SANS | Research Program

Survey

SANS Application and API Security Survey 2024

Written by **David Hazar** and **Matt Bromiley**

June 2024

Cobalt
Pen Testing as a Service

Executive Summary

The velocity at which companies are building and changing applications seems to be increasing each year. Technologies such as containers, services in the cloud such as Kubernetes, development methodologies such as DevSecOps, design patterns such as microservices, and coding assistance through generative AI (GenAI) have enabled development and operations teams to keep apace. Although security testing capabilities have also improved, the value of individual testing capabilities has changed in response to increased threats and changing application architectures.

Application and API security is not quite as ubiquitous as infrastructure vulnerability management when looking at what is included in organizations' security programs. However, it is becoming more and more important as cloud platforms, serverless, zero trust, and other security efforts have helped to mitigate more of the infrastructure threats. Application and API vulnerability identification is more complicated than infrastructure vulnerability identification and requires a more diverse toolset, which can lead to more complexity and frustration during the rollout of an application security program, as well as later, as it continues to operate. Although complicated, applications and APIs have become the lifeblood and gatekeepers of many organizations' most sensitive information—so applications and APIs cannot be ignored.

In this survey, we asked participants to tell us about where they are getting the most value from their application security program by trying to better understand which identification technologies or processes have been most effective or have provided the most value based on the application architecture. It was interesting that manual penetration testing scored high in nearly every category for value even though it tends to be the most expensive and time-consuming type of testing.

One possible explanation for the high scores given for manual penetration testing is that automated tools struggle to cover a variety of different types of vulnerabilities such as broken access control and business logic flaws. They are prone to more false positives because penetration test findings are pre-validated. When humans are involved in the testing process, they can understand better how the application is supposed to function and, with this additional context, can find ways to circumvent the intended functionality. Another explanation could be that the prevalence of this type of testing is higher due to specific compliance requirements. Finally, it could also be that this type of testing is historically less reliant on in-house resources, and many organizations perform it without having fully implemented a formal application security program (which is less common for the other testing types).

Software Composition Analysis (SCA) has emerged from relative obscurity over the last decade to become one of the easier, cheaper technologies to integrate into the software development lifecycle. Although it is not perfect, it tends to not be as cumbersome to implement and maintain as other testing technologies, which aligns with the results of our survey.

Static Application Security Testing (SAST), Software Composition Analysis, and Dynamic Application Security Testing (DAST) came out at the top of many questions in our survey. Although they weren't always rated the highest in terms of value for each of the different application types, they are easier to implement, maintain, and get support for from development teams than some of the other testing methods.

One clear finding from the survey was that it is important to test throughout the application lifecycle using a variety of methods. Although testing early continues to be important, having visibility into and being able to monitor and test deployed applications is still critical.

Survey Demographics

For this survey, a larger percentage of respondents came from organizations providing security, technology, and application development services, followed by finance and education. When asking about staffing within the respondents' organizations, application developers and software engineers tended to be found in greater numbers than other roles, with security and operational roles having fewer numbers.

For organization size, the respondents leaned a little bit toward small to midsize, but large to extremely large enterprises still made up a good percentage. Looking at survey respondents, security-oriented roles garnered the top three spots, followed by application developers and business managers. Many respondent organizations had headquarters in the United States and Europe, but more than 20% had operations in almost all regions of the world (see Figure 1).

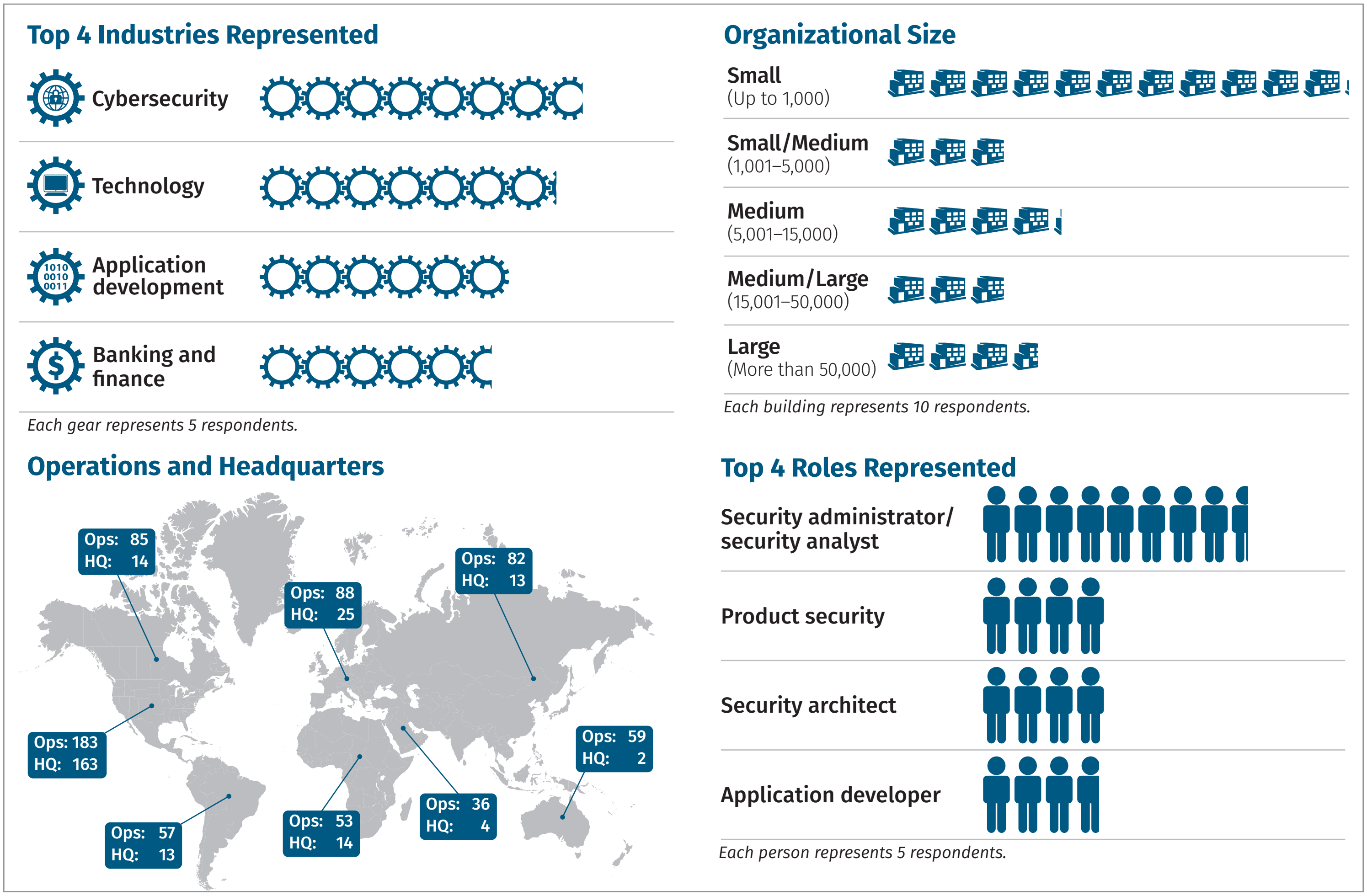

Figure 1. Key Demographic Information

Survey Results and Key Findings

About 80% of the respondents indicated that their organization developed and supported applications or APIs. These are the respondents who were included in the full survey. See Figure 2.

Although traditional form-based web applications still make up slightly more than 60% of our applications, REST APIs are close behind at 56%, followed by single-page web applications at 48%. In addition, the other application types are still well represented, with even SOAP APIs and GraphQL APIs coming in at over 20%. This points to continued and increasing diversity in the application development and security ecosystem and highlights the need for a diverse set of solutions for securing our custom-developed software. See Figure 3.

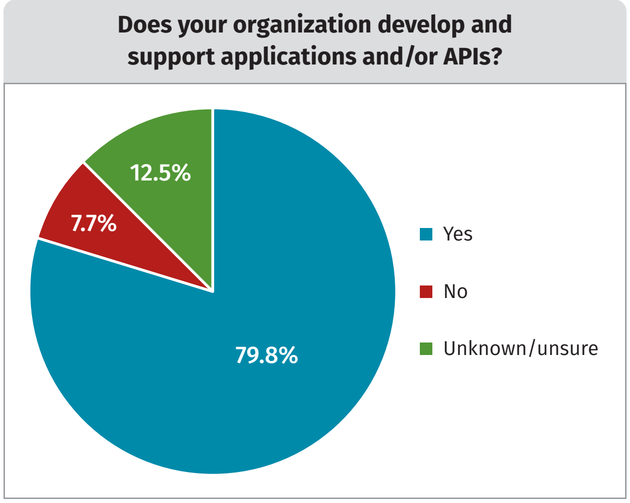

Figure 2. Development and Support of Applications and/or APIs

About 35% of respondents consider more than half of their applications or APIs to be microservices, while 11% said they had less than 10%. These statistics may be highlighting a trend away from monolithic applications to smaller applications and APIs. Adding to this hypothesis, the numbers for APIs are only slightly lower than those for microservices, which aligns with the fact that many microservices are designed as APIs. Containers are more widely used than serverless functions for running APIs, but surprisingly not by an extremely large margin. See Figure 4.

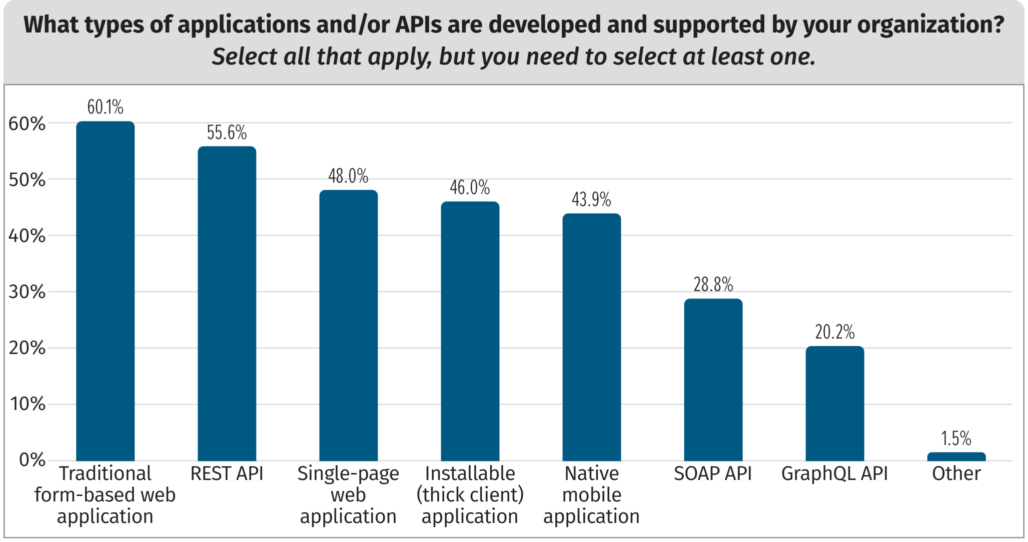

Figure 3. Supported Applications and/or APIs

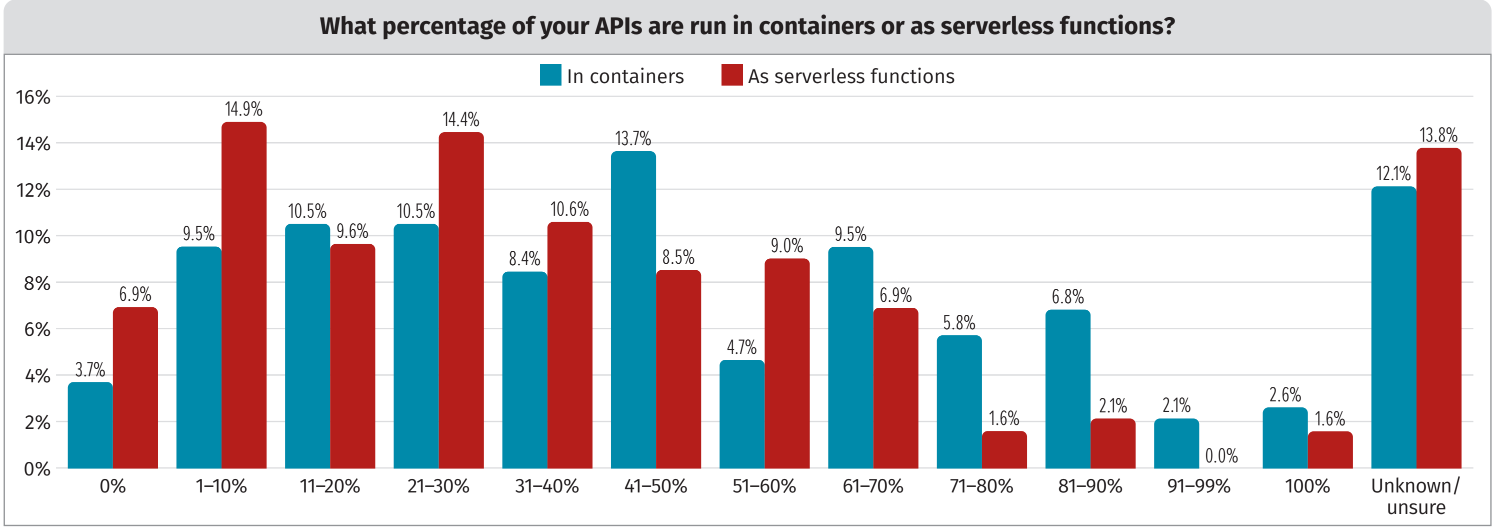

Figure 4. API Usage in Containers vs. Serverless

The most interesting statistics for us were those related to the value associated with using different testing techniques across different application and API types. See Figures 5 and 6.

We personally knew the value associated with manual testing, but it was great to see that acknowledged in the survey results. Manual testing is costly and time consuming relative to other testing methodologies; however, the additional understanding and insight that humans can provide into how an application's logic or authorization controls are supposed to work versus how they actually work is invaluable.

As mentioned earlier, because the tests are pre-validated by the testing team, there is much less noise associated with these findings. This can help streamline remediation efforts for development teams because they are not required to do as much of their own triage or analysis to determine if the risk is real. In addition, newer services such as penetration testing as a service (PTaaS) may streamline the effort, lower the cost associated with engaging in manual testing activities, and fit better into agile development methodologies. Many of these firms have also gotten much better at providing findings in a consumable format that is more easily integrated into the development workflow and backlog management tools, which has been a struggle in the past.

The survey shows that DAST tools are viewed as the most valuable form of testing for traditional form-based web applications and are the most effective automated tool for API testing. This makes sense, because form-based web applications are much easier to spider to determine what requests and parameters must be evaluated during the testing phase. Alternatively, single-page web applications can be written in a myriad of ways using numerous JavaScript libraries or even using company- or developer-specific JavaScript. This can make

Which application security testing technique provides the most value when testing your application types? ***Select your top choice for each application type.***

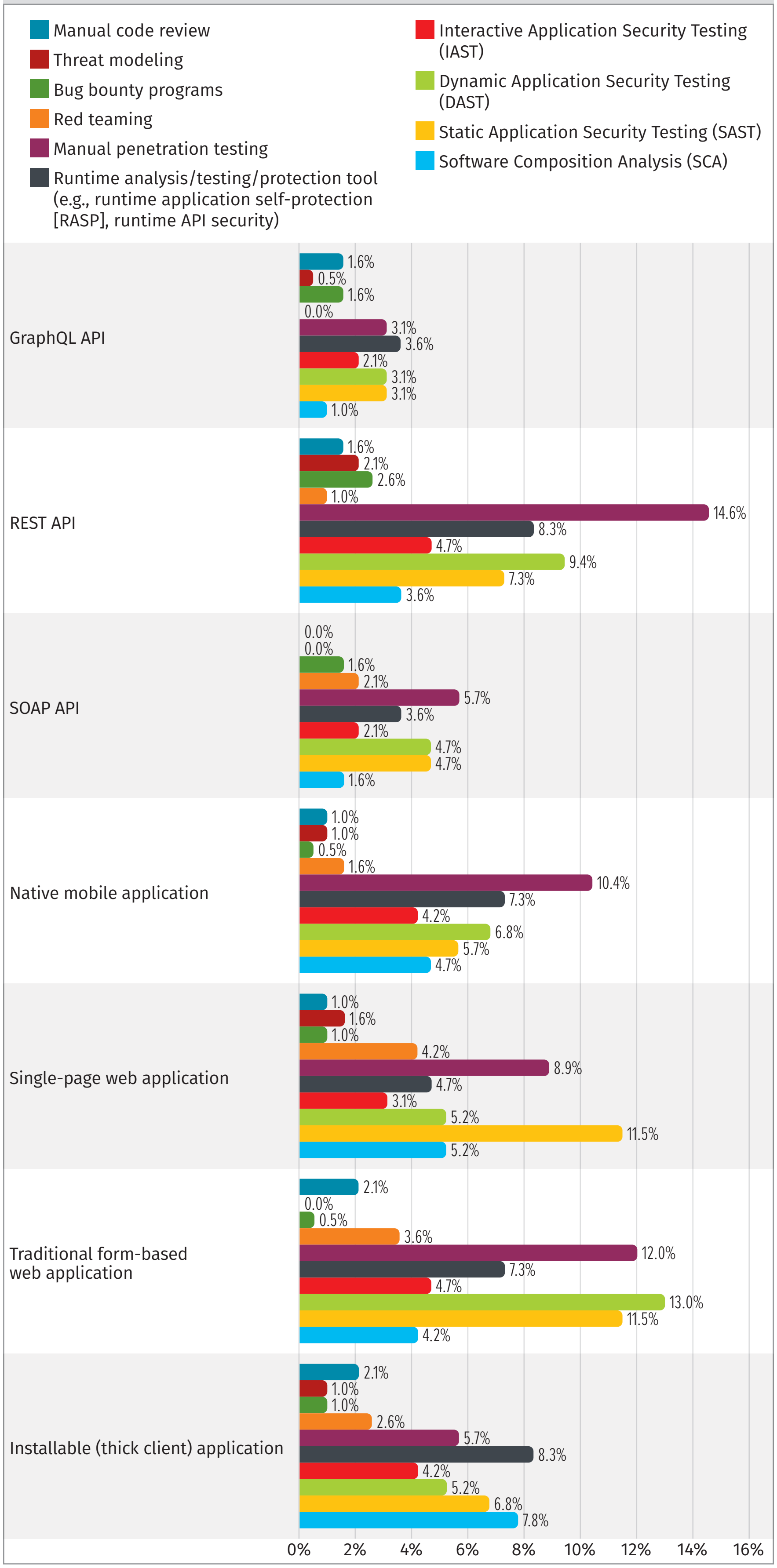

Figure 5. Value of Techniques When Testing Application Types (1 of 2)

it more difficult for scanners to properly identify all the requests and parameters throughout the applications, because it requires them to be able to interpret and understand all these coding formats, which, we have seen, can be a struggle. If you have good test coverage, you can usually train the scan engine using these tests. Some vendors have focused heavily on updating their scan engines to support newer application designs, but not all have kept pace.

We have worked with companies that have successfully run scans that take hours to run and produce results, only to find that, when we look at the details of the scan requests being made to the application, they are not even valid requests. Parameters are being submitted to the wrong pages or not being submitted at all. With any automated testing tool, validation of the results and what was scanned is important. Although it is easy to get some coverage for discoverable websites and services, successfully and safely using DAST technologies requires organizations to thoroughly configure and update scan profiles for each application to ensure adequate coverage and to avoid impacts to the application. Even though it is recommended to use DAST tools in non-production environments, if the designated test environment is shared, impact to application availability still affects the organization's velocity. Keep in mind that back-end and other asynchronous processes might be included as part of your application. Because these components of your application are not directly exposed or accessible by the front end, they are rarely visible to your DAST tools. Despite some of these implementation considerations, DAST remains an important part of any testing program and is not as prone to false positives as some other automated tools.

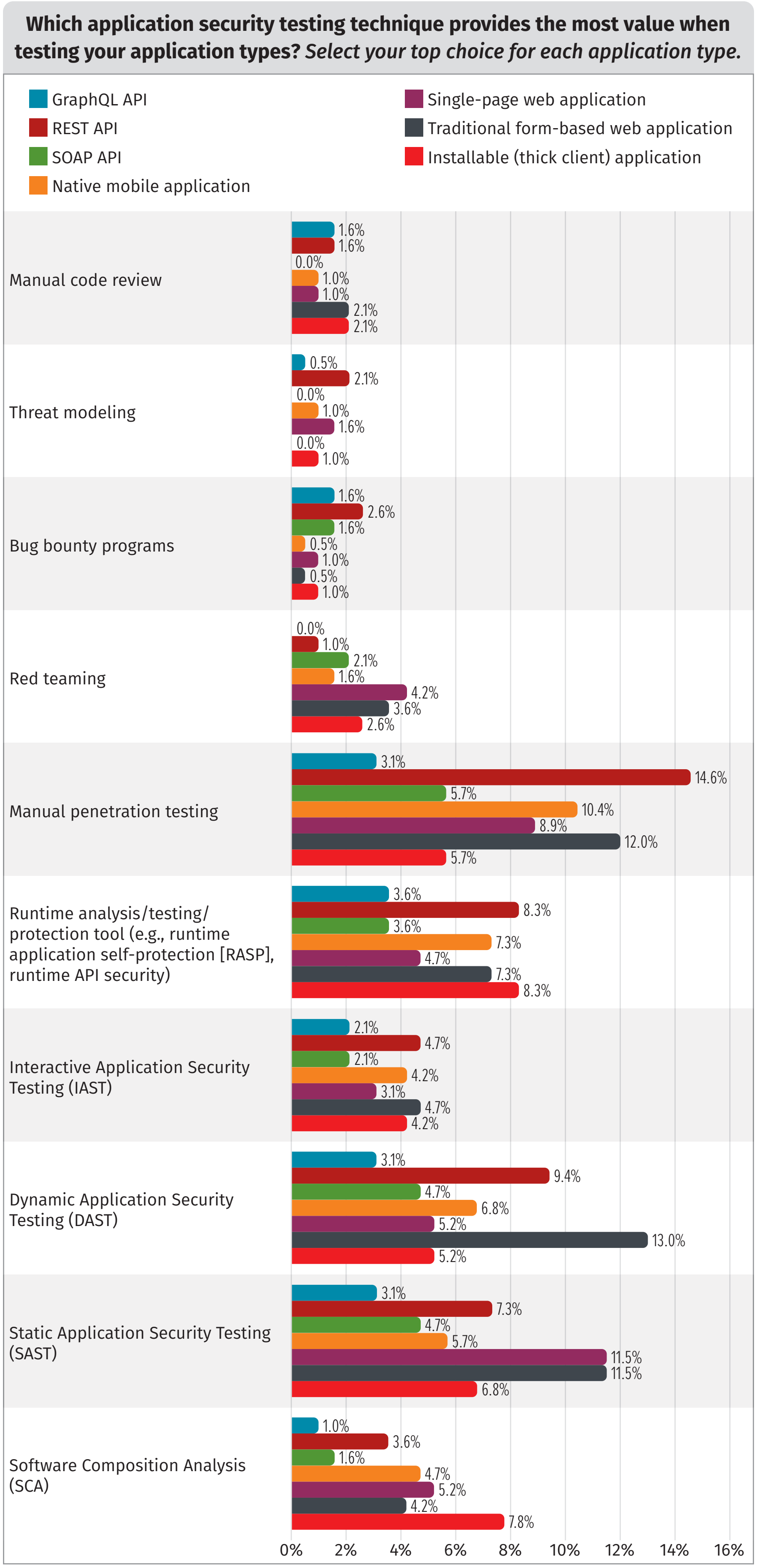

Figure 6. Value of Techniques When Testing Application Types (2 of 2)

SAST and runtime analysis are the other higher value testing methodologies according to our survey results. The runtime analysis technologies such as runtime application self-protection, web application firewalls, and other API security technologies provide some of the same benefits as DAST, but do not suffer from the same difficulties because they are not required to spider or crawl the application to understand the threats. In addition, there is value in seeing exactly how the application is being leveraged and for the inline runtime tools being able to take action to block the threats.

SAST, when done early, can provide deep insight into an application regardless of type as long as the language and underlying framework being utilized are supported by the technology being leveraged. Depending on the number of languages and frameworks being used by the organization, there may not be a one-size-fits-all solution. Many companies rely heavily on a specific commercial vendor while still having to support other commercial or open-source scan engines to get the rest of the coverage they need. Although that does add some complexity on the back end for consolidating and communicating results, the survey results show that due to how early you can integrate these tools, the ease with which they are integrated, the overall expense, and the level of support you can expect from development teams or the business if these tools are implemented properly, this still qualifies as a high-value security process. For example, Figure 7 illustrates which testing techniques are most easily accepted by development teams and the business.

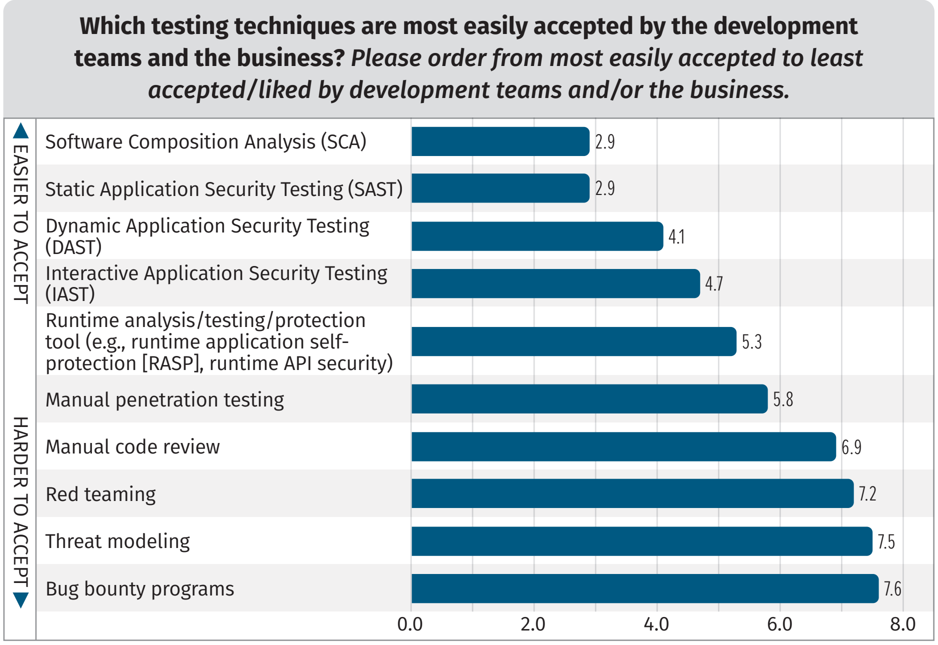
Which testing techniques are most easily accepted by the development teams and the business? ***Please order from most easily accepted to least accepted/liked by development teams and/or the business.***

Figure 7. Techniques Most Easily Accepted by Dev and Business Teams

Some might question the ease and developer support associated with SAST, and we can agree that historically SAST has been a bit painful. Nevertheless, the effort required by development teams to integrate this type of scanning is becoming more minimal each year, which we are confident factors into these results. The complexity of SAST can primarily be attributed to the fact that these tools can generate thousands, if not hundreds of thousands, of findings for larger applications when they are scanned for the first time. One way to handle this is to treat and catalog the initial scan results differently. This allows organizations to still track and acknowledge the technical debt associated with these initial scan findings to more quickly implement the normal processes for dealing with new findings from subsequent scans. This will allow development teams to better understand the true workload associated with leveraging these tools in an ongoing manner without the taint of these massive backlogs of vulnerabilities from years of development. We cannot fail to acknowledge the noise associated with SAST where a large percentage of the findings may be false positives, but if testing is automated and often, the effort to triage and identify the real vulnerabilities can be easily worked into existing backlogs and become just another task to be completed throughout the week. There are also promising efforts to leverage AI to help reduce some of the noise associated with these scans.

The other advantage of treating the initial scan results as something different is that it forces the organization to acknowledge the debt it has accumulated instead of forcing programs and teams to try and deal with it on their own. This can lead to additional funding, support, or resources for teams to help deal with this other backlog of issues or at least help the business understand why features may not be completed as quickly as they have in the past until the backlog is resolved. One other alternative that we see for dealing with these initial scan results is to turn the task of triaging the issues, or identifying which are real and which are false positives, into a game or internal bug bounty program where employees can receive rewards for the effort they put into reducing the debt. This can become a temporary side gig for those that need some extra money or have some extra time and is an alternative to just adding it as another task that teams must do without any compensation. We would caution against using this approach for anything other than the initial scan results, and be sure to make that clear to the participants.

We expected to see SCA score a bit higher in terms of value due to its ease of implementation and the relatively small expense and maintenance associated with this type of scanning. Although the ease and expense were acknowledged, we think the value is lower because relative to other testing methods this has not been as widely adopted until recently. The number of scanning options has exploded over the last 5 years as more and more organizations are leveraging third-party and open-source code within their applications. In addition, with the inclusion of reachability analysis in many of the newer versions of these products, we expect the adoption and satisfaction with this technology to grow in the coming years. Reachability analysis adds context to the scan results by helping the user determine whether the vulnerability in the included library is “reachable” in their application code. Reachability could be determined by the features or methods of the library being utilized or by the configuration of the library or components within the library. For example, if the vulnerable method is never called in your code, the vulnerability is still there but “not reachable.” If the vulnerability is only triggered when a certain configuration option is set to “true” and it is set to “false” in your application, then the vulnerability is also not reachable. This will allow teams to better prioritize these upgrades.

One other finding of note is that other forms of human-led testing outside of manual penetration testing were not listed by nearly as many organizations as valuable. These testing methods consistently showed up at the bottom of the list in terms of ease of automation, early implementation, acceptance by development teams, and expense. In addition to these methods being cumbersome and expensive, it is possible that they are not implemented as much outside of larger enterprises due to the complexity and expense or even due to the lack of compliance requirements focusing on these types of testing. Therefore, don’t let the results dissuade you from implementing these types of testing, because they definitely have a place in your testing program and can provide valuable results if implemented intelligently.

Finally, it appears that over 50% of organizations are implementing some form of runtime protection for their applications and APIs. See Figure 8.

These total percentages break down as follows:

- 54% for any of their applications
- 53% for any of their APIs
- 53% percent for implementing just-in-time authentication and authorization controls for any of their applications or APIs

Although there are cost, complexity, priority, and availability concerns with implementing these types of controls, the increased threats organizations are dealing with are starting to overcome some of these concerns.

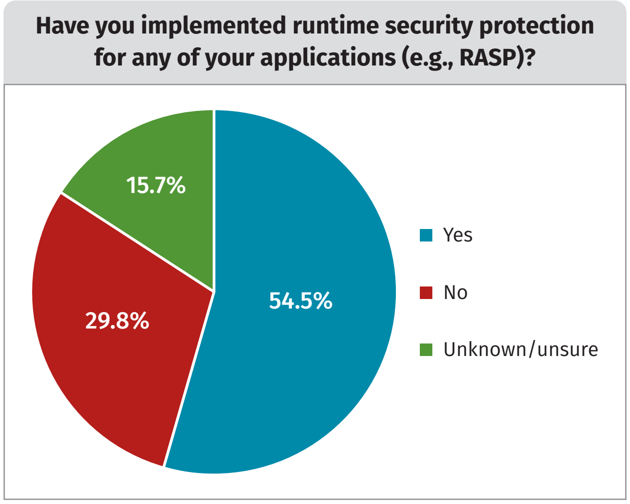

Figure 8. RASP Implementation

Summary and Final Recommendations

Application and API security can sometimes feel like the final frontier for security organizations. We put it off because either other gaps are larger and more pressing, we lack understanding of the risk, or we appreciate and understand the added complexity. However, the truth remains that much of our most sensitive data is protected by these applications. Although the effort is not trivial, it is vital that organizations take steps toward dealing with these risks. Start small and iterate until you reach a level of risk your organization can tolerate. We recommend starting with something extremely easy and inexpensive to implement like SCA, unless you have specific compliance requirements that force you to start somewhere else.

Starting small gets teams used to receiving findings, triaging them, and working them into the backlog. Then, move on to something very high value like manual penetration testing. Once these processes are in place and functioning well, start to layer on other methods like SAST, DAST, or runtime security testing, depending on your application architecture and design. Talk to others to help you prepare to implement these testing processes. Make sure you understand the common pitfalls so you can avoid them or at least communicate the pain ahead of time. For example, make sure you have resources available to consult with the developers on how to fix specific findings and that they understand that you will need their help reviewing the findings for false positives in addition to fixing the true positives.

Pain can sometimes be easier to accept if you were warned ahead of time. Make sure these warnings reach not only your stakeholders, but also their stakeholders. Talk about the benefit these tests will have for the organization in terms of not only security, but also privacy and availability, which tend to resonate a bit more with certain stakeholders. Help them see how these processes can help provide stability, which allows the organization to achieve its other business objectives. There will be setbacks, and expect complaints. Don’t let it get you down. There will always be problems to solve, but by working together with development we can find solutions to help us achieve the best possible results.

Sponsor

SANS would like to thank this survey's sponsor:

Cobalt

Pen Testing as a Service