Securing APIs: The New Application Attack Surface

Version 1.4
Released: April 2021

Securosis, L.L.C. 515 E. Carefree Highway Suite #766 Phoenix, AZ 85085 **T** 602-412-3051
info@securosis.com www.securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on [the Securosis blog](), but has been enhanced, reviewed, and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Securing APIs

Table of Contents

Application Architecture Disrupted 4

Modern API Security 8

Empowering Security 12

About the Analyst 17

About Securosis 18

Application Architecture Disrupted

When we think of disruption, a common image is a tornado coming through and ripping things up, leaving towns leveled and nothing the same. But disruption can be slow and steady, incremental in how gradually everything you thought you knew changed. Securing cloud environments was like that, initially trying to use existing security concepts and controls, which worked well enough. Until they didn't, forcing a re-evaluation of everything we thought we knew about security. The changes were (and for many still are) challenging, but overall very positive.

We see the same type of disruption in how applications are built, deployed, and maintained in most organizations. Macro changes include the ongoing cloud migration disrupting the tech stack, new application design patterns bringing microservices to the forefront, and DevOps changing dev/release practices. As we've been slowly navigating this sea change, the common thread across these changes is increasing reliance on Application Programming Interfaces (APIs).

APIs have quickly emerged as the most attractive and least-protected target within new applications because they have access to critical data and services.

For security, this new dependence on APIs changes the source of risk: it's not just the front end under siege from traditional attacks and reconnaissance to map out backend processes. APIs have quickly emerged as the most attractive and least-protected target within new applications because they have access to critical data and services. So we decided to document this disruption and its impact on how we need to view application security moving forward.

This paper will work through how application architecture and attack surfaces are changing, how application security needs to evolve to deal with these disruptions, and how to empower security in environments where DevOps rules the roost. Because that is the way.

Application Architecture Today

Let's start with how we see application architecture evolving. There's no one size that fits all requirements, and it's unlikely all these aspects apply to your current situation. But we're confident you will encounter these changes — it's just a question of how much and when.

- **Smaller:** First let's highlight microservices. This approach breaks traditional monolithic applications down into sets of services woven together using defined APIs. It adds modularity (yes, we used to call it reusable components), flexibility, and consistency, because developers don't need to reinvent the wheel. It's also heavily dependent on open source components which provide the basis for many services.

- **Faster:** With the embrace of DevOps practices across many application teams, the objective is to eliminate the typical walls between Development and Ops (and Security, to a point), to create shared accountability and focus everyone on not just *building* but *deploying* and *operating* applications at higher velocity and with better resilience. A key to making DevOps work is leveraging automation to manage deployment. Automation spans from code check-in to testing (including security tests), and ultimately through deployment into production. How can you manage CI/CD (Continuous Integration/Continuous Deployment) pipelines and all the ancillary services they orchestrate? Through APIs, of course!

- **Cloud-Native:** The computing platforms where applications run have also evolved significantly. Given the requirements above for modularity, flexibility, and velocity, applications need to run in a more agile infrastructure. It might be public or private cloud, containerized, serverless, or a combination. When we say *cloud-native,* that can encompass all the permutations — not just containers. Regardless, you interact with your computing platform via (you guessed it, right?): APIs. And increasingly infrastructure is described as code, which increases the application surface for security testing.

Another hallmark of modern application architecture is assembling applications instead of *writing* them. Using pre-built microservices to get started, building only the components you need, enables you to weave the application together without writing *everything* from scratch. This approach democratizes technology and enables business professionals to play a more prominent role in building the applications they need, potentially without the need for IT's "help." That's a bit harsh but it's where we're headed.

The reliance on APIs to integrate the components of the application stack and facilitate data exchange makes APIs a sweet target for attackers, so let's examine what the attack surface looks like.

The reliance on APIs to integrate the components of the application stack and facilitate data exchange makes APIs a sweet target for attackers.

The API Attack Surface

Per usual in security, protection starts with visibility. You have options to enumerate the API environment: you can leverage an inventory (such as a Swagger file repository, if it exists) or discover APIs via scanning and network monitoring, although monitoring offers limited API context.

But visibility doesn't only help you. Attackers can (and do) use the same techniques to enumerate your API surface. Especially given that API requests and responses may travel over accessible networks, and Swagger files are often accessible in public git repos (either intentionally for public APIs, or inadvertently for private APIs). This breadcrumb trail provides an opportunity for attackers to discover API parameters, and potentially access application data.

API Attacks

OWASP has done an excellent job of documenting standard API attacks in its OWASP API Security Top 10 list. These attacks range from the simple, like randomly changing resource IDs to discover and access other customers' data (Insecure Direct Object Reference), to the more advanced, such as brute force attacks, to identifying weak links in API authentication. The list also includes input attacks meant to cause API failures, as well as traditional flaws like buffer overflows.

More complicated attacks involve gaming an application's permission structure by invoking admin-level APIs without proper authorization or authentication. We also see application defects such as excessive data exposure when an API returns more data than necessary, or sends entire data sets to the API caller. Finally we have availability attacks, such as Denial of Service against the API to overwhelm the system.

API attacks share similarities with other application attacks: attackers can target application logic, input, availability, or permission structure. Unfortunately, they always seem to find weak links.

Traditional Defenses Miss Modern Attacks

We are drawing analogies to traditional application security, so we need to consider how traditional defenses work against these attacks. By traditional defenses we are talking about WAFs, API Gateways, and managed application security services. Let's highlight some challenges in using these defenses against API attacks.

- **WAF:** Within the context of new API attacks, think of a WAF as equivalent to an email gateway trying to stop a web-based attack. It speaks a different language. A WAF can detect some API attacks (such as injection that's clearly an application attack), but the proxy architecture and limited rule sets present limitations in defending APIs.

- **API Gateway:** These gateways emerged to centralize API traffic for performance and reliability. Security was mostly an afterthought, and as with WAFs they are limited in the protection they can offer. Basic malformed requests, brute forcing attempts, and injection attacks are simple to handle, so long as they follow well-defined patterns. But anything aiming to enumerate API surface or exploit logic or excessive data flaws will go undetected.

- **Managed Application Security Services:** A new class of managed offerings has emerged to combine WAF, DDoS protection, API gateways, and some bot mitigation, to create a combination service. They are typically offered by CDNs or cloud providers, since they see all the traffic anyway. Using a managed version of the previous solutions addresses some operational issues but cannot remove limitations in the underlying technologies.

Additionally, tactics such as application security testing also have a place in securing APIs by scanning Swagger files to find potential vulnerabilities and exposures. To be clear, we are **not** saying these traditional approaches are irrelevant in our new API-centric world. *They are necessary but not sufficient.* It's not a matter of either/or. The point we'll make through the rest of this paper is that you need to consider API security as an additional aspect of protecting critical applications, not just a part of existing application security tools and processes.

To be clear, we are **not** saying these traditional approaches are irrelevant in our new API-centric world. *They are necessary but not sufficient.* It's not a matter of either/or.

Modern API Security

An API Security strategy requires more than traditional application security. Traditional application security tactics, including SAST/DAST for security testing, and WAFs and API Gateways for threat protection, are critical parts of any application security program. We need to build on existing application security structures to protect modern applications and APIs.

So what does an API Security strategy look like? We wouldn't be analysts if we didn't think in terms of process and lifecycle. We've practiced security for decades, and one of the only truisms which has held up over time has been *visibility, then control*. There are a hundred ways to describe it, such as "you can't manage what you can't see," and they are all true. Let's use that prism to take a closer look at API security, starting with visibility.

API Visibility

The key to any security visibility effort is to figure out what data is needed, and then where you can get it. First start with the APIs you know about, which are documented. That leads you to the various API specifications, which provide details on the operations the API supports, its parameters and functions, authentication and authorization requirements, and other relevant information. With documented specifications you can figure out what each API does and identify potential security issues.

The key to any security visibility effort is to figure out what data is needed, and then where you can get it. First start with the APIs you know about, which are documented.

The reality is that developers probably haven't fully documented all the APIs in use, or might have failed to keep the documentation current as the APIs change. They're busy shipping code, don't you know? Kidding aside, you can make a strong case that building documentation as the API is defined is the *right* way to do things, but that doesn't always happen under the pressure of deadlines. So we need other ways to identify API usage.

1. **API Gateways:** Despite the ongoing debate around whether API gateways provide real security value, they definitely offer a central point to manage the performance, authentication, and authorization of existing APIs, routing requests to appropriate destinations. That means they mediate API traffic and can provide rich data about API usage.

2. **Application Security Testing:** Although it's not the most efficient way to discover APIs, you can scan each application or IP address space to enumerate available APIs and determine which web interfaces are open and potentially exposed.

3. **Passive Monitoring:** Finally you can look at traffic on the network to identify and enumerate API usage in the data you see flying past. A similar technique monitors networks to identify endpoints, and can even perform vulnerability scanning without endpoint agents.

Once you find the APIs, it's time to ensure data exposed through them do not violate compliance policies or regulatory mandates. It is possible to tackle the task manually, though rarely scalable considering the volume of API traffic and the wide range of PII in most organizations. Some API security offerings provide a capability similar to Data Leak Prevention (DLP), identifying common sensitive data types (SSN, Account IDs, health records IDs, cardholder numbers, and other forms of PII) and scanning for sensitive data exposed via APIs.

Detection and classification are only the first steps. You need to figure out the proper operational response once you discover sensitive data incorrectly made accessible via an API. Who receives a notification, and under what circumstances will you block an API response? But that's getting a bit ahead of ourselves. At this point the focus is still on finding potential exposure of sensitive data.

Detection and classification are only the first steps. You need to figure out the proper operational response once you discover sensitive data incorrectly made accessible via an API. Who receives a notification, and under what circumstances will you block an API response?

Once you have a handle on the APIs in use and any sensitive data accessible through them, we recommend building and maintaining a comprehensive API inventory. New and changed APIs can be compared against this inventory to quickly determine what changed and whether it complies with security policies. This process is very useful for tracking API attack surface to ensure adequate protection. Again, while it is possible to maintain such an inventory manually, most organizations should seek the aid of tooling to automatically maintain a current API inventory. Speaking of protecting APIs…

Securing APIs

Protection starts with understanding the threats that you face. We went through some attacks previously, but selecting the right protection requires understanding the threat model. With the large spectrum of business logic and data exchange APIs are designed for, as well as the interconnectedness of modern architecture, producing threat models is often difficult or even impossible for many organizations.

For API security, you need to protect against many threats including authentication and authorization failures, denial of service, vulnerability exploits, business logic abuse, sensitive data exposure, privacy impacts, and more. Some of these are covered in the OWASP API Security Top 10, but that is just the tip of the iceberg. Increasingly organizations face sophisticated automated attacks such as content scraping and credential stuffing or (throttled) brute forcing to achieve account takeover. Such attack types do not fit the exploit patterns that traditional threat protection mechanisms such as API gateways and WAFs are designed to catch.

You can search on “OWASP top API attacks” to find sites with detailed descriptions of the OWASP Top 10 attacks alongside mitigation techniques, so we’ll focus on the capabilities you need to protect against *all* attacks.

1. **API Scanning:** The first step in protecting an API is to make sure it doesn’t have API definition issues, if the definition exists. The API security capability should also be able to produce such API definitions based on actual traffic. Basically this capability provides a static API scanning capability which looks for weak authentication and loose definitions for parameters, responses, payloads, etc. This scanning capability should check APIs against the organization’s security policies and trigger automatically within DevOps build pipelines during deployment. Analogous to application security testing, these API scans can be either static (looking at the API code or API definition) or dynamic (sending incorrect data to the API to trigger misbehavior or exploitable conditions).
2. **Detection and Blocking:** If it looks like an attack it probably is, and an API security solution must be able to detect and block attacks such as those described in the OWASP API security list. To enforce a positive security approach, you may also want an API security solution to explicitly allow only the parameters set in the API contract.
3. **Anomaly Detection:** Shocking as it may sound, new analytics driving better detection of attacks still use a similar approach to network anomaly detection, which first appeared 20 years ago. These solutions use improved algorithms and analytics built for an API context. This translates to more accurate baselines, which enable API security solutions to define *“normal”* API traffic down to the user or process level. This enables detection of obfuscated, low and slow attacks, and helps to discriminate between innocent activity and malicious intent. As APIs change it’s essential to keep the baseline current, to maintain this context and inform security controls, which would be unreliable without advanced analytics.

As you consider an API security solution you’ll be pulled into the age-old question of inline (requiring a proxy or agent implemented within each micro-service or container) or out-of-band (monitoring the infrastructure for API activity and integrating with existing proxies). Inline solutions deploy within the application’s data path, so they can enforce policies and block attacks directly. But this means you need to install code within each application or micro-service or instantiate additional proxies, which incurs extra processing and inevitably adds latency.

The alternative out-of-band approach involves monitoring traffic to all APIs. These solutions discover hidden (or unpublished) APIs through monitoring traffic, and don't add application latency. But they require integration with other solutions (API gateways, firewalls, etc.) to block attacks.

What about the application security defenses you already have? As discussed above, these tools (specifically WAF and API Gateways) aren't particularly well suited to provide protection against the wide range of potential API security issues. They do well enough on simple attacks identified with signatures, particularly attacks that target off-the-shelf software. But they lack the application and API context (available from API contracts and baselining API traffic) to block API attacks, which are more sophisticated and subtle.

But at some point developers may need to make changes to underlying API code to address security issues, so they need to understand how to remediate and why these changes are important. *"Because security said so,"* isn't viable over the long term.

Finding API security issues is one matter, and stopping an attack with runtime controls is another. But at some point developers may need to make changes to underlying API code to address security issues, so they need to understand how to remediate and why these changes are important. *"Because security said so,"* isn't viable over the long term. It didn't work well in the world of waterfall processes, and it doesn't work well with agile methodologies and modern development practices.

Empowering Security

Historically enterprises have taken baby steps to adopt new technologies; experimenting and finding practical boundaries to meet security, reliability, and resilience requirements before fully committing. Because companies must trade off security against speed, it often takes years for new technologies to achieve widespread usage. But today's businesses don't have that luxury – the mandate is *move fast and break stuff*, innovating fast enough to satisfy customer demand.

Organizations need to find a middle ground where they can implement security as part of the tech stack. They need to ensure adherence to security policies, including protection of critical data, while moving fast enough to deliver new business value in each application sprint.

As a result, DevOps organizations don't play by the old rules governing IT adoption of new technologies. DevOps arose because corporate IT couldn't move fast enough. DevOps teams adopt technologies first, and may ask for permission later. Agile methodologies and DevOps practices promote iterating quickly and failing fast. To hit the right balance and innovate safely, organizations need to find a middle ground where they can implement security as part of the tech stack. They need to ensure adherence to security policies, including protection of critical data, while moving fast enough to deliver new business value in each application sprint.

The Promise of DevSecOps

Getting organizations aligned to deliver secure applications has *always* been problematic. Incentives and metrics for development teams focus on delivering code on time and within budget. Security can impact those goals by forcing changes and delaying shipment of new features. Even when security finds an issue and prevents a crippling data breach, it's still tough to be the bearer of bad news. So even when security is right, they are often perceived to be wrong. More importantly, in the eyes of the business, development is often viewed as revenue-generating, while security is an expense.

Doesn't DevSecOps change all that? The idea is to build security into development and deployment processes from the start, and integrate and automate security testing directly in the pipeline, so security becomes everyone's business. In this manner, security *shifts left* (yes, another buzzword) and happens earlier in the development cycle. In effect, DevSecOps makes the entire system more secure, right? That's the promise. The reality can be much murkier — especially when you start factoring in multiple build pipelines, staffing challenges, and a mix of legacy and modern technology.

Now, let's add another factor to increase the potential impact of DevSecOps: Infrastructure as Code (IaC). Everything is code in this world — not just applications but also APIs and infrastructure elements such as networks, servers, load balancers, etc. These DevSecOps concepts apply to the entirety of the tech stack. Very exciting indeed!

But once again the reality is a bit different than the promise. DevSecOps requires a genuine cultural shift to topple the traditional walls separating Dev, Ops, and Security. Many DevSecOps initiatives have been scuttled by politics and organizational resistance to change, particularly when there isn't sufficient buy-in from the top. Of course DevOps and agile development are happening, and fighting against the future is not viable over the long term, but facing resistance to the new way of doing things certainly complicates things in the short term.

Finally, DevSecOps doesn't mean security becomes an equal partner. The reality remains that security findings are still issues, and they are lumped together with new feature requests and bug fixes when application sprints are defined. Security needs to fight to get changes included in each sprint, and doesn't always win.

How do these broader challenges relate back to our API Security topic? It turns out that pretty much every modern development initiative (yes, particularly DevOps) uses APIs heavily. So securely coding and testing APIs is an integral part of DevSecOps. To reach DevSecOps utopia we need to ensure developers have adequate training, a means to ensure there aren't issues with the API code or schema definitions, and automated checks on the code as it moves through build pipelines.

The deployed code is still at risk for manipulation, misuse, and business logic errors, which no amount of pre-deployment testing can ever catch. Why? Because those gaps are evident only at runtime – they require API execution to manifest.

There's No Time Like Runtime

Let's assume (however unlikely) that your DevSecOps initiative goes perfectly. The DevOps teams get it, and they've instrumented the CI/CD pipeline to ensure API security policies are tested and enforced before any code deployment. This aspect is likely only one component of a much larger design, and it's still only half the battle. The deployed code is still at risk for manipulation, misuse, and business logic errors, which no amount of pre-deployment testing can ever catch. Why? Because those gaps are evident only at runtime – they require API execution to manifest. So what's a security lead to do?

Deploy the tried-and-true solution: runtime security. Runtime is where you catch misuse, drift, human error, and other issues that violate application or API security policies after deployment. You need runtime monitoring to detect these issues. This API and application security monitoring can look an awful lot like other monitoring techniques, though context is king for APIs even more than in other areas. Logging and monitoring events or transactions is one thing, but piecing it all together to provide application and API-layer context is the name of the game. You start by collecting and aggregating data about application/API usage, and then watch for signs of misuse. And misuse can only be identified by establishing baselines built from your own company's applications and APIs. You will need to look for clear attack patterns (Indicators of Compromise and Attack), and use advanced analytics (machine learning) to detect when application usage varies from a typical baseline. You then also need to distinguish anomalies that are simple user error from those indicating malicious intent.

So what happens when you discover a security issue? Who is responsible for fixing it? Is it Ops? Does a developer have to update the code in the template immediately? Security's role (or lack thereof!) in fixing security issues can cause frustration among security folks, especially when the Ops team doesn't perceive the same level of urgency to address the issue. As we've described, DevOps happened because IT wasn't responsive enough to the business, so DevOps teams certainly don't want to go back to the old ways of waiting for someone in Security to get around to fixing their stuff. Additionally Security brings context that Dev and Ops lack, because they aren't immersed in security all day, every day. So it works much better when Security and DevOps can work together to address runtime issues.

How do the two teams find a workable middle ground? It's a concept we call *guardrails*, which are security policies the organization cannot violate. We've taken to calling them a very technical term — *no-no's* — because these are things that must *never* happen in a production environment. When a guardrail trips, Security is empowered and expected to fix the issue. Everything else goes into the normal queue of issues and defects to address in due course by Dev or Ops during a regular sprint.

How do the two teams find a workable middle ground? It's a concept we call *guardrails*, which are security policies the organization cannot violate.

Of course there are challenges to implementing guardrails for APIs — security issues don't always follow well-defined patterns. But observing application traffic and API usage, with machine learning to detect non-standard application and API behavior, can help define guardrails, and keep them current as the APIs change.

Another point of caution is that the no-no's require careful consideration — they trigger a *take action now, ask questions later* response which violates normal expectations and roles. For API Security we recommend you start with the OWASP API Top 10, the most common and potentially most damaging issues. Of course Security needs the right tooling to identify these violations at runtime and shut them down when appropriate.

Fool Me Once...

Whether remediation happens via an automated guardrail or is performed by the Ops team, once you address the immediate issue you need to think about taking a more strategic approach. If you keep handling issues on a case-by-case basis, and don't put in the work to prevent them, you will just keep playing Whack-A-Mole during runtime. How can you squash as many of these issues as possible, early in development, and replace the detection-and-response dance with prevention?

Sadly, developers don't come out of the proverbial womb understanding security and safe coding. Conversely not all security practitioners understand the inner workings of application code and systems design. We all need to keep learning. We recommend a *Security Champions* program, where developers take on additional responsibility to represent security within their DevOps teams. This aligns with another critical role for Security in the API-centric DevOps world: *providers of guidance and education.*

This training approach also works from the other perspective, where security teams get more involved with development teams and workflows. This dual-pronged approach creates amplifying effects which help to reduce friction between teams and improve awareness.

Any discovery of a security issue offers a teachable moment, when developers can learn how to avoid making the same mistake again. It's also essential to ensure that you are testing for the security issue within the pipeline (as well as at runtime, of course), just in case it takes the developer a few times to get it right. What's important is that developers learn the lessons of detected security, and security monitoring technology ensures issues are not missed next time. Security issues, particularly incidents and breaches, also offer learning opportunities for security teams, and can be a catalyst for scrutinizing security tool effectiveness and identifying other issues in play beyond developer misfires, ultimately improving security processes.

Everybody in Alignment

The key to success in shipping secure code is to ensure that alignment exists within the organization, including a collaborative relationship between Security and DevOps. It's essential to embrace teams' mutual dependence to reduce friction. DevOps cannot meet its objectives without Security, and vice versa. If these teams view themselves as adversaries instead of partners, things will not work. This observation seems intuitive and straightforward, but human nature requires we find someone to blame when mistakes happen. In modern IT mistakes are inevitable. Organizations need to focus on how quickly they can detect and respond to failures. And *post-mortem* analysis should focus on what went wrong rather than whom to blame.

It's critical to make very clear that everyone is on the same team, with aligned objectives. The organization needs these teams to deliver the most functionality possible, on time, within budget, with strong security.

With that, your objective is clear. The development and deployment of modern applications, including a heavy dose of APIs, requires a new and different security approach. It's about more than just shifting left and integrating testing into the pipeline – you also need a clear understanding of the application attack surface, so you can empower the Security team to find and address the issues that present the greatest risk.

If you have any questions on this topic, or want to discuss your situation specifically, feel free to send us a note at info@securosis.com.

About the Analyst

Mike Rothman, Analyst and President

Mike's bold perspectives and irreverent style are invaluable as companies determine effective strategies to grapple with the dynamic security threatscape. Mike specializes in the sexy aspects of security — such as protecting networks and endpoints, security management, and compliance. After 20 years in and around security, he's one of the guys who "knows where the bodies are buried" in the space.

Starting his career as a programmer and networking consultant, Mike was an analyst at META Group prior to founding SHYM Technology, and then held executive roles at CipherTrust and TruSecure. Mike then started Security Incite in 2006 to provide a voice of reason in an over-hyped yet underwhelming security industry. After taking a short detour as Senior VP, Strategy at eIQnetworks, Mike joined Securosis with a rejuvenated cynicism about the state of security.

Mike published *The Pragmatic CSO* <http://www.pragmaticcso.com/> in 2007 to introduce technically oriented security professionals to the nuances of what is required to be a senior security professional. He also possesses a very expensive engineering degree in Operations Research and Industrial Engineering from Cornell University. His folks are overjoyed that he uses literally zero percent of his education on a daily basis.

About Securosis

Securosis, LLC is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services. Our services include:

- **Primary research publishing**: We publish the vast majority of our research for free through our blog, and package the research as papers that can be licensed for distribution on an annual basis. All published materials and presentations meet our strict objectivity requirements, and follow our Totally Transparent Research policy.

- **Cloud Security Project Accelerators**: Securosis Project Accelerators (SPA) are packaged consulting offerings to bring our applied research and battle-tested field experiences to your cloud deployments. These in-depth programs combine assessment, tailored workshops, and ongoing support to ensure you can secure your cloud projects better and faster. They are designed to cut months or years off your projects while integrating leading-edge cloud security practices into your existing operations.

- **Cloud Security Training:** We are the team that built the Cloud Security Alliance CCSK training class and our own Advanced Cloud Security and Applied SecDevOps program. Attend one of our public classes or bring us in for a private, customized experience.

- **Advisory services for vendors**: We offer a number of advisory services to help our vendor clients bring the right product/service to market in the right way to hit on critical market requirements. Securosis is known for telling our clients what they NEED to hear, not what they want to hear. Clients typically start with a strategy day engagement, and then can engage with us on a retainer basis for ongoing support. Services available as part of our advisory services include market and product analysis and strategy, technology roadmap guidance, competitive strategies, etc. Though keep in mind, we maintain our strict objectivity and confidentiality requirements on all engagements.

- **Custom Research, Speaking and Advisory**: Need a custom research report on a new technology or security issue? A highly-rated speaker for an internal or public security event? An outside expert for a merger or acquisition due diligence? An expert to evaluate your security strategy, identify gaps, and build a roadmap forward? These defined projects bridge the gap when you need more than a strategy day but less than a long-term consulting engagement.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors. For more information about Securosis, visit our website: <http://securosis.com/>.