Welcome to Week 2

Cloud Accelerator Program

DevSecOps

DevelopIntelligence
A PLURALSIGHT COMPANY

Hello

About me...

- 26+ years in the industry
- 21+ years in teaching
- Certified Cloud architect
- Passionate about learning
- Also, passionate about Reese's Cups!

Why study these subjects?

In modern software engineering, our ability to quickly deploy incremental innovation, ensure its quality, and scale to meet customer demand proves critical to our success

- Cloud is everywhere and it’s not going away
- As with many topics in technology, there are multiple options and multiple dimensions to those options
- Building a deeper understanding of Cloud and its offerings helps prepare you for modern IT
- Understanding DevSecOps and the important role of “shifting left” to ensure the quality and security of the systems we build acts as an invaluable enabler

My pledge to you

I will...

- Make this interactive
- Ask you questions
- Ensure everyone can speak
- Use an on-screen timer

Agenda

- DevSecOps – what it is and why it is needed
- The value of “shifting left”
- Options available in AWS for securing our Cloud workloads

How we're going to work together

- Slides and words to highlight key concepts
- Demos to bring those concepts "to life"
- Lab work (which will take place in sandboxes provided by "A Cloud Guru") for hands-on reinforcement
- NOTE: I welcome being interrupted – if you need more info, or clarification, or anything else, just break in and ask. I am here to help you.

DevSecOps – The Mindset

DevSecOps

What is it?

Discipline that takes into consideration how People, Processes, and Tools Automation combine to ensure we have:

- Disciplined build of security into all phases of SDLC
- Just-in-time security assessment and testing
- Security as a “shared responsibility”

DevSecOps

Why is it important?

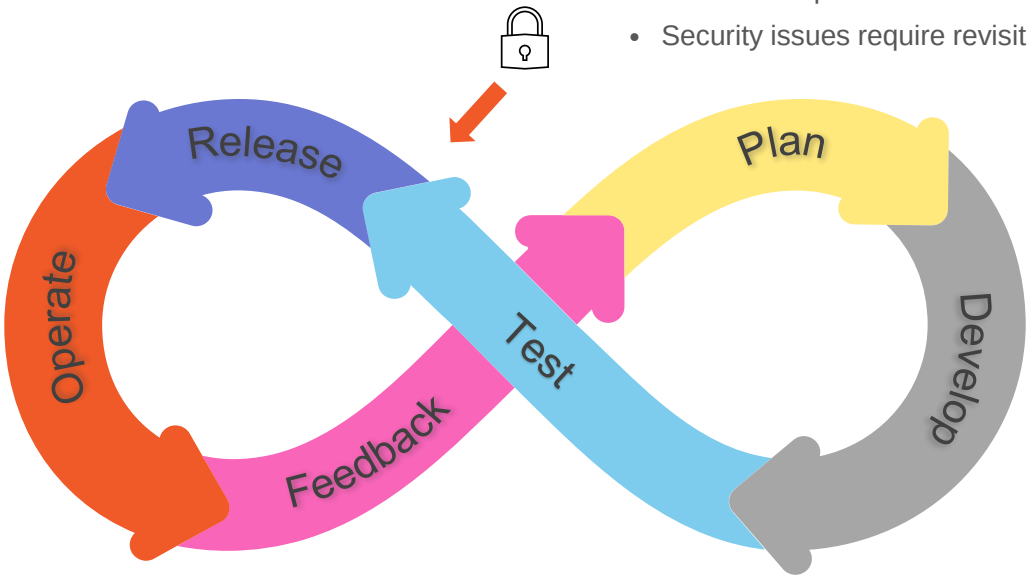
To maintain a competitive advantage:

- We need to deliver software quickly
- We need that software to have high quality
- We need that software to be secure

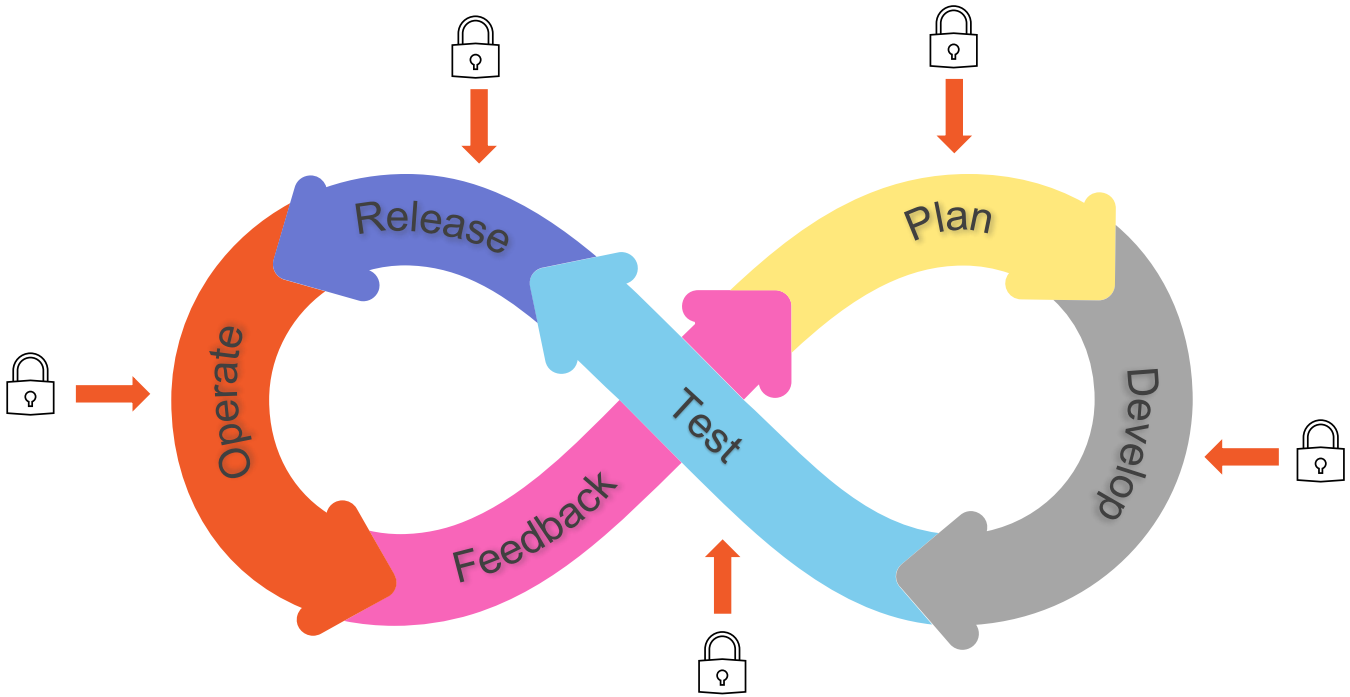
We want to merge speed & agility with security & quality!

Security in Software Engineering – The Traditional Way

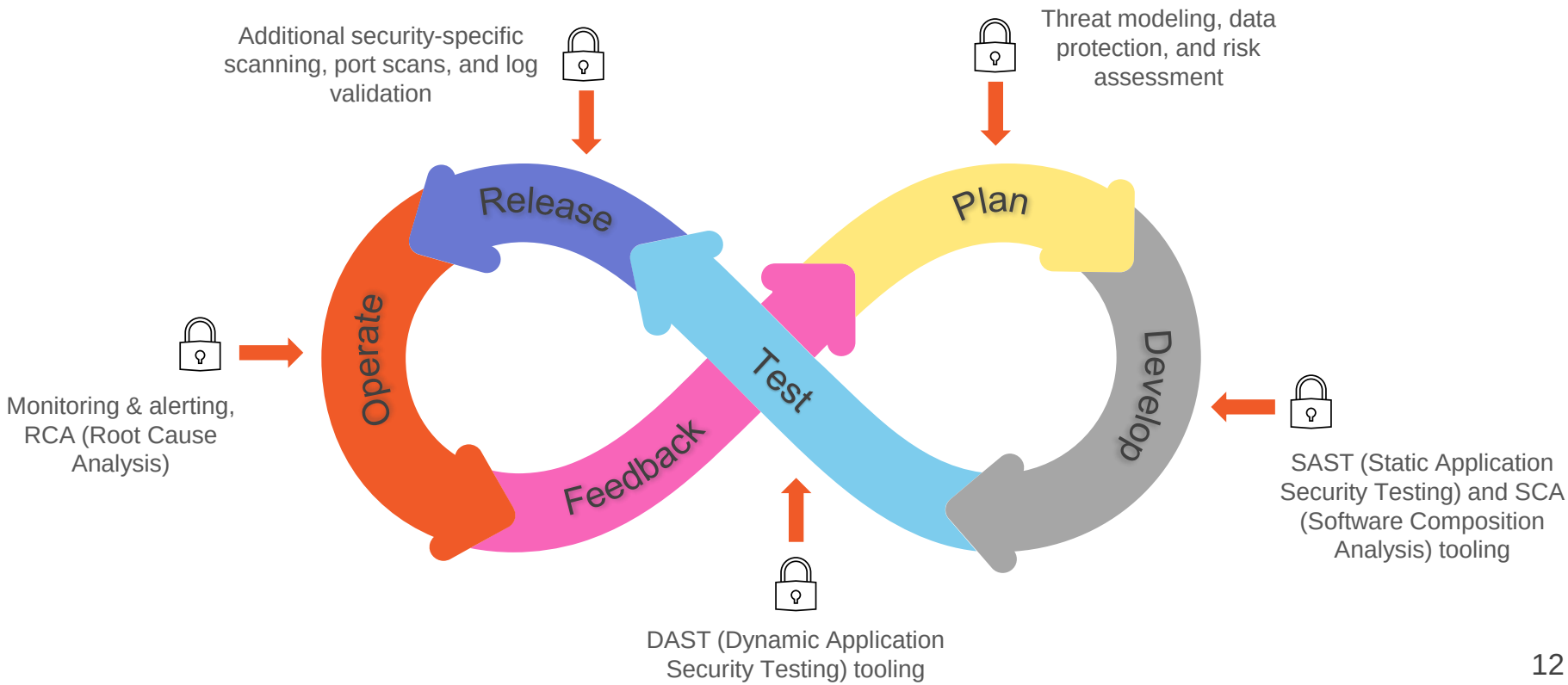
- Not enough time in schedule to absorb changes required to remediate
- Activities required to remediate may be complex
- Security issues require revisit of one or more previous phases



Security in Software Engineering – A Better Way

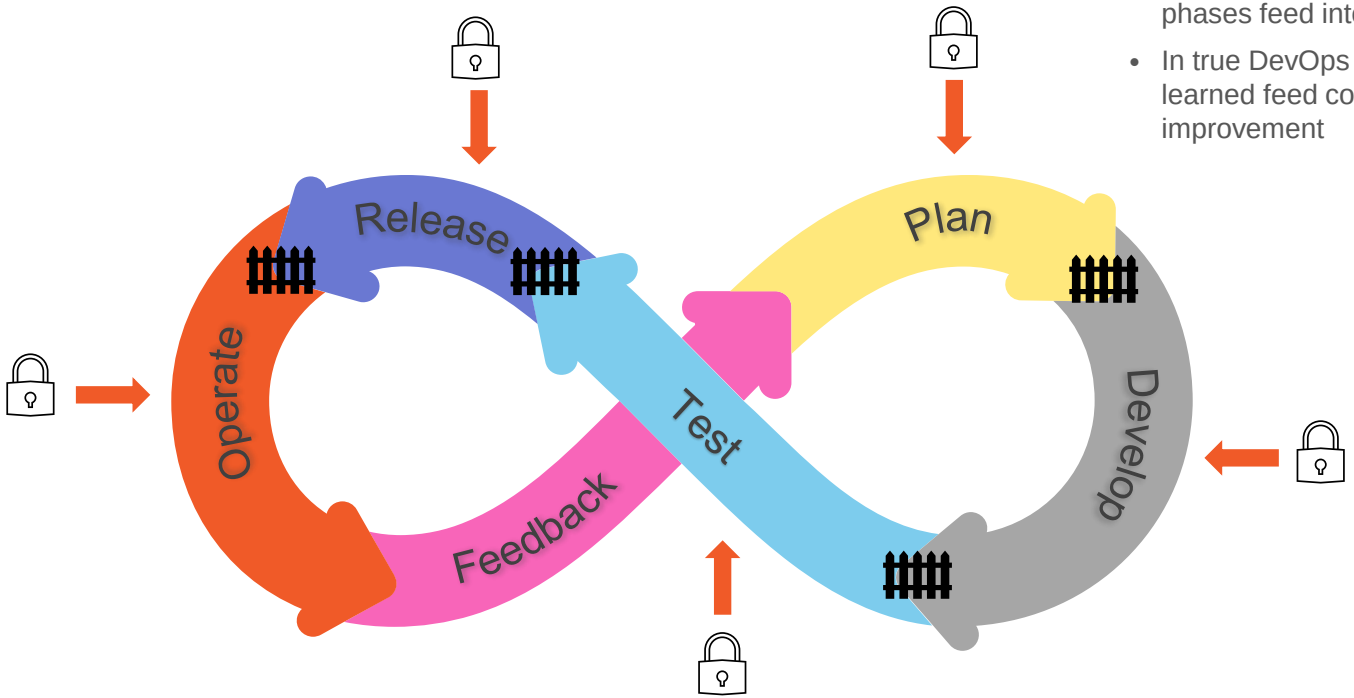


Security in Software Engineering – DevSecOps



Security in Software Engineering – DevSecOps

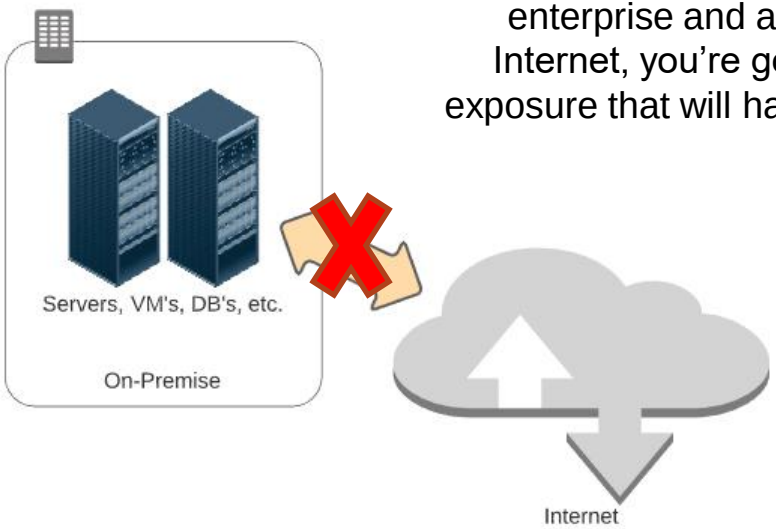
- Information gathered from early phases feed into later phases
- In true DevOps fashion, lessons learned feed continuous improvement



Quality gates guard against moving security defects forward

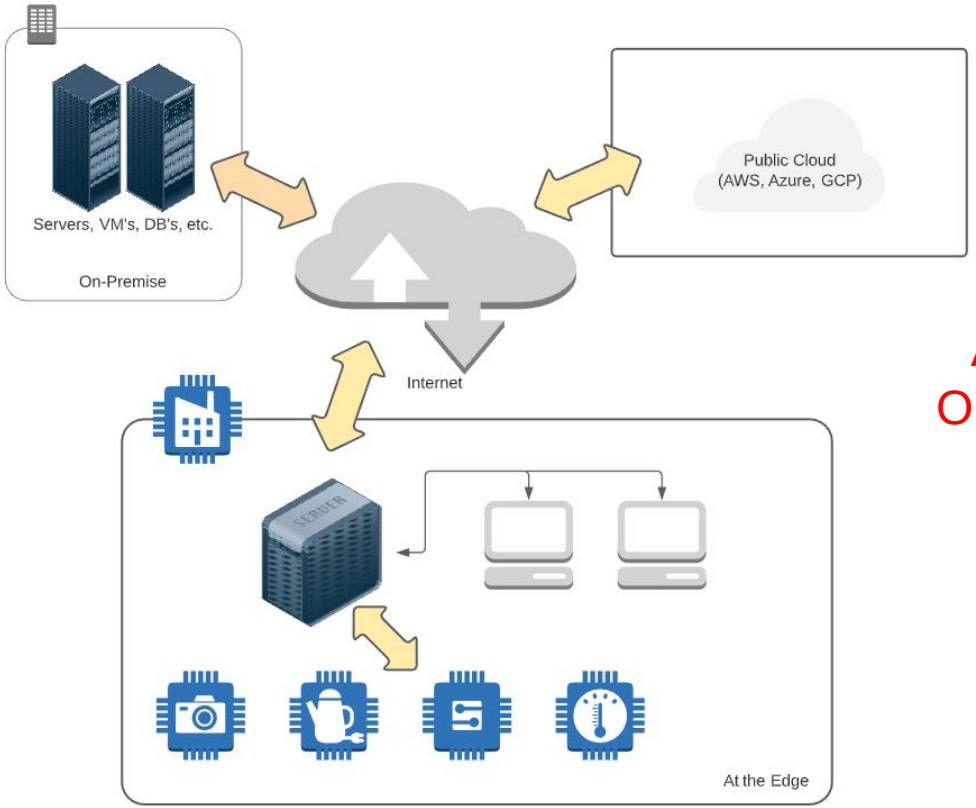
So…What About the Cloud?

On-Premise



Unless you decide to disconnect your enterprise and applications from the Internet, you're going to have security exposure that will have to be accounted for!

Hybrid Cloud



Attack surface ONLY EXPANDS – there is no ignoring the threats!!

So…What About the Cloud?

- For each AWS service type, we will have to remain CONTEXTUALLY vigilant!
- As we proceed throughout future weeks (and specific AWS services), we will review security best practices for each type
- This is a  , not a 

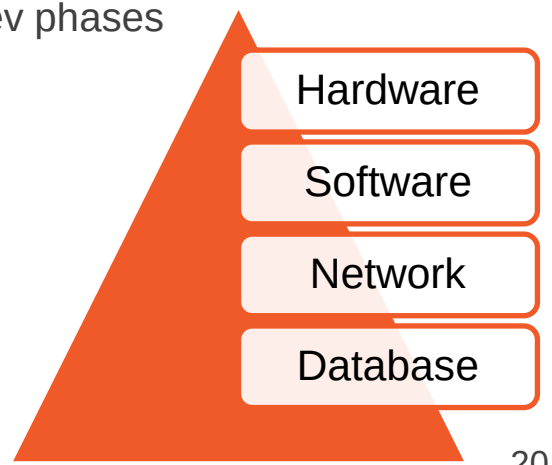
Shifting Security Left - Plan

Assessing Security Risks (Plan Phase)

- To secure a solution, attack surfaces and potential threats must be identified
- Common practice utilizes something called threat modeling
- Includes modeling and analyzing possible attack vectors based on application

Assessing Security Risks

- Risk assessment should account for different “zones” of execution
- Security requirements must be understood in context of specific use cases
- Ideally, threat modeling would be executed during design & dev phases



Threat Modeling

Threat Modeling

Can be viewed in two different, but related, contexts:

- Implementation of controls mapped to security requirements & policy (prevention)
- Implementation of countermeasures against possible known attacks (remediation)

Threat Modeling

Multiple approaches:

- Attacker-centric (think like an attacker!)
- Asset-centric (what do we have to lose?)
- Application-centric (what are we building & testing?)

Threat Modeling – Key Considerations

Valuable principles:

- Defense in Depth
- Principle of Least Privilege
- Secure by Default

Shifting Security Left - Develop

Shifting Security Left – Develop

- Ideally, security flaws in code would be caught on a developer’s machine
- Can use plugins for IDEs to provide automated static analysis
- In some cases, tools can be configured with custom rules and priorities

Shifting Security Left – Develop

- Rulesets and priorities should be driven by results of threat modeling
- Scanning tools if used should be a standard part of all developer's code tooling, to ensure consistently applied (standard dev build)
- Goal is to catch issues VERY early on in the SDLC

Common Security Attacks

- Denial of Service (DoS or DDos)
- SQL injection
- Large files
- Cross Site Scripting (XSS)
- Credential stuffing

Denial of Service (DoS)

- Server “flooded” with so much bogus traffic that systems are unable to serve valid requests
- Alternatively, instruction(s) received that trigger a server or system “crash”

Denial of Service (DoS)

Common flooding attacks:

- Buffer overflow (most common type)
- ICMP flood (AKA “smurf attack” or “ping of death”)
- SYN flood

Denial of Service (DoS)

Crash attacks:

- Often involves send of data targeting common classes of bug
- Request used to crash or severely destabilize the system

Distributed Denial of Service (DDoS)

- Similar profile as DoS but uses multiple systems to orchestrate the attack
- Provides attacker with advantages

Distributed Denial of Service (DDoS)

Potential advantages for attacker:

- More agents means more power behind the attack
- Location of attack is difficult to detect (often globally-routed)
- Easier to shut down a single attack machine than multiple
- Identity of attacker is more easily disguised

Defending Against Denial of Service

- Ensure service has good AuthN/AuthZ in place
- Utilize a proxy or gateway and configure throttling
- In Production, disable ICMP pings or use rate-limit for ICMP requests (e.g., using iptables)

SQL Injection

- Attacker injects SQL (Structured Query Language) queries into app flow (e.g., UI)
- In some cases, injected SQL used to retrieve additional sensitive detail
- In other cases, injected SQL used to alter or damage a company’s critical data

Types of SQL Injection Attack

- In-band
- Blind

In-band SQL Injection

- Uses existing channel of communication for an attack
- Error-based – attacker performs actions that cause errors in order to gather “intel” about database structure
- Union-based – attacker takes advantage of UNION SQL operator to fuse multiple SELECT statements into single response

Blind SQL Injection

- Attacker sends separate, independent data queries to a server
- Called blind because results of query are not sent back to attacker
- Instead, attacker observes results to infer vulnerabilities

Blind SQL Injection

- Relies on response and behavior patterns of server so slower to execute
- Boolean – attacker sends SQL query to database and determines if attack valid based on response received (true or false)
- Time-based – attacker sends SQL query to database and determines if attack valid based on amount of time taken to process

Defending Against SQL Injection

- Use well-defined contracts to explicitly map expected results from application
- Sanitize inputs and use parameterized queries in code
- Use a Web Application Firewall that includes protections at the application layer (including protection against SQL Injection)

Large Files

- Sometimes resembles another form of Denial of Service
- Attacker attempts to send one (or several) very large files as upload
- Could also occur with extremely large payloads (JSON or XML bodies)
- As a result, network connectivity to servers or services can become “clogged”

Defending Against Large Files

- Use configuration to limit file/payload sizes and number of concurrent connections from a client
- Utilize timeouts judiciously to prevent large file operations from completing
- Can also leverage MIME types as a way to limit acceptable types of data
- Finally, proxies or gateways (WAF) can be configured for mitigation at the network layer

Cross Site Scripting (XSS)

- A type of injection – but script instead of SQL
- Attacker uses inputs to attempt injection of a <script>…</script> element
- An example could be posting a comment with a link that routes to a malicious site

Cross Site Scripting (XSS)

- Without inspection for malicious content, <script> can be returned (and executed) in user's browser
- Malicious content can include JavaScript, HTML, Flash, etc.
- Really, any code that browser can execute

Cross Site Scripting (XSS)

Common forms of attack:

- Stealing cookie or session information
- Redirects to web content controlled by an attacker
- Executing malicious operations on user’s machine
- Leveraging impersonation for elevated privilege

Cross Site Scripting (XSS)

Stored XSS attacks:

- Injected script permanently stored on target servers
- Could include storage in database, forum, comment field, etc.
- Malicious script returned to browser as part of retrieval from storage

Cross Site Scripting (XSS)

Reflected XSS attacks:

- Malicious script is indirectly transferred back to browser
- When user clicks on malicious link, code gets injected into vulnerable site
- Malicious code then reflected back to user under the cover of “valid” site interaction for immediate execution

Cross Site Scripting (XSS)

DOM-based XSS attacks:

- Takes advantage of sites that copy input to DOM without validation
- Similar to reflected in that victim is tricked into sending malicious code to vulnerable site
- However, input lands in DOM in the browser for execution instead of being reflected back

Defending Against Cross Site Scripting

- Encode and validate everywhere – do not trust user inputs, escape outputs, and manage response headers
- Use libraries with utility handlers where possible (e.g., Jinja or Django)
- Quote every attribute of every tag in HTML

Defending Against Cross Site Scripting

- Use HttpOnly directive on custom cookie response headers (i.e., “Set-Cookie” header)
- Use the “X-Content-Type-Options: nosniff” to prevent MIME type sniffing (i.e., dynamic changes to Content-Type header)
- Leverage network components like WAF with built-in protection to intersect at the network layer

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Credential Stuffing

- Attackers use lists of compromised credentials to try and find a breach
- Based on assumption that many users reuse same credentials across sites
- Uses bots, automation, and scale

Credential Stuffing

- Like a brute force attack
- However, instead of random strings, uses existing lists of known credentials
- Powered by broad availability of compromised info and increasing sophistication of bots & automation

Credential Stuffing

- Attacker sets up bot able to attempt login for multiple accounts in parallel
- Uses an automated process to test effectiveness
- Monitors for breaches and pulls/retains sensitive detail when found
- With parallel attempts, often fakes IP addresses to make difficult to trace

Defending Against Credential Stuffing

- Leverage MFA (Multi-Factor Authentication)
- Use a CAPTCHA (though I hate them!)
- Gather details about user devices to create a “fingerprint” for incoming sessions – if same “fingerprint” is logged several times in sequence, block

Defending Against Credential Stuffing

- Leverage IP blacklisting
- Rate-limit non-residential traffic sources (like public Cloud)
- Block headless browsers based on JavaScript calls used
- Disallow e-mail addresses as user IDs

Shifting Security Left - Build

Shifting Security Left – Build

- Changes from a developer are ready to be integrated with others
- Typically driven through submission of Pull Request in GitHub
- Notifies a peer that changes are pending and need review

Shifting Security Left – Build

- In addition to logic/syntax errors, peer reviewer can focus on areas of high risk/high impact identified in threat modeling
- In some cases, a specialist peer reviewer (e.g., one from InfoSec) can be engaged to validate security of code

Shifting Security Left – Build

- Coupled with manual review, SAST (Static Application Security Testing) tools can be integrated into CI/CD to automate security checks
- Some even include ability to capture scan results in a build report and use results of scan as quality gate to prevent move forward on failure

Shifting Security Left – Build

- SCA (Software Composition Analysis) tools also provide benefit
- Tools can be integrated into CI/CD for automated security scans of Open-Source components
- Ideally, governance would be in place to monitor and manage “accepted” set and versions of Open-Source tools approved for usage

Shifting Security Left – Build

- SCA tools can provide a couple of benefits
- Primarily, security scans can be executed, and Open-Source vulnerabilities identified
- Secondarily, can help an organization catalog the Open-Source components (and versions) “in play” already

Shifting Security Left – Build

- Resulting report can be used to identify components that have not been properly vetted through governance
- Results can be merged with overall build output
- Quality gates (manual or automated) can be built around results to prevent move forward if insecure

Shifting Security Left - Test

Shifting Security Left – Test

- Includes changes ready for move to QA for additional testing
- Dynamic Application Security Testing (DAST) tools provide more sophisticated vulnerability checks
- Dynamically exercise application’s runtime interfaces using injected data

Shifting Security Left – Test

- DAST tools can use complex combinations of invalid input via fuzzing
- Provides multiple layers of protection

Shifting Security Left – Test

- Two types – passive scan and active scan
- Passive scan less aggressive and minimizes use of fuzzing
- As a result, not as robust in its ability to find exposure but also takes less time to run

Shifting Security Left – Test

- Because of that, passive scan can be good fit for CI/CD pipelines, especially those that look to push updates at a greater frequency
- For the C (continuous) part, need flows to complete quickly

Shifting Security Left – Test

- Active scans are more aggressive in their attack approach
- Make extensive use of fuzzing to elevate level of attack sophistication
- Send multiple requests, continually altering request data to simulate attack payloads

Shifting Security Left – Test

- Can also make use of a technique called “spidering”
- Allows the scan to crawl a site or service to simulate sequenced or multi-level attacks (stateful attacks)
- Can be very effective in securing a site or service but also takes much longer to execute (due to added complexity and sophistication)

Shifting Security Left – Test

- Because of its more aggressive nature, should only be executed against owned sites and in a sandboxed environment (risky)
- Often active scans are scheduled to occur out-of-band on a regular basis (weekly, nightly, etc.)

Shifting Security Left – Test

- In addition to the automated scans, traditional functional test scripts and manual testing may be used
- Should include security-focused testing as well (or in separate PEN testing)
- Tests should be informed by prioritized set of vulnerabilities identified during threat modeling

Shifting Security Left - Release

Shifting Security Left – Release

- Software confirmed ready for release
- Often deployed to a pre-prod environment (like staging)
- Can include automated “smoke” testing

Shifting Security Left – Release

- This phase may also include security-specific scans as a “last mile” protection
- SAST, SCA, and DAST scans may be repeated in this environment as a double-check
- Port scans can also be used (depending on how closely this env matches prod) to help validate exposure at network communication layer

Shifting Security Left – Release

- Additionally, this stage can utilize application logging as a type of validation
- Provides a unique opportunity to confirm correct sanitization of log detail
- Also, can include verification of specific AuthN/AuthZ controls prior to a prod release

Shifting Security Left - Operate

Shifting Security Left – Operate

- In this stage, monitoring and alerting become paramount for capturing and notifying support of any security exposure “in the wild”
- System logs, telemetry, and event detail become useful
- Likely not everything will be found prior – ongoing vigilance is a must

Shifting Security Left – Operate

- When (not if) something happens, Root Cause Analysis (RCA) is important
- Helps with understanding the why
- Can also help with identifying short-term workarounds (to stop the bleeding) and long-term fixes (for enhancement prioritization)

Shifting Security Left – Operate

- Preceding detail can feed dashboards or reports to help stakeholders visualize the security posture of the system and org
- System logs can also be used to satisfy auditing requirements (depending on the industry)
- Key Performance Indicators (KPIs) can be used to measure

Shifting Security Left – Operate

Examples

- Number of security-related tickets
- Build or release delays caused by security alerts
- Mean time to compliance
- These can all help support the Continuous Improvement function in the area of security focus

Shifting Security Left – Operate

- One additional layer of protection that a company can employ involves security bug “bounty hunters”
- Ethical hackers that can be paid to independently attempt to attack a site or system
- Can result in a set of reports on uncovered vulnerabilities and recommended remediations
- A SIEM (Security Information and Event Management) solution can also help with detection, analysis, and response

Other Key Considerations

Other Key Considerations - SAST

- Be aware that SAST tools have the propensity for false positives (depending on tooling employed)
- Tools may err on the side of caution with security scans
- Might require that an organization customize the tool to more accurately report on environment and application
- Otherwise, teams run the risk of wasting time on non-value add activities, impeding “frictionless security”

Other Key Considerations - Containers

- If application profile includes containers or container orchestration (k8s), additional consideration required relative to security of images
- Tools like Prisma Cloud (https://docs.paloaltonetworks.com/prisma/prisma-cloud/) can help with scanning of container images

Other Key Considerations – Sensitive Config

- Sensitive configuration data and app secrets (e.g., connection strings, passwords, etc.) should never be hardcoded in source
- Run the risk of checking into and exposing via source control
- Tools can help with things like GitHub repo scanning

Other Key Considerations – Sensitive Config

- Other platforms include built-in support
- Services like AWS KMS can provide dynamic linking of secure config into the CI/CD pipeline

DevSecOps

Potential Benefits

- Security “baked” into the process end-to-end
- Promotes secure by design and defense in depth
- Increased efficiency in verifying security through automation
- Faster recovery times if security breach occurs
- Enables measurement and objective assessment of security posture

DevSecOps

Potential Benefits

- Leveraging established security best practices, existing reference architectures, and existing patterns helps minimize risk
- Also, helps ensure you meet critical security requirements in a “low friction” manner
- Means as a developer you don’t have to try to mitigate all threats on your own – there are resources available to assist

DevSecOps

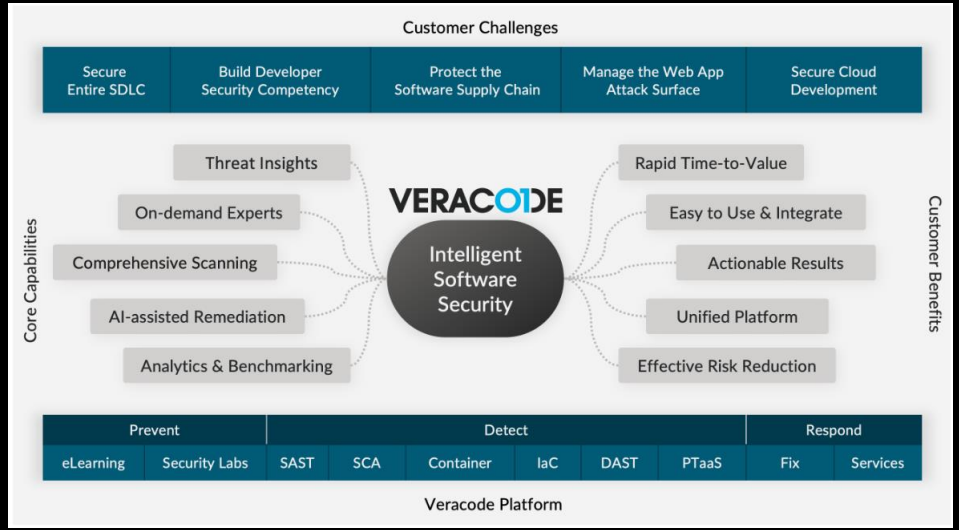
Potential Issues

- Knowledge gaps about security, a complex subject
- Additional complexity and friction in pipeline
- Potential for developer overload
- False positives breed “noise” and can lead to “boy who cried wolf”

WALKTHROUGH:

Tooling

Review Veracode capabilities – https://www.veracode.com/



WALKTHROUGH:

Tooling

Review Prisma Cloud capabilities – https://www.paloaltonetworks.com/prisma/cloud/container-security

Securing AWS Resources

LAB:

Securing AWS Resources

Execute the “Hands-On” lab available at
https://learn.acloud.guru/handson/6fc1cc38-73cd-4abf-bdc4-2d718b1f1cd1

LAB:

Securing AWS Resources

Execute the “Hands-On” lab available at
https://learn.acloud.guru/handson/e4e6a251-06af-4046-992b-84f0ece1d3fb

LAB:

Securing AWS Resources

Execute the “Hands-On” lab available at
https://learn.acloud.guru/handson/f3a6e65f-261a-4337-816f-5875ed4dd3e7

LAB:

Securing AWS Resources

Execute the “Hands-On” lab available at
https://learn.acloud.guru/handson/c084edc8-8e1f-4dfe-9c89-237a229f61d0

LAB:

Securing AWS Resources

Execute the “Hands-On” lab available at
https://learn.acloud.guru/handson/82ac8bc4-ccd3-4f28-8a96-124923392764

Thank you!

If you have additional questions, please reach out to me at:
asanders@gamuttechnologysvcs.com

DevelopIntelligence
A PLURALSIGHT COMPANY