Impact of Pricing Tiers on Posture

n4jm4

Classical pricing models carry a substantial impact on the overall security posture of applications and other software components.

Consider the C programming language. It is often the ass of cybersecurity jokes. Yet, C is armed with a powerful shield that trendy dynamic languages lack: The humble, static type checker. Python can’t shake a stick at it.

In clang and gcc, the type checker feature are available at no financial cost, with no rate limits. Consequently, security bugs arising from data type and arity manipulation are significantly reduced. By comparison, dynamically typed projects experience a flood of bugs. Which propagate downstream to infect other software projects, due to the lack of access to type checking. Some engineers spend a lot of time working around limitations in dynamic type systems, even going so far as to fuzz their projects. But most are unaware of the problem, unaware that mitigation tools and techniques exist. Regardless, for even those who practice such workarounds, nonetheless the system at large remains vulnerable.

Consider Go. Its notoriously bloated executables can stretch download times to a crawl. Yet, Go’s garbage collector wards off an incredible number of security bugs.

In Go, the garbage collector feature is available at no cost, with no rate limits. Consequently, security bugs arising from memory management are significantly reduced. By comparison, projects with manual memory management experience a flood of bugs. Which propagate downstream to infect other software projects, due to the lack of access to safe memory management. Some engineers spend a lot of time working around limitations in manual memory managed systems, even going so far as to fuzz their projects. Many are unaware of the problem, unaware that mitigation tools and techniques exist. Regardless, for even those who practice such workarounds, nonetheless the system at large remains vulnerable.

Consider Node.js. Its weak type system is the sustenance of clowns. Yet, its `npm audit` SCA command catches dependency vulnerabilities dead in their tracks.

In Node.js, npm audit is available at no cost, at sufficiently high rate limits so as not to impede any engineers, no matter how prolifically they publish packages. `npm install` automatically generates security report summaries at the earliest opportunity. Providing invaluable data at the developer fingertips. Consequently, security bugs arising from consuming vulnerable dependencies are significantly reduced. By comparison, projects with opt-in, costly, SCA tools experience a flood of bugs. Some engineers spend a lot of time working around limitations in enterprise SCA systems, even going so far as to SAST third party source code. Many are unaware of the problem, unaware that mitigation tools and techniques exist. Regardless, for

even those who practice such workarounds, nonetheless the system at large remains vulnerable.

Some bright cybersecurity efforts publish their work as FOSS resources. But too many cybersecurity firms instead sequester their tools, technical documentation, and findings behind paywalls. That paranoid model made more sense in the 1980s, before FOSS blossomed. Today, even the smallest of applications among the sea of software we rely, on is typically based on hundreds of FOSS components. None of which allocate funds for commercial cybersecurity tools. By fixating on enterprise customers, we leave an absolute feast on the table for attackers. The upstream components are unaccounted for.

Cybersecurity software pricing models occur along quite strange boundaries like that. The upstream project, the most impactful resource to secure, tends to receive no security features or support from cybersecurity funds. Enterprise engineering teams that consume FOSS components, contribute neither enhancements nor basic security patches. On a good day, replacement of vulnerable portions of the tech stack occurs. But not repair.

Cybersecurity software pricing constructs artificial boundaries around platform features. SCA vs. SAST vs. DAST vs. other capabilities. They charge by which shields we hold. And so projects with any funds at all for security, fail to purchase essential scanning capabilities. They pay for SCA but not neglect SAST and DAST. They purchase SCA but do not purchase “IaC” features that close gaps in the base SCA/SAST product offerings.

As a reminder, the effective degree of security posture is not the amount of blood, sweat, and tears spent on any one particular portion of the system. It’s the weakest link in the chain. Whatever products are not purchased, whatever features are unavailable to the legions of nonprofit software projects everyone uses, results in an ecosystem with effectively no security Imagine the number of security bugs left unreported due to commercial SAST systems not proactively scanning each project on GitLab alone, nevermind other prominent code exchanges.

It would be one thing for firms to charge by the number of bytes processed. From here, it looks like security firms are betting on whales to rush order “the works” far too late. The dentist who happily hands out candy, benefiting knowingly or unknowingly from repeat customers.

In terms of container security, we have no security. Container scans inherently involve higher storage requirements than other forms of SCA scans.

Rate limits for security scans wouldn’t fit one developer, one seat, one ticket, one stop along the Hello World train. Star development teams and personnel are punished for being prolific, since every project, every interaction with a security system quickly eats into rate limit credits. This creates a perverse incentive: Don’t contribute, build monoliths, don’t scan early, don’t scan often, or in fact at all.

CI/CD system fails to provide sufficient disk space to be able to run container scans to completion. This seems to be true of many free CI/CD services. The problem is further

worsened by the impulse to shove everything and the kitchen sink into CI/CD agents, wasting valuable resources that could easily go toward container scans.

Beyond security, there is a tectonic shift to convert formerly good citizen FOSS projects into proprietary mazes. How many attacks must we suffer until this changes? What kinds of laws, charities, education, etc. etc. would encourage healthier FOSS contribution from all parties? Maybe consumer malaise is responsible for the industry's descent into bitrot.

There's presently nothing to distinguish secure customers from insecure ones. Once shocking news articles of cybersecurity debacles and privacy advocates alike, essentially national defense debriefs, have since crashed below the noise floor.