rezilion

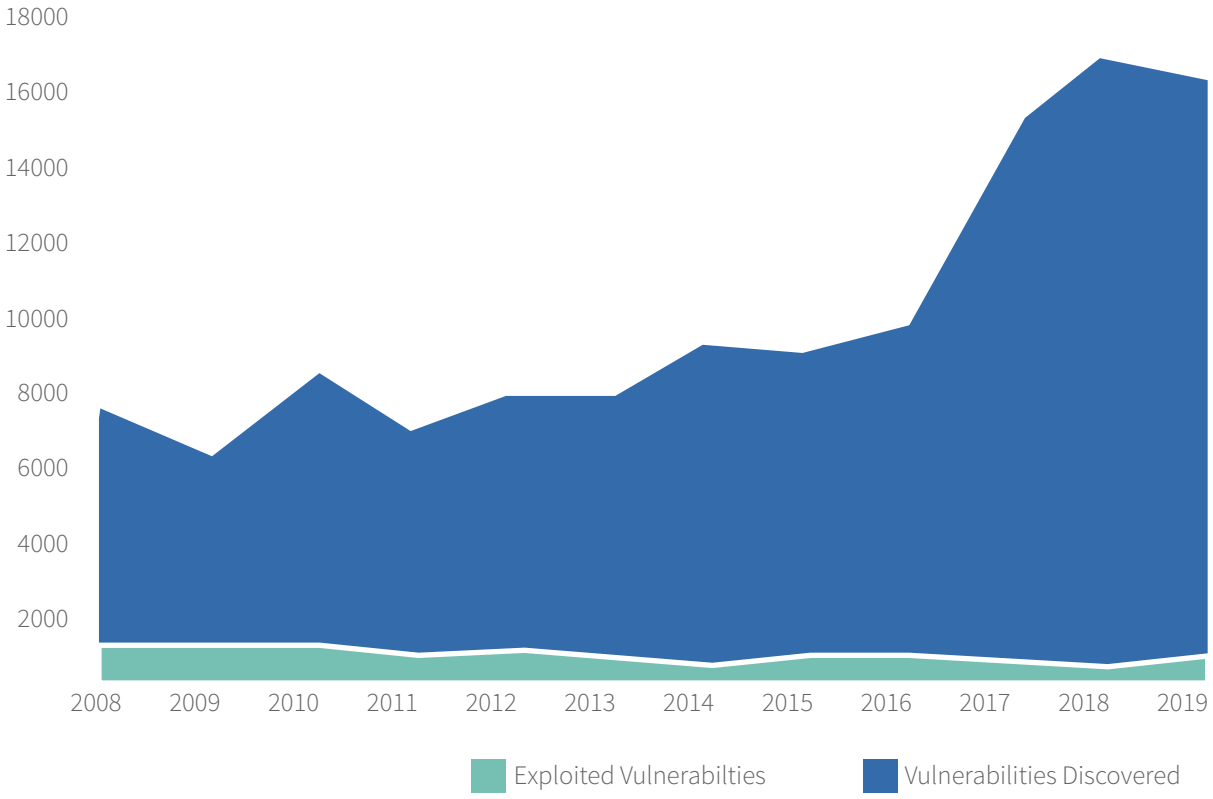
Runtime Memory Analysis: A Better Way Forward for Vulnerability Management

According to analyst firm IDC, large-to-very large enterprise companies are spending 7-10% of their security budget on vulnerability management.[1].

Security teams are inundated by vulnerability scanners that overload them with voluminous scan results. In order to cope, they prioritize vulnerability remediation using antiquated best practices and limited data, which provably do not reduce their risk or attack surface. In fact, a recent RAND Corporation analysis found no notable reduction of breaches in organizations with mature vulnerability management programs.[2]

Firms with good security posture are equally breached by known vulnerabilities as those with poor security posture.

Vulnerability management firms often point at the multiplicative growth of CVEs as a proofpoint for increased investment in their tools, but there is no correlation between CVE sprawl and breaches due to vulnerability exploitation. In fact, an analysis of the IBM X-Force Vulnerability Database[3] reveals that although there were 16,000 new vulnerabilities discovered in 2019, less than 800 led to exploitation. This paints a murky picture of the enterprise attack surface.



[1] Worldwide Security Spending Guide, IDC
[2] Improving Vulnerability Remediation Through Better Exploit Prediction, RAND Corporation
[3] Source: https://exchange.xforce.ibmcloud.com/

What is the best approach for vulnerability prioritization?

A popular prioritization methodology is the Common Vulnerability Scoring System (CVSS) — an open framework for communicating the characteristics and severity of software vulnerabilities.

CVSS consists of three metric groups: Base, Temporal, and Environmental. The Base metrics produce a score ranging from 0 to 10, which can then be modified by scoring the Temporal and Environmental metrics.

However, a vulnerability is only as dangerous as the threat exploiting it, and 95% of vulnerabilities with "high severity" CVSS scores have never been seen in the wild nor linked to breaches.[4] This means that assigning a global critical/high/medium/low rating to any vulnerability is flawed because:

- Attackers don't care what threat score a vulnerability has and regularly exploit lower-ranked vulnerabilities if they're the easiest successful attack vectors.
- The known quantity and explosive growth of identified vulnerabilities makes it impossible to remediate them all.
- Not all vulnerabilities have patches or can be patched.

It is the very nature of the CVE process that is itself inhibitive to proper risk rating due to the fact that often, even when there is a known vulnerability in a software package the vulnerable components are often never actually executed and thus represent no risk of exploitation.
Capsul8 recently made some headway along this thinking in their analysis of Linux hardening when they concluded:

"Even in the presence of a CVE, it is not directly obvious if a vulnerability is exploitable." [5]

Heartbleed (CVE-2014-0160), one of the most devastatingly exploited vulnerabilities in the history of the internet, received a CVSS score of only 5.0/10

[4] State of Vulnerability Risk Management Report, NopSec
[5] Linux hardening in the wild, Theofilos Petsios, Capsul8

Heretofore there has been no industry standard for prioritizing vulnerabilities that reflects actual risk.

Developing a risk appetite formula requires an accounting for likelihood of exploitation, in other words, a mechanism to identify which vulnerabilities pose actual threat. Traditional approaches to vulnerability management have also exacerbated the culture of friction that exists between security and DevOps teams who often have competing deliverables: security vs. uptime.

With cloud workloads, DevOps are deploying code and infrastructure more rapidly than ever, and with this rate of change, it's critical for security teams to develop a model for managing risk across the entire attack surface. For example, in sourcing data for this report, Rezilion engineers took several snapshots from 20 of the most popular container images on DockerHub[6] that have been downloaded and deployed billions of times:

Nextcloud 100M+ downloads	Jenkins 100M+ downloads	Drupal 50M+ downloads	MariaDB 1B+ downloads	Tomcat 100M+ downloads
kibana 50M+ downloads	WordPress 100M+ downloads	elastic 1B+ downloads	redis 1B+ downloads	RabbitMQ 500M+ downloads
ghost 100M+ downloads	Memcached 1B+ downloads	PostgreSQL 1B+ downloads	APACHE HTTP SERVER PROJECT 1B+ downloads	Consul 500M+ downloads
NGINX 1B+ downloads	cassandra 500M+ downloads	MySQL 1B+ downloads	mongoDB 1B+ downloads	sonarqube 100M+ downloads

- All containers were executed using their recommended/default run settings.
- Each container was scanned for vulnerabilities using the Vuls[7] vulnerability scanner.
- A list of vulnerabilities and their respective packages was assembled.
- Memory analysis was conducted and each file loaded in memory was mapped to the package it originated from.
- Last, the list of packages that had at least one file loaded to memory was compared to the list of vulnerable packages initially enumerated, and the delta between vulnerabilities found and vulnerabilities running in each container environment was assessed.

[6] Source: https://hub.docker.com/search?q=&type=image&image_filter=official
[7] Source: https://vuls.io/

Analysis of these 20 container images found over 1,776 known vulnerabilities.[8] Patching all these vulnerabilities at once would be tedious and practically impossible. However, during the course of testing — on average — only approximately half of found CVEs were ever loaded into memory, thus not posing any threat. To vet the data, Rezilion conducted the same tests and analysis twice during a three month span.

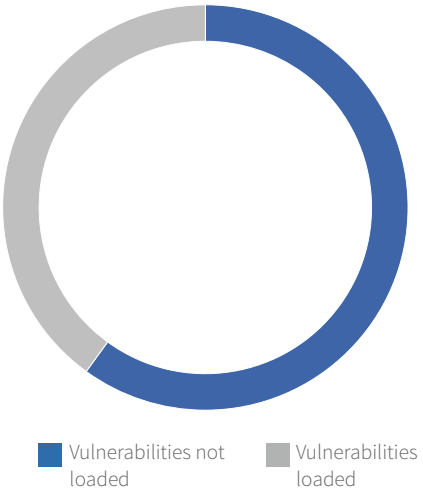
	November 2019			February 2020		
Container name	Total CVEs	Not Loaded*	% Not Loaded*	Total CVEs	Not Loaded*	% Not Loaded*
Nextcloud	198	106	54%	198	103	52%
Jenkins	139	90	65%	139	90	65%
Drupal	271	180	66%	270	174	64%
MariaDB	58	34	59%	57	27	47%
Tomcat	139	93	67%	139	93	67%
kibana	38	34	89%	39	30	77%
WordPress	200	110	55%	198	105	53%
elastic	35	28	80%	36	24	67%
redis	40	24	60%	41	24	59%
RabbitMQ	29	14	48%	36	17	47%
ghost	95	62	65%	41	23	56%
Memcached	45	25	56%	46	24	52%
PostgreSQL	68	25	37%	68	22	32%
APACHE HTTP SERVER PROJECT	61	34	56%	62	34	55%
Consul	2	2	100%	1	1	100%
NGINX	71	44	62%	72	44	61%
cassandra	141	89	63%	141	89	63%
MySQL	82	49	60%	82	49	60%
mongoDB	44	13	30%	54	25	46%
sonarqube	55	32	58%	56	33	59%

* Not found in memory during the course of testing and analysis.

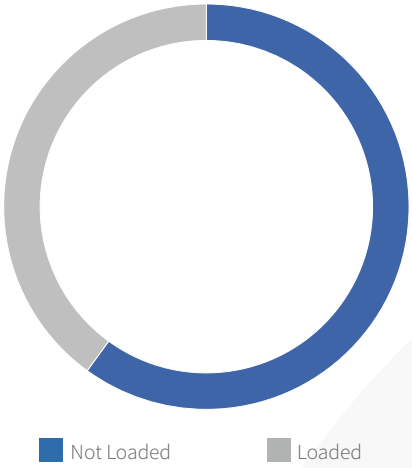
[8] Vuls identified 1,811 CVEs in the November 2019 sample, 1,776 CVEs in the February 2020 sample

During the course of testing and analysis:

60% of vulnerabilties were never loaded.



71% of CVSS "high severity" vulnerabilities were never loaded.



In the February baseline, 67% (152 out of 225) of vulnerabilities with "high severity" scores were never loaded into memory, and in November that percentage climbed to 75% (179 out 236 "high severity" vulnerabilities). If one were prioritizing vulnerability management based on CVSS scores, they would run the risk of spending upward of 70% of their time and effort on vulnerabilities that posed no risk to their production environment.

Detection of installed vulnerable components does not edify which code parts utilize them—akin to testing for operating system dependencies that inform which vulnerable libraries are installed, but cannot tell which apps are actually using these libraries. Monitoring which libraries are actually loaded in runtime is the right approach to successful vulnerability prioritization.[9]

[9] Note that source code scanning is not a viable approach to solving this problem in that the method can't know what components will end up being used once in production, and recipes often pull in a library not specified in a manifest file or integrate external third party components as part of the build.

Gartner posits that modern vulnerability management requires "continuous adaptive risk and trust assessment"[10] or CARTA.

The first phase of a CARTA-based approach to vulnerability triage is:

- Identifying business importance and criticality for services in production.
- Identifying those vulnerabilities that are actually running in them.

Followed by:

- Prioritizing treatment of loaded vulnerabilities that do not have defenses or compensating controls such as anti-exploitation, intrusion detection and white-listing tools.
- Prioritizing treatment of loaded vulnerabilities commonly targeted by exploit kits, malware, ransomware and threat actors, while also considering asset criticality and external exposure.

Vulnerability prioritization helps guide mitigation efforts by providing a reliable triage mechanism.

Implementing a risk-based approach to prioritizing vulnerability remediation focuses efforts on vulnerabilities, for which imminent threats prevail. This reduces risk, friction, and—by channeling remediation efforts at vulnerabilities that represent true risk—also reduces overhead.

This approach will result in a reduced attack surface and will provide “breathing room” for additional patch installation. Adopting a CARTA strategic approach to vulnerability management enables security teams to:

- Use more context and runtime visibility for continuous, adaptive risk-based decision making, rather than static “score”-based or manual policy driven binary “allow or block” security decisions.
- Enable their risk management teams to move beyond risk management and compliance checklists to real-time contextualized security control decisions.
- Implement autonomous compensating controls and mitigations.
- Provide continuous risk visibility feedback to DevOps and business units to adjust acceptable risk levels and controls as necessary.

[8] Implement a Risk-Based Approach to Vulnerability Management, Gartner