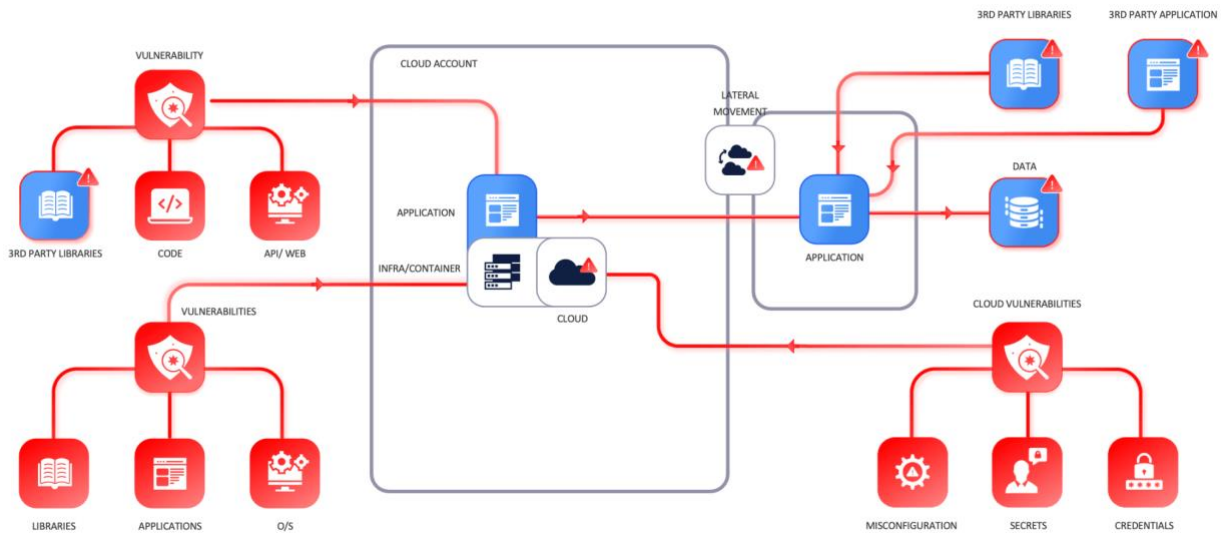



Image 3 - Asset Landscape in modern organization/enterprises

The asset landscape of modern organization is complex and widespread. Organization build application dynamically putting together internal code, libraries from 3rd party, open-source operating systems and software into container deployed in cloud.

We can break down the assets that organization leverage in the following macro categories:

- **Runtime**: Where software or application run/execute
  - Operating systems (in datacenter servers, in cloud workloads)
  - Container systems (in a datacenter, in the cloud)
  - End user computing (laptops, mobile device)
- **Software artefacts**
  - Built software: an application build from another 3rd party organization and/or purchased (e.g., software like adobe).
  - Software asset: libraries, compiled code and similar partially built software that can't be run on its own but forms part of a build file
  - Code: pure code files that need to be compiled
  - Open-source software/libraries: code built by a community that can be used (to a degree of freedom dictated by the license)
  - Software on endpoint or container system that is fully compiled and is written/purchased from a 3rd party
- **Website/Web API**
  - Frontend websites:
  - Web libraries or servers that enables web applet to run

The vulnerability landscape in modern organizations is complex; we can categories vulnerability types or security misconfigurations into several categories. Vulnerabilities in the various categories have different behaviors and require different levels of attention.

We can categories assets into the following categories:

- **Application Security** - Vulnerabilities in software, 3rd party libraries, and code running in live systems
- **Infrastructure Cloud security-related -** Vulnerabilities that concern images or similar infrastructure systems running in the cloud
- **Cloud security** - Misconfiguration of cloud systems (Key manager, S3)
- **Network security / Cloud security** - Vulnerability affecting network equipment like WAF, Firewall, routers
- **Infrastructure security/Endpoint Security** - Categorized as everything that supports an application to run, that is traditionally Server, Endpoints, and similar systems
- **Container vulnerabilities** are derived from either the image in a register/ deployed, or from the build file that composes them.

The systems used to measure the security posture and the security health of different elements that compose our system are quite wide, which is illustrated in the picture below.

The resolution time, hence, SLAs, are fairly different between asset types across the various categories.

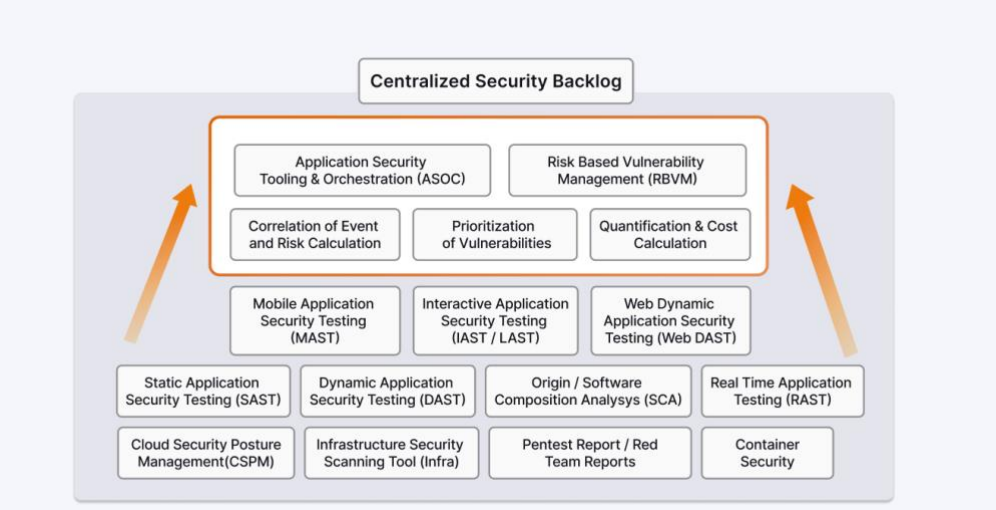



Image 4 - Vulnerability Tooling Landscape (note RAST often referred to as RASP)

## Vulnerabilities Timelines

- Image 4 explains the complexity of dealing with multiple metrics. It's easy to implement SLAs, but when do you start the timer for remediation compliance? Consider the three timelines below.
- The timeline' shown in image 5 shows the lifecycle of a vulnerability
- The second line shows the timeline of vulnerability in your organization from the date of discovery to patch/confirmed resolution
- The above line shows the timeline of risk acceptance/mitigation

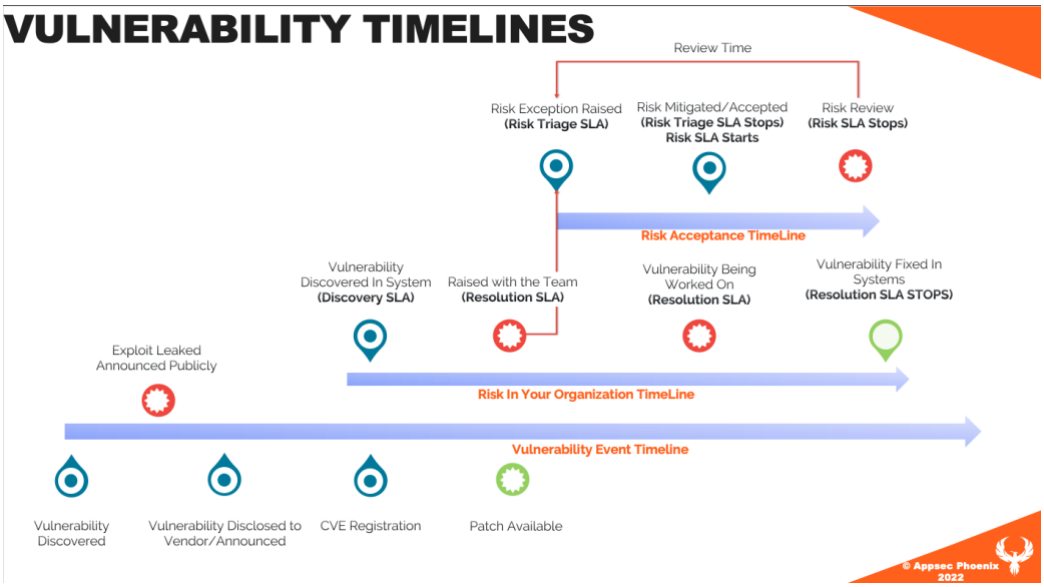



Image 5 - Vulnerability Timelines

Timelines to fix vulnerabilities are dictated by several events. They are composed, in reality, of several timelines. We start from the official public timeline (Image 4) that determines the public or private disclosure of a vulnerability till the time of the release of a patch/bugfix.

At any point in this evolution, your system can detect the vulnerability.

Usually, this happens when Application Security Tooling releases a signature or detection for a vulnerability discovered. The zero-day time that spans between the vulnerability being released and the patch/fix released by the vendor. (second timeframe)

When a vulnerability is disclosed in public security scanners, vendors tend to release the vulnerability detection within hours or days to enable organizations to detect vulnerabilities.

The exposure window is usually the time from the release of the vulnerability to the time of resolution in your system. Nonetheless, in reality, the timers for exposure windows start from the time the vulnerability gets identified in your system to the time the vulnerability gets resolved.

SLA or SLO usually are the target times from the vulnerability being identified and discovered in the system or the ticket being raised with the individual team (resolution SLA).

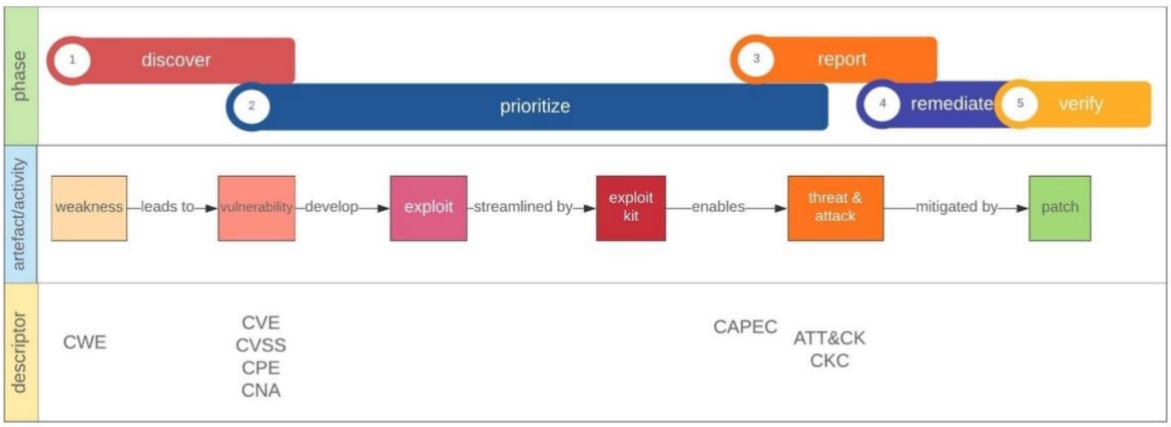



Image 6 - Simplified Disclosure timeline

Discovery to Declaration to CVE - This timeline is usually the most dangerous because the discovery of the vulnerability means there is no patch available, and the systems are at risk for zero days.

- Disclosure in the wild of vulnerability usually involves the vulnerability being disclosed widely on the web for various reasons, giving the vendor no chance to fix the vulnerability. The resolution time/mitigation time becomes critical.

- CVE Registration - The CVE register acknowledges the vulnerability, and the vulnerability does receive a specific code.
- PoC - Proof of Concepts made available - Usually, the PoC is a piece of code that exploits vulnerabilities in systems.
- Vulnerability identified in network/container/code.
- Vulnerability being worked on by a team - Not time a vulnerability/ patch is straightforward to fix. Sometimes, an update is quite straightforward and requires only a few updates, whereas other times, it requires extensive testing and careful planning.
- The vulnerability is being remediated by the team.
- Vulnerability remedy being confirmed (PenTest, Security scanner).

Resolution of vulnerabilities can also be driven by architectural restructuring and upgrading to more modern systems.

A system could also be out of support and maintenance, and the patch might not be available anymore by the vendor.

The risk of those vulnerabilities needs to be addressed and adjusted depending on the criticality of the system and the blast radius it could cause.

Nonetheless, a deeper analysis of those systems might lead to overall compensating controls (like WAF, RASP, System lockdown, and restrictive access control) that lowers the overall probability of exploitation and hence reduce the overall risk

## CVE Timeline

To add to the context a CVE don't get immediately registered following a picture representing the vulnerability timeline for a CVE from first seen to APT37 (cozy bear) exploit group use of the vulnerability

## Risk-Based Vulnerability Use Case

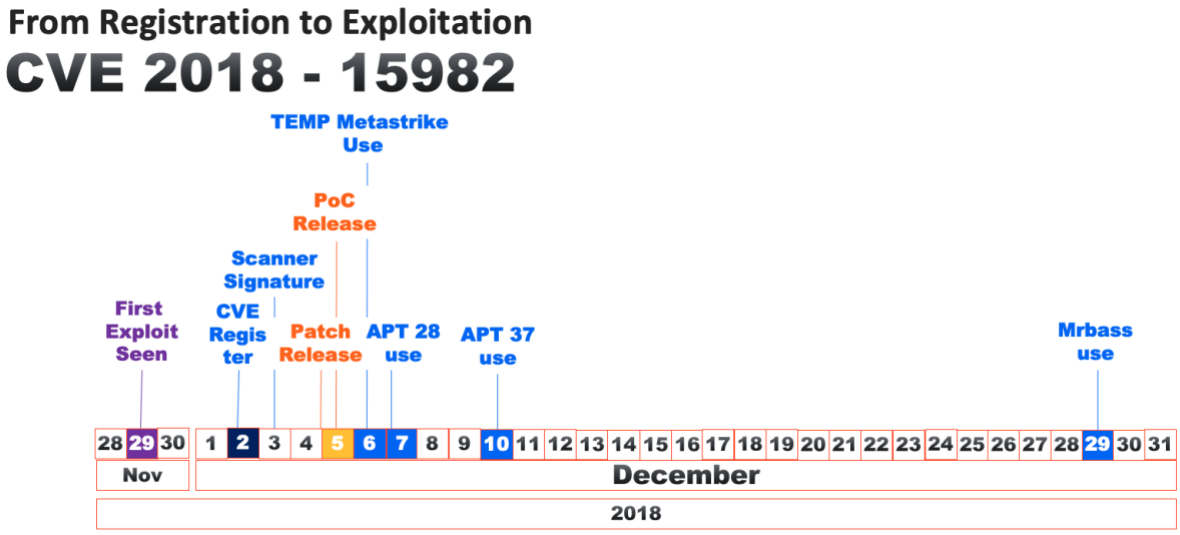



Image 8: A timeline of events for CVE-2018-15982

CVE-2018-15982 affected Adobe Flash Player versions 31.0.0.108 and earlier, a Mozilla Firefox compatible web browser plug-in. The vulnerability was a use-after-free vulnerability that could give an attacker arbitrary code execution. The CVSS base score was 9.8 and labelled as "Critical". A timeline of events related to CVE-2018-15982 is shown above in Image 8. For simplicity, we will only consider how the probability of exploitation impacts contextual risk.

To calculate the probability of exploitation for CVE-2018-15982, we need to combine several Cyber Threat Intelligence inputs, including

- Which attacker groups are actively exploiting the vulnerability?
- Whether or not an official patch has been released by the vendor
- Whether a publicly available PoC exists
- Whether fully mature malicious exploit code is available
- Is the vulnerability observed to be actively exploited in the wild?

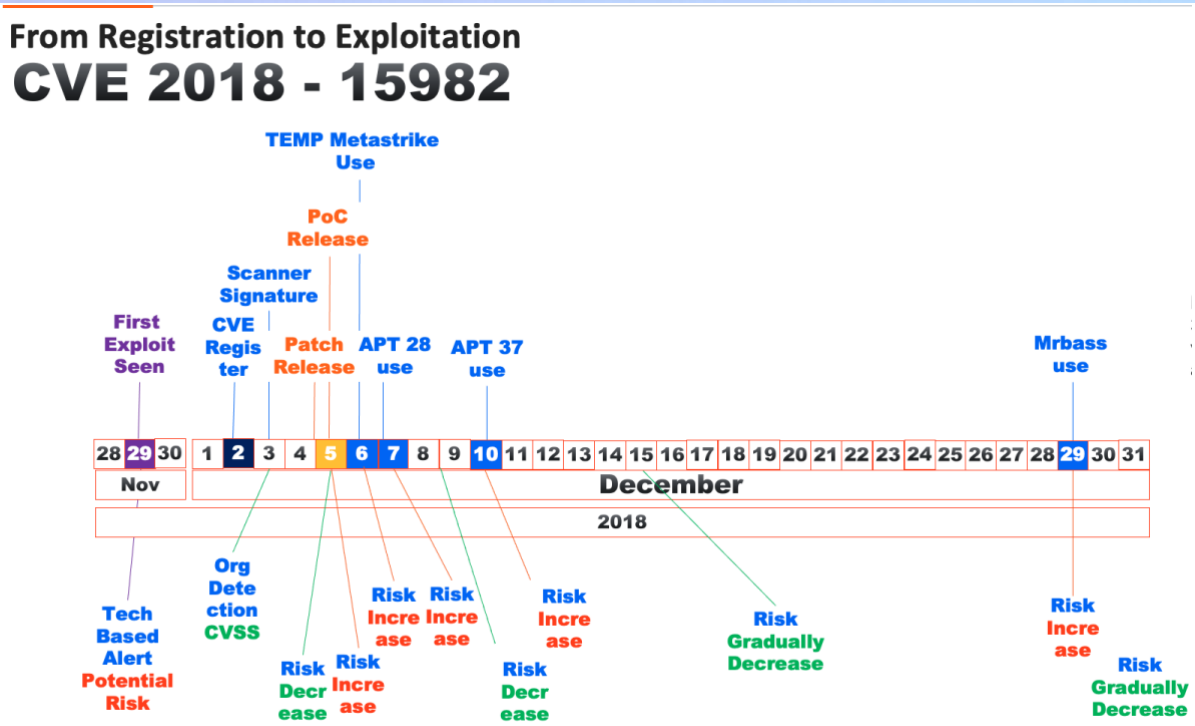



Image 9: Timeline for CVE-2018-15982 with relative risk assessment

Image 9 shows the same vulnerability timeline for CVE-2018-15982 with a relative risk assessment below the timeline. Analysis of the vulnerability for a mature triage includes the following significant events:

- November 29, 2018 - the vulnerability is first seen in the wild before the vendor officially acknowledges its existence. This scenario is a “zero-day” vulnerability. It implies that organizations are not prepared to detect it because traditional security products such as malware scanners have no prior knowledge of its existence. The vulnerability in Flash Player can be categorized as “potentially exploitable”.
- December 2, 2018 - CVE is officially issued for the vulnerability and assessed with a CVSS score of 9.8 - critical severity.
- December 3, 2018 - A malware signature is available. The impact on the risk of exploitability is highly contextual depending on the malware signature’s availability to each organization. The contextual risk is reduced if their installed malware scanner has added the signature and scanning applications have been fully updated.
- December 4-5, 2018 - In quick succession, the PoC’s source code and an official security patch from the software vendor are released. The impact on the contextual risk of exploitability quickly diverges. If an organization can quickly deploy the security update, its risk is reduced. Otherwise, the overall risk of exploitation is increased because PoC is available, meaning that attackers now have easy access to source code that can be weaponized.

- December 5-29, 2018 - Increased interest on the web, Twitter chatter, and security researchers are publishing blog articles describing how the vulnerability works, but more importantly, advanced persistent threat (APT) groups are exploiting CVE-2018-15982 scale and mature malicious exploit code is being shared on dark web forums. The contextual risk for this vulnerability increases dramatically for organizations.

Only using the CVSS score to plan a defensive strategy indicates that this is a critical vulnerability and demands attention. However, when the exploitability factor is enriched with CTI, the contextual risk score changes over time. Also, an opportunity to dramatically save valuable IT security team efforts becomes available through a platform that can automate the aggregation and contextualization of Cyber Threat Intelligence.

To optimize prioritization, security team members could have been automatically notified immediately after the CVE-2018-15982 was published, allowing human analysts to deploy temporary workarounds such as disabling Adobe Flash Player or blocking websites that contain Adobe Flash content. Furthermore, remediation efforts could be optimized by prioritizing CVE-2018-15982 immediately after the official security patch is released. For a human-only team of threat, analysts to optimize prioritization in this way would require a significant number of manhours to monitor and communicate the situation.

Today CVE-2018-15982 has a 50% exploitability, low attack surface (Adobe flash being decommissioned), and almost 0 activity in the wild from most threat actors. The risk level is medium to low. Nonetheless, a CVSS of 9.8. Image 12 below shows the contextual exploitability risk score evolution over time. The Picture below shows the importance of context-based prioritization. [6]

[6] A wider whitepaper focus on context and impact-based prioritization is available at: https://appsecphoenix.com/whitepapers-resources/whitepaper-vulnerability-management-in-application-cloud-security/

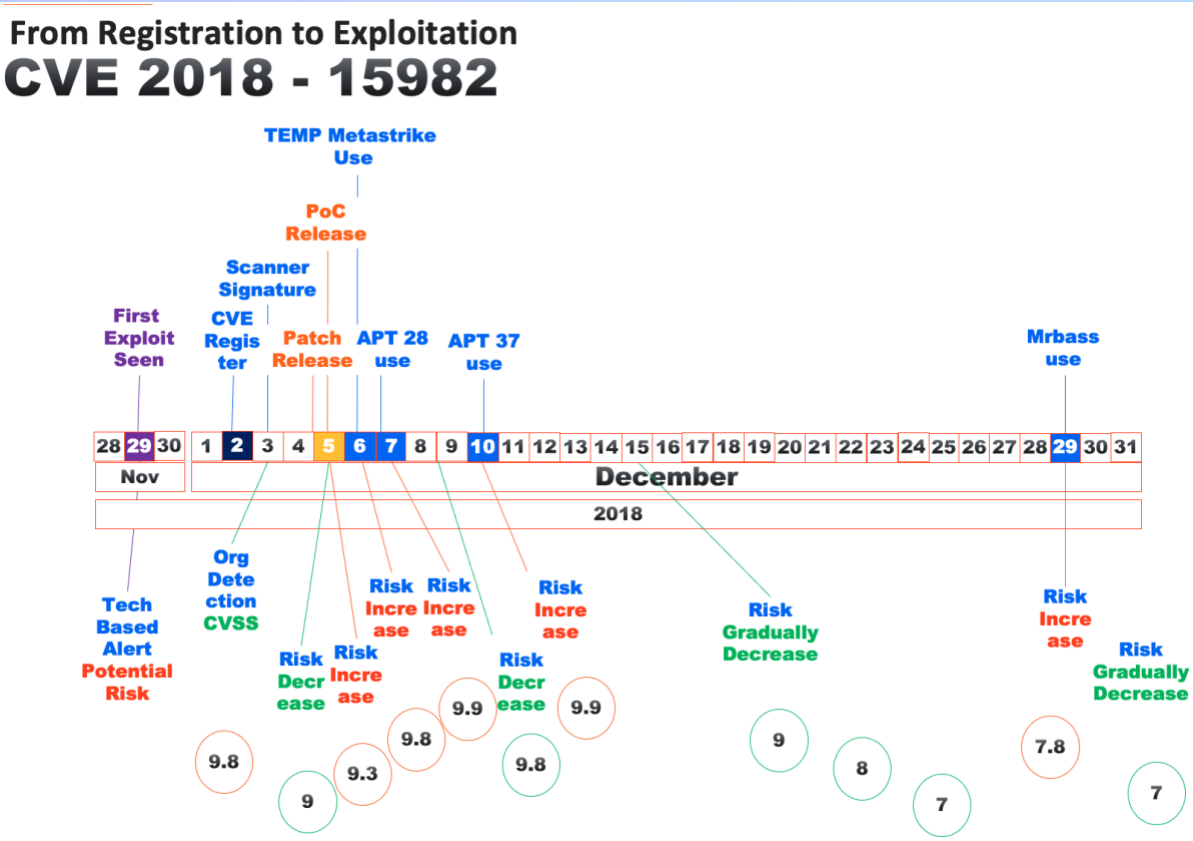



Image 10: Contextual exploitability risk score evolution for CVE-2018-15982

## SLA, SLO and Vulnerability Timelines

The Image 11 adds the commonly used timers and when they start/ finish. Those timers are important as the Security SLA/SLO (for now on referred as SLA, SLO for simplicity) set by organization are based on some or all those timers. Usually, the timer that get used the most is Mean time to Resolve that generally is considered from the vulnerability getting discovered to the resolution. Caveat that most of the vulnerabilities that remain unresolved add up the Mean time to resolution and clattering the data. A better time for SLA is usually Mean time to Resolve from Acknowledgement:

- True Exposure is the timeline from vulnerability released/discovered (publicly or not) to the time it get fixed in your organization. Caveat the vulnerability might get discovered or might be hidden in your organization, it all depends when the signature of a scanner or a vendor/ bug bounty programmed discloses the vulnerability to your organization.
- Zero-day exposure is usually the time when the vulnerability get released and when a patch is made available by the vendor (divided here for clarity between hidden - when the vulnerability is actually discovered and explicit when the discovery is made available on the web)

- Mean time to Resolve - MTTR - is the average time the ticket takes to get resolved is usually captured from ticket acknowledged and ticket resolved
- Mean Time to Acknowledgment - MTA - is the time between the discovery of a vulnerability is made, and the time it gets taken into work by the dev team
- Mean time to Resolution from Acknowledgment - MTTR - A - this timer counts the amount of time it passes from Acknowledgment of the ticket to resolution of the vulnerability in the ticket from developers. This is a better indicator of work being done and average time it takes to resolve a vulnerability
- Mean Time to Open MTO - is the time between ticket being raised and ticket being worked on. This is usually similar to MTTR - D
- Mean Time to Resolve from Discovery MTTR-D is the overall time from discovery of a vulnerability to full resolution

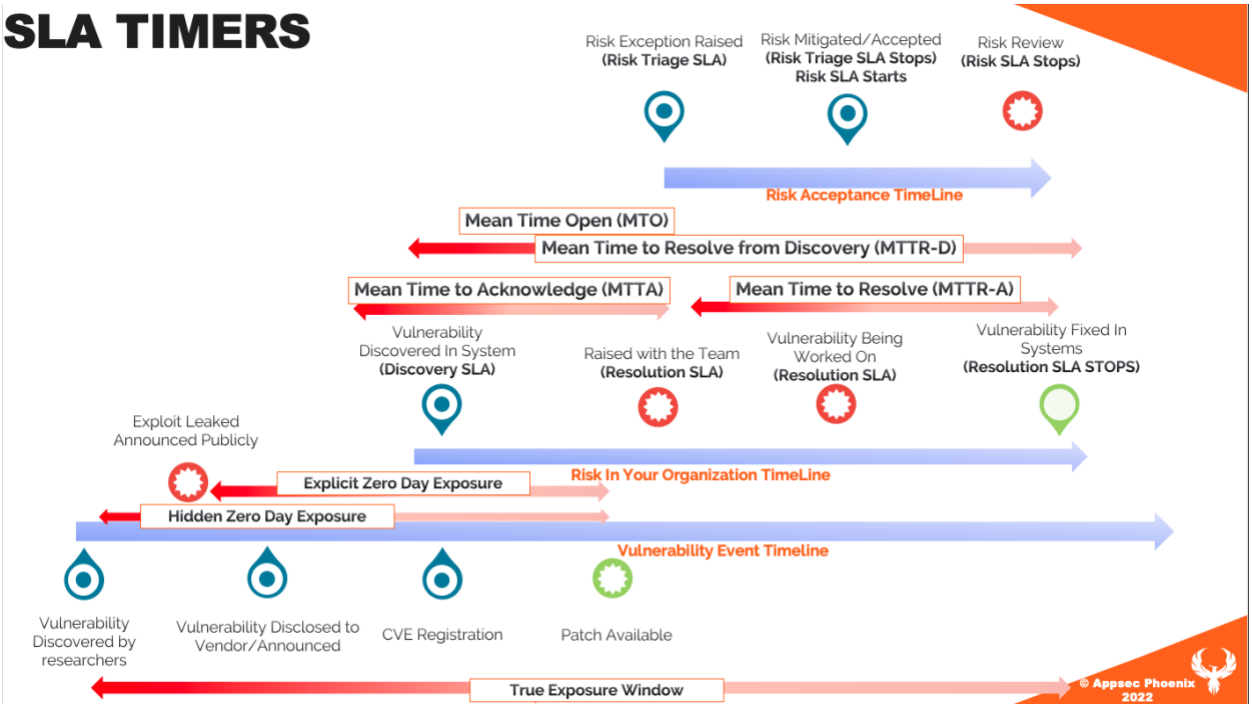



Image 11 - Security SLA Timelines & Timers

## Security SLA/SLO Definition

SLA/SLO based on severity lacks the context elements (importance, criticality of asset/data), while the SLA/SLO based on risk is more precise, but it could vary over time depending on the variation of threat intelligence, exposure etc....

- Based on Severity of vulnerabilities - does not account for context and is fixed
- Based on Risk - account for the criticality of assets and varies over time

Ultimately it is up to you which SLA you would like to use, and it matters in the context of an agreement with the development teams.

## Vulnerability Related SLA:

- Discovery SLA = This SLA provides the agreed time on how long a team should fix the vulnerability from the time of discovery (in the system) to the time of resolution.
- Resolution/Acknowledgment SLA = This SLA provides the agreed time on how long a team should aim to fix a vulnerability. Usually, the clock starts when the ticket gets acknowledged or after triaging it.

## Risk Related SLA

- Risk Triage SLA = This SLA provides the agreed time on how long it should take to triage a risk and accept/reject it or mitigate it.
- Risk SLA = This SLA provides the agreed time on how long the risk should be in the risk status - accepted, signed off (Maximum Risk time)

## Timers and Statistical Indicators

There are several indicators used to measure performance and average resolution times for SLA, SLO

MTTR (mean time to resolve) is the average time it takes to resolve a failure fully. This includes the time spent detecting the failure, diagnosing the problem, and repairing the issue

MTTA (mean time to acknowledge) is the average time it takes from when an alert is triggered to when work begins on the issue. This metric is useful for tracking your team's responsiveness and your alert system's effectiveness.

# Notes:

Some notes on the above SLA/SLO

- **The Discovery Timers** are controversial as it does not calculate exactly the time when a ticket was raised with the team that needs to solve it but gives a good idea of the age of a vulnerability in the organization. Those timers are often referred to as SLA
- MTTR resolution timers need to account also the business downtime unless you have teams that follow the sun and can work on resolution around the clock

- MTTR and other resolution considerations should account for release cycles. When a vulnerability and story are resolved might not be detected by the scanner immediately, so there should be compensation for this buffer in the calculation

# Part 2 - Maturity Model & Vulnerability Management Framework

Vulnerabilities are everywhere. If you want to start approaching an application security programmed or a vulnerability management programmed, the level of upskilling and understanding is very high and steep.

When starting one of these programs, the first question is where are we in the journey, and the second one that arises - where do we go from here?

Introducing the Vulnerability Management Maturity Model

We look at several maturity models from NIST to NCSC guidance and SANS. Despite being good guidance, they are disjointed and look at the problem not from a risk perspective

In the effort to improve vulnerability management maturity calculation and move towards a more risk-based approach, we propose a model that encompasses application security, patching, vulnerability management, and cloud security.

The practices and procedures will focus on the identification and remediation of vulnerabilities.

This process as a vulnerability management process is ongoing and will continue to evolve and be refined.

## What is vulnerability management Lifecycle

Vulnerability management was historically seen as managing risk around the operating systems and endpoint software.

Patch management is the baseline of all Information Technology organizations and must meet and be treated as such.

An owner (whether individual or group) of each asset is responsible for keeping that asset secured via a patch management program.

Vulnerability management is the process and procedure of maintaining a healthy and risk-based posture and environment.

## What is modern Vulnerability management

A Modern vulnerability management includes the following pillars and process:

- Prepare asset register and individuals
- Assess maturity and identify issues

- Prioritize vulnerabilities
- Act on vulnerabilities and measure actions (SLA, SLO, MTTR)
- Re-assess the assets
- Iterate on vulnerability management process and procedures

# Vulnerability Maturity Model Levels

The levels of maturity measure from very immature (L0) to a highly mature (L5). The methodologies considered vary from an absent process (L0) to a more data-driven measured, and controlled process (L5). In the model we refer as well to SANS vulnerability management framework[7]

# Maturity Model & Framework

The model was put together to consider both scanning, and maturity in the asset management and DevSecOps resolution methodologies. The model's objective is to enable teams to self-assess where they are on the scale of self-sustainability and a risk-based view of vulnerabilities.

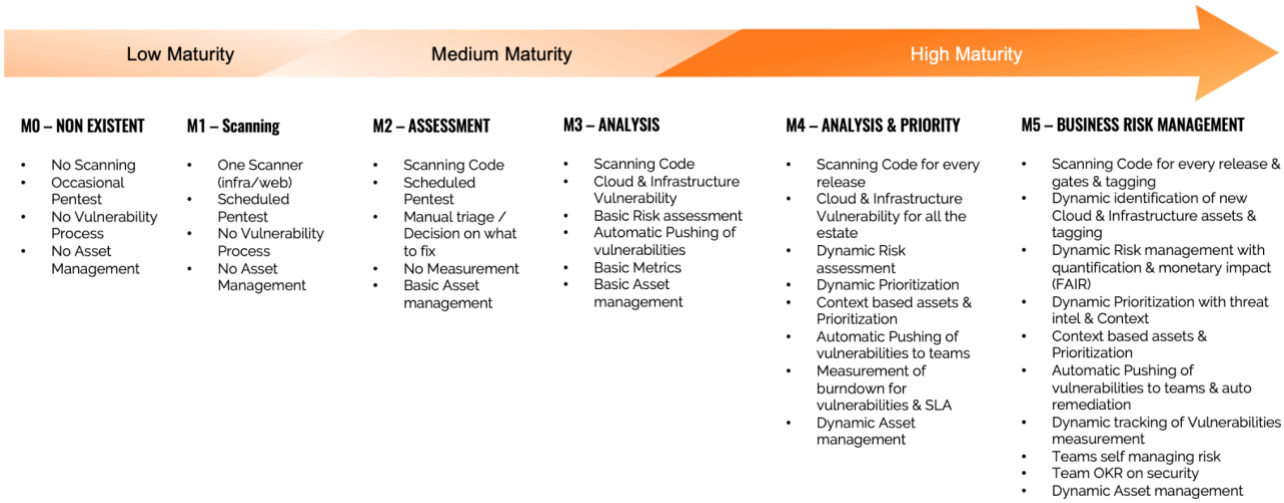



Image 12 - Vulnerability Maturity Level

7

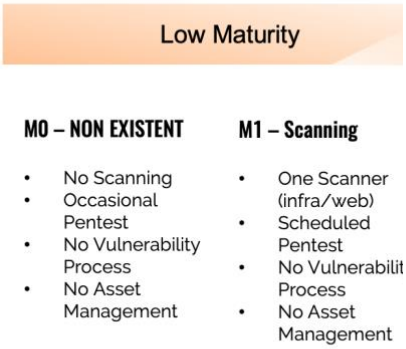
For a more comprehensive view of Maturity Models in DevSecOps refer to the modern application security and DevSecOps Book

For a more comprehensive list of Maturity in vulnerability management, refer to the SANS Maturity Model

Every organization should go through different maturity levels and SLA/SLO exercises. The following is a good indication of maturity level.

**Maturity 0 (mapped to SANS VMM Level 1) - Non-Existent**

- No Asset Register
- No Scanning Capabilities
- No Vulnerability Management Process
- No-Risk assessment of vulnerabilities
- Occasional PenTest or manual assessment
- No scanning capabilities.




**Maturity 1 (mapped to SANS VMM Level 1) - Scanning**

Some scanning capabilities (early stage)

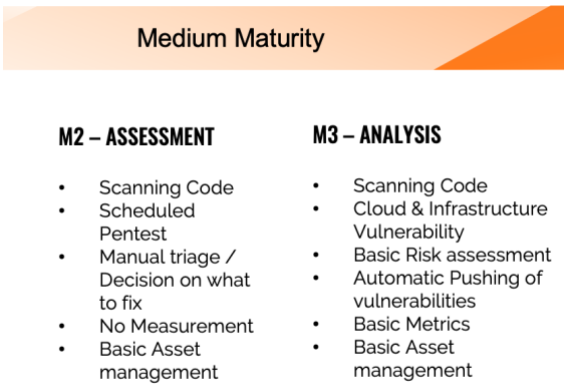
- No Asset Register
- One or two scanners (infrastructure, code)
- Some Pen testing activity (internal/external)
- No Vulnerability management process
- Just Fix vulnerabilities when there is time.
- Vulnerabilities are fixed when and if discovered

When organizations are at maturity 1-2, the best and most efficient way is to start with a smaller team, scan and document assets, and demonstrate good control on those projects.

After this it is easier to replicate the model at scale with a systematic approach while teams gradually mature.

**Maturity 2 (mapped to SANS VMM Level 2)**

- Scanning Code, Assessing software with DAST or some dynamic application testing capabilities
- External attack surface tested
- Critical assets pen tested regularly
- Manual triage or some Basic SLA (for a whitepaper on SLA see here)
- Vulnerabilities fixed when and if discovered
- No asset management or some basic level
- Some non-formalized vulnerability management process
- No risk acceptance or assessment process

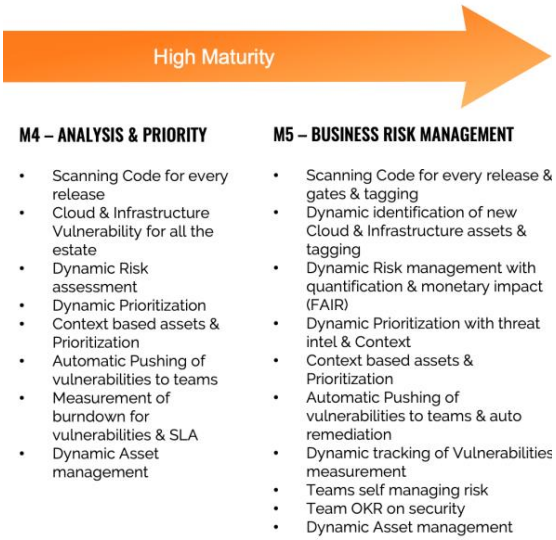



**Maturity 3 (mapped to SANS VMM Level 3)**

-
- Start Using SLA for the whole (for a whitepaper on SLA see here)
- Policy & mandate when SLA fix
- Some basic level of the vulnerability management process
- Some basic level of asset management
- No major measurement of vulnerabilities
- Not a consistent measurement of resolution

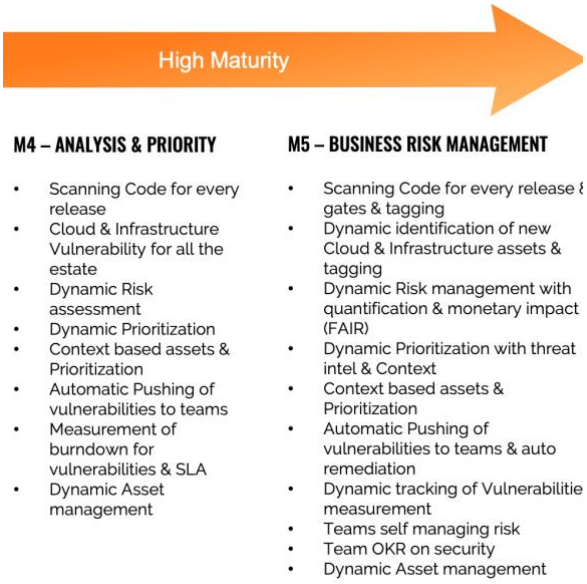
**Maturity 4 (mapped to SANS VMM Level 4)**

- Start Using SLA for the whole organization.
  - Consistent use of Severity Based SLA
  - Move to Exposure Based SLA or Risk based SLA
- Consistent Pipeline approach for vulnerabilities scanning
- Scheduled/Regular Pentest, assessment
- Vulnerabilities fixed when and if discovered
- No asset management



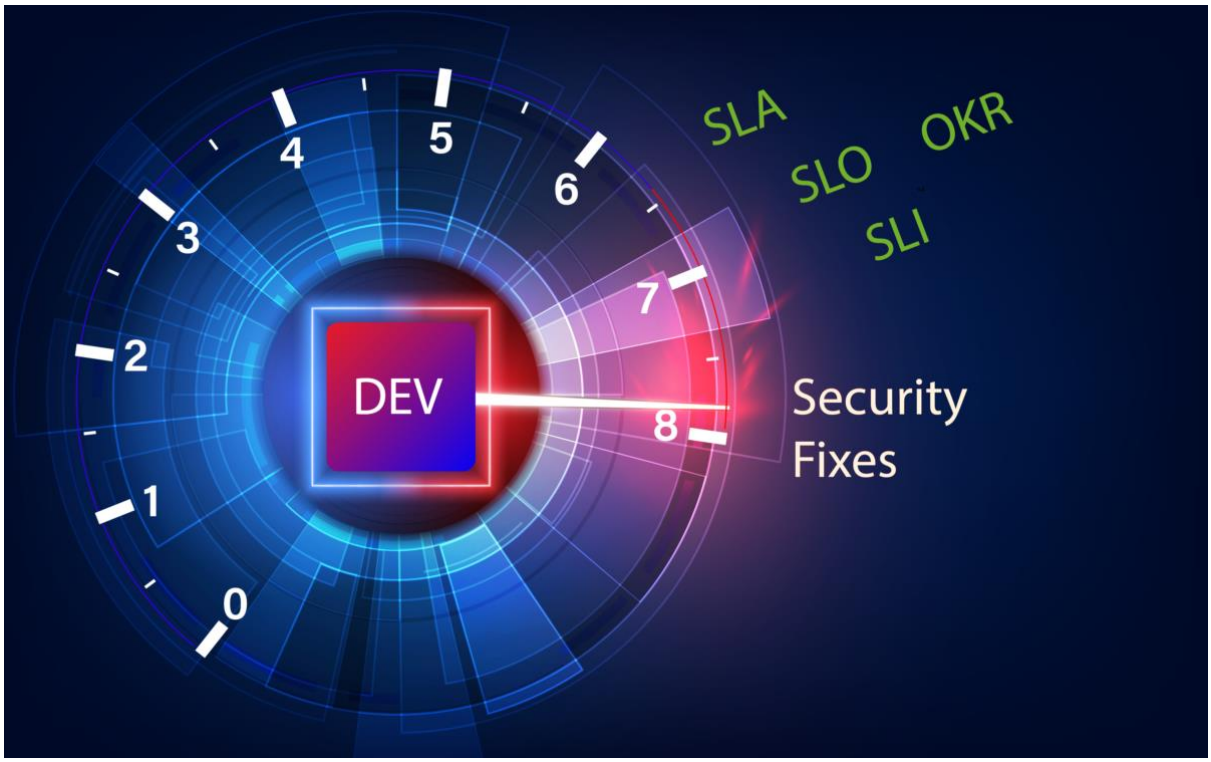

**Maturity 5 (mapped to SANS VMM Level 4)**

- Creating Customized SLA/ SLO for different teams/complexity.
- Implementing SLA Levels based on Type of asset and risk
  - Consistent use of Severity Based SLA
  - Move to Exposure Based SLA or Risk based SLA
- Embedded in Feedback Loops
  - Creating feedback loops to Customize SLA / SLO for systems in different categories.
- Confidently breaking pipeline
- Using team's OKR to:
  - Burning down regularly the Backlog of vulnerabilities
  - Slack and ticketing system used actively to deliver vulnerabilities resolution to teams
  - Measuring team performance & feeding it to higher management
- Product owner report on vulnerability resolution and risk



- Risk-based approach to vulnerability burndown
- Scheduled/Regular Pentest, assessment
- Vulnerabilities fixed when and if discovered
- Automatic asset management composition driven by either the vulnerability scanners, CI/CD, cloud services
- Measured Vulnerability management process
    - Start measuring and feedback the Mean Time to Resolution (MTTR) and Mean Time Open (MTO).
    - Measuring Vulnerability timelines

# Part 3 - Data-Driven Approach on Vulnerabilities SLAs, SLOs How to improve measurement with Data and OKRs




## Elements to consider when fixing SLA

Applying patches and mandating SLA seems straightforward. Still, many factors influencing, fixing and resolving vulnerabilities are a bit of a learning curve.

Reasons for such a complex landscape are various, for example,

- Systems are composed of a number of components that grow consistently and exponentially.
- Software is even more complex than a hosting system with multiple realms (code, open-source libraries, proprietary library, licensed libraries)
- Different layers of patches (application, O/S, Drivers, or container layers).
- Testing required, and several teams/clients involved in testing.
- Number of teams involved in the fix, Time zones for fixes, Release Cycles

- Exposure to clients and criticality of systems (Downtimes allowed).
- Maintenance windows, Operational SLA, Downtime allowed
- Context-based situational awareness and understanding

Fixing vulnerabilities in the various systems should not be delegated to just one team but rather have an objective and measurable target to discuss in the context of risk.

Some of the elements above can be measured in various ways. Some of that measurement could also be automated, with a degree of complexity directly related to the organization's size, if not exponential.

Every system has different contracts and requirements. Nonetheless, those can be fed as business requirements for SLA and determine the bucket or categories of the system for SLA/SLO/OKR.

Regardless of the methodology and target like SLA enables the organization to measure and drive adoption.

Usually, systems are categorized into the following categories

- **Critical system** - uptime is critical, and windows for updating are tight
- **Medium criticality** - uptime is essential, but windows for an update are more relaxed
- **Low criticality** - downtime update windows are very flexible.

The following list highlights several elements to consider for patching and adjusting the time for SLAs, SLOs and writing OKRs. It must be noted that whilst this list is a helpful guideline, it is by no means an exhaustive list of topics.

Locality and business context are absolutely critical when prioritizing those vulnerabilities

**Patching / Infrastructure Vulnerabilities**

- SIMPLE - Patch (easy) upgrade with simple testing, maintained by only one team
- SIMPLE/MEDIUM - Patching a system with some testing requirements maintained by one or two teams
- COMPLEX - Complex system with multiple dependencies and configurations
- VERY COMPLEX - Complex system with multiple applications and multiple teams maintaining different part and vast client surface

**Fixing Cloud Misconfiguration/ Vulnerabilities**

- SIMPLE - Updating a system rule in a register, settings in a cloud element (e.g., S3)
- SIMPLE/MEDIUM - Changing the role, IAM rule, Permission rule
- COMPLEX to VERY COMPLEX - Restructuring the architecture of service, introducing controls, changing access methodologies, changing settings that involve user interaction

**Fixing Containers vulnerabilities**

- SIMPLE - Library or dependency without major impact
- MEDIUM - Changing a critical library or OS (operating Systems) that potentially breaks dependencies
- COMPLEX - Updating container image with multiple dependencies and different running containers utilizing the image
- VERY COMPLEX - Updating a container's image that is utilized widely across the organization e.g., Linux, with the deprecation of a function utilized by one or multiplied systems

**Updating Libraries**

- SIMPLE - Updating a library without critical dependencies
- MEDIUM - Changing a critical library that potentially breaks dependencies on one application
- COMPLEX - Changing a library from minor to major versions, with deprecated libraries that require swap of those functions in multiple parts of code
- VERY COMPLEX - Similar to Complex but when the number of teams is multiple and distributed

**Update Code/ Bug Fixing**

- SIMPLE - Changing a simple variable, function with a minor regression testing
- MEDIUM - Changing a function or a section of code with medium impact in the individual file or limited number
- COMPLEX- Updating a major section of the code, changing inputs from one or multiple user perspectives that requires extensive testing

# Setting SLA - Some examples and guidelines

Setting SLA as illustrated can be complex. SLA and SLO are down to the various organizations. We will explore some tactics and examples with advantages and disadvantages.

## Guidance

Setting SLA should start with setting the objectives to meet. This can further on be a useful method and tool to further expand security OKRs

**Objective #1: Improve the efficiency of the vulnerability assessment process**

Vulnerability Assessment: It is a systematic review of security weaknesses in an information system. It evaluates if the system is susceptible to any known vulnerabilities, assigns severity levels to those vulnerabilities, and recommends remediation or mitigation if and whenever needed.

**Objective #2: Improve patch management process**

Patch Management: It is the process of distributing and applying updates to applications. These patches are often necessary to correct errors (also referred to as “vulnerabilities” or “bugs”) in the software.

We will explore more Security OKR with the key metrics in the section below.

Work with Dev Team to establish and agree upon metrics they’ll be measured against.

Being transparent on how the metrics are being calculated and measured is key.

Protecting the integrity, as well as the transparency of the metrics, is crucial to keep those metrics trustworthy.

SLA and metrics that are not precise or too variable are ignored.

Ultimately the collaboration aspect is key, and the agreement shall be based on data and metrics all agree.

Risk-based metrics build vs fix parameters and similar measurements result in the best-established discussion and objective to define the dev teams’ OKR and objectives.

## Examples of SLA

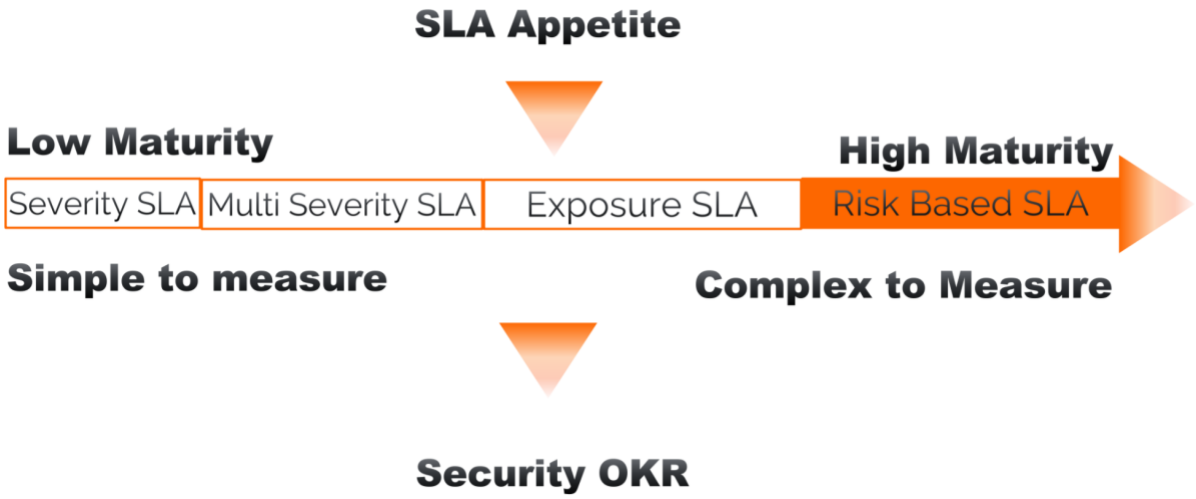



Image 13 - from SLA to security OKr

## Setting SLA/ SLO based on vulnerability severity

A good reference measure of SLA for software and vulnerability resolution can be found in Palantir blog

> For vulnerabilities on the underlying infrastructure, containers, or hosts, we adhere to the following maximum SLAs:
>
> - CRITICAL: 72 hours
> - HIGH: 30 days
> - MEDIUM: 90 days
> - LOW: 120 days
>
> For vulnerabilities in Palantir-developed software products, which may be significantly more complicated to remediate, we adhere to the following maximum SLAs:
>
> - CRITICAL: 30 days
> - HIGH: 30 days
> - MEDIUM: 90 days
> - LOW: 120 days

### Vulnerability Severity Based SLA

The most common level and objective when fixing vulnerabilities at an early stage.

The SLAs are usually set to Different levels of vulnerability criticality

- Critical severity vulnerabilities fixed in X amount of time /days
- High severity vulnerabilities fixed in X amount of time /days
- Medium severity vulnerabilities fixed in X amount of time /days
- Low severity vulnerabilities fixed in X amount of time /days[8]

| Advantages | Disadvantages |
|---|---|
| - Simple to set<br>- Initial starting point for discussion<br>- Easy to embed with scanners | - Too simplistic<br>- Sometimes teams can't achieve the targets<br>- Can cause frictions |

## Asset Criticality Based SLA

Those SLAs are a bit more sophisticated and rely on a different level of criticality of the assets.

Conversations with different teams have surfaced with confusion when working with different SLA levels.

The SLA is usually set to Different levels of vulnerability criticality

**Critical Services**

- Critical severity vulnerabilities fixed in X amount of time / days
- High severity vulnerabilities fixed in X amount of time / days
- Medium severity vulnerabilities fixed in X amount of time / days
- Low severity vulnerabilities fixed in X amount of time / days (caveat most organizations don't get around fixing low or medium severity vulnerabilities)

**Non-Mission Critical Services** - General speaking more time to fix vulnerabilities

- Critical severity vulnerabilities fixed in X+Y amount of time / days
- High severity vulnerabilities fixed in X amount of time / days
- Medium severity vulnerabilities fixed in X amount of time / days
- Low severity vulnerabilities fixed in X amount of time / days

| Advantages | Disadvantages |
|---|---|
| - Medium Complexity | - Multi SLAs can be confusing |

[8] Low and Medium might not even get addressed. A low vulnerability or a medium that is not exploitable might not get or need the attention required

| | |
|---|---|
| - Take into consideration asset criticality<br>- Multiple level prioritize work on what's most critical | - Sometimes teams can't achieve the targets<br>- Requires more information on asset<br>- Difficult to keep consistent<br>- Requires middleware to enrich scanner data |

## Exposure Based SLA

The SLA are based on exposure of asset, usually assets that are externally facing. The exposure level is generally more complex to measure and relies on asset management accuracy or some form of tag-based strategy in container and cloud

| Advantages | Disadvantages |
|---|---|
| - Medium Complexity<br>- Take into consideration asset criticality<br>- Multiple levels prioritise work on what's is most critical | - Multi SLA can be confusing<br>- Sometimes teams can't achieve the targets<br>- Requires more asset information<br>- Difficult to keep consistent<br>- Requires middleware to enrich scanner data |

## Risk-based SLA

Risk-based SLAs are more sophisticated and rely on composite metrics to set targets. Those SLAs are the best to use but also the most complex to implement

- ➤ Risk Triage SLA = This SLA provides the agreed time on how long it should take to triage a risk and accept/reject it.
- ➤ Risk SLA = This SLA provides the agreed time on how long the risk should be in the risk status – accepted, signed off (Maximum Risk time)

| Advantages | Disadvantages |
|---|---|

| | |
|---|---|
| - High granularity<br>- Take into account different factors<br>- Risk can be based on probability of exploitation<br>- Risk can be based on asset criticality | - Those SLA are not the most intuitive<br>- SLA Can change based on the landscape change<br>- Same vulnerability can have different SLA based on the deployment<br>- SLA can change while in work (rare but to consider)<br>- Requires middleware to enrich scanner data |

### Additional Considerations

When setting SLA sometimes no one rule fits all. Is important to keep in mind that settings SLA shall serve a purpose. Driving metrics and resolution time down or keeping resolution time consistent.

## The Security OKRs

A good indication of maturity for the organization is the adoption of security OKRs for development teams. OKRs are a good way for the team to self-regulate and create an objective to strive for and achieve at every sprint.

Security OKRs should consider the SLA or SLO to create more feasible metrics.

SLA Timing should be consistently reviewed and re-assessed to account for the difference in organization priorities or changes in complexity that could impact the resolution time.

An example is a system that globally takes ten days to reboot, and a resolution time below ten days for vulnerabilities would be difficult to achieve.

Please note the key metrics and the example below act only as guidance


Image 14 - SMAR OBJECTIVE Definition

Note the OKR review and time for measurement should be consistent and frequently reviewed

OKr Should be:

- S — Specific.
- M — Measurable.
- A — Achievable.
- R — Relevant.
- T — Time-bound.

## Examples of OKRs

### Patch Management OKR:

**Objective:** **Improve patch management process**

Patch Management: is the process of distributing and applying updates to applications. These patches are often necessary to correct errors (also referred to as “vulnerabilities” or “bugs”) in the software.

Key Metrics:

- Reduce the number of critical vulnerabilities from X to Y in X sprints.
- Meet the SLA for resolution time from X time every month.
- Decrease the Mean time to Resolution from X to Y days.
- Reduce the Mean time to patch from X to Y.

Key Metrics Examples

- Decrease the average mean time to patch (MTTP) from 90 days to 60 days

- Increase the percentage of systems that have the latest OS or application patches installed over time from 90% to 95%
- Improve the efficiency of automated patch deployment process from 50% to 75% to ensure patches are automatically deployed based on the deployment policies

## Code Vulnerability Management OKr

**Objective: Increase the security Quality of the code**

Bug Fixing: Is the process of identifying and correcting code or upgrading libraries. These software remedies are often necessary to correct errors in code (also referred to as "vulnerabilities" or "bugs").

Key Metrics (examples):

- Reduce the number of critical bugs from X to Y in X sprints.
- Meet the SLA for resolution time from X time every month.
- Decrease the Mean time to Resolution from X to Y days.
- Decrease the number of security bugs for each release from X to Y.
- Reduce the Mean time to resolution from X to Y.

Some of the elements that can influence the timing and description of SLA/SLO are:

- Change in complexity.
- Reconfiguration of teams.
- Increase workload.
- Changes in priorities.

## Vulnerability Assessment OKR

**Objective: Improve the efficiency of the vulnerability assessment process**

Vulnerability Assessment: It is a systematic review of security weaknesses in an information system. It evaluates if the system is susceptible to any known vulnerabilities, assigns severity levels to those vulnerabilities, and recommends remediation or mitigation, if and whenever needed

Key Metrics:

- Decrease the number of undetected intrusions attempts within a given period from 2% to 0%
- Reduce the number of unidentified devices on a network at any point of time from 5 to 0
- Reduce the number of spoofing attacks – IP Address Spoofing, DNS Spoofing, HTTPS Spoofing – from 200 to 10 per day

- Decrease Number of False positive requests
- Decrease the number of re-open tickets

# Risk Considerations

## Challenges to calculate a meaningful number for risk

Why is it so challenging to quantify the complexity of a system/organization?

Currently, risk and severity are two words used interchangeably when they should not. Severity is a non-actualized risk, while risk expresses not only the potential of a risk to manifest but also the impact and the probability of it manifesting.

Let's look at the risk element of vulnerability management aspect throughout the whitepaper, risk levels for vulnerabilities are often misunderstood and difficult to contextualize due to the complex nature of how we build modern applications and the complexity of tracing where they are run.

An organization that wants to mature to a higher level should be considering risk, not severity, of individual vulnerabilities.

Vulnerabilities need to be translated into risk, but how? The answer is context and business information. When associating the assets' information with business context (like criticality) and environmental context (like if an asset is exposed to the web), the risk of a vulnerability to materialize and causes a specific impact can be calculated.

While risk is in direct relation to its impact, the time and complexity of patch implementation or bug fixing could affect risk but is challenging to quantify. A precise SLA is complicated to define and measure due to the sheer volume of vulnerability data originating from multiple phases of the software development lifecycle.

Some of the metrics that we could consider in calculating the resolution time are impacted by the complexity of the organization, the number of users affected by the change, and the length of test required for the change.

## What's wrong with risk as it stands

Following some of the elements and the challenges that risk in an application and cloud security has:

- Risk as it stands is not dynamic and is often just severity from CVSS.
- Real-Time - Risk scoring is traditionally static or on a sheet assessed and updated based on impact and probability analysis, but those values are often arbitrary and not quantifiable. Modern organizations have a dynamic landscape where the risk constantly changes every hour
- Contextual - CVSS score provides a base severity that needs to be tailored to the particular environment where that vulnerability appears, but is complex to determine manually
- Threat intelligence - usually analysis of threat intel is left to analysis that need to look at reports and individual vulnerabilities to extrapolate risk.
- Not Quantifiable - Risk that is based on data and in real-time is quantifiable. Risk assessed manually by individuals based on opinion is qualifiable.
- Not Actionable - risk scoring and tracking is usually done offline or on a sheet of excel and reviewed quarterly. That can lead to confusion

Is challenging for business to understand purely CVSS severity. For the above reason  we are currently facing a language barrier and conflict between business not prioritizing security, development team not being given time. A vulnerability management programmed not based on risk is deemed to be challenge. Security and business should communicate in a risk-based scoring that enables the business to make informed decisions. Engineering teams should be provided as well with a number of actions to take in order to meet risk expectations

## A new and improved way to calculate risk

In order to calculate a better risk and move from pure CVSS to a more actionable risk we need to consider several elements. Risk calculation in security has been mostly qualitative so far, with the CVSS calculator being the only quantitative analysis of security. The element below of an improved risk formula calculation make the case of more quantitative and less qualitative risk analysis. The elements considered in the risk formula will be Probability of exploitation driven by a number of factors, context and impact analysis.

Let's look at the definition of risk:

**Risk = Probability (Likelihood of exploitation, Locality)  * Severity * Impact**

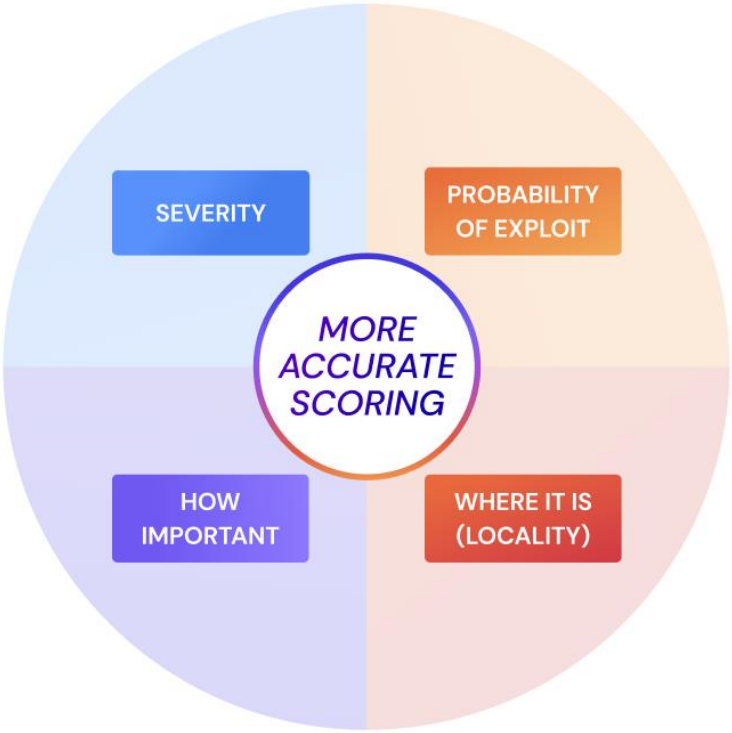



Image 15 - Elements used to calculate Risk

**Contextual aspects are based on:**

- Severity of a vulnerability - how much a 3rd party vendor has declared that vulnerability to potentially be dangerous
- Probability of exploitation - how likely is that vulnerability to be exploited
- Locality is a factor of the probability of exploitation
- Impact (also known as a factor of the Business Impact assessment) that communicate how much damage a vulnerability could cause to the organization

Risk-based threat assessments  are usually done by security professionals, but this is an impossible job due to the increase and variable factors that must be considered.

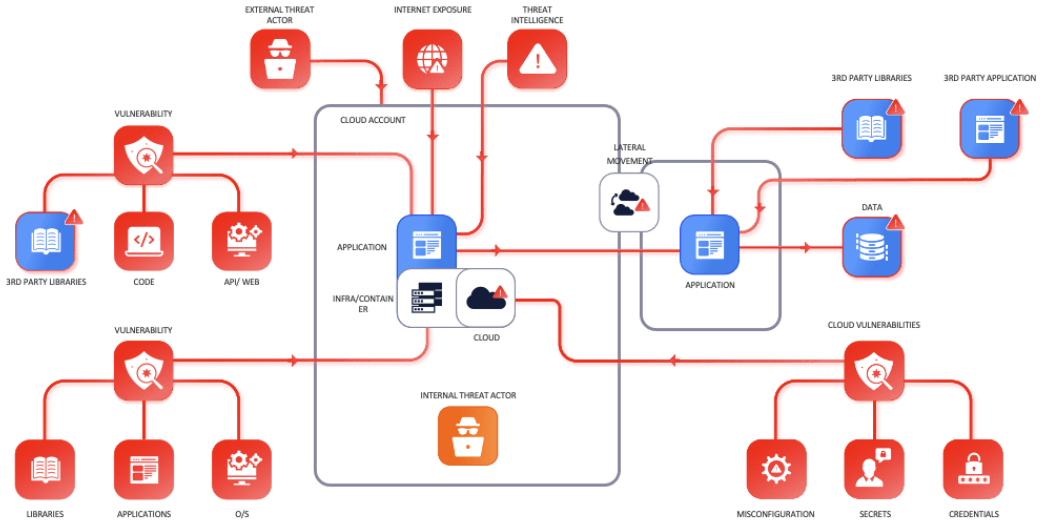



Image 16 - A topography of IT components in a typical enterprise

Following a list of elements security professionals need to consider when triaging and deciding which vulnerabilities to fix first :

The following questions should be considered when triaging which vulnerabilities to prioritise and remediate:

- How is the application is being built
- Where is the application deployed (which network, which environment)
- What kind of data does the application process
- How many the components are external or internal
- Where are the encryption keys stored
    - Are there any misconfigurations in the storage system
- Is there any threat actor group targeting a specific vulnerability / system
- What are the vulnerabilities of the source code, libraries, and APIs, containers, and infrastructure that the application is building
    - What is the blast radius if one of those components get compromised

**The following elements can be considered when calculating the risk elements**


Image 13 - from basic CVSS to Modern Risk

**Probability of exploitation**

- Severity of a vulnerability (CVE, CWE, CVSS and CWSS)
- Locality of an asset also known as Context
- Exploitability of a vulnerability based on availability of Proof of concept or code snippet
- Probability of an attacker to target the vulnerability
- Active exploitation of the vulnerability from threat actors' groups
- Discussion in Twitter, LinkedIn, Reddit and other forums
- Freshness of vulnerability (in first 40 days vulnerability are exploited/target more frequently)

**Impact on system**

- What data is being processed by the system
- How many users are accessing the system or could be impacted
- How much revenue could be impacted if the system is unavailable and for how long
- Contractual impact - how much damage/ credit clients need to be compensated for a failure in a system
- Brand image damage - how much new business
- Stock/Share price damage - how much a public disclosure affects the trust of the stock market (in direct relationship with

We written recently a whitepaper that expands on this problem: Vulnerability management prioritization in cloud and application security

# Conclusions

In this whitepaper, we saw the metrics that matter, the timeline and evolution of vulnerabilities and the maturity score elements of vulnerabilities.

With organizations ever-changing, there is no one size fits all; we hope that this whitepaper has provided a perspective and an informed view of the complex process that is measuring vulnerabilities in various landscapes.

The SLAs or rather SLOs, are a good starting point to establish guideline principles. SLO should be assessed frequently and used by the organization to measure the target for each team. SLAs should be agreed upon as an organization and captured in service agreements with clients.

Any organization that starts with SLO or, better, OKr will help the development teams to understand objectives. Those objectives should be used as tools for discussion, and resolution should be in line with the risk discussion of various teams.

The important factor is that the vulnerabilities get consistently reduced week on week. This generates a healthy habit for development teams to fix and achieve consistent objectives sprint on sprint.

Those objectives should be reported and fed back in loops to application owners and the security team to steer the SLO/A and improve the objectives.

The recommendation is to have a risk-based or metric-based conversation with the various teams.

A very powerful change force gets generated when an organization's security teamwork in conjunction with the development teams and product owners.

Where before, the objectives from security were fuzzy due to scaling and communication issues working with objective shifts the organization and security left but also upward.

When all of the teams in organizations have reached a higher maturity level, then it is time to switch to more trust-and-verify models where engineering teams can confidently release software and fixes knowing the timelines for fixing the vulnerabilities.

Fixing vulnerabilities in code, patching servers, and running new containers, can be mandated with SLA, risk based OKRs guidance's or other methodologies.

A risk-based approach is often recommended as a good risk formula can incorporate several elements that are often used in triaging.

Risk is also a well-understood metric across the business and helps non-security professionals to make decisions.

# Reviewer & Contribution

Authors

- Francesco Cipollone - Appsec Phoenix CEO & Founder

**Peer Reviewer**

- Chris Romeo - Chief Security and Strategy Officer - Security Journey
- Brook Shoenfield - independent
- Richard Grey - CISO FreeAgent
- Ruchira Gupta - Director of application security - Box
- Larry Maccarone - Contrast Security
- Amanda Alvarez - Sr. DevSecOps Engineer - Trace3

## How to Scale metrics and augment observability - Phoenix Platform

For medium and large enterprises, strategically prioritizing assets is key. Addressing all aspects of vulnerability management, application security, cloud security, critical infrastructure, and considerations for legal and regulatory compliance can be daunting and could lead any security professional to be overwhelmed with tasks and information.

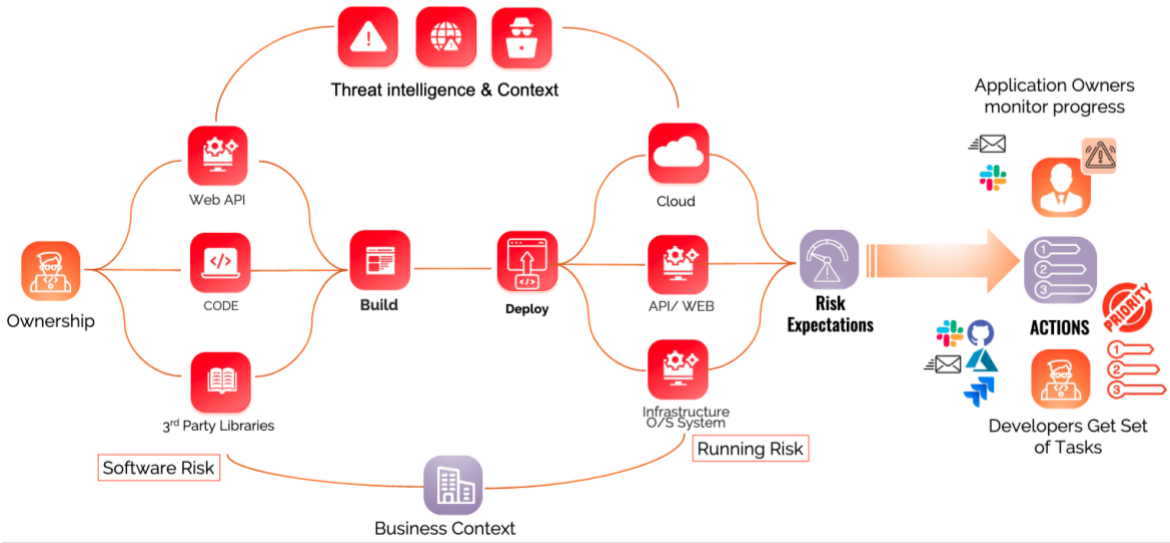
Security and development teams face significant burdens with a large number of vulnerabilities and triaging and prioritizing vulnerabilities manually is a complex process that can lead to inconsistent decisions and impact the resulting degree of IT security assurance. The consequences of ineffectively implementing a Vuln Management program can be dire to an organization's operational resilience.

Measuring and maintaining metrics that matter and running vulnerability management and application security programmed is complex.

Appsec Phoenix comes to the aid of empowering business calculation of SLA and SLO, setting risk-based targets and SLA that gets automatically translated in work to execute.

With Phoenix platform you can set risk-based metrics that matters, automate triage, prevent burnout.

metrics that matter to them and overall build a positive relationship with the various development teams.




Phoenix is on a mission to help organizations and security teams move from addressing individual vulnerabilities into consolidated and actionable risk reduction. AppSec Phoenix is a next-generation Security Orchestration and correlation (ASOC) and Risk View System allowing organizations to manage IT security operations in a data-driven/risk-based way.

Phoenix supports Vuln Management program execution with technology that provides both high-level overview and low-level insights on what to focus on and aids security professionals to set appropriate and achievable priorities. Our solution properly coordinates vulnerability scanning activities with threat intelligence and synthesizes data into a structured information system, ensuring that a Vuln Management program is accountable and stays on track.


The Phoenix system provides Application & Cloud Security Posture Management (ASPM), Application Tooling Orchestration (ASTO), and prioritization at scale leveraging up to 15 data sources.

## Risk Augmentation / prioritization


Any good Vuln Management program needs to have executive sponsorship and a consistent feedback loop to enable executive risk/data-based decision-making that is in line with the risk management goals of the organization.

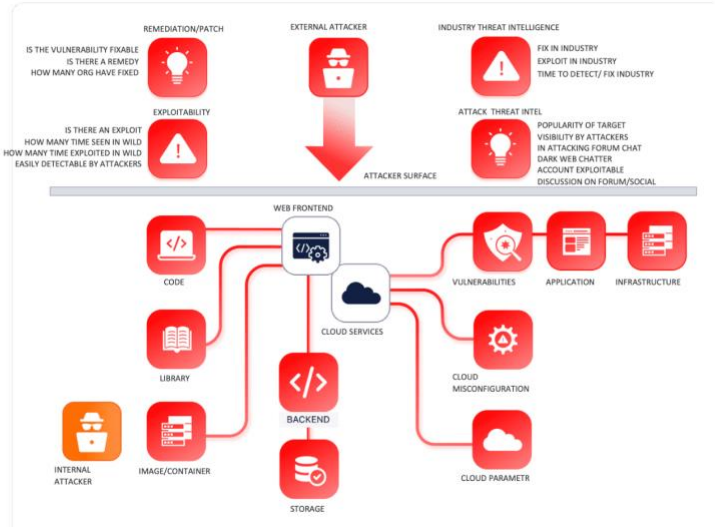
### WHAT WE DO?

- Appsec Posture management & Cloud Security Posture Management
- Quantification of Exposure and potential monetary loss
- Selection on what to fix first

### WHY IS HARD?

How can a security/dev decide without insights?

- Asset Management & inventory
- SBOM : Software bill of material
- Attack Simulation & Attack Vector
- Cloud Misconfiguration
- Container Misconfiguration
- Infrastructure & Application Vulnerability
- Network Exposure, Internal and External actor




ACT now on vulnerability get a demo at https://appsecphoenix.com/request-a-demo/

PHOENIX SECURITY

ACT today on vulnerabilities
ACT today on RISK

ACT today with Phoenix Security

ask@phoenix.security
www.phoenix.security
+442031953879