PRACTICES TO SHIELD YOUR APIs FROM ATTACK

SOLSYS

hello@solsys.ca
solsys.ca
Solsys LinkedIn

TABLE OF CONTENTS

SOLSYS

APIs at Risk

One of our customers, a leader in the Canadian Telecom Market, found that the need for a single business innovation service gateway was inhibiting business and imposing unrealistic business costs.

Their legacy API Gateway was missing many features, more over the code was outdated with no way to observe or measure traffic. This platform was performing the vital role of securing access to many business-critical APIs.

With planned significant increases in API usage on the horizon, driven by new business services to be offered by Media, Health, IoT, and 5G, Solsys was engaged to work with them to establish a next-generation API connectivity platform. With an eye towards quality and security, an essential part of this engagement has been to modernize security to address the API attack vector risk while retiring the legacy system.

APIs as an Attack Vector

Our client is right to be concerned about API attack vectors. They are large and increasing security concerns for corporations, at the same time that API usage is increasing and becoming critical to strategic business operations and for enabling innovation with partners and cloud services. According to Forbes and as

predicted by Gartner, 2022 saw a large spike in API security threats. The API attack vector is still a high-risk issue in corporations, and 2023 kicked off with T-Mobile revealing the theft of the personal data of 37 million of their customers from an exposed API endpoint in the last few months.

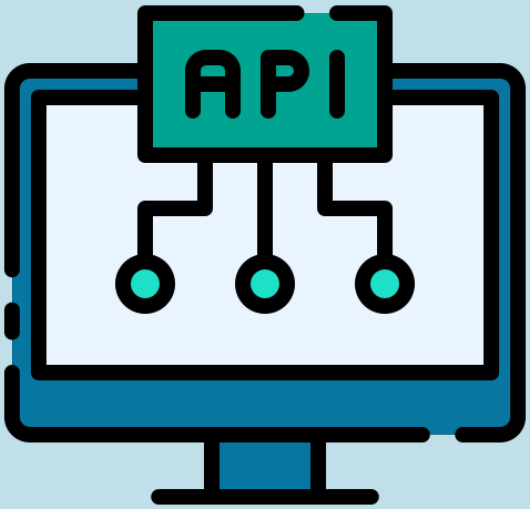
Our Security Journey & Lessons Learned

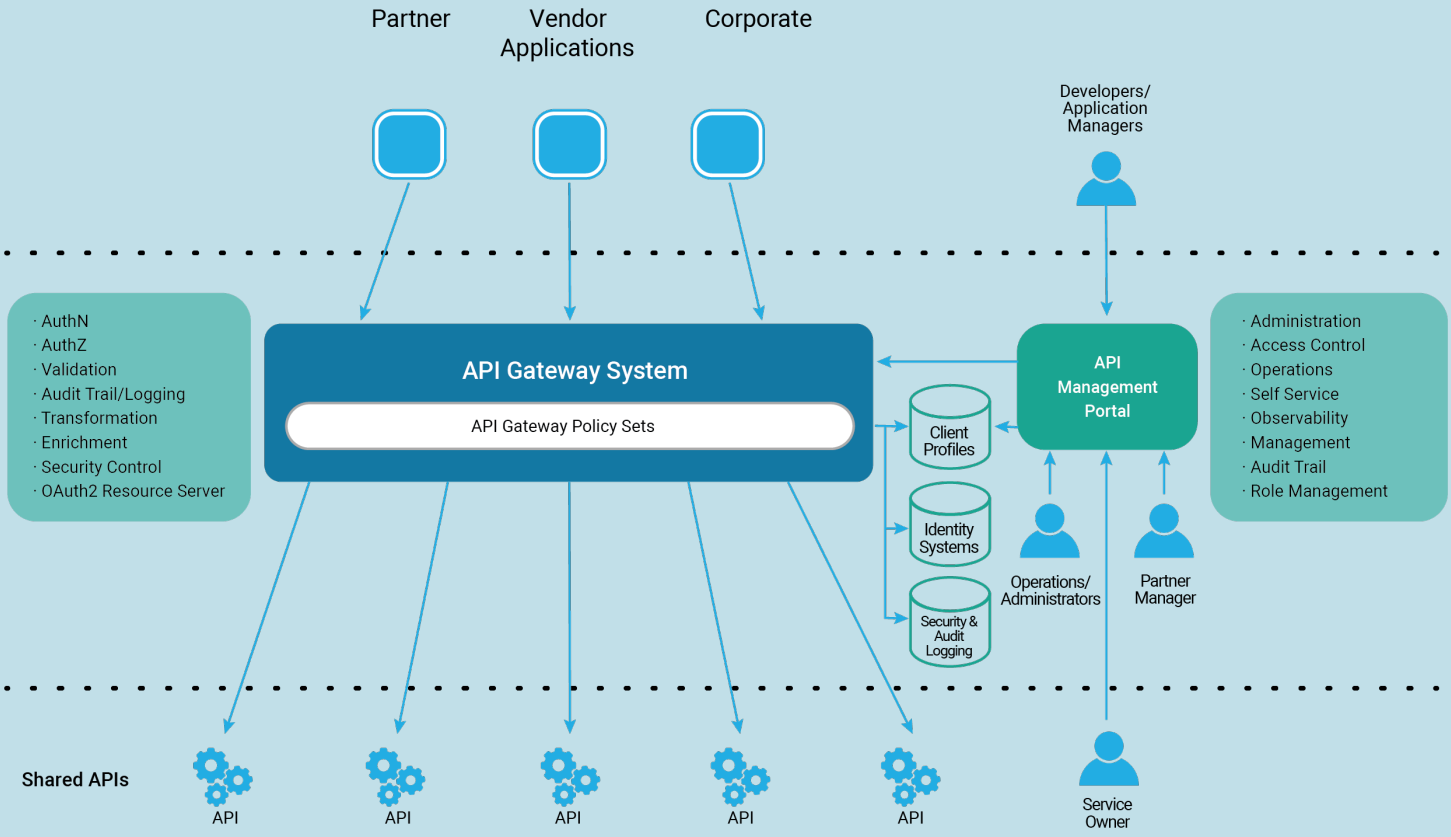
In our multi-year journey with our client, we have significantly improved their security posture, provided full observability and security audit events, retired their legacy high-risk API platform, and delivered many API management capabilities to enable innovation and faster time-to-market.

In this ebook, we will share the practices of our product development teams in building and securing APIs and implementing a defense strategy with this customer to protect and enable the use of their API assets. We will discuss the practices (activities, approaches, tools, platforms, systems, ideas, architecture,

and designs) that have enabled our development teams to securely develop this secure API platform product with our Canadian client and how we continue to ensure that this software remains secure. While the title of this ebook relates to API security and this journey, the practices discussed in this series apply to all software development.

Later in the ebook, we will discuss specific security risks and how these working practices are applied to help avoid them. We will be reviewing the OWASP top 10 security risks, and explaining how the practices we have outlined help to avoid these problems.





SECTION 1:

15 Practices to Shield Your APIs from Attack

Practices 1-4: Approaches & Activities

Practices 5-9: Ideas, Architecture, and Design

Practices 10-15: Tools and Systems

SOLSYS

Approaches & Activities

These approaches and activities are extremely valuable to improving security during software development and largely revolve around DevSecOps Teams, Automated Tests and Threat Analysis.

01: DevSecOps Product Teams

Use DevSecOps Product Teams enabled with ownership of security & operations, agency to prioritize security, and security related tooling.

02: Automated Security Testing

Full automated testing of security posture using programmatic tests, work prioritized and automated by the DevSecOps team.

03: External Security Services

Externally provided penetration testing and security reviews are vital inputs to the product team.

04: Security Consulting as a Service

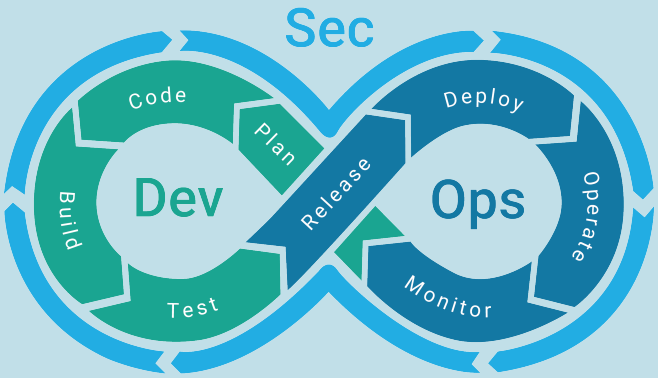
Ensure teams have access to security expertise and guideance when they need it.

01 DevSec Ops Product Teams

Dedicated DevSecOps development teams are a key approach to ensure a reduction in security exposures. Dedicated product teams that own a product's code base, can iterate independently and are aware of security as an ongoing requirement can be very effective at reducing security risks. Enabling these teams with tooling and training is important, but also allowing their Product Owner to work with the team to prioritise security requirements and repairs, is also critical.

Enabled DevSecOps teams tend to work as a group to look at the big picture with a product. Owning deployment and configuration (CI/CD) pipelines ensures they have an eye for pushing products and configurations consistently and reliably.

Stable DevSecOps teams require business sponsorship to ensure consistent dedicated team members over time. From a business perspective, stable and consistent teams owning a product pay off. Not only can these teams build a sense of pride and ownership around their work, but the ability to maintain and adapt these products to new business needs becomes rapid and costs less overall than bringing in people every time changes are required. Our client measured in one example year that they saved $100K in terms of administrative overhead alone, by simply being able to ask for particular changes to the product from an existing team and not having to go through the expenses of engaging new



people every time. This saving does not even measure the business case offered by having this product team able to rapidly respond to these new business needs and implement a solution sustainably, ensuring the solution is secure, tested, and fits well within the product codebase.

02 Automated Security Testing

Teams concerned with product security and quality as a whole tend to prioritize automated testing. Our two teams working on this particular system have over ten thousand (and growing) fully automated programmatic functional tests for the platform. Programmatic just means that the tests (even for user interfaces) are orchestrated by functions in the code. This means that if part of the interface changes, the functions dealing with that part of the interface are quickly changed, and the orchestrated test continues to pass. It is worth knowing that the DevSecOps team as a group owns the development and maintenance of all these tests so that tests

are applied at the time of the development or change in a feature. In fact, our team with this client spent several years doing this testing without any specialized or dedicated QA Developer on the team at all.

Several thousand of these automated are exclusively concerned with validating security policy and 'edge case' conditions that could result in a security failure or exploit. This is over and above countless unit tests in the code. These tests are programmatic, meaning the development team maintains the functions and steps that enable them every sprint as they are building functions and features. Tests have to pass completely to have a candidate build to deploy to any other system, and the tests are baked into the CI/CD pipeline. Agile teams might need to change any part of their code base regularly, and Test Driven Development (TDD), and good functional test coverage help teams design code for change.

03 External Penetration & Security Testing

Product teams benefit from feedback. As such, we also ensure that external code evaluation and penetration testing by a third-party group is engaged from time to time. This type of testing gives the team insight into potential improvements or risk areas and may find existing gaps in the system.

These activities can be performed without much engagement from the product team until it is time to review the results and decide how to prioritize identified improvements. This is a great 'service' for established partners or security departments to offer to product teams in their organization.

External tooling can help the team a lot. In a future post, we'll discuss how some of these tools work within automation pipelines to measure and check for security issues. However, running external scanning tools periodically as a 'state-of-the-nation' assessment update is also valuable. An example of external tooling our DevSecOps team has used is an SSL checking tool, like the SSL Labs from Qualys.

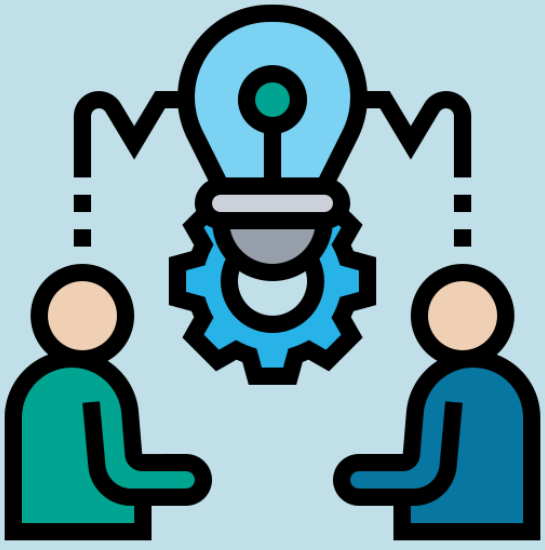
Empowering our DevSecOps team to own the product and prioritize security and quality work has enabled us to act on the input from security reviews quickly, depending on risk and priority. Agility and automation are what enable our teams to roll out fixes to any identified gaps at low cost and close any high-risk findings in production rapidly and with confidence.

04 Security Consulting as a Service

It is unlikely that every team has the expertise for all types of security questions. While a cross-functional DevOps team should include the required day-to-day security skills whenever possible, there will always be situations where additional expertise and advice is needed, and sometimes that means more than finding online articles.
Our team benefits greatly from approachable security professionals in other parts of our customer's organization. Sometimes this was because we needed to know what systems or software may already exist to meet a need, or if we were building some secure component, that the approach we were taking made sense and was addressing the correct things. Sometimes we need to know if specific security policies exist around a particular type of software implementation, and being able to reach out to other experts can help.
A prior example in our case was when we needed to implement SSL certificate signing and verification services. Some of this function was supported in the products we were already using, but parts of it were not. When considering something like a certificate signing service, we wanted additional support to ensure we were complying with our customer's organizational standards and concerns, and that we weren't building something that we could leverage from elsewhere in the organization.

In our case, using security expertise from a trusted professional within the customer's organization enabled us to meet these goals. We were lucky to find that our specific customer experts are approachable and supportive in providing direction and advice. So often in large organizations employees tend to avoid security groups for fear of them shutting down work until they can 'fully review' the design, audit the implemented code, or other 'work stopping' blockers. In these cases, security is sometimes seen as an obstacle to avoid, rather than a consultation that will help.

It takes awareness and trust to work with Agile teams who are dealing with constant change and support them as they try to improve. A security department with a service mindset will be more readily engaged by those teams, will have much greater influence over those teams, and therefore greater success at improving an organization's security.



Ideas, Architecture, & Design

Practices 5-9: Concepts, Cryptography, & Credentials

These ideas, architectures, and design are extremely valuable to improving security during software development, especially around API exposure. These concepts are either specific architectural ideas related to APIs, or 'best practice' approaches or ideas related to networks, infrastructure, or software programming in general.

05: Network & Credential Encryption

Use best practices behind network set up, and data encryption & protection, especially with regards to credentials.

06: Rotate, Repave, Repair

This concept is common in security thinking and is enabled through pipeline automation (CI/CD pipelines).

07: Policy Sets

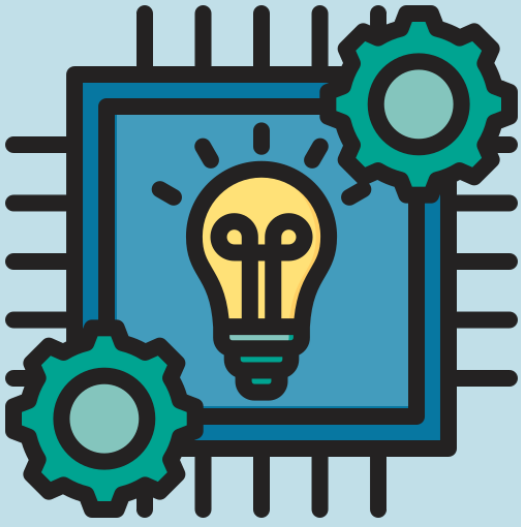
This architectural idea is related to API deployment and a mechanism to ensure accepted security exposure patterns are encoded into API deployment choices to prevent security deployment errors.

08: Identity & OAuth2 Policies

This API specific architecture implements policies that work with the identity platform to enforce identity validation at the request level, and ensures APIs are defended at the Resource Gateway.

09: eXtreme Programming Concepts

Some of the eXtreme programming ideas can help teams to improve code security by spotting errors faster, reducing complexity, and focusing on 'fit for purpose' design hygiene.



05 Network and Credential Encryption Practices

Simple network security approaches are key. Always insist on the basics like using HTTPS everywhere, and locking down ports by default (deny-by-default, only open ports as needed, and ideally only open endpoints within private subnets). We also focus on common credential 'hygiene', such as storing credentials in an encrypted format, and providing interfaces so customers can self-serve changes to their API access keys and credentials. It is paramount to avoid transmitting API secrets to API customers over insecure channels such as email. This practice also keeps us on the right side in a 'non-repudiation' situation, whereby only clients have knowledge of their credentials and are responsible for their security. This restricted access does, of course, require an interface where a specific customer is authorized to manage a particular application profile, but others are not. Whether a security setting is changed via an API or the management portal, we track ALL security events in detail, including who changed what, when, and from where. These details enable future investigations into activities like credential change or management. We'll cover more in a future post about Observability & SIEM.

All these concepts are kept in a simple checklist, maintained by the team, to review when they decide to make architectural or software changes. Other security agreements and team practices are captured

in other checklists and are a working agreement between the members of the DevSecOps team.

06 Rotate, Repave, Repair - Automation & APIs

Another practice captured and encoded into our build and deployment pipelines, are the concepts of Rotate, Repave, and Repair (the Three R's of security). Wherever possible and practical, the product team ensures that their pipeline pulls the latest versions of containers or libraries into the product build process. Even if a firmware, library, or platform is a manual step and decision, the pipeline and fully automated tests enable the team to validate the new change (e.g. a library replacement), with confidence that no defects have been introduced with the update. Staying current on the latest versions of dependencies helps to close security gaps because older code tends to be more exploitable (not just entropy at work, but also attackers have had more time with the older code to discover and exploit gaps that were there all along).

A popular infrastructure vendor announced a critical security flaw in their system. Our team was able to redeploy that infrastructure on the latest (fixed) version quickly and validate its functionality, prior to swapping out the component at risk. This swift response was only possible because we automated our deployment, set-up, and testing. The result? Speed, security, reliability, and no impact on customers - hallmarks of our team as a result of these practices.

07 Policy Sets: Security Patterns Pre-Encoded

Our API gateway introduced a specific deployment pattern for APIs in production that our team invented. Called 'policy sets', we have designed the workflow for the day-to-day operations team responsible for deploying customers' services on the gateway so that they select from security patterns pre-encoded in sets of policies.

Using policy sets means that it is practically impossible for someone on this team, regardless of skill level or experience, to configure and deploy an API without at least basic levels of authentication and security logging. Other policies can be selected from the set or are required to be configured in the set, depending on the pattern the policy set represents.

08 Identity, OAuth2 and the Resource Server

One of these policies and policy sets is responsible for enforcing OAuth2 security. Solsys had previously worked with our Canadian customer around Identity and Access Management (IAM) to establish an Identity Platform (IdP), which hosts their customer's identity and can issue OAuth2 tokens expressing the customer’s authorized relationship to their products and services.

The API gateway provides OAuth2 as a service to downstream APIs (without API developers having to validate customer tokens in their physical API) and also works with the IdP to enable this policy, when needed, to play the full role of the Resource Server in the OAuth2 model. As a result, our gateway can be configured, on an API and method-by-method basis, to enforce account or product validation for a specific customer.

What this means is that any client accessing an important API protected this way has to provide an OAuth2 token, generated by customer authorization, that the gateway

validates to ensure that the client is only accessing accounts or products for which the customer has authorized access. For example, accessing an account balance by billing account number can require the token to represent a customer who has that billing account associated with their identity profile.

This extra level of checking on the gateway protects the APIs from accidental mistaken access, but also malicious use of software clients' credentials should their API key become compromised.

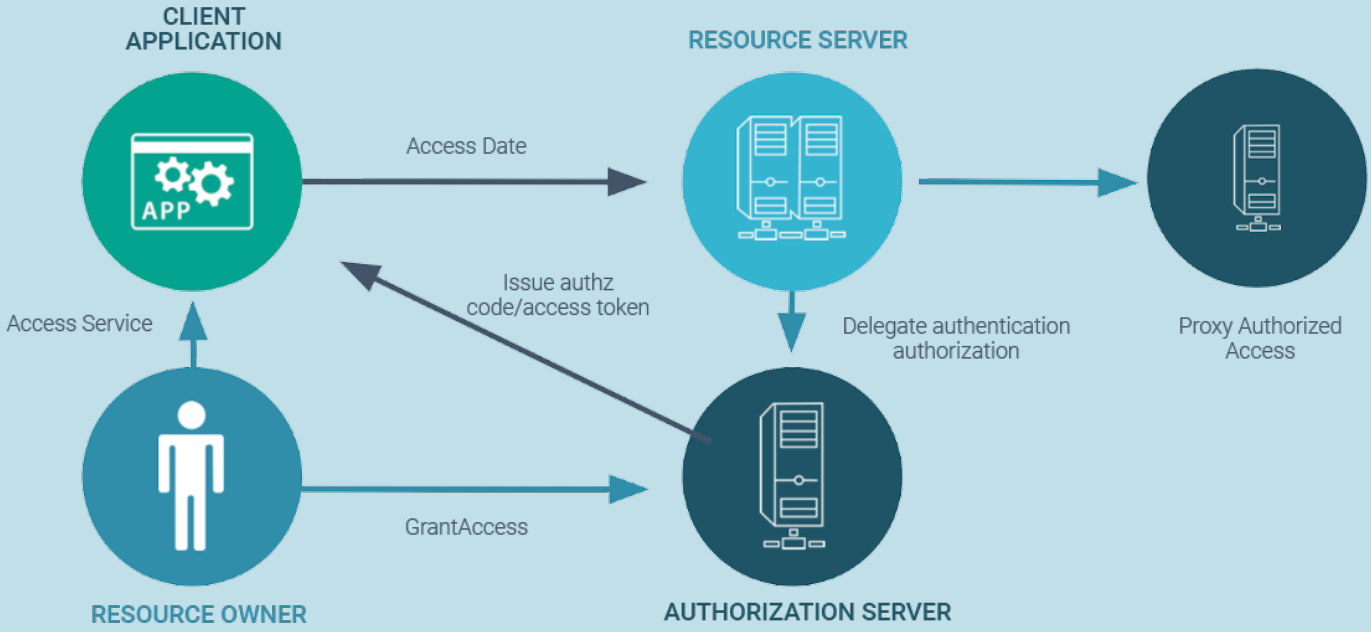
Kong, as an API gateway and critical partner, provides the framework for fully orchestratable and customizable policies. We were able to use this capability to adapt OAuth2 policies to perform the specific validations required and overlay our policy set concept.

09 eXtreme Programming Concepts

Lastly, concepts from the eXtreme Programming (XP) movement play a prominent role in software security in general.

If you're not familiar with XP, many of these ideas pre-date Scrum and the Agile Manifesto. Most of the concepts in XP are programming practices and concepts, whereas in general terms Agile concepts can apply to all kinds of complex work. Going into depth here is out of the scope of this series, and our teams like most, follow XP practices to varying degrees. Some ideas in XP however can really help to ensure software security.

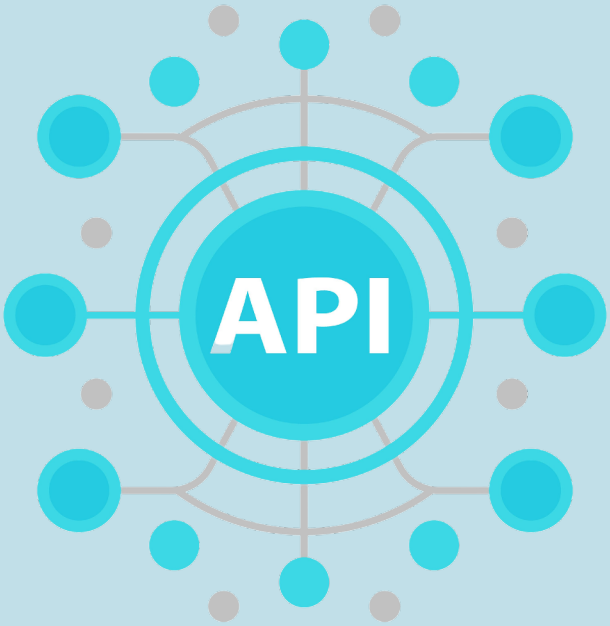
For example, Pair Programming, the idea of two developers working together to produce code, can help a team ensure that common development mistakes are caught



immediately. It also helps teams to be confident that everyone's code is adhering to agreed working practices - including those related to security. Pair Programming tends to be more comprehensive and likely to catch issues than post-development code reviews (although these aren't mutually exclusive practices).

The benefits of simplicity in code, another XP idea - and its impacts on security - cannot be overstated. Building the least amount of code to meet the need is a great way to avoid building software that contains security vulnerabilities. Code that isn't there, can't have a security defect! Our team operationalized this principle during joint design sessions, Pairing, and code reviews. Complexity in code is where we often find unintended behaviours and security holes. Removing unused code, continuous refactoring (boy scout rule as it's sometimes called), and team-agreed coding standards all help towards this goal. Our teams also agree to measure some of these coding standards through code analysis tools.

The concept of simplicity in code can also be reflected in general design hygiene. The idea of keeping designs minimal (simple), but also considering whether a function should even be there or not, is important. For example, having considered RESTful design for any implemented APIs means that the team is more likely to implement an API correctly. There will be fewer 'strange gaps' in terms of how the API works, operations aren't just left in because they might be needed, rather they are built only with intention. Following good design principles also means that your software will make appropriate use of the functions and patterns in an API implementation library instead of working against them. When we use the tools we select for their intended purpose, we're more likely to benefit from their functionality (that might protect us from security problems), rather than introduce unexpected behaviours (which can lead to new security problems).



Tools & Systems

Practices 10-15: Tools & Tech

These tools and technologies (platforms, and systems) are extremely valuable to improving security during software development. These practices cover specific types of tooling and platforms that can be introduced by the team or provided by the organization to support teams developing software products.

10: Code Scanning & Analysis Tools

These tools help detect software security issues as part of an automated CI/CD pipeline. DevSecOps teams can set build breaking gateways around issues as needed to detect issues early in development.

11-13: Technologies for Enabling the Three R's

These sections cover Cloud Platforms, Automation Tooling, and APIs - technologies important in implementing the Rotate, Repave, and Repair practice more easily.

14: Monitoring

Automated monitoring (synthetic transactions) can check and validate security edge cases, while providing continuous output to monitor platform behaviour as it related to security.

15: Observability & SEIM

The practices of making products monitorable - observability and security event logging - can greatly assist in making a product more secure, especially when integrated into a SEIM.

10 Code Scanning & Analysis Tools

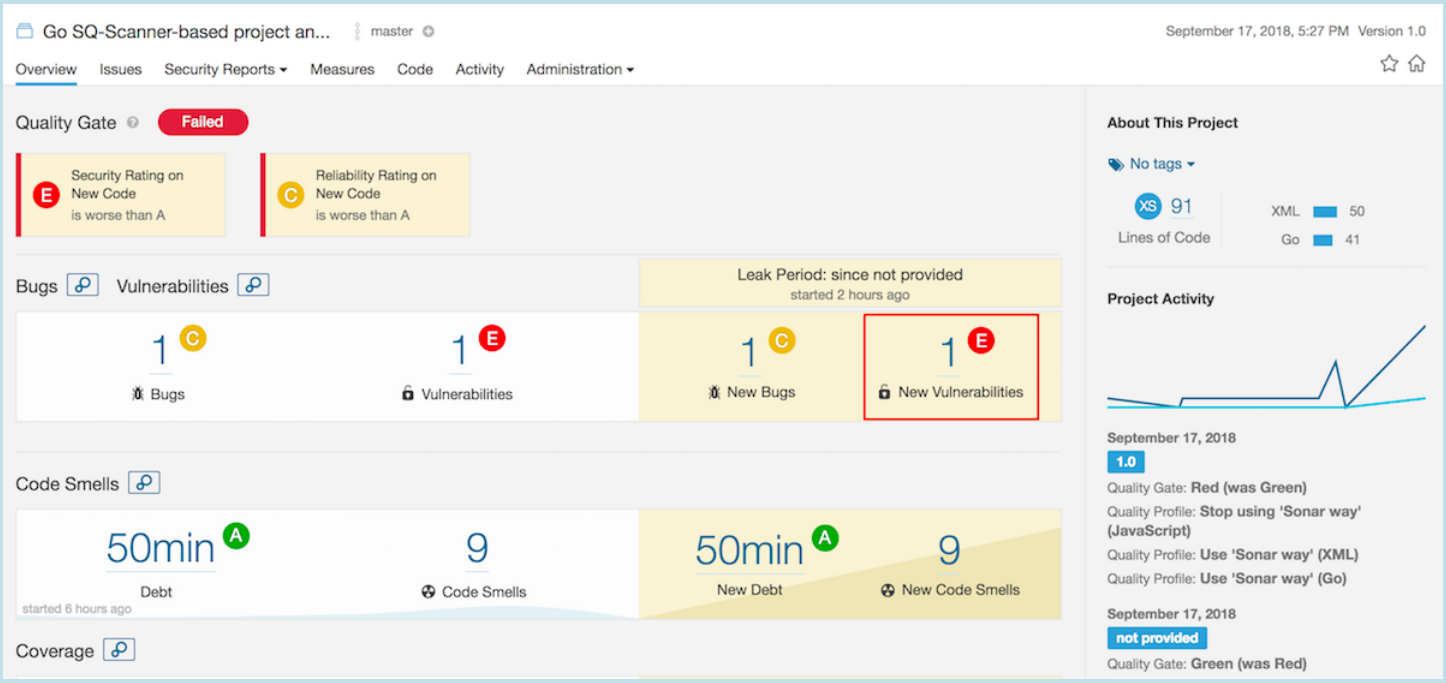
Security tooling is extremely valuable for DevSecOps product teams in providing information to enable them to make decisions about their implementations.

We use services to scan our 3rd-party libraries and report on things like CVEs that are applicable. There are several types of these tools on the market, and the selection of them would be dependent on language, cost, and licensing choices.

Another type of code scanning tool, an example of which is SonarCube, can perform static analysis on code to identify security exceptions or bad practices. Such tools provide metrics for unit test coverage, code complexity and instances of poor security practices.

As the security owner for their solution, the DevSecOps product team orchestrates these tools as well as evaluates the information they provide. While it can be useful to provide advice and training to the team about their tools, the ultimate decision about the impacts of the analysis metrics, and any resulting code changes, is made by the team since they are the experts closest to the current implementation.

Invoking these tools is usually included as a step in the build (DevOps/CI/CD) pipeline to ensure that reports and measurements are kept up to date. The team can even agree to fail a build if certain thresholds in these metrics are exceeded, which helps the team ensure they're sticking to their working agreements. This level of automation is something our teams do in the pipeline to ensure security.



Example dashboard from SonarQube showing a scan of a code base with a possible security vulnerability, along with other metrics

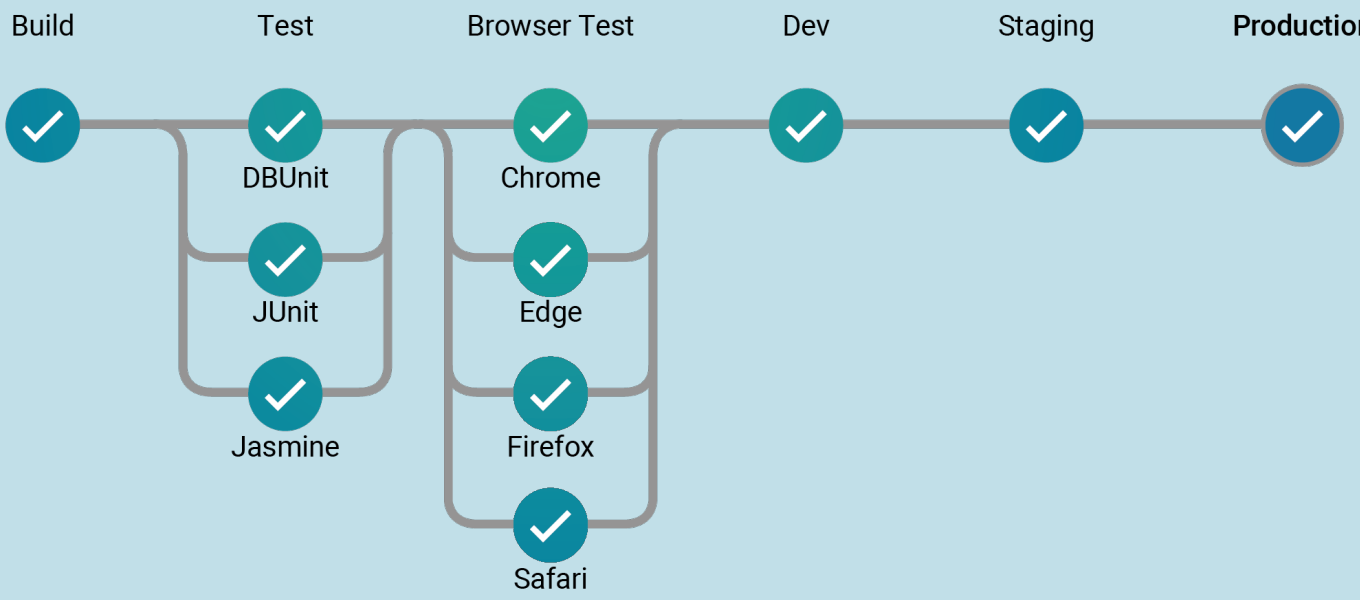
11 Enabling The Three R's - Cloud Platforms

The three R's of security - Rotate, Repave, and Repair - are key to the DevSecOps product team. The ability to execute these ideas, however, is often dependent on tooling. A massive benefit of cloud-based infrastructure, such as AWS or GCP, is that APIs exist to perform these kinds of activities more easily. Cloud Platforms are designed to be easy to orchestrate, configure, and reconfigure. They also provide repositories of updated containers and operating system images since cloud platforms are always virtualized infrastructure.

12 Enabling The Three R's - Automation

Our DevSecOps teams use CI/CD pipelines that automatically perform these activities as part of the deployment, grabbing the latest libraries and containers for deployment purposes, and ensuring stateless API services that enable a 'red green' style of infrastructure change-over, all with little to no developer intervention at deployment time.

In addition to this, our team ensures that credentials and SSL certificates (either internal or for clients) are regularly rotated. Tools like Jenkins and it's associated plugins are usually leveraged here as an orchestrator to this pipeline. There are also many other scripting, code, and deployment orchestration languages being called to achieve full automation, however. These can include tools like Chef, Salt, Ansible, Terraform, and Puppet. All of these languages excel at orchestrating software deployment, calling cloud APIs, and configuring systems.



13 Enabling The Three R's - APIs

Our API management platform provides services to enable clients to rotate and manage credentials themselves and encourages good security behaviour and practice by making it easy to invoke operations via software. If the tools (in this case APIs) are not available to automate tasks like credential rotation, teams are not likely to find the time to manually invoke the processes required. APIs enabled us to write software to 'set it and forget it', which ensures credential security over time.

14 Monitoring

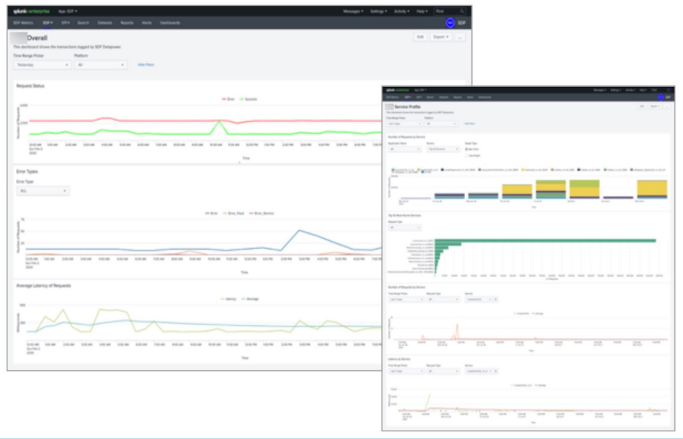
Automated monitoring tools that deliver synthetic transactions provide another means of ensuring security. By validating connectivity, behaviour, and response times regularly, the DevSecOps team is immediately alerted to possible security compromises triggered by errors in the synthetic transaction. We'll talk more about the kinds of security vectors these measures help identify and protect against later in the post on specific attack vectors identified by OWASP's top 10.

15 Observability & SIEM

Observability and SIEM (Security Event & Incident Management) tools play an enormous role in ensuring security. Security breaches happen in the dark when no one is watching, so the best way to improve a system's security posture is to ensure it is well-observed (see our eBook Convergence Security & Observability for more information!). We make it a core principle for our team to know what's happening in our product so we can support it. Being a DevSecOps team means that the people building the product also operate and support it - and no one wants to be sitting in the dark when something goes wrong.

Our DevSecOps team iteratively improves logging and observability information produced by the product over time. Enabling a team to make decisions about what kinds of transactions are relevant to report on to improve operations and security, means the experts on the product are always improving their capability to observe more and increase visibility for possible attack vectors. Observability platforms can be set up to monitor and alert on specific types of events that could indicate a security problem. For us, Splunk provides the observability platform, and our product puts millions of events and tracking records into Splunk daily, to give full insight into the API platform product for our DevSecOps team as well as for external operations and security groups.

A DevSecOps team can't do everything themselves, however. The ability to channel logs into a SIEM platform further enhances security by providing automated use case monitoring of possible attacks. Leveraging machine learning can help to spot aberrant patterns in client behaviour, and highlight possible attacks. Such a platform can check for malicious actors and activity, and alert the team of a possible security gap. This is a great example of a security service that can be deployed by enterprise security to service DevSecOps teams like ours. The product we've built already includes well-instrumented and defined access and audit events that track all the information needed to trigger security events, and exemplifies how the product development team can collaborate with the corporate security team to identify attacks and breaches. In the case of the T-Mobile exposure that happened over several weeks, this type of observability and a SIEM platform could have helped them identify the issue sooner.



Splunk is used with our client as a platform that can provide reporting directly from audit and security logs over 13 months of millions of data events

SECTION 2:

OWASP TOP 10 ATTACK VECTORS

01 Broken Access Control

02 Cryptographic Failures

03 Injection

04 Insecure Design

05 Security Misconfiguration

06 Vulnerable and Outdated Components

07 Identification and Authentication Failures

08 Software and Data Integrity Failures

09 Security Logging and Monitoring Failures

10 Server-Side Request

SOLSYS

01 Broken Access Control

Every few years, the Open Worldwide Application Security Project (OWASP) updates its top attack vectors. To illustrate how our work and the practices we've discussed in this ebook truly help improve security posture, this post goes through the first of the top 10 attack vectors and reviews which of the practices helps to prevent this kind of attack.

Broken access control involves clients obtaining information or operations that they are not permissioned to access. In APIs, access control can be complicated and hard to validate. Of the practices we have talked about, our teams rely on the following to prevent this kind of security risk.

Automated Security Testing

Having the policies involved in access control continually tested prevents accidental failures and new bugs from being introduced. These tests cover failure scenarios as well, and having the team adopt this testing mindset ('how can I break this code?') is important in ensuring these tests have coverage. Part of our testing involves automated validation after deployment to all environments, including production, which allows the team to ensure access control policies are behaving as expected. We also test for valid encryption of identity tokens being used by the system, and tests involve providing fake and manipulated tokens to ensure the system does not accept them - a risk directly identified by this OWASP vector.

Network and Credential Encryption Practices

OWASP calls for deny-by-default as a practice for ensuring no access to unnecessary resources. Our network practice of closing ports by default is a good example of this one.

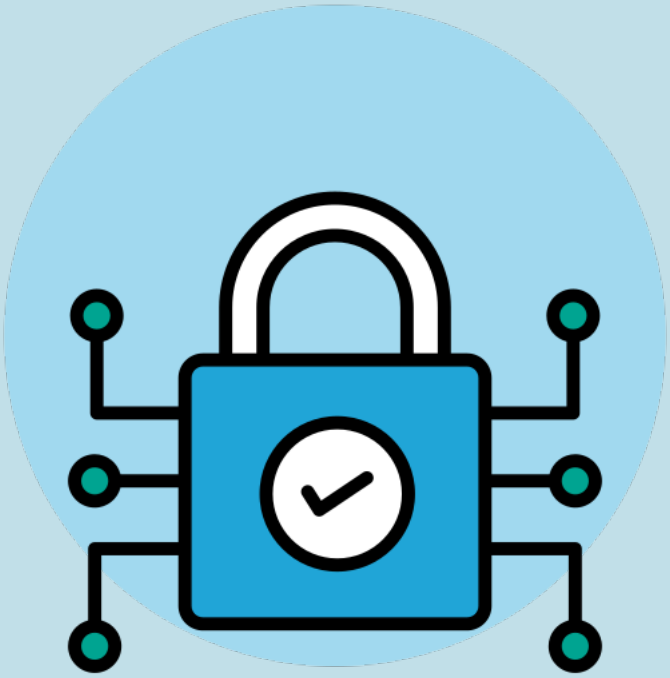
Policy Sets

The concept of policy sets in our API gateway means that we prevent accidental deployment of API endpoints that miss the access control policies. None of the API deployment operators can ever configure an endpoint with missing authentication and access control, as a result of this design. It also means that, implicitly, access control mechanisms are being reused across the system.

Identity, OAuth2, and the Resource Server

Compromised software credentials for one of your clients is almost inevitable. In this scenario, a simple B2B API is automatically vulnerable, granting access to all the operations permissioned for the client, which might mean the ability to access hundreds or thousands of customer accounts and their related information. Our OAuth2 policy in the gateway can protect against this type of breach. Having the customer who owns the data in the API call flow, as identified by an Identity Platform generated authorization token, provides an extra layer of security that makes it virtually impossible for an attacker to brute-force an API to download thousands of

results without also tricking every single customer into authenticating and authorizing their calls at the same time, which is far more difficult, unlikely, and impractical. Does losing thousands of customer records to an attacker sound like a familiar issue? It should be, it happens to corporations all of the time. Such an 'OAuth as a service' policy might have protected the compromised T-Mobile API, depending on the particular attack used. This kind of policy helps our client stay off the front page of the news with this kind of damaging announcement by ensuring access control is specific, not just to the API user, but also to the customer information being accessed.



02 Cryptographic Failures

Cryptographic Failures occur when data is transmitted in the clear, or with old or weak encryption algorithms that enables information to be compromised by a third-party. Of the practices we have covered, our teams rely on the following to prevent this kind of security risk.

Network and Credential Encryption Practices

Lack of cryptography certainly counts as a cryptographic failure. Our practice of "SSL everywhere" and practicing good credential encryption hygiene avoids these issues.

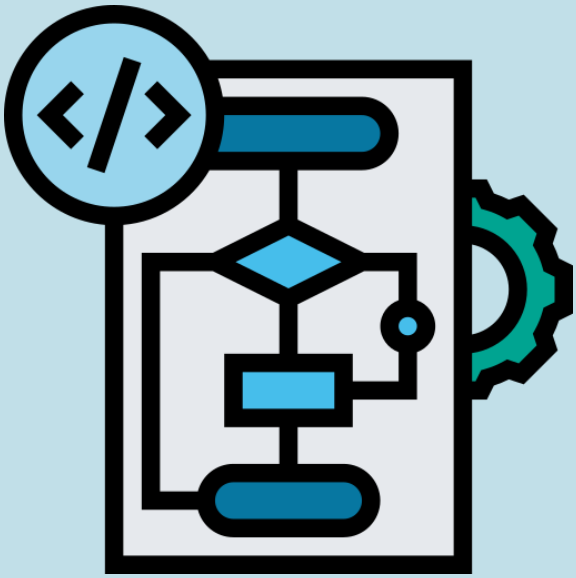
Encryption everywhere (SSL on the network, encryption at rest, etc.), as a default practice, prevents data from being transmitted in the clear. Our Automated Security Testing also validates some of these configurations, ensuring HTTP connections are being appropriately redirected or rejected, for example.

External Pen-Testing & Security Review

Having external teams and tools validate and check for outdated SSL algorithms, for example, helps the team to identify where the product may have weaknesses. Interestingly, just because your product might have to support older SSL encryption protocols for legacy clients, it does not preclude your platform from prioritizing and negotiating for the updated protocols by default, thus protecting customers who can support newer and more secure protocols.

Rotate, Repave, Repair - Automation & APIs

This practice, combined with Automated Security Testing (in fact, automated testing as a whole) means that upgrades to newer versions of a platform or a library that support new encryption protocols can be done confidently without introducing regression errors. The A06:2021-Vulnerable and Outdated Components security risk talks more about these kinds of risks and how this process helps us address them.



03 Injection

Injection attacks involve an attacker tricking your software into executing instructions, supplied through additional or specially formatted parameters.

It's hard to protect against these attacks without following basic development practices and using well-built and supporting software libraries. Checking for and even discovering such attacks can also be difficult. Of the practices we have talked about, our teams rely on the following ones to prevent this kind of security risk.

External Pen-Testing & Security Review

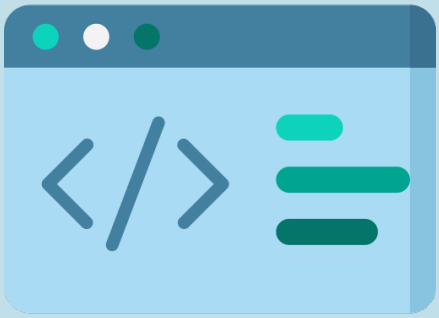
Good pen-testing services will attempt to exploit injection techniques specifically. One of the early reviews of our system found that we were allowing XML style-sheets in a request to reference an off-site URL, which could perhaps expose us to injection or tracking risks. The configuration option to disable off-site requests was missed by the team. Rather than simply fix the issue in the code and move on, the team encoded the check for the issue into an Automated Security Test to ensure the proper behaviour, forever. They also did some research to see if there were similar patterns in other parts of the system. Having the DevSecOps Product Team and their Product Owner empowered to prioritise this work means similar issues have never been detected by future pen tests.

Code Scanning & Analysis Tools

Injection often takes advantage of unparsed or unhandled customer-provided strings. Code analysis tools can check for these mistakes. Additionally, eXtreme Programming Concepts like Pair Programming make them less likely to occur in the first place (the old adage of "two heads are better than one" definitely applies to complex and creative practices like software development).

Observability & SIEM

OWASP requires access control failures to be logged. Our fully-instrumented observability and security events do exactly that. We push them to the Splunk platform for continued monitoring and auditing. Having a SIEM platform that can trigger and watch for unusual activity, perhaps even employing machine learning to detect these aberrant patterns, would further enhance the detection of access control failures.



04 Insecure Design

Insecure Design is not about insecure software implementation, but about missing or ineffective design control. This category is new for OWASP and reflects issues with ill-considered systems architecture, missing consideration for appropriate security risks, or missing software controls. These practices help our team avoid insecure design.

DevSecOps Product Teams

Having accountable teams that can discuss and share ideas about design, and prioritise small experiments to test implementation ideas, tends to help to establish team practices and build working agreements. By incorporating ideas from eXtreme Programming Concepts and approaching the work as team-owned (versus individually owned),we have found that development by DevSecOps teams tends to encourage more discussion about the work being performed and an increased likelihood of making better choices. In addition, the nature of incremental work removes the challenge of "big think" requirements. Answering the question "what will these small changes do to our system's security?"", is easier than asking the question "will this entire system I've designed, and will build, going be secure?".

Automated Security Testing

Once a design choice is made, ensuring that choice is maintained over time is best done by encoding it into a test. For example, we have tests to verify that particular ports are inaccessible, and to ensure that insecure requests (e.g. non-HTTPS requests) are rejected. We also encode choices in our automated deployment pipelines, ensuring that infrastructure and settings are deployed consistently through the explicit steps of a pipeline.

External Pen-Testing & Security Review

Assuming that design issues might occur, having an external expert test the system, and review how things work on a periodic basis, is always a good idea.

Security Consulting as a Service

Access to security professionals within your organization who are available to consult on architecture, design, and implementation choices, readily extends the effectiveness of the DevSecOps team.

Network and Credential Encryption

The practice of "locking-down-by-default" or "deny-by-default" means that failsafe designs are part of the team's mindset. To some extent, leveraging Cloud Platforms also helps with this kind of design, because they tend to have easy-to-automate controls (API control of set-up and service). They also encourage "deny-by-default" network practices and have services that are built for purpose, which makes it easier to enable the segregation of services across system and network tiers.

05 Security Misconfiguration

Security Misconfiguration covers a wide-range of security setup mistakes. There are so many ways security misconfiguration can manifest that it is difficult to suggest that a handful of specific practices will help. A lot of things need to come together to prevent this type of attack. However, here are the top practices that helped us:

Rotate, Repave, Repair - Automation & APIs

This practice makes the process of deployment and hardening repeatable. This requires us to use the automation aspects we discussed in Enabling The Three R's - Automation. Our pipeline runs Automated Security Tests to validate that if a security configuration exists, it's working. A subset of our tests - those most critical to validate that security behaviours are working in each environment - are explicitly shared across multiple environments to verify behaviour throughout the deployment lifecycle. This subset includes negative tests that, for example, ensure things designated as inaccessible, still are, or that protocols we want to reject, are being rejected.

eXtreme Programming Concepts

The idea of simplicity in code specifically means our team works to avoid complexity or unnecessary functions that can leave 'unattended' components in the system with out-of-date or incorrect security

Code Scanning & Analysis Tools

The use of these tools in our pipeline means our team can check for common configuration issues in libraries and set defaults for missed, or poor, configuration settings. Choices like default password configurations, plain-text or hard-coded passwords, or even insecure connections, can all be detected and are alerted to the team, with critical issues causing builds to stop until the offending implementation is either corrected or accepted as a risk by the team.

Enabling The Three R's - Automation

Automation of a CI/CD pipeline to ensure repeatable deployment allows us, for example, to encode into our container configuration firewall lockdown for outbound traffic thus ensuring it is never missed, which often happens in manual deployments.

06 Vulnerable and Outdated Components

Vulnerable and Outdated components risks are system components that have not been upgraded, or that contain known software vulnerabilities. Defending against this vector means being aware of the components you're using, knowing if they have vulnerabilities, and responding quickly to patch them. These are the practices we are following that help us to prevent this risk:

Rotate, Repave, Repair - Automation & APIs

Ensuring we use the latest and most correctly configured platform when deploying the product is important. Specifically, automation allows for the rapid deployment of the entire system. This practice, along with Automated Security Testing (in fact, automated testing as a whole), means that the team can proceed with upgrades to the platform or a library with confidence knowing that a successful test run means no regression errors. Validating our entire product against the latest operating system, container, run-time, firmware, library, etc. with just a few clicks means that upgrading vulnerable components is low-risk and easy, using the automated CI/CD pipeline

Code Scanning & Analysis Tools

In this practice we discussed tools that sit in our pipeline and specifically check for third-party components and versions with known vulnerabilities. This routine drives us to make continuous migrations to new libraries whenever applicable security concerns are flagged within our dependencies.

Enabling The Three R's - Automation

These platforms make it easy to automate the selection of containers and operating systems as part of the CI/CD pipeline. Without infrastructure that can be built and deployed via APIs, it becomes much harder to ensure the latest and least vulnerable versions of containers and operating systems are used.

07 Identification and Authentication Failures

Identification and Authentication failures are all attacks related to poor management of credentials and gaps in authentication. The following practices are instrumental in helping us to avoid these:

Automated Security Testing

Verifying all of our session and token-based authentication (and associated authorization) through automated testing ensures that our chosen practices around credential management continue to be enforced as the product evolves.

Observability & SEIM

As we have learned, being able to see the patterns in the data of behaviours indicative of a security gap means we can catch a brute-force attack, for example, before they're successful.

Network and Credential Encryption Practices

The non-repudiation practices we implemented - credential management best practices - make it much harder for passwords to be passed to people who don't need them within our customer's organization. These tests also validate the Identity token and identity system dependencies we talk about in the section on Identity, OAuth2 and the Resource Server. The APIs we implemented (Enabling The Three R's - APIs) enable clients to rotate credentials regularly and automatically, making credential compromising much harder.

08 Software and Data Integrity Failures

This attack vector involves using code and infrastructure that does not check for integrity violations. Examples might include pulling code or data from content delivery networks without validating it, using libraries that contain malicious code, or updating containers, firmware, or platforms without validating the source. This vector is quite a specific type of attack, but thanks to tools like npm that can be easily triggered to pull in chains of dependency libraries, it is one that can definitely cause issues. These practices helped us avoid some of these problems:

Code Scanning & Analysis Tools

Perhaps the most important of these practices, we use tools in our pipeline that scan all library dependencies for known security alerts and vulnerabilities, including known malicious code and insecure libraries. Alerting the team immediately, and ensuring we review these alerts on a sprint-by-sprint (weekly) basis as part of sprint review meant the DevSecOps Product Team is constantly making informed decisions about possible problems. Included in these scans are the code analysis tools that trigger when bad practices around data management and in the code are detected, and again these are included in the CI/CD pipeline to ensure every change to the product code base is scanned.

eXtreme Programming Concepts

Having developers pair and review code, and make decisions about implementation as a DevSecOps Product Team means the team has practices like the intentional selection of dependencies with more than one person involved in the discussion. Reviewing implementation decisions every sprint means it is less likely that someone grabs a questionable library without some level of discussion around the choice.

Enabling The Three R's - Cloud Platforms

Thanks to the practice of leveraging the cloud and the latest container and operating system images from our cloud provider, we automatically benefit from the practices of the security teams at the cloud provider that make those pieces of software available. Cloud security is one of the top concerns of all the leading cloud providers, and critical to trust in their business. As a result, they will almost always be more focused on, and better at, reviewing the security of platforms they make available, than most of our customers or their development teams. This practice is one more level of assurance provided by cloud platforms and services.

09 Security Logging and Monitoring Failures

Security Logging and Monitoring failures are about gaps in logging and monitoring that occur within a system, resulting in being unable to detect, escalate, and respond to security breaches. This vector is mitigated very well with these two specific practices:

Observability & SEIM

This practice means that we constantly improve our logging and ability to observe the platform. The DevSecOps Product Teams building the product have to support it, so everyone on the team wants to be able to see what is going on. Monitoring practices ensure we always have a mix of good and failed (e.g. unauthenticated) transactions hitting the system to verify activity, even when the platform is quiet. Thanks to all these practices, our team has over 40 billion security and logging-related events at our fingertips (over a year) to look into breaches, find them, track them, and action them. For example, in one instance one particular API that had not been well considered and was actively exposed through the gateway, resulting in some data was being made available, inappropriately, to users of the API. Thanks to our gateway and its level of observability, we were able to immediately audit who had accessed the service, if they had called the particular operation on the API that was exposing the sensitive information, and which accounts had been accessed. This visibility meant that the corporate security team could follow up with partners and affected customers quickly and effectively. Instead of having to publicly inform all customers that they might be at risk as a precaution, they were able to inform only the small group of customers that were actually exposed.

Automated Security Testing

The practice of automated testing helps us ensure we're not failing to log some detail. We have tests that run on every build that verify, based on a transaction, that the log entry is being recorded and contains the important details relating to security. These tests prevent accidentally failing to capture critical information for a security event, and to ensure we are not searching through logs during a possible security breach and wondering where our critical audit information has gone!

10 Server-Side Request

The Server-Side request attack is perhaps the most specific of the top ten vectors. It involves a piece of software fetching a remote resource without validating its destination first. As a result, an attacker can cause an application to request information or services from an unexpected destination, despite other controls (like firewalls) that might try to prevent it. As a specific type of attack, it is hard to identify general practices to prevent it, however, the following help us most:

Automated Security Testing

Our best friend, automated testing, helps to check that specific functionality to validate user-provided URLs (whether via API calls or user interfaces) are checked and validated correctly. For example, making sure that only HTTPS URLs are accepted is important.

Network and Credential Encryption Practices

The practice of deny-by-default comes in here. Leveraging Cloud Platforms also helps us here, because the deny-by-default state of networks and load balancers in our infrastructure are supported and reinforced by cloud services. Cloud platforms also usually provide tools that help to scan your set-up and raise alerts when new routes or access points get exposed. Making sure that our CI/CD pipeline (discussed in Automation & APIs) does infrastructure set-up helps to ensure a consistent set up all the time and that deny-by-default configs are continuously deployed.

Enabling The Three R's - Automation

Pair programming helps developers to spot possible mistakes, like failing to check or validating user-provided URLs, for example. Two heads are better than one, and pair programming builds this approach into your software development.

SECTION 3:

GETTING STARTED

Which Practices are Most Valuable?

How Do I Get Started with These Practices?

Who Can Help My Organization with This?

SOLSYS

And Finally, Getting Started...

In this ebook we have looked at 15 practices to shield your APIs from attack, and shown how many of these practices directly protect us from the top three of the OWASP Top 10 attack vectors.

These practices are applicable to securing all kinds of software development and not just thwarting API attacks, but given the breadth of the problem, you might be wondering how to get started. This section will help you prioritize and begin applying these practices, especially if you have teams already building software.

Which Practices are Most Valuable?

Of the practices we have covered, three stand out as starting points as they are fundamental to progressing to more secure software over the long term. These also appear most frequently when looking at the different attack vectors in the OWASP Top-10.

Rotate, Repave, Repair - Automation & APIs

The Agile Product Team, using a DevSecOps mindset, is the foundation of agility as well as the best place for security ownership: with the developers of the product, at the time of software creation. Working with a DevSecOps mindset requires an organization, as well as each individual team, to develop practices, technical (and soft) skills, and organizational support and mechanisms to enable the team to work effectively. Without improvements in all three of these areas, it is hard to get the full benefits of such a team, but many organizations around the world are tackling these challenges now to get the benefits of development agility and security.

Automated Security Testing

Automated testing for any product that will be in market or service for more than a few months is fundamental, if the goal is business and software agility. They are especially reassuring during technology migrations and infrastructure upgrades to confirm equivalent function in the new code. Focusing some of these tests around security is most critical. Building software that can be tested, and building the automated programmatic tests to go with it is a goal that can be achieved over time. Software without good test coverage is just technical debt, or a legacy platform, waiting to cause trouble. The skills to reduce this risk are available and are worth the investment if security, quality, and reliability are concerns for your business.

Observability & SIEM

Visibility into a software system is key to spotting security-related and unusual behaviour. It is also valuable with respect to tracing impacts of security events. Knowing exactly what transactions are occurring on a system, and how these transactions are flowing is a great start to observability, but pulling events into a place where they can all be analyzed and correlated is even more valuable. These practices don't just help with security, and their value is incalculable with respect to being able to improve a system, reduce time to resolve customer issues, and measure operational performance against SLAs.

How do I get started with these Practices?

There are many ways to get started. Your organization's experience, and your team's experience, will influence your direction, but, assuming you've not tried these practices before, below are some key ideas to get started with each practice. See the next section for where you can go for more advice and help.

DevSecOps Product Teams

- Build an agile team - it's never too early to get started.
- Your agile team for one product needs to be no more than 10 people, who are fully dedicated to developing the product all the time. They will need a Product Owner too.
- Hiring good Agile Product Owners who understand software development is tough, but they can make all the difference.
- Bring in an agile coach to help the team, and your organization, with the change. Change is hard and coaches are experts at facilitating change.
- Some people start by modeling their existing work via Kanban boards, and adapting from there.
- Start small, and remember that it takes time and effort to work differently.
- Automate, automate, automate - CI/CD pipelines are a key practice in getting to DevSecOps, and these pipelines and deployment tools must

ultimately be owned by the product team, and not by a separate group.
- There are lots of books on agility and adopting these practices, and many are worth reading. Here's a guide to some of them, with The Phoenix Project being at the top of that, and my, list.

Observability and SIEM

- Encourage your product development team to include explicit logging for audit events, and include items like timings. In a micro services architecture, correlating these audit events can be made easier by services passing around trace or request IDs.
- Check out our eBook on the things to log for best security observability.
- Bring your logs into an Observability platform like Splunk Enterprise or Splunk Observability. These tools make it much easier to identify operational patterns, not to mention make short work of creating dashboards.
- Investigate building reports and triggers around your observability data in these tools.
- Start small, and look for obvious issues to report to your SOC (Security Operations Center).
- Take a long view: growing to more mature SIEM tools (like Splunk Enterprise Security) takes time and skills. See the next section for where to go for help.

Automated Security Testing

- Adopt a tool or framework like Cucumber or Serenity BDD and just build a few tests across key areas of your product.
- Authentication steps are good.
- Testing is a mindset, and your product team needs to adopt the idea that their job is to break things that they have created.
- Developers of your product must be the key people building and maintaining tests, even though many developers have not taken ownership of tests before.
- Bring in an Automated Test Engineer to augment the team if the developers on the team are new to these ideas.
- Incorporate your automated tests into your CI/CD pipeline, and run them every time you want to validate a build candidate to be pushed beyond the development environment. And when they fail, do not proceed with the push until they pass.

Who can help my organization with this?

In addition to these work practices, there are plenty of tools and information available. OWASP is a great organization to look at for help, with their Top 10 reports, and their SAMM assessment (Software Assurance Maturity Model). If you're concerned with API security specifically, OWASP is coming out with a refresh this year of their API-specific Top 10 attack vectors, and you can look out for a post from us talking about how these practices apply to those specific vectors in the future.

Even running these types of assessments, however, can be difficult, and adopting the above practices often takes experience and advice to be effective.

Bringing in expertise, and coaching can help tremendously with improving your security and in adopting these practices. Consulting support can also help you determine what is best for your organization, depending on where you are now. You might benefit from an audit or review of your current API security practices, compared to where you want or need to be, which can help focus your efforts and provide a road map for next steps. Taking into account your specific situation and current security exposure combined with looking at your specific business needs for scaling, security, business agility, and the features you may need to safely expose your specific API assets, can all help in making strategic decisions about the next steps your organization needs to take.

At Solsys, we help clients with these very kinds of decisions. We have workshop services available to help you assess your current API security posture and software development approaches, and provide some details on what might come next for you and your team. We can focus on your specific API security practices and identify gaps you might want to close. We have also have a number of patterns across these practices that we have instrumented in our Center of Excellence and Innovation Lab that is super helpful in diving deeper into design options.

About the Author

John Tobin

John has worked with Solsys Corporation since 2013 as a Technical Product Owner & Systems Architect supporting the Digital Identity, Observability and API practices. In addition to working with customers on strategic initiatives in the information systems field, John has been engaged to support CISO in their API security standards work.

John has worked with most of the Canadian Telecom companies, and also in the finance sector, in a variety of consulting roles, before coming to Solsys where he has worked with customer strategic initiatives as a Technical Product Owner.

LEARN HOW TO GET STARTED WITH SOLSYS

Contact us to secure your APIs today

hello@solsys.ca

solsys.ca

Solsys LinkedIn

SOLSYS