

WHITEPAPER

**Models or Mayhem?
How Application
Security will become
Cybersecurity's Grand
Master**

Abstract

Computer security teams organize themselves according to function. In one common pattern, application security is separated from functions like cryptography, incident management, operations, compliance. In some organizations, application security teams burn most of their hours in patch management. They may check for misconfigurations, check compliance boxes. But the ubiquity and pervasiveness of software will likely reshape traditional cybersecurity swim lanes. Nine questions are posed which are intended to assess the capabilities of cybersecurity teams. Depending on how these questions are answered, in information technology are identified which suggest that AppSec teams will find themselves the highest-regarded among all cybersecurity teams.

Toppling the Stack

Software's layer cake continues its decades-long steady march toward complexity, differentiation, and specialization. Code production has always been a mix of cut-and-paste, design patterns, even trial and error. Any two developers tasked with solving the same problem produce different code. Any two code reviewers will take different amounts of time to review code written by any two developers.

This is not new. What's new is the "software-ification" of more and more aspects of information technology. In one extreme case, consider the Google Data Center, where its Jupiter and Orion software gives high granularity configurability. Software-based tooling enables Google engineers to ". . . shift moveable compute tasks between different data centers, based on regional hourly carbon-free energy availability" said Ross Koningstein, co-founder of Google's Carbon-Intelligent Computing project.

In short, Google engineers created a software-based IT fabric. Most developers used to draw a bright line at the "hardware" level, requiring service tickets to be issued to infrastructure teams to deploy, for instance, server or service account assets needed to run applications. "IT Administrator" was a specific role in the industry. But now, in many settings, that *software fabric* now goes deeper and broader; great swaths of infrastructure tooling can be exposed to code. "IT Administrator" employment is in decline, while developer talent remains in high demand.

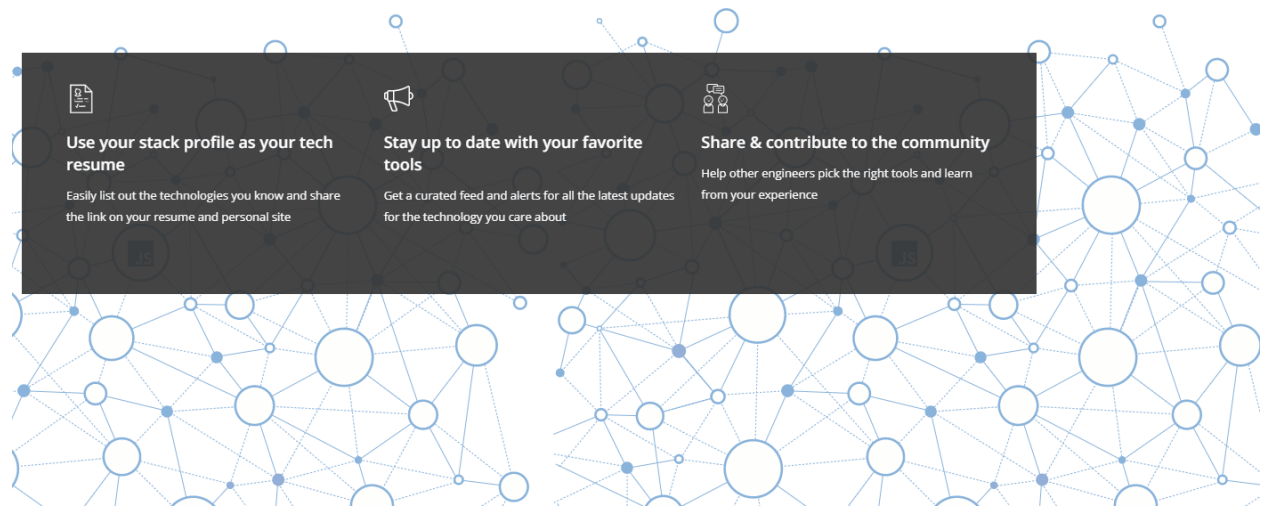
For security teams, whether these trends are regarded as good or bad is debatable, but they are clearly irrevocable. New attack surfaces will emerge. More and more configuration settings leave underlying assets accidentally unprotected. Visibility and

understanding of the deeper-broader stack – the suite of interconnected components needed to run an application or tool -- is essential. This perspective is on display in publications like New Stack <https://thenewstack.io/> and lightweight quasi-registries like StackShare.



Welcome To StackShare

Why share your stack?



The graphic features a dark grey horizontal bar with three white icons and text blocks, set against a background of a blue network diagram with nodes and connecting lines.

- Use your stack profile as your tech resume**
Easily list out the technologies you know and share the link on your resume and personal site
- Stay up to date with your favorite tools**
Get a curated feed and alerts for all the latest updates for the technology you care about
- Share & contribute to the community**
Help other engineers pick the right tools and learn from your experience

























































StackShare users identify software components -- and these are high level components – deployed in applications. The list of components long and growing.

Add tools to your personal stack

Search for or select tools & services you use in this stack

Send me weekly updates on these tools

Popular Tools

 JavaScript	 GitHub	 Python	 Git	 Node.js	 jQuery	 Docker	 React	 Visual Stu...	 HTML5	 PHP	 Java	 Slack	 Google An...
 MySQL	 NGINX	 Gmail	 npm	 MongoDB	 PostgreSQL	 Postman	 WordPress	 Google Dr...	 Ubuntu	 TypeScript	 ES6	 Stack Ove...	 CloudFlare
 Apache H...	 AngularJS	 Google Fo...	 Jenkins	 Jira	 GitLab	 Redis	 Amazon S3	 Kubernetes	 Amazon E...	 Vue.js	 C#	 Google Ta...	 Trello
 Font Aw...	 Bitbucket	 CSS3	 Visual Stu...	 IntelliJ IDE...	 Sass	 Google M...	 Firebase	 Django	 Webpack	 Sublime T...	 jQuery	 Flattop	 G Suite

Screenshot from StackShare, 2021

“Cloud native” is a trend in software engineering that leverages cloud computing which emphasizes scalability and composable services, typically using public cloud services and often extensively incorporating open-source projects. Trends in the cloud native movement suggest that this already long list of stack components will continue to grow. The resulting complexity is part of what fuels the benefits associated with cloud native design patterns: resilience, scalability, isolated states, elasticity and “loose-coupledness” (Fehling et al., and Kratzke et al.)

Left-Shift

Because of these and other trends, much of cybersecurity is already a logical subset of software engineering. The so-called Left Shift movement (see <https://devopedia.org/shift-left>) moves engagement of software engineering practitioners – to earlier stages of the development process. Left Shift has moved the focus toward nonfunctional requirements such as observability, telemetry, resilience, and security are integrated into planned builds. With Left Shift, as Devopedia’s author notes:

Shift Left doesn't mean "shifting" the position of a task within a process flow. It also doesn't imply that no testing is done just before a release. should be seen as "spreading" the task and its concerns to all stages of the process flow. It's about continuous involvement and feedback.

Where security was once left to “software engineering education” and late-stage testing, Left Shift distributes security concerns throughout build and test processes.

Continuous Integration / Continuous Deployment (CI/CD) is a relatively recent practice of more frequent, increasingly automated software production and deployment. When using CI/CD methods, developers must also design test harnesses which will enable testers access to developer artifacts, and where possible, enable test automation. Left Shift for security teams is directly analogous to test engineering obligations. The Left Shift trend highlights overlapping areas of responsibility, notably for penetration testing, code review, compliance with approved stack and repository components and instrumentation for Security Information and Event Management (SIEM) visibility.

These changes have been incremental but have accelerated with the cloud native movement. Not surprisingly, security teams have seen the number and heterogeneity of components and technologies increase.

Lines of Evidence for the Ascendancy of AppSec

How extensive are these trends and what do they mean for information security teams?

Supporting evidence for the pervasiveness of this transformation comes from multiple sources. It's not a coordinated transformation with an agreed-upon Gartner / Forrester calling card (e.g., “zero trust”). Instead, there are transformations across a broad front. The table below lists attack vectors involved in nine information technology domains.

EVIDENCE DOMAIN	SAMPLE ATTACK VECTOR
Microservices	Shodan, Git repo search for API's; GitHub residual code snips & secrets. APIs unprotected by Captcha or input checking (recent SYF incident***)
DevOps	Brand and typosquatting of Maven & Jenkins plugins
Software defined networks	Attacks on software-defined network (SDN) controls (One analyst categorizes 8 other types of SDN attacks); use of service mesh e.g., Istio for defense; Google Data Center Jupiter and Orion
Left-shift test methods	Insecure continuous deployment and integration pipelines due to accidental secrets disclosure
Agile and SRE	Secure Scrum; SRE to AppSec Engineering skill enrichment
Cloud Native, Kubernetes	Attacks on Docker Daemon ports; Kubernetes Attack/Defense analysis
Infrastructure as Code	CloudSploit scanning of CloudFormation
GitHub & "Repo Reliance"	Solarwinds exploit; dependency com attacks
Streaming services (e.g., Spark, Kafka)	Unsafe deserialization; RCE Apache Spark REST API; unsecured access to Kafka metadata

For more details on these attack vectors, see "AppSec Attack Vector References" below.

Nine Questions

Cybersecurity teams which believe they can simply perform vulnerability scans and call it a day are doing worthwhile and nontrivial work but missing a big part of the application security picture.

1. Can you test what you can't understand? *What if the software is performed a complex biomedical process with someone's life on the line?*
2. Do you have access to experts at each software abstraction layer? *If not, how can you identify a proper configuration from a malicious one?*
3. Have you leveraged automation for assurance, health as well as penetration testing? *To automate, you must produce code, either through low code tools, AI or traditional programming languages.*
4. Are your risk and trust levels explicit? *In building most applications, components will be drawn from multiple sources. Some sources, such as internally verified reusable libraries may be highly trusted, whereas open- source components with*

few recent contributions or contributor might be less trusted. Vendor-supplied software is often somewhere in the middle.

5. Have you leveraged AI, including stack and domain knowledge for the application, to help manage application complexity? *AI can enable domain experts to participate in application assurance, identify risks, implement new countermeasures.*
6. Are you prepared for the specialization paradox? *Specialists will be needed to secure each black box, even as the number of black boxes increases. Staffing can't be indefinitely augmented to include every specialization.*
7. Have you identified and instrumented the policy decision points (PDPs) where security controls, such as access and logging, can be implemented? *System complexity can multiply PDP's at a terrific pace.*
8. Are security principles fully distributed through development, test, deployment, and production monitoring? *The principle of zero trust has shifted security reliance from end points, but has yet to fully encompass the software development life cycle.*
9. How are you keeping developer, test, and infrastructure teams abreast of the latest open-source tools, security frameworks, and standards? *Projects like OpenTelemetry and Kubernetes are changing both the attack surface and the available countermeasures.*

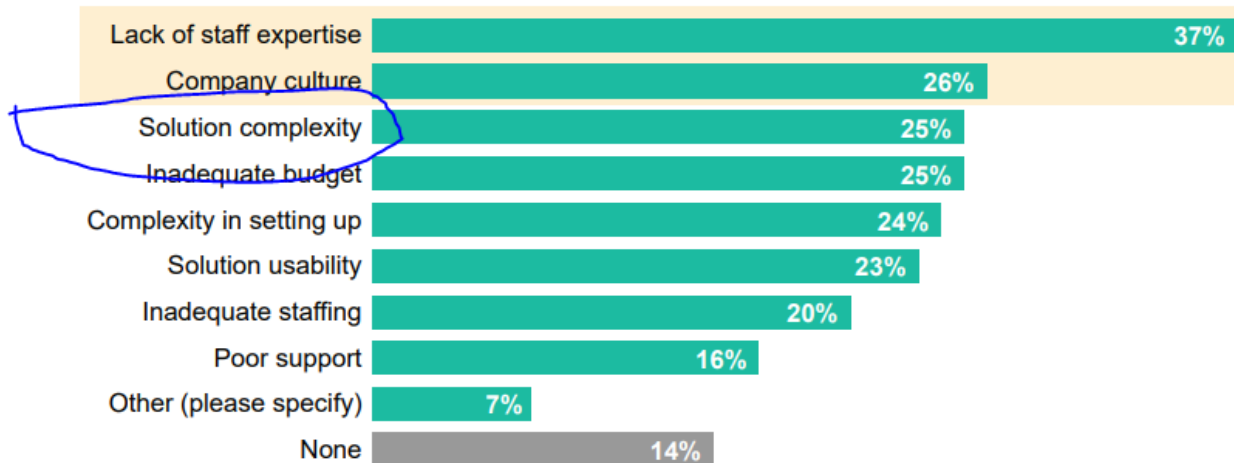
Mayhem Management

There's widespread concern over these challenges to application security, as shown in this recent survey.

Application Security Testing Tool Inhibitors

Source: 451 Research's Voice of the Enterprise: Information Security, Vendor Evaluations 2019

Q. What inhibitors has your organization encountered in adopting or fully utilizing your application security vendor's technology? Please select all that apply.



Application Security Testing Tool Inhibitors (451 Research, 2019)

Industry voices echo the worry. Let's listen in.

- “We need to reimagine all of our security testing techniques so that they make sense in a continuous environment. We also need our security experts to become coaches and toolsmiths¹ rather than the ones to chase down every vulnerability -- because that will never scale” (Williams, 2014).
- “If every time you’re building some new microservice, you have to think about all of those concerns about security, where you’re going to host it, what’s the IAM user and role that you need access to, what other services can it talk to—If developers need to figure all that stuff out every time, then you’re going to have a real scaling challenge” (Sargent, 2021).
- “Like in the equivalent of it takes a village, it takes a team to keep a microservice healthy, to upgrade it to make sure it’s checking in on its dependencies, on its rituals, around things like reliability and SLO,” Mike Tria, head of platform services at Atlassian recently told *SD Times*. “So, I think the good practices [folks] have a team [working] on it. For example, Atlassian has about 3,000 developers and roughly 1,400 microservices. Assuming teams of five to 10 developers, this works

¹ Brooks, F. P. (1996). The Computer Scientist as Toolsmith II. *Commun. ACM*, 39(3), 61–68. <https://doi.org/10.1145/227234.227243>

out to every team owning two or three microservices, on average . . . “ (Sargent, 2021).

- “. . . There is a real cost to this continuous widening of the base of knowledge a developer has to have to remain relevant. One of today’s buzzwords is “full-stack developer”. Which sounds good, but there’s a little guy in the back of my mind screaming “You mean I have to know Gradle internals and ListView failure modes and NSObject quirks and Ember containers and the Actor model and what interface{} means in Go, and Docker support variation in Cloud providers?” (Bray, 2014).

The developer concerns aired by these industry voices correspond *directly* to the concerns of security engineers. Some want to place the onus of creating secure software on developer. Not only has that not worked, but in an era of highly specialized, component-based development practices, it can’t scale. Just building the test harnesses is several steps beyond what most application developers can manage.

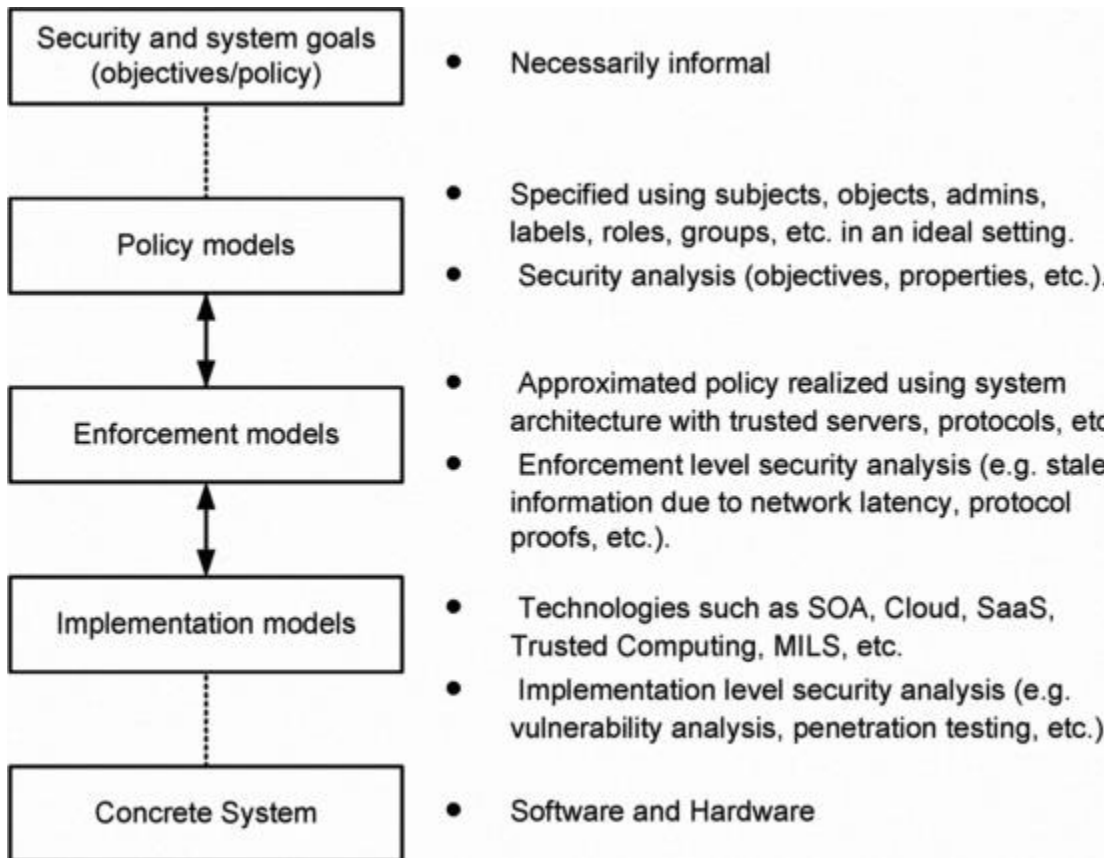
Simplification is Not the Future, or Why AI

Just as all products and services are increasingly software enabled, designed or managed, securing cyberspace is increasingly tied to securing that touch on all facets of IT.

In part, the relative importance of bug bounty programs reflects the greater importance of application security. Technologists with specialized skills to identify and test vulnerabilities may not be available, even when best build or deployment practices are followed. Surveillance, health checks, test probes, log analysis -- each of these can require considerable specialization: tools, test environments and expertise.

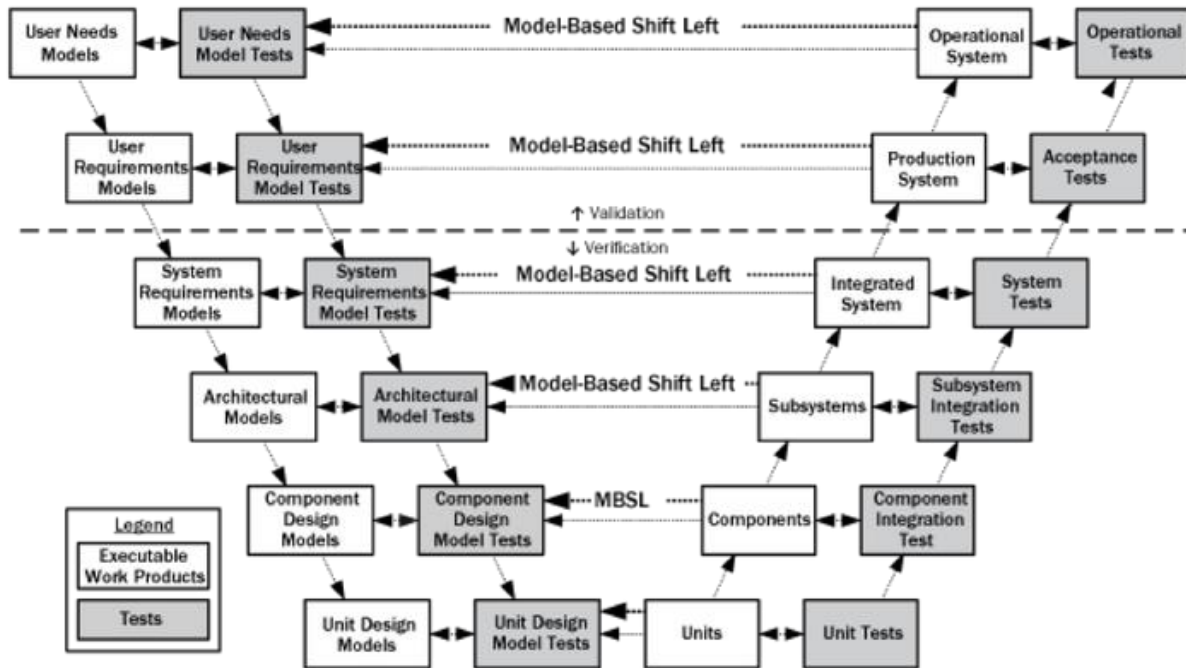
To address this complexity, software technologists -- developers, test, quality, and security engineers -- must work with high levels of abstraction. Abstractions are a force multiplier; by using models and other constructs, they enable limited resources to accomplish more. For security teams, abstractions also enable DevSecOps and automation, key to prompt response to a greater diversity of threats, as well as greater variety of alerts from layered defenses.

For instance, consider the emerging security models that were identified in 2015 by Firestone. Each of the models depicted can be complex, requiring a dynamic mix of supporting infrastructure, rich metadata, domain-specific awareness, and talent.



Information Security Models (Sandhu, 2009)

It's also argued that models are essential to left-shifted testing. The approach depicted by Firestone shows how models can inform the full range of application build, test and operations. Security is a fabric that must be draped over each of these processes.

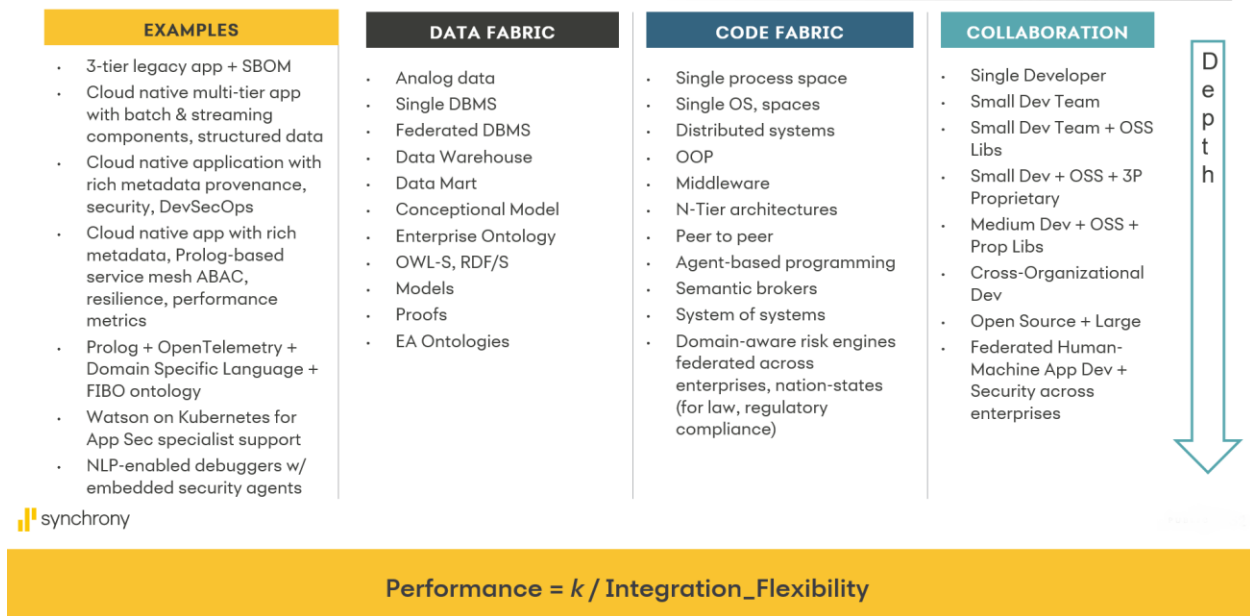


Model-based Shift Left Testing, Figure 5. (Firestone, 2015)

Even the mere *interpretation* of myriad alerts emitted by complex applications is challenging. Is that alert a problem that security should address or an event that should be processed by a domain specialist? Or both?

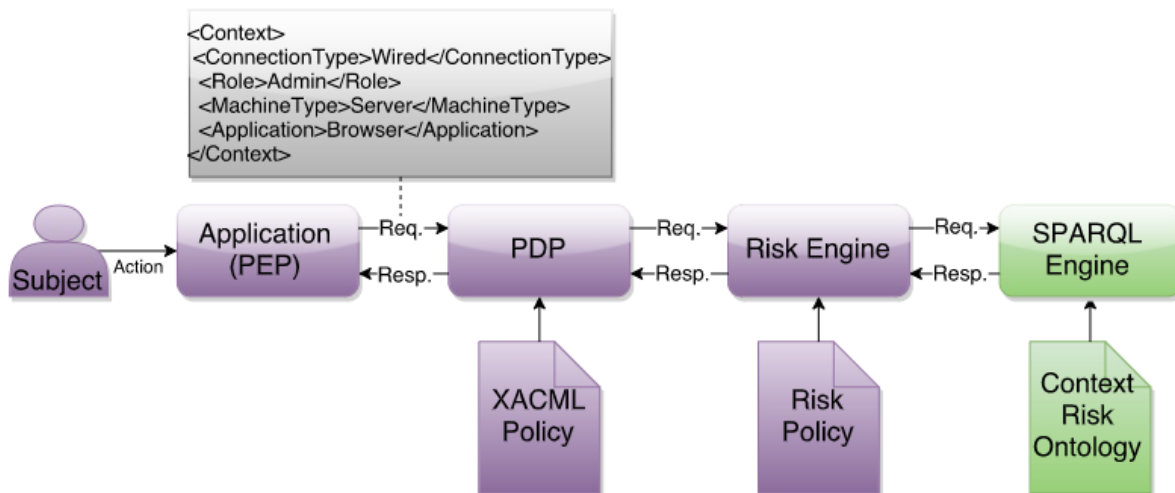
This challenge is depicted in a MITRE analysis (Obrst, 2016). This analysis, in part, identified the need for automated, knowledge-based reasoning within and across applications to assist in securing systems.

Trends in Application Semantics: Tighter Coupling, Explicitness



Trends in Application Semantics: Tighter Coupling, Explicitness (Adapted from Obrst, 2016)

As a result of this gradually evolving understanding – that alerts, events, “data” must be anchored in frameworks that enable automated reasoning – some projects report success in building security automation processes with deeper awareness of applications.



Ontology-based Security Analytics (Riesco, and Villagra, 2019)

In a demonstration by Riesco and Villagra, the processing begins with an application context where Policy Decision Points (PDP) exist, such as an ETL operation into a data lake, a user web form login, or an attempt to connect an API end point. Policy declarations can be represented in eXtensible Access Control Markup Language (XACML), processed, then moved to a risk engine to further decision-making.

A risk engine can do more than grant or deny access; armed with additional domain-specific knowledge, it can require additional authentication, send notifications, check for special circumstances (such as pandemic-related exceptions). It can launch coordinating events, such as machine learning to detect unusual behavior or fraud scoring using an enterprise model. In this project, the additional domain awareness is enabled through the SPARQL Protocol And RDF Query Language (SPARQL), a SQL-like semantic web language which consults knowledge structures. The knowledge structures can be assembled using the humble building blocks of key-value pairs.²

What’s the underlying challenge? Costa et al. (2018) argued:

“Cyberspace is a highly dynamic man-made domain with a high degree of uncertainty and incomplete data which must be transformed into

² Key-value pairs are assembled into a database-like structure called an RDF triple, subject / predicate / object. In MongoDB, the native document key-value pattern is mapped to an RDF format through a plugin.

knowledge to support precise and predictable cyber effects estimation. Current systems have to rely on human subject matter experts (SMEs) for most tasks, rendering the cyber asset planning process too time consuming and therefore operationally ineffective.”

Researchers in the cybersecurity ontology community believe that by supplementing current cybersecurity systems with automated reasoning, some of these concerns can be mitigated. Toward this end, MITRE has sponsored a community effort, the Unified Cybersecurity Ontology (UCO) <https://github.com/Ebiquity/Unified-Cybersecurity-Ontology>. At the least, such tools can improve human-machine collaborations through knowledge-based processing of alerts, countermeasures, threat models and adversary tactics.

Future Implications

This survey of application security covers important trends but leaves still other topics unmentioned.

Not addressed, for instance, is the importance of data science for security analytics, or the emergence of machine learning models in tools such as Exabeam and CrowdStrike -- models whose training sets, capabilities and limitations may not be fully visible to or understood by security teams. But even in those omissions the importance of a model-based understanding becomes clear. Application security teams must partner with model owners. Future work will consist of tasks such as Hardening R, reviewing Scikit-Learn script libraries, or implementing data controls for TensorFlow.

In the game of chess with adversaries, AppSec teams need to strive for grand master status. AppSec is increasingly the superset above other cybersecurity specializations. It's all about the code, and that code will have to be defended if the incentives -- like ransomware -- line up to attract adversaries. Security teams which cultivate minimal knowledge of diverse, complex applications are inviting attacks by more sophisticated adversaries whose developer skills will be weaponized.

References

R. Sandhu, "The PEI framework for application-centric security," *2009 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2009, pp. 1-5, doi: 10.4108/ICST.COLLABORATECOM2009.8382.

Firesmith, Donald, "Four Types of Left Shift Testing," Software Engineering Institute Blog, CMU, 2015-03-23, <https://insights.sei.cmu.edu/blog/four-types-of-shift-left-testing/>, accessed 2021-08-14.

Smith, Larry (September 2001). "Shift-Left Testing". *Dr. Dobb's Journal*. 26 (9): 56, 62.

Devopedia. 2021. "Shift Left." Version 5, June 28. Accessed 2021-09-09.
<https://devopedia.org/shift-left>

Riesco, R., & Villagr a, V. A. (2019). Leveraging cyber threat intelligence for a dynamic risk framework: Automation by using a semantic reasoner and a new combination of standards (STIXTM, SWRL and OWL). *International Journal of Information Security*, 18(6).
<https://doi.org/10.1007/s10207-019-00433-2>

F. Fischer *et al.*, "Stack Overflow Considered Harmful? The Impact of Copy & Paste on Android Application Security," *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 121-136, doi: 10.1109/SP.2017.31.

Obrst, L. "The Ontology Spectrum & the Range of Semantic Models," MITRE Corporation, Information Semantics Group, 2016.

K. A. Torkura, M. I. H. Sukmana, F. Cheng and C. Meinel, "Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications," *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, 2017, pp. 90-97, doi: 10.1109/SmartCloud.2017.21.

C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud application architecture patterns", in *Cloud Computing Patterns*, Springer, 2014, pp. 151– 238

N. Kratzke and P.-C. Quint, "Understanding cloud native applications after 10 years of cloud computing: a systematic mapping study", *Journal of Systems and Software*, 2017.

Brooks, F. P. (1996). The Computer Scientist as Toolsmith II. *Commun. ACM*, 39(3), 61–68. <https://doi.org/10.1145/227234.227243>, accessed 2021-10-13.

Williams, Jeff. "The Staggering Complexity of Application Security," in *Dark Reading*, 10 November 2014, <https://www.darkreading.com/application-security/the-staggering-complexity-of-application-security>, accessed 2021-08-15.

451 Research, “Addressing Complexity and Expertise in Application Security Testing,” 26 October, 2020, https://info.whitehatsec.com/rs/675-YBI-674/images/451_Advisory_BIB_WhiteHat.pdf, accessed 2021-08-15.

Sargent, Jenna. “Microservices at scale: A complexity management issue,” in *SD Times*, 2 July 2021, <https://sdtimes.com/microservices/microservices-at-scale-a-complexity-management-issue/>, accessed 2021-08-15.

Bray, Tim. “Discouraged Developer,” blog post, 17 July 2021, <https://www.tbray.org/ongoing/When/201x/2014/07/17/Discouraged-Developer>, accessed 2021-08-15.

Cassel, David. “Where is the complexity Of modern software coming from?” in *The New Stack*, 18 October 2020, <https://thenewstack.io/where-is-the-complexity-of-modern-software-coming-from/>

Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, and Emerson Murphy-Hill. 2018. Advantages and disadvantages of a monolithic repository: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18)*. Association for Computing Machinery, New York, NY, USA, 225–234.
DOI:<https://doi.org/10.1145/3183519.3183550>

Andonov, N. “WordPress Malware Camouflaged as Code, *Wordfence* post, 13 August 2021 <https://www.wordfence.com/blog/2021/08/wordpress-malware-camouflaged-as-code>

Killian, C., Ferguson, A.D., Gribble S., et al. “Orion: Google’s Software Defined Networking Control Plane,” *18th USENIX Symposium on Networked Systems Design and Implementation*, 5 May 2021, https://youtu.be/W_tSOj6hDKU, accessed 2021-08-16.

Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., Germano, P., Kanagala, A., Provost, J., Simmons, J., Tanda, E., Wanderer, J., Hölzle, U., Stuart, S., Vahdat, A. Jupiter rising: A decade of clos topologies and centralized control in Google’s datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (2015)*, ACM, 183197.

Haranas, M. Google's Data Centers To Use 'Carbon-Intelligent Computing', *CRN*, 19 May 2021, accessed 2021-08-15. <https://www.crn.com/news/data-center/google-s-data-centers-to-use-carbon-intelligent-computing->

Attack Vector references

<https://blog.sonatype.com/malware-removed-from-maven-central>

https://owasp.org/www-pdf-archive/Microservice_Security.pdf Unsafe deserialization

<https://www.hardening-security.com/vulnerability/cve-2020-9480> (Apache Spark exemplar)

<https://alibaba-cloud.medium.com/alibaba-cloud-security-team-discovers-apache-spark-rest-api-remote-code-execution-rce-exploit-a5fdb8fbd173> Spark RCE REST API

<https://opsmatters.com/videos/cloudsploit-aws-cloudformation-security-scanner-demo>

<https://www.routerfreak.com/9-types-software-defined-network-attacks-protect/> (SDN attacks)

<https://thenewstack.io/the-biggest-security-risks-lurking-in-your-ci-cd-pipeline/> CI/CD risks

<https://www.semanticscholar.org/paper/Secure-Scrum%3A-Development-of-Secure-Software-with-Pohl-Hof/ece4559a2c0b15aa8fe57297482a22a961bc4ccf> Secure Scrum

<https://breanneboland.com/blog/2020/01/27/how-an-sre-became-an-application-security-engineer-and-you-can-too/>

<https://portswigger.net/daily-swig/open-source-ecosystem-ripe-for-dependency-confusion-attacks-research-finds>

<https://www.inguardians.com/wp-content/uploads/2021/06/WWHF-Kubernetes-Attack-and-Defense-RealGenius.pdf>

<https://docs.cloudera.com/runtime/7.2.9/kafka-securing/topics/kafka-secure-deltokens-hardening.html>

