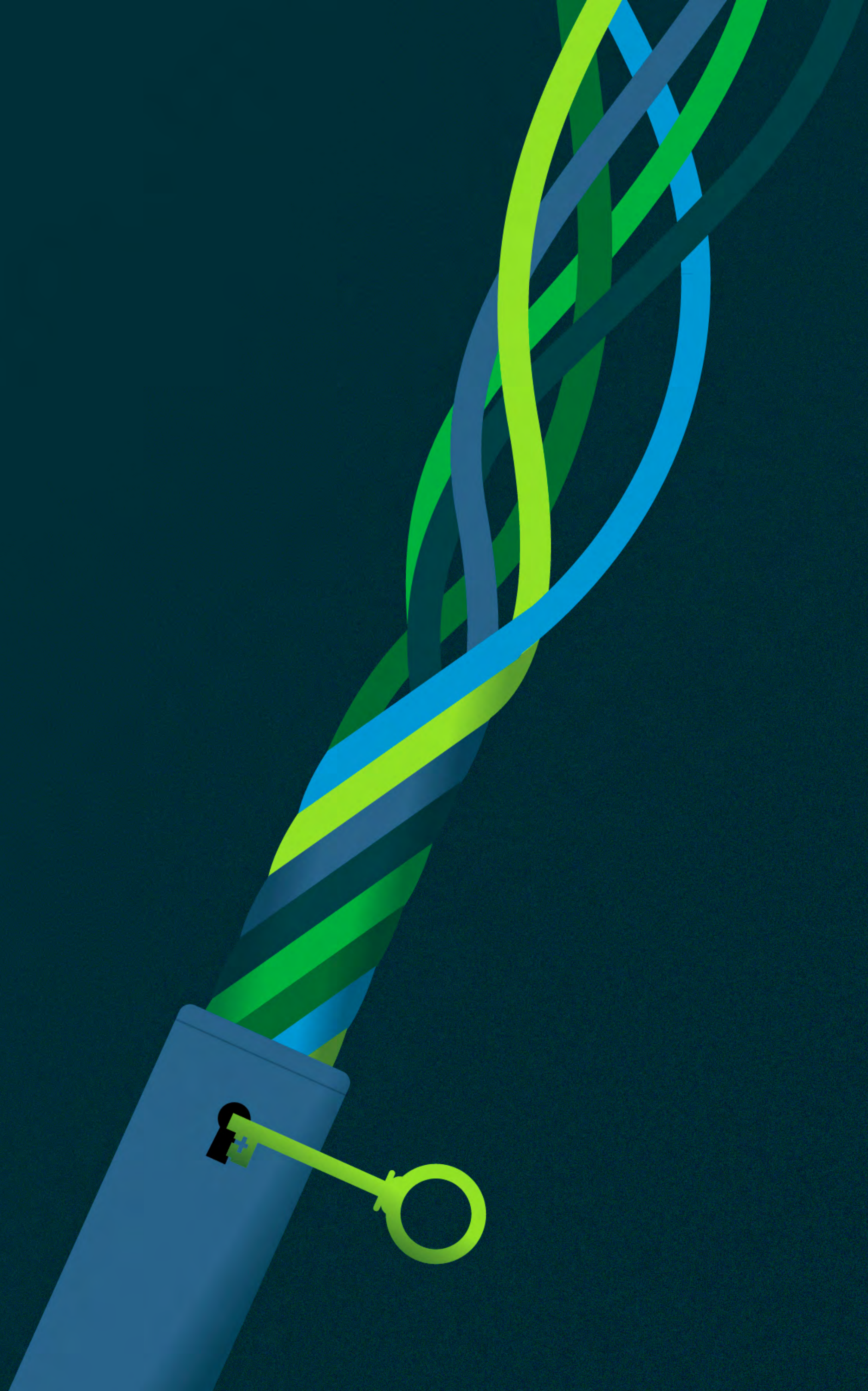
6 Paths to Application Security

By Ron Powell



circleci

Introduction

Enterprise applications are becoming more central to organizations' value streams, as companies across various industries look to accelerate their digital transformation. The consequent and exponential adoption of cloud services and applications is further fueled by remote work as teams seek to maintain productivity with effective and transparent processes.

This evolution often involves shifting app development to containers and microservices, which is great for rapid development and deployment, but leaves open the door for security vulnerabilities to sneak in. As development practices evolve, new threats and pitfalls emerge, such as external attacks, privilege abuse, and data theft.

Most organizations fail to adopt application security best practices that work to protect software, data, and users. In application security company Veracode's annual State of Software Security Vol. 11 report, 76 percent of the 130,000 applications tested have at least one security flaw, while 24 percent contain high-severity flaws (those rated by Veracode as level 4 or 5). The most common flaws are information leakage, cryptographic issues, carriage return and line feed (CRLF) injection, and code quality flaws.

The researchers note that most of the flaws do not pose severe risks to the application. However, these flaws do slow production in the long run.

Organizations can secure their assets and defend their software by integrating application security best practices into their software development life cycle. For example, integrating security tools into your application development environment can make the security process and workflow more straightforward and effective by making security issues more visible, automating auditing, and providing real-time insights to threats and vulnerabilities.

In this ebook, we examine the top security risks modern enterprise applications face and help you understand the journey toward mitigating each risk.

Vulnerabilities in the server, VM, or container operating system

Virtual machines

Virtual machines (VMs) are isolated from the physical operating system. However, virtualization software exploits can spread malware to the physical system, particularly when hypervisor software runs the VMs.

For example, scripts can run on a compromised host and interact with logged-in guests. An attacker can maliciously install trojans on the host and guest machines. They can also create an account with admin privileges, enabling the attacker to read, collect, or destroy company data.

Incorrect VM use can also impact a company's efficiency and undermine security efforts. An example of this is VM sprawl, which is when developers create VMs for testing but fail to delete them when they no longer need them. Developers often leave these test VMs unpatched and without security updates. If teams fail to track the number of VMs, along with when and where they are deployed, it not only causes security concerns, but also consumes valuable hardware compute power and disk space.

Companies should uninstall all obsolete applications and get new security updates to patch missing files and prevent attacks. Developers should regularly update running host and virtual operating systems through manual updates upon starting a virtual system. That being said, the best way to control VM sprawl is through regular lifecycle management using VM inventory reporting software.

Teams should also implement role-based access control (RBAC) to define appropriate roles and permissions for different users. RBAC ensures that only certain people have the ability to create VMs and snapshots, reducing the chance of wayward VMs.

If a file is suspicious, VM applications enable you to take a snapshot (both manually and automatically) of your current VM configuration. Then, if the file or service causes an unrecoverable VM problem, you can quickly revert to a previous stable-state VM snapshot. However, for greater security, short-term snapshot storage helps prevent attackers from stealing valuable information from snapshots.

Automated software monitoring tools can track and monitor your virtualized environment for anomalies and alert administrators of possible threats. An effective backup and disaster recovery plan helps ensure you retrieve your files after an attack.

Install the latest firmware on hosts and the latest security patches on virtualized infrastructure. Additionally, network elements (switches, routers, and more) should have the newest firmware.

Containers

Containers provide a lightweight, portable, scalable, and easy way to build, test, and deploy across various environments throughout the software development lifecycle. However, containerization lacks isolation from the host operating system (OS). A vulnerability in the host kernel or OS can impact all containers and let intruders access everything else in your stack.

Docker containers and Kubernetes deployed on the same IP space make it possible to attack other containers, spreading the attack. Further, what appears to be their advantage can also be their downfall. Container lifespans are short, but they can still be long enough for an enterprising hacker to get in and gain access to confidential information. The short lifespan can also challenge container observability, making it harder to detect the occurrence of attacks after the container is deleted.

Images and their dependencies are also highly vulnerable, especially if your DevOps team builds them from untrusted sources, including images embedded with Bitcoin miners and malware. A container registry can host thousands of images, and any intrusions or vulnerabilities within the registry provide a means to compromise running applications.

Container security requires a baseline so you can compare container environments in their normal state versus those containing anomalies or attacks. This is challenging to do manually as you may not realize a container is compromised, because it has a short lifespan, and any evidence of an attack is destroyed when a container is destroyed.

You can mitigate the chances of your data being undermined by choosing images with fewer libraries and only dependencies from trusted bodies. However, any image you manually scan and find secure may not be secure in the future. New threat data may identify vulnerabilities in what previously seemed like secure components.

Update your Docker or Kubernetes version and support applications regularly to ensure access to patches and bug fixes that address older versions' vulnerabilities. Use both dynamic and static scanning.

For optimal results, scan early and often. Continuous integration and continuous deployment (CI/CD) tooling can run a full range of scans and analyses every time an engineer commits code to a feature branch. If anything goes wrong, the engineer will get a report so that they can fix their code immediately. Containers need continuous end-to-end monitoring to ensure secure use throughout the CI/CD pipeline.

Servers

Servers are prone to various attacks from enterprising hackers, including security misconfiguration, cross-site scripting, and unvalidated redirects. A typical example is structured query language (SQL) injection attacks, where hackers exploit code in a web application.

The application sends user input to an interpreter as part of a command or query, which tricks the interpreter into executing unintended commands and giving unauthorized data access. Hackers usually do this by tweaking the URL to include malicious commands. The attack enables hackers to steal and alter information, such as modifying database data and executing administrator operations on the database itself.

Another example is broken authentication, which is, effectively, a digital identity shift where an attacker can impersonate a user's identity on the server. Credentials stored in plain text, exposed session IDs, or session IDs that do not time out or expire, can cause this type of attack.

Fortunately, encrypting server credentials such as hashing, using Secure Sockets Layer (SSL) encryption, and enforcing strict cookie control offer protection. Even better, audit regularly and continuously monitor updates and patches to software, servers, scripts, and applications.

Bugs in language runtimes

Runtime errors occur while a program is running in an interpreter or after successful compilation. Common examples of user-caused runtime errors include:

- Inputting string format data when the computer expects an integer
- Dividing by zero
- Passing an argument that is not in a valid range or valid value for a method

However, runtime errors may point to a bigger problem. Senior Security Consultant Fernando Arnaboldi found significant flaws in interpreters for five popular programming languages, which put applications they parsed at risk. For example, he noted Python has "undocumented methods and local environment variables that can be used for OS command execution." Software developers may unknowingly include code in an application that hackers can use in a way the designer did not foresee. This poses a security risk to applications securely developed according to guidelines.

Attackers continually use automated software to seek out code vulnerabilities as a means to attack. They can exploit any vulnerabilities within or between code repositories and servers to change and commit code to the primary, or exploit other resources and launch multiple application attacks.

Code reviews are critical to writing secure code, as the earlier you find errors, the faster, easier, and cheaper they are to resolve. In a DevOps workflow, code reviews should be embedded into the development process as early as possible. Debugging and testing should also be integrated throughout the development pipe flow. Developers can spend up to 75 percent of their time searching for errors and performance problems using logs and customer reports. Integrated development environments (IDEs) such as JBuilder and Eclipse can help with debugging.

The use of static analysis tools can identify problems with code security. In this instance, code is not running or executed, but the tool itself executes using the source code as its input data. The static analysis enables developers to use tooling to discover security issues early or even in real-time while writing code. The tools scan as developers write, flagging any security issues in the engineer's integrated development environment (IDE) or editor. By looking at data flow paths through an application, static analysis tools can identify where code produces unintended outcomes or data is mishandled. As developers push code directly into production, static code analysis also validates code quality, decreases future errors, reduces bugs, and prevents the opening of backdoors for attackers.

Error monitoring platforms can also identify problems as they occur, creating higher quality software development lifecycles. Errors can be efficiently triaged and assessed through crash monitoring and real-user monitoring. Then, developers can remediate these errors to prevent their reoccurrence.

Bugs and vulnerabilities in build tools like compilers

Compilers are a valuable feature of an IDE. However, compiled code may contain bugs that could be exploited. This includes bugs in your source code, bugs in the compiler and libraries, or undefined behavior in your source code that the compiler turns into a bug. A single bug can introduce a security vulnerability in your program or make it compute the wrong result. Unfortunately, compiler bugs are hard to detect, and once triggered, can hide in programs for a long time.

This is not a new problem, though. Computer science pioneer Kenneth Lane Thompson designed the original Unix operating system and invented the B programming language, the direct predecessor to C. In 1984, he spoke of “the many ‘chicken and egg’ problems that arise when compilers are written in their own language” and asserted, “You can’t trust code that you did not totally create yourself. (Especially code from companies that employ people like me.) No amount of source-level verification or scrutiny will protect you from using untrusted code.”

Academics at Pennsylvania University researched compilers (specifically the open-source GNU C compiler and the Microsoft C/C++ compiler that is part of the Microsoft Visual Studio package). They found that errors in the compilation process can produce vulnerabilities in executable code. They highlighted two main reasons:

- Undefined behavior in a programming language grants freedom to the compiler implementation on how to handle that behavior, which might lead to security vulnerabilities.
- The compiler (correctly) optimizes source code designed for security purposes in a manner that voids the intended security guarantee.

The academics discovered vulnerabilities related to standard vulnerability classes, side-channel attacks, undefined behavior, and persistent state violations. Some vulnerabilities are more dangerous than others because they provide either more information or more control to a potential attacker. Compiler bugs impact application code and even lead to severe damage, especially when a buggy compiler collates safety-critical applications.

The researchers found that many compiler optimizations occur in the front end of the compilation phase and not the optimization phases as expected. This gives the developer no control over these transformations, creating security vulnerabilities.

These researchers are not the first to find problems. Researchers at the University of Utah previously created an open-source tool called Csmith for stress-testing compilers, static analyzers, and other tools that process C code. Csmith found bugs in every tool it tested, including more than 400 previously unknown compiler bugs.

The Pennsylvania University research concludes that developers must understand low-level source code representation, because the program may appear to function correctly. Still, the developer's intent is not always realized at the machine code level.

Any software developed with compilers should follow a stringent software development process that includes quality assurance. CI/CD tests should test the build for vulnerabilities to avoid introducing bugs. Until compiler writers optimize code in more secure ways, testing is critical to ensure compiler bugs are found and their impact is mitigated.

Bugs and configuration errors in provisioning, infrastructure, and deployment tools

Infrastructure as code (IaC) enables DevOps teams to test applications in production-like environments early in the development cycle. It eliminates the need for developers to manually provision and manage servers, operating systems, database connections, storage, and other infrastructure elements every time they want to develop, test, or deploy a software application.

Configuration management tools such as Ansible, Chef, and Puppet enable developers to automate many cloud deployment and provision tasks. Instead of manually setting up on-premises and cloud environments, administrators and architects can automate them with IaC. They can shift from manual tasks prone to errors, inconsistencies, and security issues in deployment, to real-time automation that simplifies large-scale configuration and management.

However, IaC tools are not without their challenges. For example, an unpatched vulnerability can serve as a threat entry point, enabling hackers to run code on compromised servers or deploy cryptocurrency miners.

Palo Alto Networks' Unit 42, a team of global threat intelligence researchers, completed an industry-first study of IaC templates, which found over 200,000 insecure templates in use. The team also found that:

- 42 percent of CloudFormation Templates (CFT) contain at least one insecure configuration
- 48 percent of AWS S3 buckets do not have server-side encryption enabled
- 55 percent of cloud user-configured S3 buckets do not have logging enabled
- 22 percent of Terraform configuration files contain at least one insecure configuration
- 26 percent of cloud user-configured AWS EC2 instances have SSH (port 22) exposed to the internet
- 17 percent of cloud user-configured AWS security groups allow all inbound traffic (0.0.0.0/0)

Unit 42 also found that attackers use the default configuration mistakes implemented by weak or insecure IaC configuration templates to bypass firewalls, security groups, or VPC policies, unnecessarily exposing an organization's cloud environment.

The researchers also note that cloud users, as opposed to cloud service providers, create the IaC misconfigurations. In the shared responsibility model, configuring IaC templates belongs entirely to the cloud user. The challenge for organizations is to consistently enforce secure IaC configurations across multiple public cloud accounts, providers, and software development pipelines.

There are many ways a company can improve its tooling security. Unit 42 researchers strongly recommend thoroughly scanning every IaC template pulled from a public repository, such as GitHub, for vulnerabilities as part of the CI/CD pipeline.

Companies can also double down on their efforts to shift security left, that is, moving it to the earliest point in its development lifecycle. This encourages software delivery teams to test code right after writing individual units of code. Shift-left security helps detect bugs and other security problems early, meaning it can help maintain the delivery schedule by increasing the speed of the software development lifecycle in releasing secure, quality applications to market.

Overall, it isn't easy to secure the invisible and unknown. Teams need to monitor their public, private, and hybrid clouds to check who is accessing data and determine if the data is altered or exfiltrated.

Bugs and misconfiguration in infrastructure tools

Organizations are rapidly embracing cloud-native technologies — and for a good reason. These technologies' advantages include enabling faster development and deployment and quicker bug fixes and patches, which lead to faster feature delivery that drives competitive differentiation.

However, a 2020 StackRox report found that 94 percent of survey respondents had experienced a security incident in their container environments over the past 12 months. Sixty-nine percent of respondents' organizations had detected a misconfiguration in their Kubernetes environment, followed by runtime issues at 27 percent, and vulnerabilities at 24 percent. Forty-four percent of respondents had to delay deploying an application due to security concerns. Almost a quarter reported a major vulnerability, 17 percent experienced a runtime security incident, and 16 percent failed a compliance audit.

Misconfigurations allow a malicious actor to access a container. A threat actor with high access privileges can potentially enter other containers housing sensitive information or infect those containers with malware. Those who download particular images from within the infected container then get that malware.

Divy's 2020 Cloud Misconfigurations Report revealed that from 2018 to 2019, the number of records exposed by cloud misconfigurations rose by 80 percent, as did the total cost to companies associated with those lost records. Gartner estimates that through 2025, at least 99 percent of cloud security failures will be the cloud customer's fault.

Fortunately, there are plenty of robust security controls and protections available, but you need to fine-tune them for your environment. It is not only about implementing solutions such as secrets management, rotating and changing secrets regularly, appropriate access privileges, and role-based access control (RBAC), but doing so correctly.

For example, you should authorize secrets access according to an appropriate access management policy that restricts access rights according to pertinent roles, time, and tasks. However, it is still possible for the secrets provider to grant an imposter access to secrets. Multifactor authentication is useful in such a scenario because it prevents the secrets provider from giving the secret to an imposter, effectively limiting access to trusted containers.

Good configuration management requires ongoing vigilance. The threat vector is forever changing as attackers increase their abilities to access structural vulnerabilities. A central management tool makes it easy to manage audits, access control, and secrets. Automate vulnerability scanning and patching to stay on top of security challenges.

Exposed secrets

Developers handle more sensitive information than ever before as infrastructure secrets multiply throughout cloud and service providers. A secret contains a small amount of sensitive data such as a password, token, or key. Applications, scripts, and other non-human identities use secrets and other credentials to communicate with other applications and tools, and securely access databases and other sensitive resources. Incorrectly implemented secrets offer attackers an easy target.

Fortunately, secrets management provides a way to control digital authentication credentials. Manage secrets according to best security practices such as credential rotation, time and activity-limited access, and auditing.

It is tempting to lean on manual secrets management steps such as a registry of passwords and keys across an organization. However, this is time-consuming and prone to human error.

Also, many DevOps teams use different toolsets for different phases of the development process. These toolsets may differ in terms of security maturity and lack interoperability with other tools. Teams may also fail to aggregate security policies and audit data, leading to security silos and inconsistencies across the workflow.

With a CI/CD platform like CircleCI, you can secure your pipeline by centralizing production keys across your organization. An effective secrets management strategy should integrate with every tool in the DevOps workflow and across cloud providers. Good secrets management provides granular access control across your ecosystem to determine which access level people and services have.

CircleCI's automation provides a real-time view of your secrets, including audit trails and the ability to manage, rotate, and monitor app credentials.

Conclusion

The industry continually discovers vulnerabilities in software libraries, software packages, operating systems, and infrastructure. Vulnerability management requires continuous scanning, classifying, prioritizing, and patching these software vulnerabilities.

While developers can perform these tasks manually, this is prone to error. Vulnerability management benefits from automation and optimization. Human error is the most cited cause of data breaches and hacks, especially in complex environments. Also, there is a shortage of skilled, trained developers when it comes to DevOps, containers, and Kubernetes, increasing the chance of human error.

Security needs to shift left and be embedded into the DevOps workflow as early as possible, then enforced across the entire software development lifecycle. Otherwise, it risks delaying production and inhibiting business acceleration and innovation.

A staggering amount of tools and resources can help organizations through the enterprise application security minefield. Still, all tools and platforms must talk to each other and avoid creating security silos.

Further, developers must be confident that their tools and resources continually update against the latest security threats.

DevOps teams must understand the threats that can attack their pipeline and develop best practices for deploying a CI/CD pipeline. Securing the pipeline configuration is also essential.

CircleCI protects sensitive information across your delivery pipeline from source code to environment variables, the runtime environment, and artifacts. Also, CircleCI orbs enable you to easily integrate third-party tools and services to secure your pipeline with just a few code lines.

Enterprises cannot take these key application security risks lightly, but the risks do not have to be showstoppers. With a bit of work, enterprises can use CI/CD to automatically detect and mitigate all these threats in a scalable manner.

To automate your security, consider CircleCI's CI/CD automation software with built-in security.