Red Hat

Red Hat Product Security risk report 2022

Introduction

The year 2022 was an incredibly transformative year for the Red Hat® Product Security organization. This year marked the retirement of Mark Cox from Red Hat. Mark was the founder and original leader who started this team 21 years ago in 2001. His retirement is a symbolic event of note in our coming-of-age story. Though Mark left the Product Security team some time ago, his impact and legacy played a tremendous role as he passed the torch to our current VP, Vincent Danen.

Red Hat Product Security underwent several changes last year. 2022 was our first full year of operation, where our Product Security Incident Response Team (PSIRT) no longer functions as our primary and largest team. While the work of our PSIRT remains absolutely critical to the company and our customers, and reflections on vulnerability response still fill the bulk of this report, we will also delve into some of the statistics and reflections of our other work, notably as the stewards of Red Hat's secure development life cycle (SDL) implementation.

Much of this was not new work. PSIRT engineers and program managers who handled both vulnerability response program work and additional security architect work or smaller specialist teams had already laid a foundation for this implementation. 2022 was a year of significant change in focus and structure. We saw the empowerment of dedicated leadership and owners of specific SDL-aligned functions, allowing us better clarity on specific goals.

In some ways, you could say Red Hat Product Security has grown up.

In the first section of this report, we share some of the summary statistics from our PSIRT operations. After that, we thoroughly examine the four major incidents. To summarize, we look at some statistics and insights from our work beyond vulnerability response and conclude with a final word from our leader, Vincent Danen. As always, we hope the information we provide here and our continued dedication to transparency in our work serve our customers and the community with valuable insight. Enjoy.

– Garth Mollett / Product Security Lead Architect

Red Hat Security Advisory (RHSA)

Red Hat Security Advisories have been the primary notification mechanism for releasing a software update containing a CVE fix for the last 20 years. While the format and other details have evolved over the years, the primary purpose remains the same.

The total amount of Red Hat Security Advisories released this year broke our previous high of 1,621 in 2020, reaching 1,635 advisories in 2022. This number contributes to a similar trend we have seen over the last three years. The minor fluctuations here are to be expected and reflect the amount of changes in supported products and components.

Table 1. RHSA by severity

Total RHSA	1635	+232 from 2021
Critical RHSA	35	-10 from 2021
Important RHSA	892	+184 from 2021
Moderate RHSA	629	+43 from 2021
Low RHSA	79	+15 from 2021

Common Vulnerabilities and Exposures (CVE)

The 2022 numbers above were similar to the 2021 numbers, with only minor fluctuation. The changes we see are more likely reflective of the evolution in our product portfolio than any kind of meaningful trend in the threat landscape. As with previous years, the vast majority of security flaws were rated Moderate. Although, as seen in the RHSA table above, they did not make up the majority of fixes. This is because a significant amount of these flaws do not present a risk that warrants backporting and are often not fixed outside of major updates, which are less frequent. The lower risk of these Moderate severity flaws is reflected in the very low percentage of reports of exploitation in the wild compared to the higher severity flaws.

Table 2. Overview of security flaws by severity rating

Severity rating	Flaw count (+/- from 2021)	In the wild exploitation (% of total | +/- from 2021)
All	1656 (+82)	7 (0.4% | -19)
Critical	19 (+8)	2 (10.5% | +1)
Important	276 (-2)	3 (1.1% | -1)
Moderate	1086 (+56)	2 (0.2% | -16)
Low	275 (+20)	0 (0.0% | +0)

Response times

One of the most important metrics for many of our customers, in terms of both risk exposure and standards compliance, is how swiftly we are able to get fixes out. While exploitation of "zero day" vulnerabilities that are unknown to vendors or the public remains a real threat from the most well-funded threat actors, the biggest threat from vulnerabilities occurs in the time between the details of a flaw being publicly disclosed and when patches are applied. Before patches can be applied, vendors must first create, test, and release them. In the most crucial cases, every day matters, so Red Hat strives to keep these numbers as low as possible. In table 3, we show the average response time from disclosure to errata for each severity level.

Red Hat

Table 3. Risk exposure time period comparison by severity

Severity	2021	2022	Change
Critical	11 days	6 days	45.5% faster
Important	42 days	36 days	14.3% faster
Moderate	151 days	108 days	28.5% faster
Low	205 days	168 days	18.0% faster

Per product breakdown

The three major versions of Red Hat Enterprise Linux® (7-9) account for the vast majority of Important and Critical errata, which is expected given both the sheer size and complexity of the platform, as well as included software being primarily written in C. It is worth noting that the middleware family of products fixed 21 CVEs in Critical errata. This is not because there was a larger number of Critical CVEs in middleware products, but it shows that middleware errata are more likely to bundle with other non-Critical CVE fixes into Critical errata.

When reading these numbers, it's worth mentioning that the CVE count is basically the amount of CVE flaws fixed in the given errata, which may not all be Critical or Important. As previously mentioned with middleware errata, lower severity CVE fixes can and do get incorporated into other releases. Many products either support multiple versions at once or related products that share a component. In these cases, the same component will be fixed in multiple errata.

Table 4. Important and Critical RHSA CVE count by product

Product	Important RHSA (CVE Count)	Critical RHSA (CVE Count)
Red Hat Enterprise Linux	659 (374)	24 (13)
Red Hat OpenShift	93 (144)	6 (13)
Ansible	7 (4)	0 (0)
Red Hat OpenStack® Platform	26 (24)	0 (0)
Middleware Products	59 (187)	3 (21)
Red Hat Virtualization	19 (56)	0 (0)
Red Hat Satellite	7 (13)	0 (0)

Common Weakness Enumeration (CWE)

An interesting data point in vulnerability analysis is less about how many vulnerabilities were addressed, but the type of vulnerabilities that they are. To highlight this, we make use of the Common Weakness Enumeration (CWE) system from MITRE.

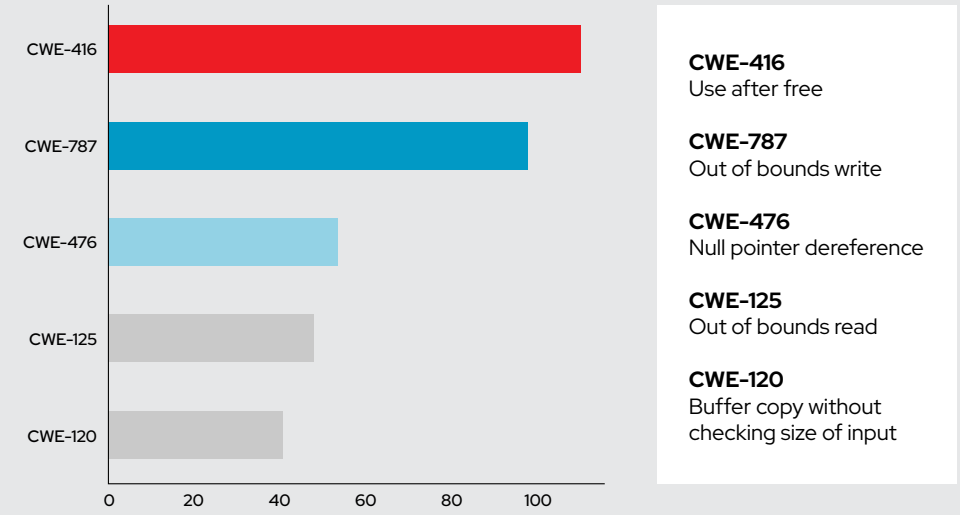

Figure 1. Top 5 CWE

This system allows us to categorize vulnerabilities based on the nature of the bug that caused it. While this system is far from perfect and modern exploit development will often combine multiple bugs together to provide usable primitives for exploitation, it provides an indication of where defensive efforts can be best applied.

Looking at the Top 5 CWE results on the previous page, what stands out is that these are classic memory safety issues. The largest contributor to this is Red Hat Enterprise Linux and code written in C, where the lack of memory safety is well understood as the price of the low-level power and performance that the language provides.

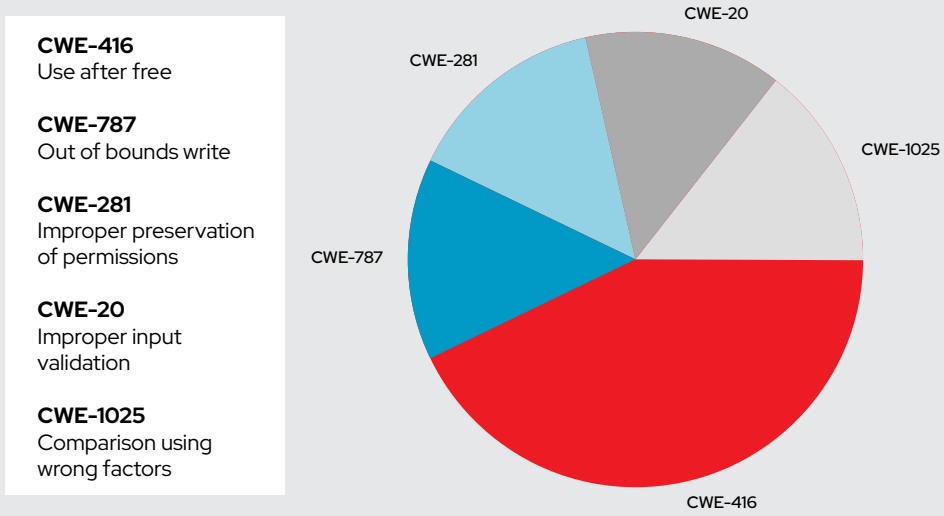

Figure 2. CWE exploited in the wild

In Figure 2, we see that CWE-416 and CWE-787 combined account for more than half of the issues that we saw exploited in the wild. Both provide fairly significant opportunities to influence the control flow of a running program. However, with only 7 flaws total and 4 between these 2 categories, this data set is not very substantial.

Major incidents

Although we saw a multitude of vulnerabilities of varying severity in 2022, as evidenced in

Figure 2 statistics, 4 incidents set off alarm bells and caused Red Hat Product Security to activate the major incident process. This process involves Red Hat immediately orchestrating efforts to provide a corporate response that addresses the issue and prioritizes remediating the vulnerability as quickly as possible upon identifying a major incident.

While Red Hat considers numerous factors to determine if an event qualifies as a major incident, the first and foremost is risk exposure. This includes perceived risk to our customers or the Red Hat brand. Security risk from trivial mass exploitation of mission-critical software ranks the highest. However, other kinds of risk, such as the risk posed by misinformation or panic from confusing or over-hyped media coverage, are also considered.

The aim of the major incident process is not just the prioritization of fixes, as that happens irrespective of whether an issue receives a Critical or Important severity rating, but also to ensure clear, calm, accurate, and well-defined communication channels stay open—both internal and customer-facing. On the customer-facing side, this is done by issuing additional artifacts such as a Red Hat Security Bulletin (RHSB), Red Hat Insights detection rules, Ansible® remediation playbooks, along with the usual security errata and CVE page entries.

This section provides a rundown of major incident events in 2022. For each event, we detail the basics, the more technical details for those who like a brief overview at the fascinating internals of how software vulnerabilities work, some statistics of interest ranging from the amount of product versions affected to a very rough estimate of time in "Red Hat associate hours", and onto how quickly we saw reports of exploitation in the wild, if at all.

RHSB-2022-001 (Polkit "PWNKit" CVE-2021-4034)

The basics: Polkit's pkexec program is a command-line program that allows local users on a Red Hat Enterprise Linux system to execute commands as another user identity (UID) based on policy definitions. In order to be able to change that UID, the pkexec program is installed with the Set User ID (SUID) bit set and the file owned by the superuser, or root. This program contained a memory safety error in the parsing of the command line. Because it runs with root privileges (required to change UID), it carried the risk that exploitable flaws in the program could allow running arbitrary code with elevated privileges and/or bypassing the policy restrictions.

The details: The execve(2) system call in the Linux kernel accepts the path name of the file to be loaded into memory and executed as an additional argument passed to the program via the command line. Tradition and standard practice dictate that the first command-line argument (argv[0] in the C API) to be the name of the program being executed. However, this is not a requirement of the kernel and is thus not enforced. Pkexec's command-line parsing code expected the command-line array (argv) to be well- formed; however, if pkexec was run with a NULL pointer value for argv, an out-of-bounds memory read occurred. While this memory read primitive alone would not usually be enough to take control of a program, the same memory location where the out-of-bound read occurs is later used to write data. As the stack memory following "argv" is almost always "env" (the program environment variables passed to execve), this allowed the parsing code to be abused in a way that allowed the rewriting of environment variables. The ability to rewrite environment variables allows unsafe values (such as LD_PRELOAD) that would normally not be permitted for a program running with SUID privileges. This would allow for arbitrary code execution with root privileges.

The statistics

Affected major product versions: 4 (Red Hat Enterprise Linux 6, 7 and 8, Red Hat Virtualization 4)

Severity rating: Important

Estimated associate time: 520 hours (approximately 13 associates involved)

Embargo time: 62 days

Time from public to first fix release: 0 days

Time from public to all fixes released: 20 days

Exploit code published: Yes, 1 day after disclosure

Exploitation in the wild: Yes, added to the CISA Known Exploited Vulnerabilities (KEV) catalog after 6 months

Estimated fix deployment: Very High

Closing thoughts: Qualys discovered this issue through excellent research and branded it "PwnKit." Qualys gave Red Hat a significant lead time on the issue, thus putting us in an excellent position to prepare our public response prior to public disclosure. While the exploitation primitives described and used in the research are somewhat novel, once they are understood, reliable exploitation is reasonably trivial. While the threat posed by the flaw is somewhat mitigated by requiring local access, polkit is a heavily deployed and used package. Exploitation in the wild would likely occur, as would at least a fair amount of media attention. In the end, media attention was limited. However, publicly available exploit code was quickly circulating with the flaw, eventually making the CISA KEV catalog. The 6-month delay from the flaw going public to inclusion on the CISA list is likely more of a reflection on process speed, alongside detection and monitoring capabilities than any real reflection on the interest in the flaw to malicious actors. Our thanks to Qualys for the lead time and collaboration.

RHSB-2022-002 (Linux kernel "Dirty Pipe" CVE-2022-0847)

The basics: A "basic" explanation of a complicated kernel flaw introduced over multiple commits is not an effortless task. Operating system kernels are generally not simple to understand by folks who do not have some operating system design and implementation theory. The Linux kernel, like most modern general-use operating system kernels, uses several kernel-based memory disk caches to speed up access to secondary storage. Files are read (paged) into one of these caches (the page cache) by the kernel and transparently made available to userspace programs. Linux contains several highly optimized mechanisms for inter-process communication (IPC) and shifting data between communication endpoints, which may be files that are currently stored in the page cache. A combination of logic errors and the use of uninitialized values in the interaction between the page cache and some of these IPC mechanisms lead to a code path where permission checks could be bypassed. Bypassing these checks allowed a normal user to write to files cached in memory that should not have been writable, including executable code. Thus a normal user could inject code into a privileged process, presenting 2 serious risks: the most concerning being that any local user could obtain full system privileges, and the other that serious data corruption could occur, even from a non-malicious user under the right circumstance.

The details: The Linux kernel pipe_buffer structure is used internally by the kernel to govern both the memory and state associated with an open pipe. The root cause of this issue is that the functions copy_page_to_iter_pipe() and push_pipe() both allocated a new pipe_buffer structure without initializing the "flags" member, allowing previous values to persist when they should not. A later commit introduced a flag value named "PIPE_BUF_FLAG_CAN_MERGE," which is used to govern if the buffers can be merged with new writes. The splice(2) system call provides a highly optimized method to move data between file descriptors inside the kernel without transferring the data via userspace. By using the splice system call to splice data between a pipe intentionally manipulated to inherit the" PIPE_BUF_FLAG_CAN_MERGE" flag and a file opened "read-only" (such as an executable binary), data could be written to page cache memory pages for the opened read-only file, thus bypassing permissions and potentially even allowing execution of arbitrary code as the root user.

The statistics

Affected major product versions: 2 (Red Hat Enterprise Linux 8, Red Hat Virtualization 4)

Severity rating: Important

Estimated associate time: 182 hours (approximately 13 associates involved)

Embargo time: 3 days

Time from public to first fix release: 3 days

Time from public to all fixes released: 7 days

Exploit code published: Yes

Exploitation in the wild: Yes, added to the CISA KEV catalog after one month

Estimated fix deployment: High

Closing thoughts: Putting the immediate risk of the vulnerability, once understood, aside for a moment, this was a really interesting vulnerability from the point of view of the inherent risk posed by our ability to understand complex systems, many of which make up security boundaries that we rely upon. This bug is an excellent example of how complex systems can fail in very unexpected ways with unexpected impacts. The development and maintenance of the Linux kernel is arguably the single most complex exercise in open source software development ever undertaken by humankind to date, both in terms of the complexity of the code and the massively distributed nature of development.

This incredible effort is almost entirely invisible to most of the world, even though its continuous operation is relied upon by everything from mobile phones and the communications infrastructure underpinning the Internet to game consoles and door bells. The security implications of the bug at the root cause of the issue described here did not manifest until a later change introduced a code path that exposed it in an exploitable fashion. The subtle and often difficult to understand difference between "a bug" and "a bug with security implications" in a project of this magnitude is a continuing point of contention between many folks in the community. There is no simple right or wrong answer here, and there are incredibly talented and dedicated people on both sides of the argument, yet this bug perfectly illustrates nicely why it is not without effort. Max Kellerman discovered the bug. Max did not intend to break any security boundaries, but was attempting to track down the source of file corruption on a production workload.

RHSB-2022-003 (Spring framework remote code execution "Spring4shell" CVE-2022-22963 and CVE-2022-22965)

The basics: This issue was a combination of 2 distinct and different CVEs affecting different Spring components. Both flaws could theoretically lead to remote code execution and were handled together as a single incident. Investigating both flaws did not find an exploitable use case in any Red Hat product.

The details: CVE-2022-22963 was a flaw in Spring Cloud Function, in which an attacker could pass a malicious expression via the spring.cloud.function.routing-expression header in an HTTP request. As the header was not properly validated, a payload of expression language code resulted in arbitrary code execution by the Cloud Function service.

CVE-2022-22965 was also a flaw in the Spring MVC and Spring WebFlux components when running on JDK 9 or newer. The root cause of the issue is a bug in the "getCachedIntrospectionResults" method, in which the class object is made accessible to the user via the method's return object, thereby allowing the attacker to manipulate the classloader directly and making multiple exploits possible.

The statistics

Affected major product versions: 11 (Red Hat AMQ Broker 7, Red Hat Decision Manager 7, Red Hat JBoss® AMQ 6, Red Hat Fuse 6 and 7 (formerly Red Hat JBoss Fuse), Red Hat Process Automation 7, Red Hat Integration, Red Hat Integration Camel Quarkus, Red Hat Virtualization 4, Red Hat OpenShift® Serverless)

Severity rating: Critical

Estimated associate time: 754 hours (approximately 15 associates involved)

Embargo time: 0 days

Time from public to first fix release: 13 days

Time from public to all fixes released: 29 days

Exploit code published: Yes

Exploitation in the wild: Yes, added to the CISA KEV catalog after 5 months

Estimated fix deployment: Unknown

Closing thoughts: While these flaws were considered to have a Critical severity rating, we were unable to find any example of exploitability in a supported use case in any Red Hat product. The code was included in all 11 affected products as part of a larger dependency, yet the vulnerable code was not actually reachable or even used. Thus, the flaw was given a Low severity rating in each individual product.

In situations like this, where the code is present but not reachable, we still call the products "affected" for a number of reasons. The first reason is that software can be incredibly complex and we are humans, using tools typically written by other humans in many programming languages. Unless the code is actually eliminated by the compiler or build process, complex and difficult to trace references could exist that human and machine analysis simply miss. Secondly, while the code may not be reachable in a system functioning entirely as designed, emergent properties such as other bugs may change this. Lastly, a future change may expose the flaw. In cases like this, where the underlying flaw is not only Critical but actively exploited in the wild, fixes are often prioritized due to an abundance of caution rather than an immediate threat.

RHSB-2022-004 (OpenSSL X.509 email address overflow CVE-2022-3602 and CVE-2022-3786)

The basics: OpenSSL likely needs little introduction as it is one of the world's most widely deployed general-purpose encryption toolkits today, providing encryption and authentication primitives for much of the HTTPS traffic on the Internet. Two stack-based buffer overflows were found in the parsing of email addresses in the X.509 certificate handling code. X.509 is one of the most common certificate standards used in public key cryptography and is an ASN.1 (abstract syntax notation) format certificate that binds a public key to an identity. ASN.1 itself is a fairly complex notation format, and many libraries that parse ASN.1 in C have been subject to bugs. The risk presented by any flaws in a critical cryptography library like OpenSSL is numerous, code execution allowing a complete remote system takeover being the most serious. Even minor flaws can potentially be used to undermine the integrity of cryptographic systems relied upon for trustworthy communications across the Internet.

The details: There were two flaws that were introduced when punycode handling, a method of encoding extended unicode characters in plain ASCII, was added in OpenSSL 3.0.0. CVE-2022-3602, the most serious of the two flaws, involved bounds checking when comparing the length of the decoded punycode string to the maximum length of the destination buffer was off by one, allowing a single unsigned integer value (4 bytes) to be written to the memory—past the end of the stack buffer in the ossl_punycode_decode() function. In theory, a 4-byte memory write primitive of this nature could allow overwriting a pointer or other critical program flow data. However, this was likely not the case in practice due to the memory layout in all analyzed builds. The second flaw was a simple 1-byte overflow with static data and would be even less likely to be exploitable on its own under any practical conditions.

The statistics

Affected major product versions: 1 (Red Hat Enterprise Linux 9)

Severity rating: Moderate

Estimated associate time: 52 hours (approximately 15 associates involved)

Embargo time: 7 days

TIme from public to first fix release: 0 days

Time from public to all fixes released: 1 day

Exploit code published: Red Hat has not seen functional exploit code for this issue on any architecture

Exploitation in the wild: Red Hat is not aware of any reports of exploitation in the wild

Estimated fix deployment: Very High

Closing thoughts: OpenSSL and Red Hat issued prenotification alerts for this issue, calling the 4-byte arbitrary data stack overflow Critical. This rating was based purely on theory and not on an actual analysis of any compiled code on any supported CPU architecture. Upon further analysis by Red Hat, and many others in the community, we were unable to find any instance where the 4-byte memory write primitive provided by the flaw on its own could be used to overwrite anything useful to gain code execution. In almost all cases, the overwritten memory was simply discarded. With this further analysis in hand, the severity rating was downgraded. While these decisions were made with the best intentions and are decisions both OpenSSL and Red Hat stand by, they were not without controversy. OpenSSL is an incredibly significant software library and attracts a great deal of attention. Confusion and even accusations of wrongdoing instantly filled online forums and support queues. In the end, while the flaw is theoretically exploitable in the right conditions, and a 4-byte stack write primitive is certainly not something to leave lingering in code, at the time of this writing, Red Hat has yet to see any functional proof-of-concept code or any reports of exploitation in the wild. Fixes were still prioritized and have been widely deployed.

Beyond vulnerability response

Trusted supply chain

Software supply chain security became an extremely hot topic in the last month of 2020 as news of the SolarWinds breach hit the tech press and mainstream media outlets. It did not take long for discussions about trust in open source software supply chains to surface.

This discussion was not a new topic to Red Hat. We've been in the business of providing trusted open source for over 20 years. However, as the world evolves, most notably the increasing sophistication of tactics and techniques employed by threat actors, so must we.

The year 2022 saw the first full year in operation of the new dedicated "Supply Chain Security" group within the Red Hat Product Security organization. This new function started in May 2021, and consists of multiple teams covering both the operational security oversight of our productization pipelines and research and development into new supply chain security initiatives.

Along with the increase in government oversight in the software supply chain, we are seeing an increase in the frameworks available for creating a "trusted supply chain". The Supply Chain Levels for Software Artifacts (SLSA) is one method designed to decrease risk and increase trust in software supply chains. Red Hat Product Security has promoted a large percentage of these activities internally. However, we are taking a more intentional approach in attesting to these controls.

There is no single solution for every software supply chain, so be wary of any one standard claiming to be the "gold standard". SLSA is a mechanism for measuring the perceived maturity of security, and we hope to provide our customers with this soon. But it is not the only standard and we will continue to explore all options that we think will give us the controls and transparency needed to gain the highest levels of customer confidence in our software supply chain.

Secure development

As mentioned in the Introduction, the start of 2022 began with some significant changes to the structure of our security-focused development programs, with increased attention to goals and standardized approaches to SDL.

The main goal of our SDL function is to formalize security controls that provide demonstrable evidence of adherence to internal security standards, following the structure of NIST Secure Software Development Framework (SSDF) and using controls outlined in NIST SP 800-53. The set of controls implemented across the portfolio includes, but is not limited to, static and dynamic security scanning, threat modeling extended by architecture and code reviews, and penetration testing, including automated scanning and white box assessments.

This led to the uncovering of 22 security vulnerabilities: 3 Important, 15 Moderate, and 4 Low, and 421 findings in the general security-hardening category. In the cases where these also affected open source projects, we contributed our results upstream so as to make certain the open source community at large benefits from our work.

Final words

Red Hat continues to take a risk-based and customer-focused approach to how we do vulnerability management. If you have not seen it already, I invite you to read the Open Approach to Vulnerability Management whitepaper, which goes into significant depth on how we rate and correct vulnerabilities. This annual risk report continues to demonstrate that Red Hat focuses on the riskiest vulnerabilities and mitigates that risk quickly. It's important to note that our proactive approach to correcting Critical and Important vulnerabilities, where the greatest probability of exploitation lies, bears itself out in the data. When we have confidence that a vulnerability is being broadly exploited, for which no fix has been provided, we promptly turn our attention to getting that fix out. However, most of those vulnerabilities are already corrected because we proactively fix all Critical and Important vulnerabilities across our portfolio of products.

Red Hat continues to look for ways to reduce the number of vulnerabilities that affect Red Hat products, and that is where our aim for security-focused development comes into play. Every vulnerability found and fixed before it enters the product life cycle is one less patch customers have to apply, so an increasing focus there is always good.

The number of newly disclosed vulnerabilities continues to increase annually, yet not every vulnerability is meaningful. The correlation between increased supply chain attacks and new vulnerabilities disclosed is not linear, meaning there is a disproportionate increase in supply chain attacks versus new vulnerabilities. One conclusion that many have drawn is that new vulnerabilities, exploited quickly, are not responsible for successful supply chain attacks. Where can improvements be made? Certainly, it is not fixing all vulnerabilities when exploitation rates are so low. Availability of patches is only meaningful if those patches are applied, which is the number 1 preventative measure. Holistically, a good security regimen requires scrutiny of not just the technology, but the processes and people. This presents the challenge organizations face to reduce risk is to use a layered approach that incorporates all 3.

– Vincent Danen / VP Product Security

North America
1 888 REDHAT1
www.redhat.com

Europe, Middle East, and Africa
00800 7334 2835
europe@redhat.com

Asia Pacific
+65 6490 4200
apac@redhat.com

Latin America
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@RedHat
linkedin.com/company/red-hat

redhat.com
#254252_0323