The State of Pentesting: 2020

Caroline Wong, Chief Strategy Officer
Vanessa Sauter, Security Strategy Analyst

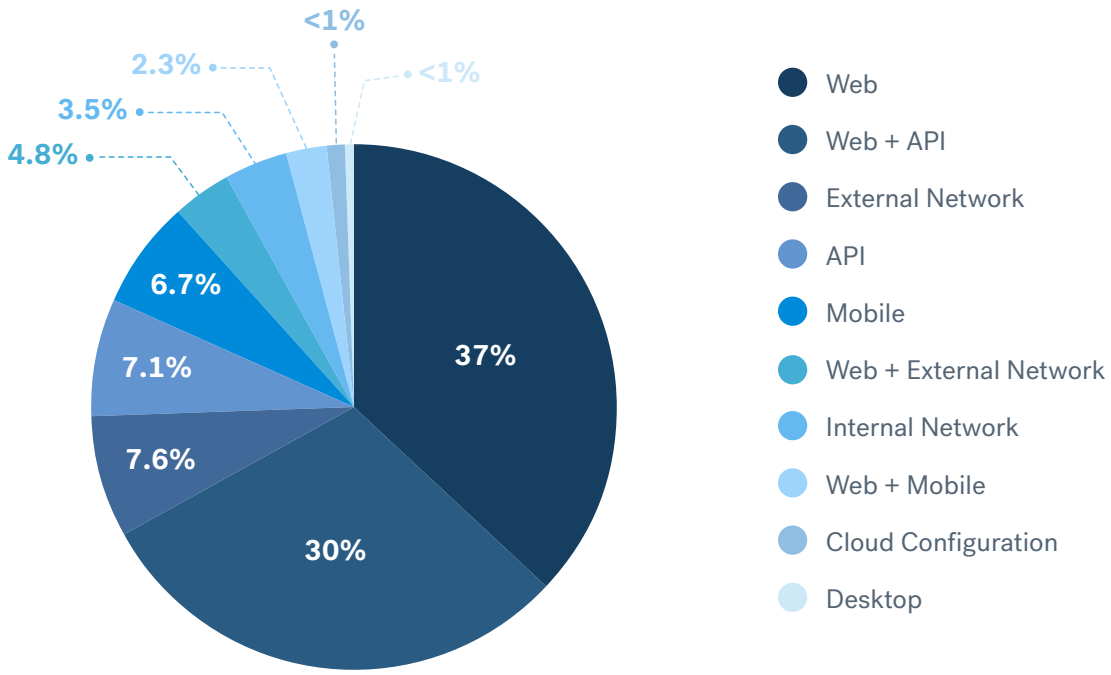
Cobalt

Contents

About The Report

Cobalt.io is a Pentest as a Service (PtaaS) platform that connects a global pool of nearly 300 vetted, certified pentesters, known as the Cobalt Core Pentester Community, with organizations who want to build quality security testing into their software development lifecycles. During the last four years, we have conducted more than **2,500 pentests** through our PtaaS platform.

Application Types Tested in 2019 (Cobalt.io PtaaS Platform Data)



Cobalt.io conducts pentesting across a variety of application types. This report features insights from aggregated data derived from nearly 1,200 pentests conducted in 2019. Web applications and web applications with APIs comprised 67 percent of Cobalt.io's testing last year. Cloud configuration, mobile, desktop, and isolated API testing were also covered, though in smaller numbers, as well as internal and external network testing.

For the last four years, we have reported on the categories of vulnerabilities that have been discovered by the Cobalt Core Pentester Community. This year, the following categories comprise our top five vulnerabilities across web applications:

1. Misconfiguration
2. Cross-site scripting
3. Authentication and sessions
4. Sensitive data exposure
5. Missing access controls

Although the OWASP Top 10 lists misconfiguration as number six in the top 10 web application vulnerability types, misconfiguration tops our list for the fourth year in a row. (In The State of Pentesting: 2019 report, we did a deep dive on the security misconfiguration vulnerability category.)

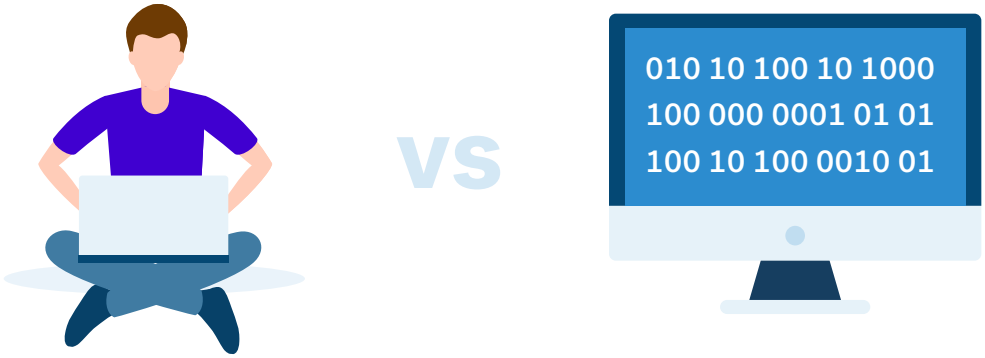
The State of Pentesting: 2020 report assesses which web application security vulnerabilities can be found reliably using machines and which require human expertise to manually identify.

We also analyzed survey responses from more than 100 practitioners in security, development, operations, and product roles regarding their application security programs. Respondents span a wide variety of industries, including information technology, healthcare, education, retail, and finance.

Finally, this report includes insights from members of the Cobalt Core Pentester Community in partnership with independent researchers, engineers, and other security practitioners.

Executive Summary

The State of Pentesting: 2020 report assesses which web application security vulnerabilities can be found reliably using machines and which require human expertise to manually identify. The scope of this exploration is black-box penetration testing ("humans") against dynamic scanning and out-of-band testing ("machines") for web applications.



We investigate the following questions:

- What vulnerability types can dynamic scanners reliably find?
- What are the vulnerability types that only humans can find (i.e. dynamic scanners cannot reliably identify them)?
- What are the vulnerability types for which scanners will not automatically populate results, but where automated tools can enhance efficiency to conduct further exploitation?

We assume that humans will use proxies like Burp Suite, Fiddler, or ZAP to modify HTTP requests, modify web sessions, and crawl sites. Any tool that can, once configured, identify a class of vulnerabilities would be considered "findable" by machines. (Note: This does not include tools like fuzzers, where the tool assists in identifying problems but does not necessarily identify what type of vulnerability it is, or how to practically address it.)

We hope that this report can help security and engineering teams make informed decisions concerning their application security programs.

Key Findings

- Humans “win” at finding the following vulnerability types: business logic bypasses, race conditions, and chained exploits.

- Although machines broadly “win” at finding most vulnerability types when applied correctly, scanning results should be used as guideposts and analyzed contextually.

- There are vulnerabilities that neither humans nor machines can independently find. Rather, they must work together to identify these issues. Vulnerability types in this category include: authorization flaws (like insecure direct object reference), out-of-band XML external entity (OOB XXE) , SAML/XXE Injection, DOM-based cross-site scripting, insecure deserialization, remote code exploitation (RCE), session management, file upload bugs, and subdomain takeovers.

- As the pace of software release hastens, so must the methodologies designed to secure applications. **More than one-third (37%) of our survey respondents release software on a weekly or a daily cadence.**

37%

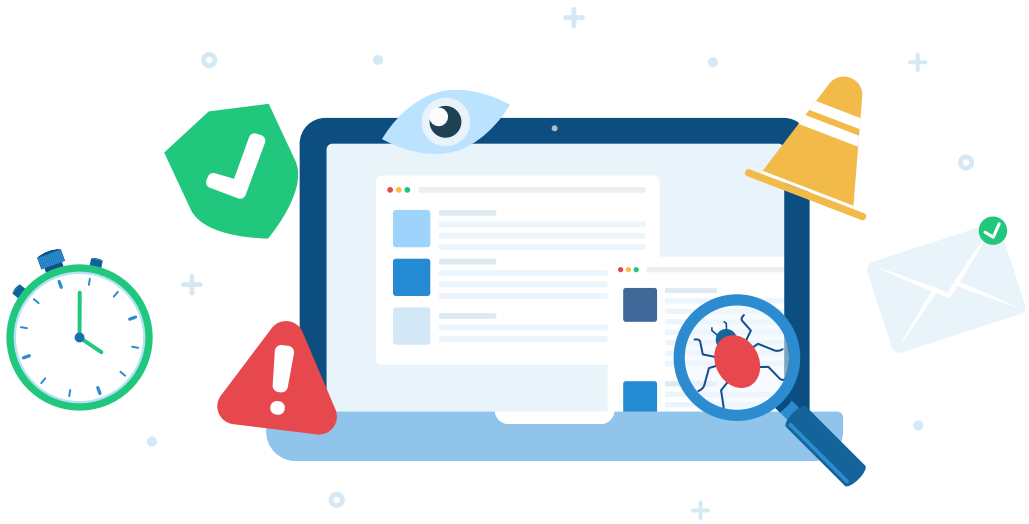
of security practitioners release software on a weekly or a daily cadence

Key Takeaways

- **Identifying a vulnerability is not the same as assessing the risk it presents.** While scanners can succeed in quickly finding vulnerabilities when properly directed, practitioners are required to apply necessary context. Among the many vulnerabilities (including false positives) you might find, the most important process is finding the true number of vulnerabilities and remediating the most critical ones. Right now, only humans can perform such tasks.

- **Good pentesters rely heavily on automation to test applications—whether writing Python code to iterate through hundreds of subdomains or automatically fuzzing inputs.** While pentesters will automate many of their processes to maximize their own efficiency, such automation is not synonymous with scanning. Scripts are often written to problem-solve and make the pentesting experience faster and more rewarding. In fact, the tools that pentesters develop and rely on to conduct security assessments reflect human creativity, persistence, and out-of-the-box thinking.

- **Scanners can only be as effective as the practitioners who deploy them.** Open-source and enterprise scanners serve as an excellent baseline to reliably identify simpler classes of vulnerabilities. When configured correctly, they can save a tremendous amount of time. All application security programs, no matter how scrappy or robust, should use static and dynamic testing. Scanners, however, are not a substitute for comprehensive application security measures, nor can they replace pentesting.

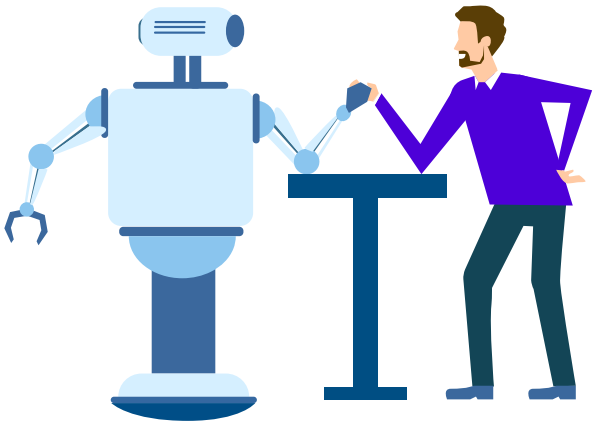


Finding Security Vulnerabilities: Human vs. Machine

To date, relatively little research has been published on the classes of vulnerabilities that machines and humans respectively excel at finding. Yet the distinction between machine-found and human-found vulnerabilities can greatly impact a security team's application security strategy, particularly with respect to allocating time and resources to the most effective find-and-fix activities.

The Machine Wins

Dynamic scanners work by injecting malicious payloads. They test access points when they are communicating with the front-end. Scanners are programmed to understand arguments and function calls and can detect vulns in headers, verbs, fragments, and DOM. They can also identify some misconfigurations and find components with known vulns.



Vulnerability types that can reliably be found by machines include: XSS (self, stored, and reflected), SQL injections (including blind and second-order), server-side request forgery (SSRF), cross-site request forgery (CSRF), sensitive information disclosure (path traversal, application errors, directory listing), missing or broken authentication, security headers (clickjacking/UI redressing), components with known vulnerabilities, local and remote file inclusion, cookie attributes, SSL/TLS-related issues, OS command injection, XXE, and cross-origin resource sharing (CORS)-related issues. Although machines broadly "win" at finding many vulnerability types, scanning results should be used as guideposts and analyzed contextually.

BLIND, SECOND-ORDER, AND OUT-OF-BAND VULNERABILITIES

When we talked with the Cobalt Core Community Pentesters about which vulnerabilities they find using machines, one of the more controversial issues was whether scanners could reliably find trickier issues like second order or blind vulnerabilities. A standard XSS or SQLi attack, for instance, produces an immediate result—like an alert prompt that says "Hello World!", data spilled into an input field, or an interesting error message. When you see that, you can immediately recognize the success of a payload. But what happens when a payload is successfully injected but the output isn't obviously visible?

A vulnerability is referred to as "blind" if the request or response is obfuscated in some way, making it difficult (or impossible) to interpret. A vulnerability is referred to as "out-of-band" if the response does not return within the same interface through which the attack was sent.

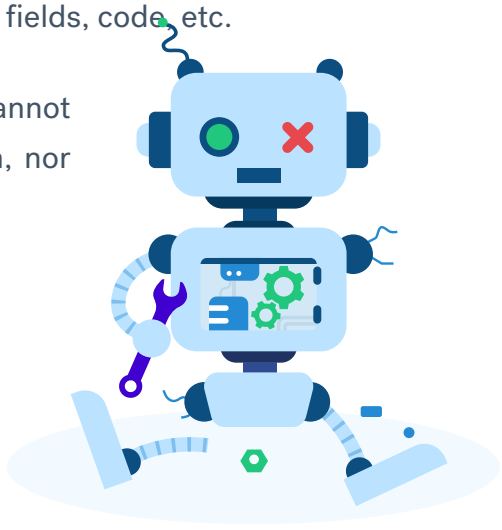
When it comes to blind or out-of-band vulnerabilities, the output won't be visible immediately. This is because an application may activate the payload at a later point in time, thus requiring pentesters to have back-end knowledge of the system to know if the payload was successful.

When properly configured (by a human), machines are able to identify blind, second-order, or otherwise out-of-band vulnerabilities.

MACHINE LIMITATIONS: SET-UP, TRIAGE, AND CONTEXT

Of course, scanners still require manual set-up and produce a significant number of false positives, so a human must configure any scanner and sift through results. Dynamic and out-of-band testing machines only search for what they're directed to look for, which means that any error in the discovery and reconnaissance phase of a web application pentest may result in an absence of scanning results for the overlooked subdomains, input fields, code, etc.

While scanners can guess the criticality of a vulnerability, it cannot properly assess the severity within the context of an application, nor can scanners indicate the possibility of a chained exploit.



The Human Wins

Humans “win” at finding business logic bypasses, race conditions, and chained exploits.

BUSINESS LOGIC BYPASSES

Business logic vulnerabilities exploit flaws in an application’s design. An attacker may be able to circumvent the anticipated workflow or alter a web application’s execution path. While most web application vulnerabilities result from a misconfiguration or absence of security controls, business logic attacks misuse an application’s unique business rules.

Examples of business logic attacks include:

- Successfully modifying the price of goods or services purchased
- Abusing weak password recovery validation
- Evading approval flows

Since business logic attacks are context-driven and exploit legitimate processes, business logic vulnerabilities elude scanners. Scanners are not capable of manipulating business logic rules or identifying misuse of an execution flow. The ability to identify this class of vulnerability requires a complete understanding of the web application and necessitates creative thinking. Business logic vulnerabilities are also difficult to identify through incident monitoring and intrusion detection systems.

Business logic vulnerabilities stem from weak design. Poor documentation of execution flows, lack of understanding of the technologies deployed, and an absence of manual testing will increase the risk of these vulnerabilities. The increasing complexity of web applications subsequently introduces greater risk for business logic vulnerabilities. Compounded by insufficient security monitoring and other practices (like mandatory vacation for arbitrage traders), business logic bypasses can have serious consequences.

To address and prevent these types of vulnerabilities, structural changes are required. Frameworks should be used consistently, the deployed technology stack must be well understood, and new vulnerabilities should be carefully assessed. Threat modeling and manual pentesting are essential to finding and mitigating business logic vulnerabilities.

RACE CONDITIONS

Race conditions are a subset of application logic attacks. They occur when two threads attempt to access the same data at the same time and both attempt to change it. Race conditions are often produced by failing to lock a file, meaning there's a race when a file is opened and not locked by the process. They can also arise when an application does not store information on a per-session or per-thread basis, and instead uses static storage. This type of vulnerability occurs for a brief period of time and must be triggered by specific circumstances. This means that there is a short window in which an attacker must "race" to exploit the vulnerability.

Login functions, password changes, and fund transfers are examples of application processes that are susceptible to race conditions.

It can be particularly challenging and time-intensive to identify race conditions when conducting black-box penetration testing, but these vulnerabilities are almost always critical when discovered.

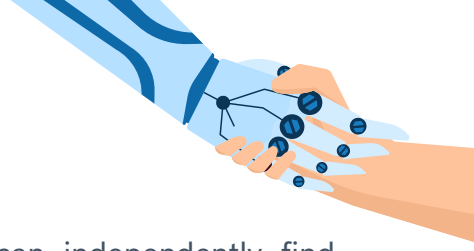
Race conditions can be reliably found using source-code review and static application security testing. However, dynamic scanners cannot reliably detect race conditions. A pentester would likely need to conduct fuzzing in order to identify race conditions, but for the purpose of this report, we consider fuzzing out of the scope for machines to "win." A pentester must first identify the possibility of a race condition before it can be tested, at which point automated tools can be deployed.

CHAINED EXPLOITS

Vulnerabilities can be chained to produce more severe vulnerabilities. "Low-hanging fruit" bugs that seem low-risk independent of each other can introduce new pathways for exploitation when chained to other vulnerabilities.

Informational or low-level vulnerabilities—the kind that a scanner can easily identify—can be an entry-point or hinge to more sophisticated attacks. Chained exploits rely on an attacker's creativity and attention to detail. These attacks are contextual and depend on a clear understanding of an application's functionality. They may even require exploitation of multiple frameworks. Neither a scanner nor some other kind of automated tool is able to effectively connect multiple vulnerabilities together in the same way that a human can.

There is no simple solution to preventing chained exploits. Instead, existing vulnerabilities must be mitigated and security measures must be improved across the application.



Together: Human + Machine

Finally, there are vulnerabilities that neither humans nor machines can independently find. Scanners cannot reliably find them or they may require intensive manual configuration. Humans, however, must rely on automated tools to successfully execute these exploits.

Vulnerability types in this category include: authorization flaws (like IDOR), OOB XXE, SAML/XXE Injection, DOM-based XSS, insecure deserialization, RCE, session management, and file upload bugs. In these cases, humans and machines must work together to exploit these vulnerabilities.

CASE STUDY: INSECURE DIRECT OBJECT REFERENCE

IDORs are produced when a user achieves unvalidated access to an object through their supplied input. An example would be modifying the URL parameters to access content, such as another person's account, by changing the input. Another example of IDOR is an attacker modifying a user's input through their id, pid, or uid that can be found in HTTP requests.

IDOR vulnerabilities can range from low-level or informational bugs (such as accessing or editing personal information like a person's name) to high severity (such as accessing and/or deleting sensitive information like credit card numbers or medical records). This type of vulnerability falls into the category of broken access control flaws.

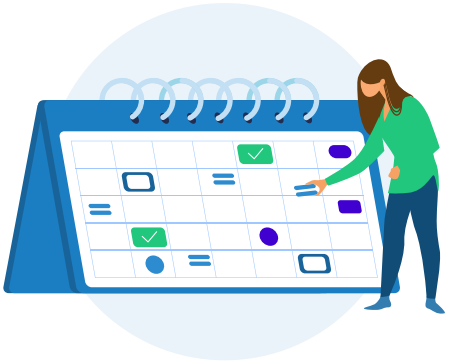
Vulnerability scanners will not always find IDOR vulnerabilities. Instead, a pentester must first identify the object reference and then use an automated tool (like Burp) to tailor the payload. Then, the automated tool can start the attack by sending thousands of iterations. Attackers can take other actions from there, like enumerating accounts.

There are multiple ways to prevent IDOR vulnerabilities. In principle, direct object references should not be exposed. Hashed values, or another value that is difficult to predict, should be used instead of normal strings or values. For example, www.bigimportantbank.com/user.php?id=99013 should be replaced with a hashed value like www.bigimportantbank.com/user.php?id=744878fbdd26871c594f57ca61733e09. Logical access controls should also be defined and enforced to prevent users from accessing restricted objects or acting in unintended ways.

2020 Application Security Trends

In addition to evaluating nearly 1,200 pentests conducted in 2019 and assessing the relative capability of machines and humans to find different kinds of web application security vulnerabilities, we also analyzed survey responses from more than 100 practitioners in security, development, operations, and product roles regarding their application security programs. Respondents span a wide variety of industries, including information technology, healthcare, education, retail, and finance.

Here's What We Learned:

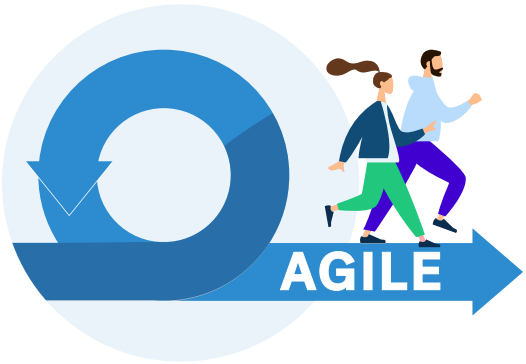


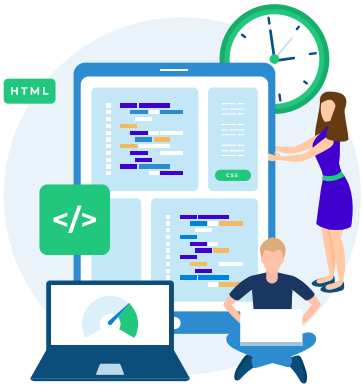
52%

indicate that their organization **pentests applications at least quarterly**, while only 16% pentest annually or bi-annually.

54%

of organizations use an **agile software development methodology**, and 30% characterize their methodology as DevOps, compared to just 5% doing waterfall.





25%

of **organizations release software on a daily basis**; 12% release software weekly; 22% release software monthly; and 13% release software quarterly. Only 1% of respondents reported that their organizations release software annually.

Organizations pentest many different types of applications. **Web application pentesting continues to be most popular**, while API, cloud, and mobile testing follow.

Cloud environments continue to present significant risk, particularly with respect to security misconfiguration. More than half (51%) of survey respondents conduct pentesting on Amazon-based cloud environments alone.

Almost three-quarters (71%) also said that they **rely on cloud environments like Amazon Web Services** or Microsoft Azure.

Many organizations are making the transition from DevOps to DevSecOps and **embracing an "everyone is a part of the security team" approach** to security.

The majority of respondents (78%) reported a strong relationship between security and engineering, and we expect that to continue to grow in the future.

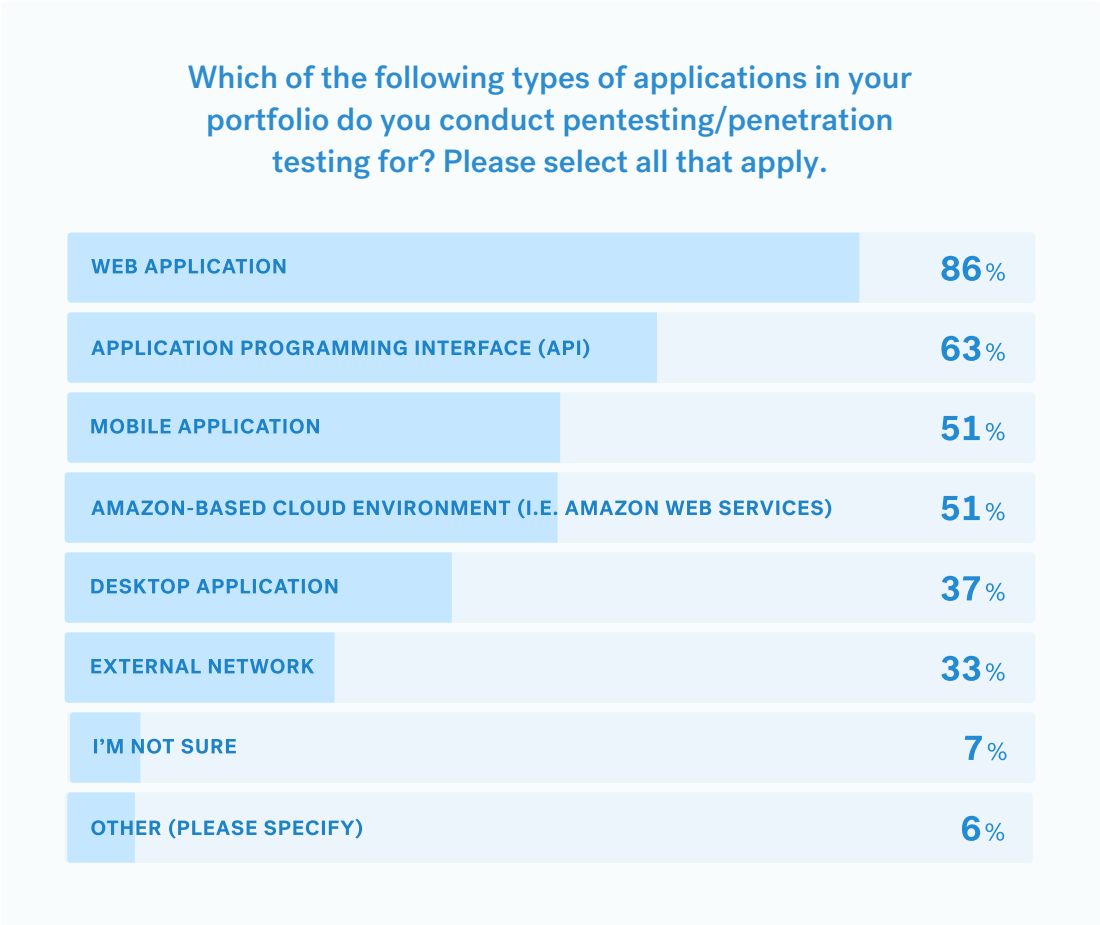
Pentest Frequency Is Up

There is a significant uptick in the frequency of pentests. In 2019, 67% of respondents said they conducted pentests annually or semi-annually. In 2020, respondents are conducting pentests on a much more frequent cadence, **with 52% indicating they pentest applications at least quarterly, while only 26% pentest annually or bi-annually.**

Organizations Pentest Many Types of Apps

✓ **Web application pentesting continues to be most popular, while API, cloud, and mobile testing follow.** This matches Cobalt's pentest platform data, where web applications and web applications with APIs comprised 67% of Cobalt's testing in 2019. Yet the increase in API and cloud environment testing is unsurprising given an increase in microservices and APIs.

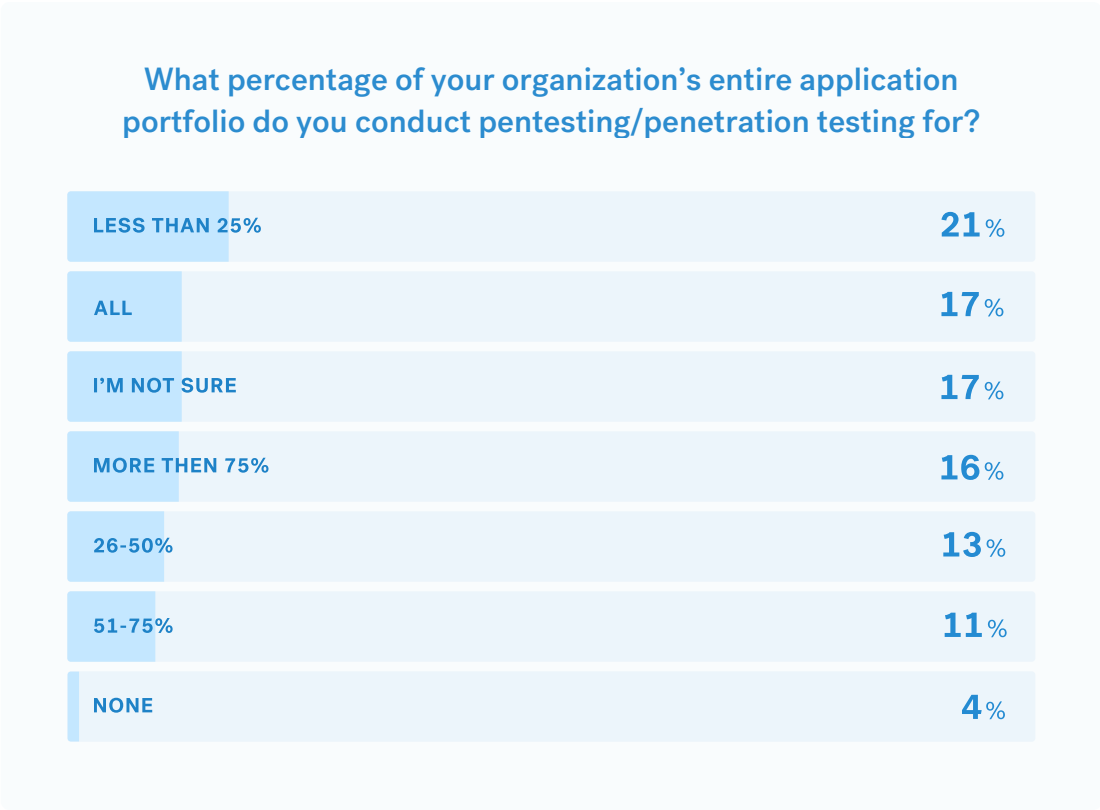
✓ **Cloud environments continue to present significant risk, particularly with respect to security misconfiguration.** We continue to see more cloud testing as web applications increasingly rely on cloud servers.



Pentesting Is a Priority

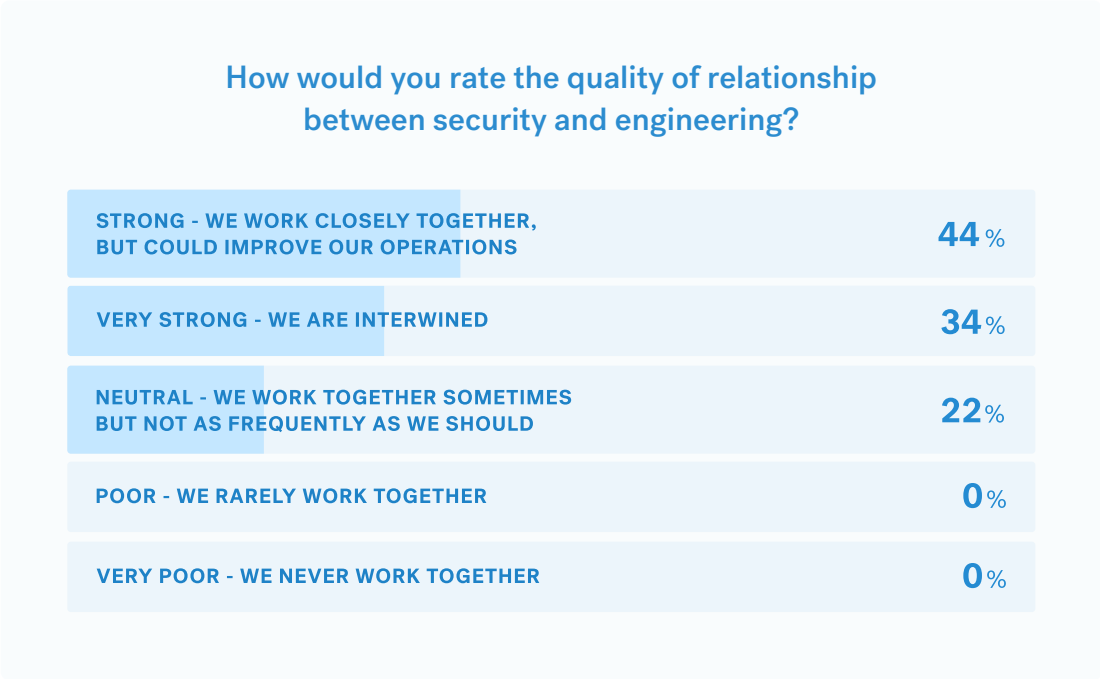
✓ **Given that more than three-quarters (78%) of respondents conduct pentesting to improve their application security posture, it's unsurprising that pentesting is viewed as a high priority.** Compliance continues to be a significant driver for pentesting, as well as procurement requirements and third-party vendor assessments.

✓ **Although there is general agreement that pentesting is a priority for organizations, the percentage of tested applications varies widely.** Given the cost and overhead of traditional pentesting, many organizations are forced to choose which applications to test and which ones to neglect. Compliance requirements, risk, and business continuity considerations all play a role in the number of applications tested. Given the variety of different industries represented in the survey, it also makes sense that the number of tested applications varies.



AppSec Requires Security + Engineering

✓ **A healthy relationship between security and engineering is essential to application security.** Many organizations are making the transition from DevOps to DevSecOps and embracing the "everyone is a part of the security team" approach to security. With this transition, of course, comes small roadblocks. It is reassuring to see that the majority of respondents are confident in the relationship between security and engineering, and we expect that to continue to grow in the future.



Conclusion

The global application security market has risen to meet the demand. Analysts predict this market will reach $9.64 billion by the end of 2023, achieving an annual compound growth rate of nearly 25%. Today, there are thousands of companies tackling application security.

As web applications continue to proliferate and the technology stacks evolve, application security engineers and pentesters alike must adapt quickly. The goal, from a defender's perspective, is to reduce the technical acumen and time required to find and fix a web application's vulnerabilities. With greater integration of DevSecOps, we hope to see a reduction over time in trivial vulnerabilities.

Scanners, which we refer to colloquially as machines, excel at finding specific classes of vulnerabilities, while humans are better able to find other classes. The benefits of pentesting are maximized when low-hanging fruit, like simple SQL injections, cross-site scripting attacks, or components with known vulnerabilities, are addressed earlier in the DevOps cycle using automated tools. This frees up time for skilled pentesters to identify trickier and more critical vulnerabilities.

The question of manual versus automated testing is now a question of ascertaining value in a results-driven market. It's become part of the strategy for choosing vendors, allocating resources, and determining the best use for the information security industry's greatest scarcity:time. If you take anything away from this report, it should be the unique value that machines and humans bring to the table. We hope this report helps you think strategically about how you invest your limited application security budget.

Team

AUTHORS

Caroline Wong
Chief Strategy Officer

Vanessa Sauter
Security Strategy Analyst

PRODUCTION

- Christina Schultz, Director of Marketing Communications
- Julie Kuhrt, Marketing Content Manager

SPECIAL THANKS TO:

- Travis McComarck, Technical Program Manager
- The Cobalt Core Pentester Community

Appendix

Web Application Security Testing Methodology and Tools

- "OWASP Web Security Testing Guide, Version 4.1," OWASP, https://owasp.org/www-project-web-security-testing-guide/v41/.
- "The Web Application Hackers Handbook, 2nd Edition," Dafydd Stuttard and Marcus Pinto, 2011, https://archive.org/details/TheWebApplicationHackersHandbook2ndEdition/page/n2/mode/2up.
- "The Burp Methodology," PortSwigger Web Security, https://portswigger.net/support/the-burp-methodology.

Vulnerability Management

- "Building an Application Vulnerability Management Program," Jason Pubal, SANS Institute, July 23, 2014, https://www.sans.org/reading-room/whitepapers/application/building%20-application-vulnerability-management-program-35297.

Integrating Testing into the CI/CD Pipeline

- "DevSecOps: What, Why, and How?" Anant Shrivastava, BlackHat USA 2019, https://i.blackhat.com/USA-19/Thursday/us-19-Shrivastava-DevSecOps-What-Why-And-How.pdf.

Deploying Web Application Scanners

- "Evaluation of Web Application Vulnerability Scanners in Modern Pentest/SSDLC Usage Scenarios," Shay Chen, Security Tools Benchmarking, November 17, 2020, http://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html.

Business Logic Flaws

- "How to Prevent Business Flaws in Web Applications," Marco Morana, OWASP, January 2011, https://owasp.org/www-pdf-archive/OWASP_Cincinnati_Jan_2011.pdf.

Race Conditions

- "CAPEC-29: Leveraging Time-of-Check and Time-of-Use Race Conditions," MITRE, September 30, 2019, https://capec.mitre.org/data/definitions/29.html.

Chained Exploits

- "Application Bug Chaining," Mark Piper, OWASP, July 2009, https://www.owasp.org/images/5/55/Application-Bug-Chaining-Live.pdf.


Cobalt