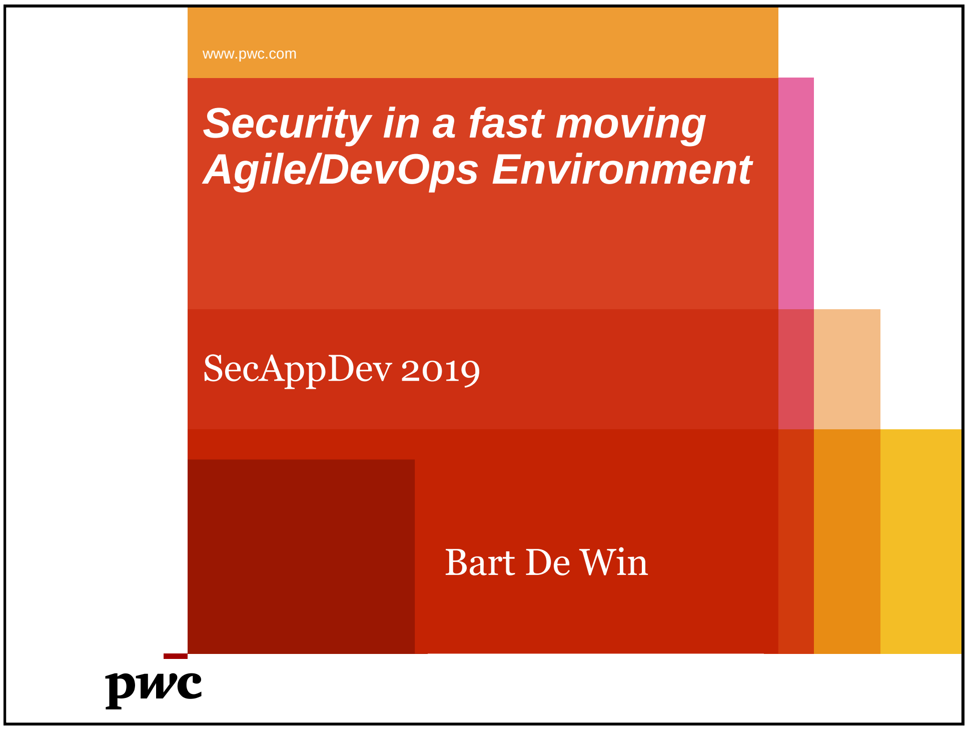


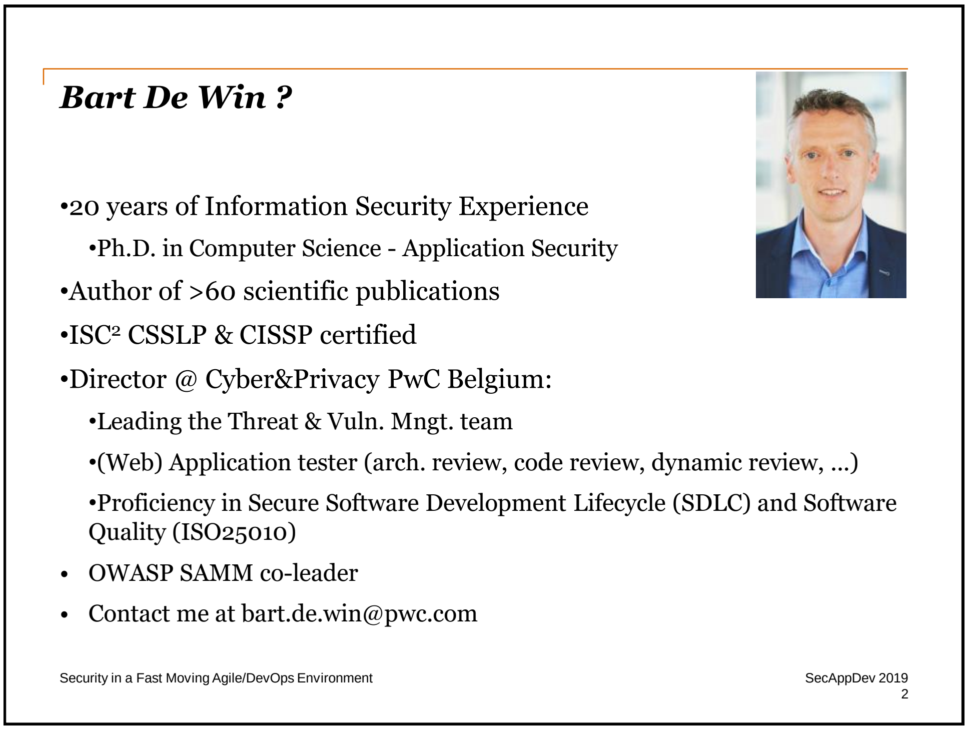
www.pwc.com

Security in a fast moving Agile/DevOps Environment

SecAppDev 2019

Bart De Win

pwc



Bart De Win ?

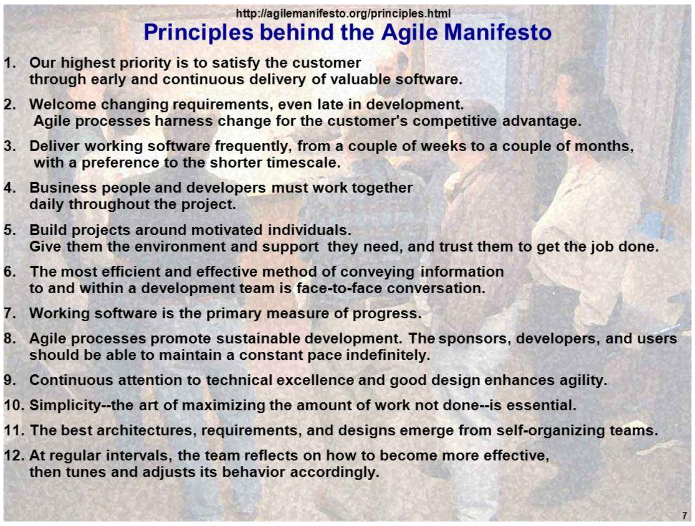
- 20 years of Information Security Experience
  - Ph.D. in Computer Science - Application Security
- Author of >60 scientific publications
- ISC² CSSLP & CISSP certified
- Director @ Cyber&Privacy PwC Belgium:
  - Leading the Threat & Vuln. Mngt. team
  - (Web) Application tester (arch. review, code review, dynamic review, ...)
  - Proficiency in Secure Software Development Lifecycle (SDLC) and Software Quality (ISO25010)
- OWASP SAMM co-leader
- Contact me at bart.de.win@pwc.com

Security in a Fast Moving Agile/DevOps Environment

SecAppDev 2019
2

Secure Agile Development

Once upon a time in 2001 …



http://agilemanifesto.org/principles.html

Principles behind the Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

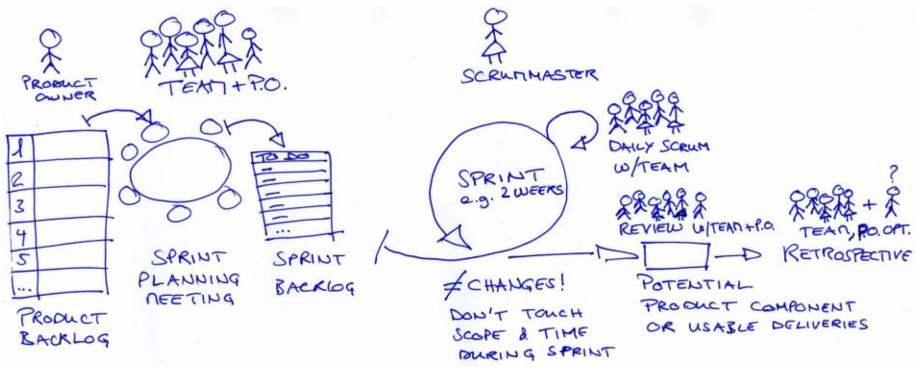
7

Boyd's Law

- *Speed of iteration beats quality of iteration*
  - Sooner to correct
  - Better for testing
- Successful companies continuously improve their software

WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com
scottadams@aol.com
©2007 Scott Adams, Inc./Dist. by UFS, Inc.

Scrum

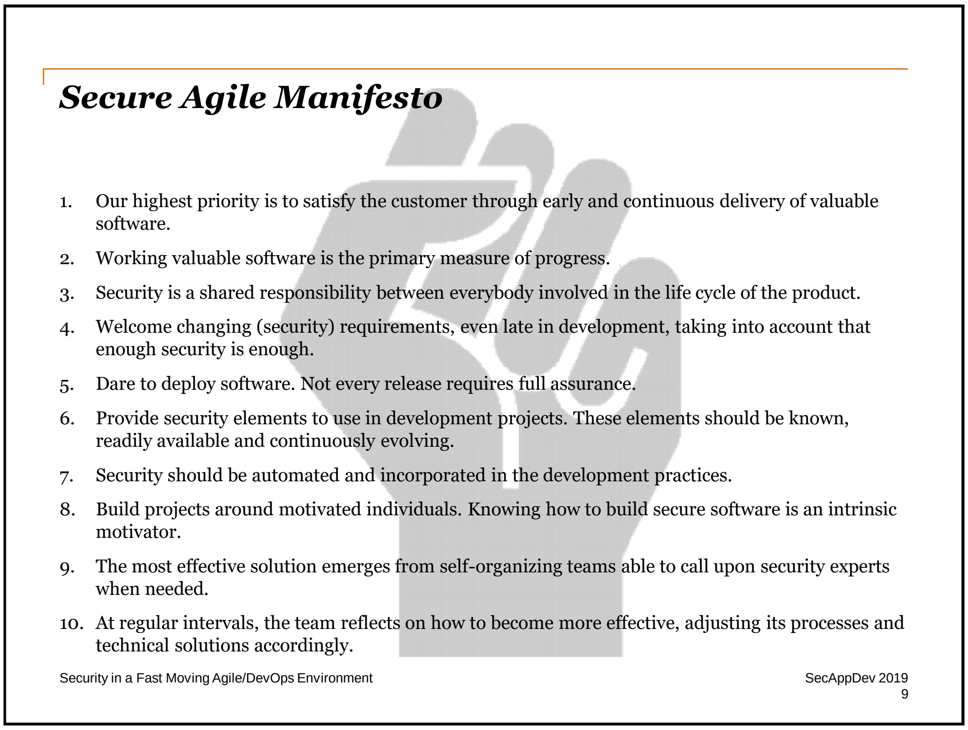


So what about security ?

Agile Development	Security
Speed & Flexibility	Stable & Rigorous
Short cycles	Extra activities
Limited documentation	Extensive analysis
Functionality-driven	Non-functional

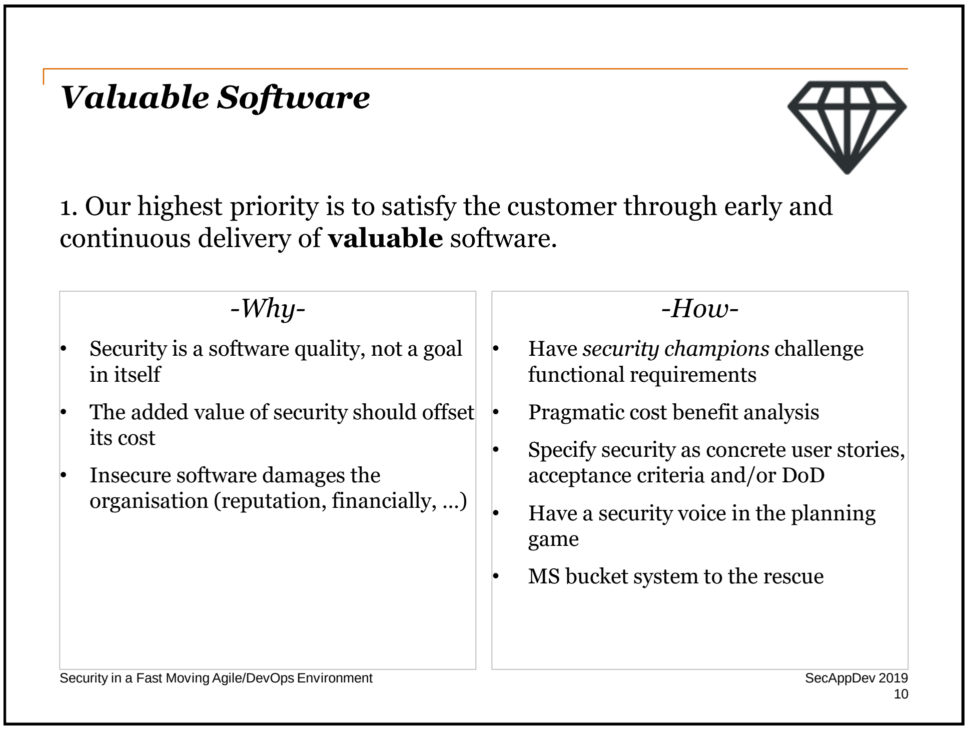
Secure agile development

If we want to be successful in developing *secure* applications, we **radically** have to change the way we organize this.



Secure Agile Manifesto

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Working valuable software is the primary measure of progress.
3. Security is a shared responsibility between everybody involved in the life cycle of the product.
4. Welcome changing (security) requirements, even late in development, taking into account that enough security is enough.
5. Dare to deploy software. Not every release requires full assurance.
6. Provide security elements to use in development projects. These elements should be known, readily available and continuously evolving.
7. Security should be automated and incorporated in the development practices.
8. Build projects around motivated individuals. Knowing how to build secure software is an intrinsic motivator.
9. The most effective solution emerges from self-organizing teams able to call upon security experts when needed.
10. At regular intervals, the team reflects on how to become more effective, adjusting its processes and technical solutions accordingly.

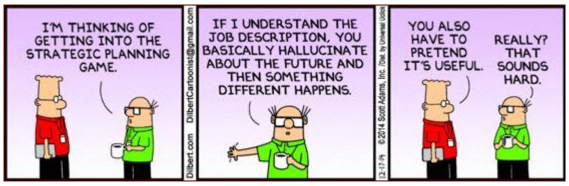


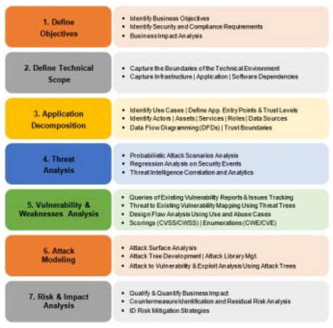
Valuable Software

1. Our highest priority is to satisfy the customer through early and continuous delivery of **valuable** software.

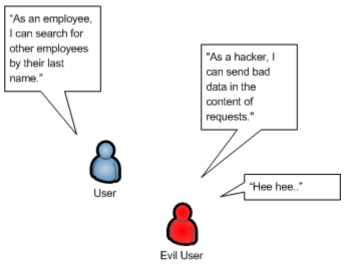
-Why-	*-How-*
• Security is a software quality, not a goal in itself	• Have *security champions* challenge functional requirements
• The added value of security should offset its cost	• Pragmatic cost benefit analysis
• Insecure software damages the organisation (reputation, financially, …)	• Specify security as concrete user stories, acceptance criteria and/or DoD
	• Have a security voice in the planning game
	• MS bucket system to the rescue

Valuable Software


I'M THINKING OF GETTING INTO THE STRATEGIC PLANNING GAME.
IF I UNDERSTAND THE JOB DESCRIPTION, YOU BASICALLY HALLUCINATE ABOUT THE FUTURE AND THEN SOMETHING DIFFERENT HAPPENS.
YOU ALSO HAVE TO PRETEND IT'S USEFUL.
REALLY? THAT SOUNDS HARD.
Dilbert.com DilbertCartoonist@gmail.com
12-17-14 ©2014 Scott Adams, Inc. Dist. by Universal Uclick



1. Define Objectives	• Identify Business Objectives • Identify Security and Compliance Requirements • Business Impact Analysis
2. Define Technical Scope	• Capture the Boundaries of the Technical Environment • Capture Infrastructure | Application | Software Dependencies
3. Application Decomposition	• Identify Use Cases | Define App. Entry Points & Trust Levels • Identify Actors | Assets | Services | Roles | Data Sources • Data Flow Diagramming (DFDs) | Trust Boundaries
4. Threat Analysis	• Probabilistic Attack Scenarios Analysis • Regression Analysis on Security Events • Threat Intelligence Correlation and Analytics
5. Vulnerability & Weaknesses Analysis	• Queries of Existing Vulnerability Reports & Issues Tracking • Threat to Existing Vulnerability Mapping Using Threat Trees • Design Flaw Analysis Using Use and Abuse Cases • Scorings (CVSS/CWSS) | Enumerations (CWE/CVE)
6. Attack Modeling	• Attack Surface Analysis • Attack Tree Development | Attack Library Mgt. • Attack to Vulnerability & Exploit Analysis Using Attack Trees
7. Risk & Impact Analysis	• Qualify & Quantify Business Impact • Countermeasure Identification and Residual Risk Analysis • ID Risk Mitigation Strategies


"As an employee, I can search for other employees by their last name."
User
"As a hacker, I can send bad data in the content of requests."
"Hee hee.."
Evil User

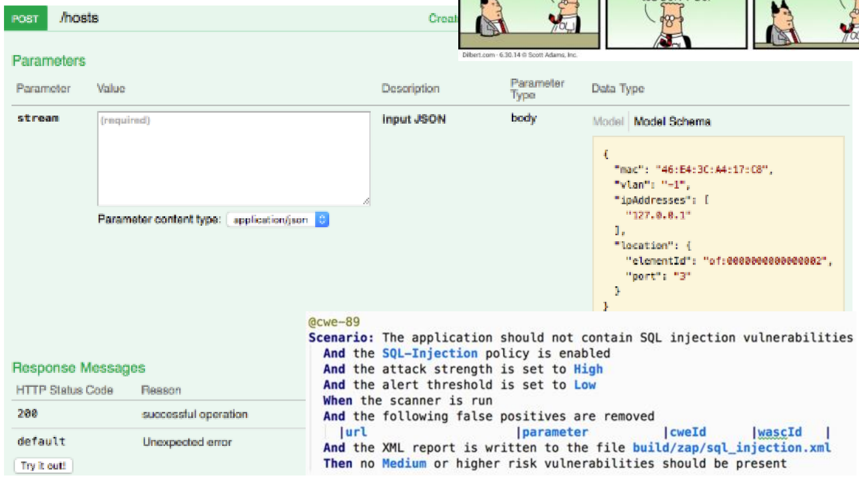
Code over Documentation

2. Working valuable software is the primary measure of progress.

-Why-	-How-
• Documentation does not always add value to the customer • Security's primary goal is supporting the business by improving the software, not producing documentation • Maintaining documentation for rapidly changing software is expensive and challenging	• Prefer actionable items over standards and policies • Strive for a single source of truth • Use expressive, readable, automated tests as documentation

Code over Documentation

YOU DIDN'T ACCOMPLISH ANY-THING THIS MONTH.
SURE I DID.
I DID THE MANDATORY TRAINING THAT HAS NO USE, ATTENDED YOUR MANDATORY MEETINGS THAT DON'T HELP, AND FILLED OUT REGULATORY PAPERWORK FOR THINGS WE DON'T DO.
ARE YOU TRYING TO MAKE A POINT?
NOPE. JUST BEING PRODUC-TIVE.
Dilbert.com · 6.30.14 © Scott Adams, Inc.



Parameter	Value	Description	Parameter Type	Data Type
stream	(required)	Input JSON	body	Model Model Schema

Parameter content type: application/json

```
{
  "mac": "46:E4:3C:A4:17:C8",
  "vlan": "-1",
  "ipAddresses": [
    "127.0.0.1"
  ],
  "location": {
    "elementId": "of:0000000000000002",
    "port": "3"
  }
}
```

Response Messages

HTTP Status Code	Reason
200	successful operation
default	Unexpected error

```
@cwe-89
Scenario: The application should not contain SQL injection vulnerabilities
  And the SQL-Injection policy is enabled
  And the attack strength is set to High
  And the alert threshold is set to Low
  When the scanner is run
  And the following false positives are removed
    |url                    |parameter          |cweId      |wascId   |
  And the XML report is written to the file build/zap/sql_injection.xml
  Then no Medium or higher risk vulnerabilities should be present
```

Shared Responsibility

3. Security is a shared responsibility between everybody involved in the life cycle of the product

-Why-

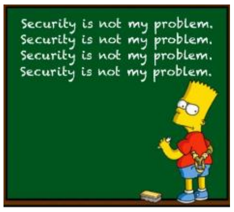
- Shorter cycles means less time to implement security
- Scalability
  - Security people are typically outnumbered
- Everybody should feel responsible, otherwise it will fail
  - developers can't fix the problem all by themselves

-How-

- Enable everybody to suggest code (security) improvements (push)
- Security team responsible for tooling, awareness and support
- No central steering for security (push vs. pull)
- Intelligent monitoring of state of security

Shared Responsibility

WEB SECURITY EXPERT
IF YOU GIVE ME LOTS OF MONEY, I WILL DO VARIOUS VAGUE THINGS TO INCREASE YOUR WEB SECURITY
WOW! I WOULD HAVE BEEN HAPPY WITH JUST VARIOUS THINGS, BUT THAT VAGUE STUFF SOUNDS GREAT TOO!
AND NOW HE'S MY RESPON-SIBILITY?
DON'T SCREW UP HIS VAGUE-NESS PLAN BECAUSE I THINK IT CAN WORK.


Security is not my problem.
Security is not my problem.
Security is not my problem.
Security is not my problem.

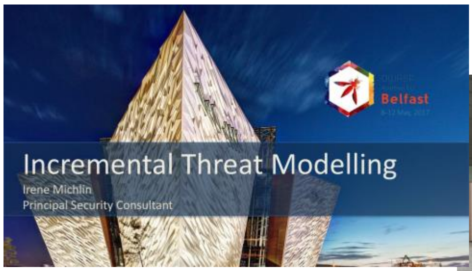
Embrace Change

4. Welcome changing (security) requirements, even late in development, taking into account that enough security is enough.

-Why-	*-How-*
• Changing requirements are a fact of life, learn to live with it • Companies loose competitive advantage when their operation is too rigid	• Use analysis techniques to work on moving targets • Ensure delivery of high quality software to facilitate changes - Embrace refactoring to improve code quality and minimize technical debt • Build flexible security solutions

Embrace Change

THIS IS RODNEY. HE'S IN CHARGE OF PRODUCT SAFETY TESTING.
IS OUR NEW PRODUCT SAFE ENOUGH TO START SELLING?
$#%@
DID THAT SOUND LIKE "SHIP" TO YOU?
Dilbert.com DilbertCartoonist@gmail.com
© 2010 Scott Adams, Inc./Dist. by UFS, Inc.


Incremental Threat Modelling
Irene Michlin
Principal Security Consultant

Dare to deploy

5. Dare to deploy software. Not every release requires full assurance.

-Why-

- As software changes often, a vulnerability might only be exposed briefly
  - This reduces the attack window
- Completely secure software does not exist. Trying to produce it is a very expensive exercise in futility.
- In case of issue, a new version can be released quickly

-How-

- Trust in your people and practices to get the balance of security right.
- Leverage A/B testing to limit application exposure and frustrate attacks
- Increase security efforts for sensitive features and important releases
- Invest in detective controls. Prevention alone is insufficient

Dare to deploy

HI, I'M ALAN, FROM THE QUALITY ASSURANCE DEPARTMENT.
DON'T BE WORRIED ABOUT QUALITY. I ASSURE YOU WE HAVE PLENTY OF IT.
THAT'S ALL YOU DO?
HEY, I DON'T TELL YOU HOW TO DO YOUR JOB.
Dilbert.com
@ScottAdamsSays

CARTOONSTOCK .com
"I ONLY LIVE FOR 24 HOURS, SO I NEED A TEMP JOB THAT PAYS BIG BUCKS."

Available security elements

6. Provide security elements to use in development projects. These elements should be known, readily available and continuously evolving.

-Why-

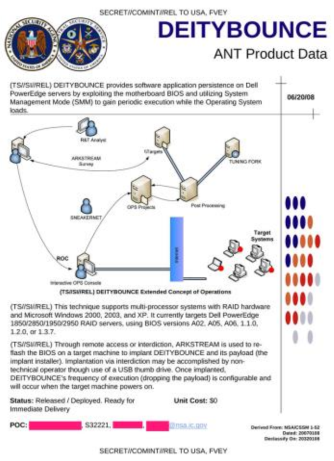
- It is inefficient for all teams to invent their own solutions
- Well-known good practices/solutions increase quality and efficiency
- Security is a specialist topic
- Different projects (might) require different variants

-How-

- Have a central security team prepare reusable practices (e.g., threat modelling, code review) and solutions (e.g., a single sign-on component)
- Provide a central knowledge repository (PULL)
- Communicate frequently and proactively around the roadmap (PUSH)
- Build them the agile way

Available security elements

DOGBERT IS CHAIRING THE INTERNATIONAL DATA SECURITY STANDARDS GROUP.
THE GOAL OF OUR ORGANIZATION IS TO MAKE YOUR SECURITY PROCEDURES SO INCONVENIENT THAT YOU GIVE UP HOPE AND DIE FROM BED SORES.
Dilbert.com
DilbertCartoonist@gmail.com



SECRET//COMINT//REL TO USA, FVEY

DEITYBOUNCE

ANT Product Data

(TS//SI//REL) DEITYBOUNCE provides software application persistence on Dell PowerEdge servers by exploiting the motherboard BIOS and utilizing System Management Mode (SMM) to gain periodic execution while the Operating System loads.

06/20/08

R&T Analyst
ARKSTREAM Survey
!Targets
TUNING FORK
OPS Projects
Post Processing
SNEAKERNET
ROC
Interactive OPS Console
Target Systems

(TS//SI//REL) DEITYBOUNCE Extended Concept of Operations

(TS//SI//REL) This technique supports multi-processor systems with RAID hardware and Microsoft Windows 2000, 2003, and XP. It currently targets Dell PowerEdge 1850/2850/1950/2950 RAID servers, using BIOS versions A02, A05, A06, 1.1.0, 1.2.0, or 1.3.7.

(TS//SI//REL) Through remote access or interdiction, ARKSTREAM is used to re-flash the BIOS on a target machine to implant DEITYBOUNCE and its payload (the implant installer). Implantation via interdiction may be accomplished by non-technical operator though use of a USB thumb drive. Once implanted, DEITYBOUNCE's frequency of execution (dropping the payload) is configurable and will occur when the target machine powers on.

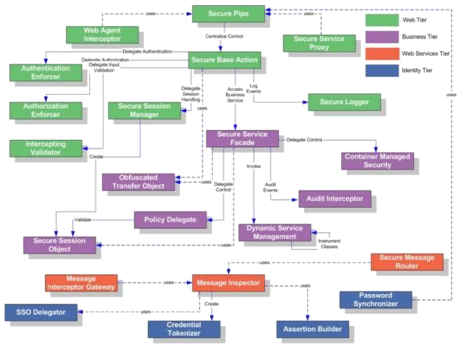
Status: Released / Deployed. Ready for Immediate Delivery

Unit Cost: $0

POC: [redacted], S32221, [redacted], [redacted]@nsa.ic.gov

Derived From: NSA/CSSM 1-52
Dated: 20070108
Declassify On: 20320108

SECRET//COMINT//REL TO USA, FVEY



Automate and integrate

7. Security should be automated and incorporated into development practices.

-Why-

- Security people are outnumbered and cannot scale
- Carrying out security tasks manually is too slow and cannot scale
- Efficiency and uptake increase when stakeholders are engaged using their own tools and language

-How-

- Maximise security automation
  - Minimize friction by accepting the constraints of automation
- Favour test quality over quantity
- Embed security in the development toolchain
  - Use for securing but also educating

Automate and integrate

OUR NEW ROBOT IS TOO SMART.
IT KEEPS THREATENING HUMANS INTO DOING ITS JOB WHILE IT DOES NOTHING BUT DRINK COFFEE.
ISN'T THAT ALL YOU DO?
I DON'T LIKE WHERE THIS IS HEADING.
Dilbert.com @ScottAdamsSays

WhiteSource QUALYS SoapUI sonarqube CHECKMARX cucumber FORTIFY acunetix

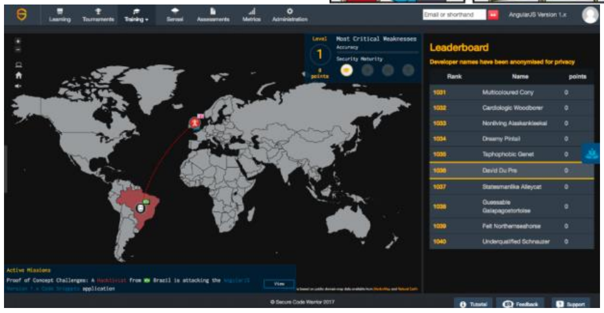
Motivation

8. Build projects around motivated individuals. Knowing how to build secure software is an intrinsic motivator.

-Why-	-How-
• Motivated people produce better software • Developers are genuinely interested in security, given concrete advise • Developers influence each other • Motivation is a critical success factor for any secure development initiative	• Disseminate concrete advice on how to develop secure software • Enable developers to better understand security but appreciate the limits of their knowledge • Encourage attending events to foster curiosity and learning • Favour reward over punishment • Gamify learning to increase engagement

Motivation

DILBERT
BY SCOTT ADAMS
I NEED SOME MENTORING.
THIS IS AWKWARD.
ON ONE HAND, HELPING YOU WOULD MAKE ME APPEAR WISE AND GENEROUS.
ON THE OTHER HAND, IT WOULD MAKE YOU A MORE CREDIBLE THREAT TO TAKE MY JOB.
I SEE YOU AS MORE OF AN ADVERSARY THAN A SUBORDINATE.
THAT'S WHY I WITHHOLD VITAL INFORMATION THAT YOU NEED TO DO YOUR JOB.
I'VE ALREADY SAID TOO MUCH.
DID YOU LEARN ANY-THING?
YES, UNFORTU-NATELY.



Leaderboard

Developer names have been anonymised for privacy

Rank	Name	points
1031	Multicoloured Cony	0
1032	Cardiologic Woodborer	0
1033	Nonliving Alaskankleekai	0
1034	Dreamy Pintail	0
1035	Taphophobic Genet	0
1036	David Du Pre	0
1037	Statesmanlike Alleycat	0
1038	Guessable Galapagostortoise	0
1039	Felt Northernseahorse	0
1040	Underqualified Schnauzer	0

Active Missions

Proof of Concept Challenges: A Hacktivist from Brazil is attacking the AngularJS Version 1.x Code Snippets application

© Secure Code Warrior 2017

Self-organizing teams

9. The most effective solution emerges from self-organizing teams able to call upon security experts when needed.

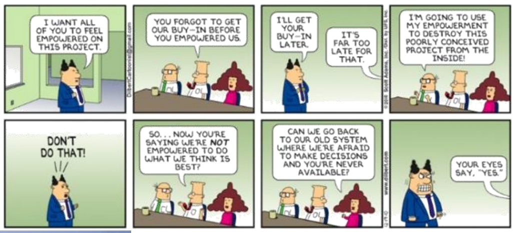
-Why-

- People are more motivated for the things they are in control of
- The team is best placed to effectively organize and work on the project

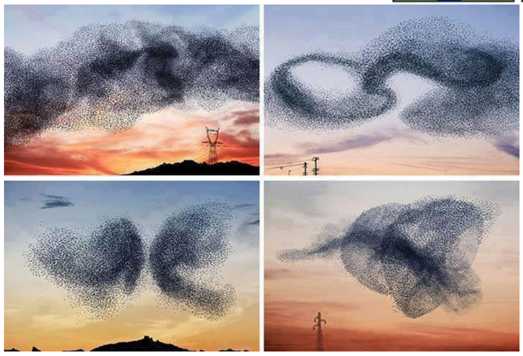
-How-

- Decide within the team who's driving security tasks
  - Security "belts" might support this
- Decide within the team how to leverage on security tooling
- Security specialists are available for advice when needed (mentoring)

Self-organizing teams


I WANT ALL OF YOU TO FEEL EMPOWERED ON THIS PROJECT.
YOU FORGOT TO GET OUR BUY-IN BEFORE YOU EMPOWERED US.
I'LL GET YOUR BUY-IN LATER.
IT'S FAR TOO LATE FOR THAT.
I'M GOING TO USE MY EMPOWERMENT TO DESTROY THIS POORLY CONCEIVED PROJECT FROM THE INSIDE!
DON'T DO THAT!
SO... NOW YOU'RE SAYING WE'RE NOT EMPOWERED TO DO WHAT WE THINK IS BEST?
CAN WE GO BACK TO OUR OLD SYSTEM WHERE WE'RE AFRAID TO MAKE DECISIONS AND YOU'RE NEVER AVAILABLE?
YOUR EYES SAY, "YES."

Reference : http://dilbert.com/strip/2010-12-19



Self-reflection

10. At regular intervals, the team reflects on how to become more effective, adjusting its processes and technical solutions accordingly.

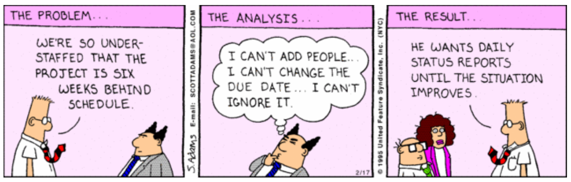
-Why-

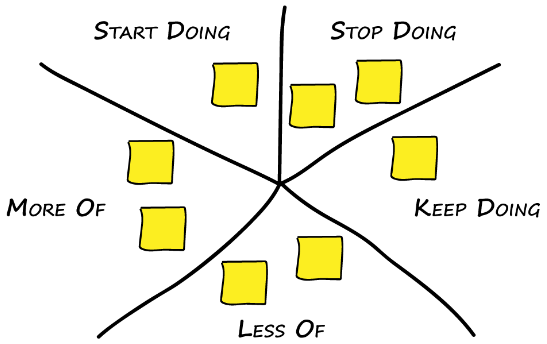
- Paradigms change. Technologies change. Organisations change. People change.
- Cater for these changes by constantly looking for better ways to organize your delivery.

-How-

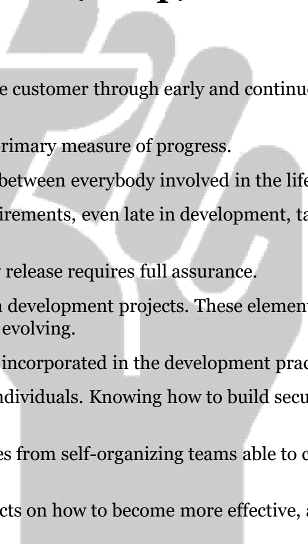
- Treat security as an explicit topic within retrospective meetings.
- Allow to fail - Failure to deliver is a learning opportunity.
- Learn from other teams what works well and what doesn’t.

Self-reflection


THE PROBLEM...
WE'RE SO UNDER-STAFFED THAT THE PROJECT IS SIX WEEKS BEHIND SCHEDULE.
S.Adams E-mail: SCOTTADAMS@AOL.COM
THE ANALYSIS...
I CAN'T ADD PEOPLE... I CAN'T CHANGE THE DUE DATE... I CAN'T IGNORE IT.
2/17
© 1995 United Feature Syndicate, Inc. (NYC)
THE RESULT...
HE WANTS DAILY STATUS REPORTS UNTIL THE SITUATION IMPROVES.



Secure Agile Manifesto (recap)



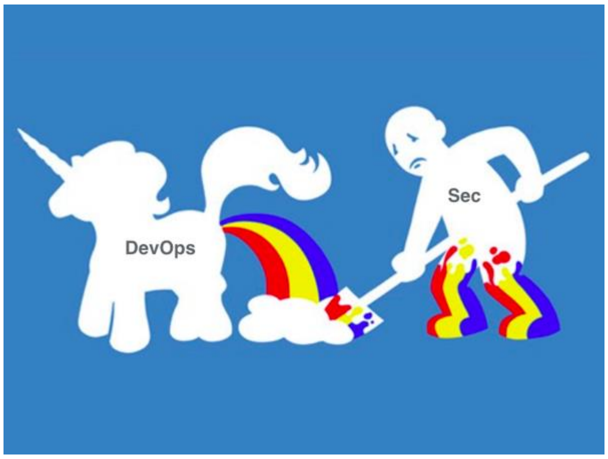
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Working valuable software is the primary measure of progress.
3. Security is a shared responsibility between everybody involved in the life cycle of the product.
4. Welcome changing (security) requirements, even late in development, taking into account that enough security is enough.
5. Dare to deploy software. Not every release requires full assurance.
6. Provide security elements to use in development projects. These elements should be known, readily available and continuously evolving.
7. Security should be automated and incorporated in the development practices.
8. Build projects around motivated individuals. Knowing how to build secure software is an intrinsic motivator.
9. The most effective solution emerges from self-organizing teams able to call upon security experts when needed.
10. At regular intervals, the team reflects on how to become more effective, adjusting its processes and technical solutions accordingly.

Final Notes on Agile Security

- Agile development changes the game for software security
- We need to start thinking in this mode and adapt our practices
  - Value to the business and speed of delivery
  - Not strive for 100% security
- Manifesto for Secure Agile
- Key success factors to achieve this are shared responsibility, automation and integration

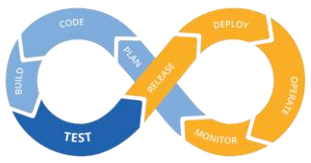
A secure CI/CD pipeline

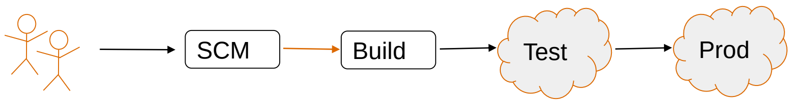
Sounds familiar?



Author: Pete Cheslock

A basic CI/CD pipeline



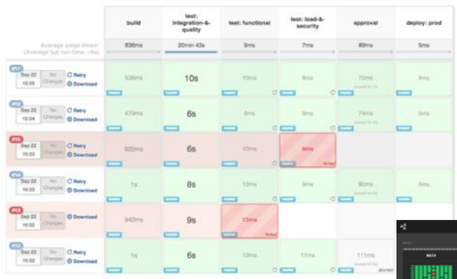


Concourse crash course

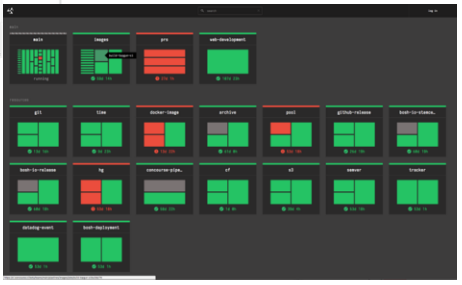
Concourse is a CI/CD server similar to Jenkins or Gitlab.

- Work is performed in jobs. A job has a build plan, a sequence of steps to execute. A job either succeeds or fails. Job execution can be made contingent on the pass/fail status of any other job.
- All jobs are executed in containers, typically on worker nodes. Keeps all worker nodes generic (cattle not pets)
- Important: No way of passing state between jobs. This is considered a Good Thing.
  - Require suitable storage for artefacts, logs, etc. Keeps the pipeline relatively independent of the choice of CICD server.
- Jobs fetch data via input resources and put data via output resources.
- The meaning of get (input) and put (output) is determined by the type of resource.
  - Git resource: input -> pull from a git repo. Output -> push to a git repo
  - Docker resource: input -> pull image from registry. Output -> push image to registry

Visualising the pipeline status



	build	test: integration-&-quality	test: functional	test: load-&-security	approval	deploy: prod
Average stage times:	836ms	20min 43s	9ms	7ms	89ms	5ms
#17 Sep 22 16:06 No Changes	536ms	10s	10ms	8ms	72ms	4ms
#16 Sep 22 16:04 No Changes	479ms	6s	9ms	9ms	74ms	5ms
#15 Sep 22 16:03 No Changes	922ms	6s	10ms	9ms		
#14 Sep 22 16:03 No Changes	1s	8s	12ms	9ms	80ms	5ms
#13 Sep 22 16:02 No Changes	942ms	9s	13ms			
#12 Sep 22 16:02 No Changes	1s	6s	13ms	11ms	111ms	



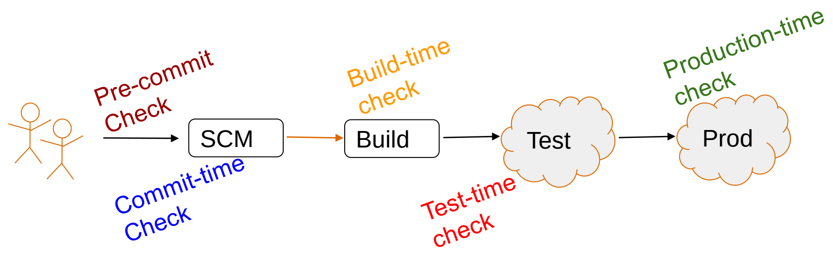
CI/CD good practices (1/2)

- Pipeline as code.
  - Text files, versioned, tested, stored in CMS.
- No secrets in repositories or images.
  - Use something like truffleHog to find any accidentally committed secrets.
- Cattle not pets.
  - Automatically deployable pipeline without hacks
  - Be explicit about versioning: Same input produces the same output
  - Don't save state in the pipeline
- Perform all work in jobs which will be scheduled on worker nodes, freeing up master.

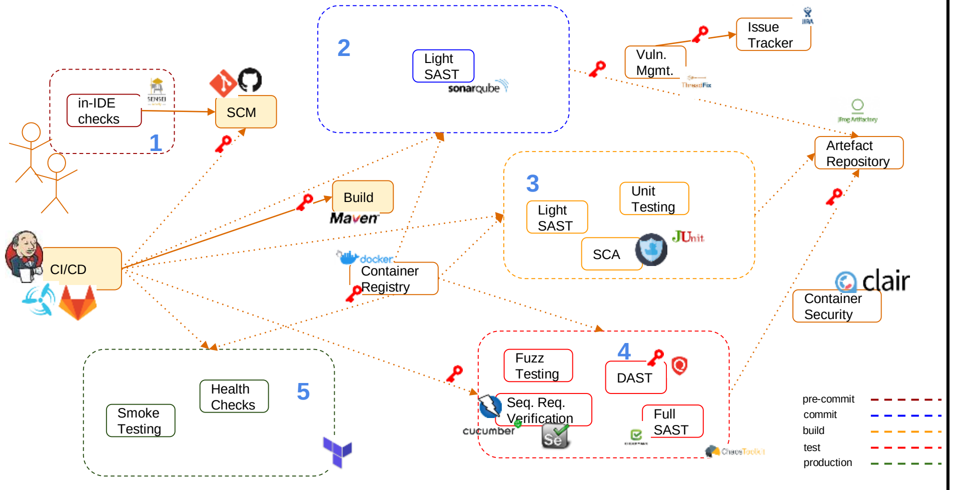
CI/CD good practices (2/2)

- Parallelise as many jobs as possible to reduce time to deploy
- Deploy the same artefacts you test
  - Reuse images promoted through the pipeline
- Add assurance throughout the pipeline (Compliance as code)
  - Promote artefacts
  - Automated quality gates
- Track every change through tickets

Integrating security into the CI/CD pipeline

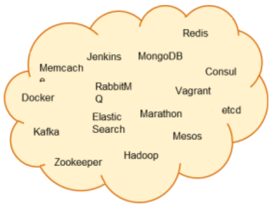


The Big Picture



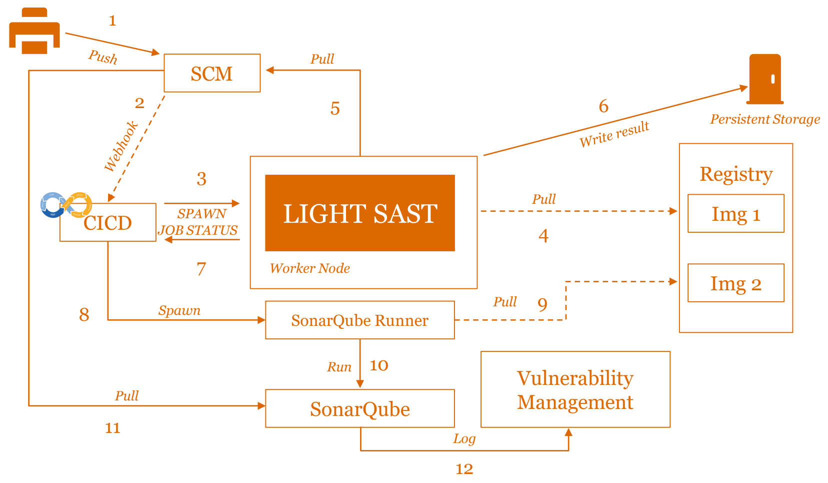
Challenges of a Modern CI/CD Pipeline

- Ensuring authentication and authorization at all relevant components of the pipeline (dev starts looking more like a prod network).
  - This is complicated by the relatively large attack surface area of many products and the need for unattended execution.
  - Kubernetes integration compounds the problem adding more secrets that need to be passed among different layers of abstraction (node, pod, container, application)
  - Temptation to pass secrets insecurely when no simple/obvious integration exists for tools
- Secure storage and retrieval of secrets in an automation-friendly manner
- Managing the security of myriad 3rd party components (in app and in container)
- Hardened configuration of the entire setup



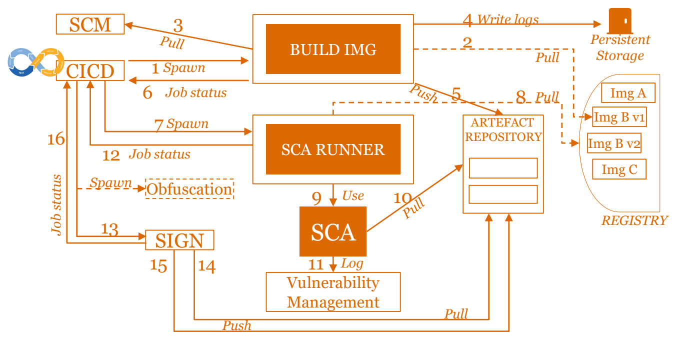
(Light) SAST

- Early lightweight static testing. Note this is in addition to pre-commit in-IDE checks.
- Only enable **fast** scans
  - SonarQube (code quality metrics and security plugins --replaces FindSecBugs)
- Only enable **low false positive** scans (good practice to have a few targeted scans specifically customized for the development environment)
- Limit the scan to the modified project
  - Consider incremental scans if tooling supports it



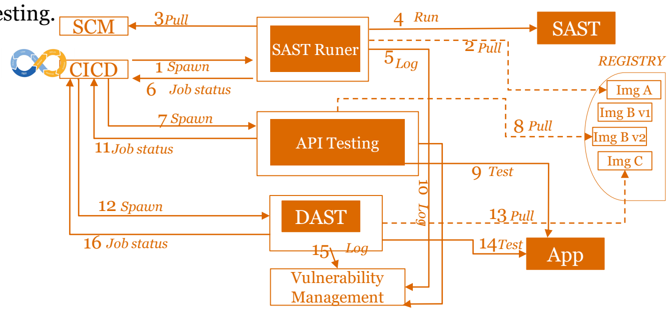
Build

- For **reproducible builds**, use explicit versions, don’t rely on default tags , such as latest.
- Run SCA scans.
- Treat databases as code, with versioned schema changes (e.g. Liquibase, Flyway..).
- Digitally **sign** the artefact



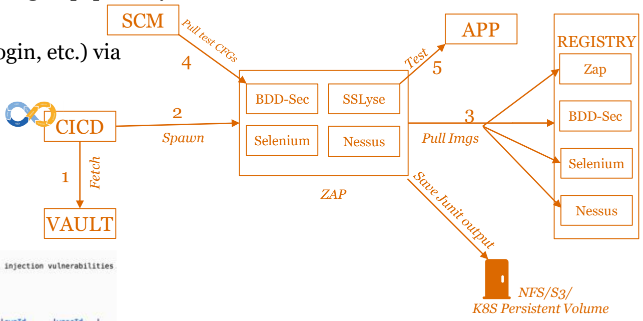
Testing

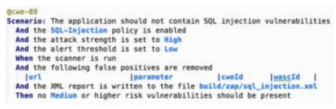
- Run multiple tests in parallel (Unit, API, Chaos..)
- Run heavier SAST regularly, but out of band (e.g. daily or weekly basis)
  - Commercial SAST solution, Free solutions currently too far behind to be useful
  - Customize rulesets to minimize false positives.
- Free up QA and testers to perform manual exploratory testing on artefacts which pass automated testing.



Security requirements testing

- Human readable tests. Can be written by non-technical analyst or business owner
- Scans the application (e.g. zap, possibly with custom checks)
- Executes scenarios (login, etc.) via Selenium



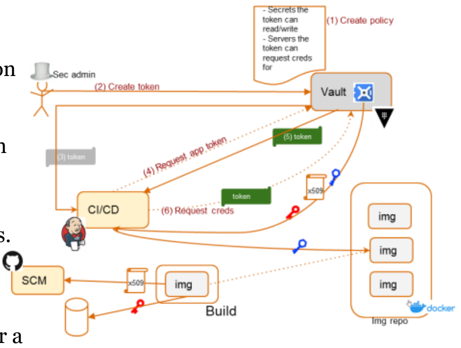


```
@cwe-89
Scenario: The application should not contain SQL injection vulnerabilities
  And the SQL-Injection policy is enabled
  And the attack strength is set to High
  And the alert threshold is set to Low
  When the scanner is run
  And the following false positives are removed
    |url                    |parameter          |cweId      |wascId   |
  And the XML report is written to the file build/zap/sql_injection.xml
  Then no Medium or higher risk vulnerabilities should be present
```

Src: https://continuumsecurity.net/bdd-security/

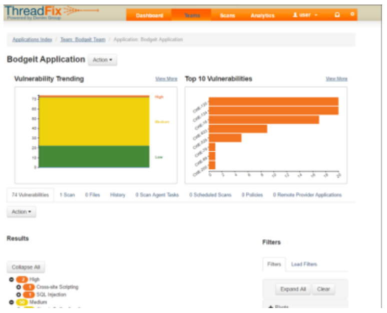
Secrets: Overview of a vault solution

- Server is administered by security.
- Create security policies (e.g. Application X identified by Token Y may access Secrets A,B,C, may request credentials for servers S,T, may request encryption or decryption under key K, etc.
- Entities identified by tokens (limited validity or uses) are mapped to policies.
- Automatic key rotation
- Dynamic creation of credentials (e.g. create and return ODBC credentials for a postgresql server and revoke them X minutes later)
- Generate X509 certificates on the fly and handle their revocation.



Application vulnerability management

- Centralise application vulnerability test results (SAST, DAST, SCA, WAF, GRC, Defects..)
- Results are reviewed by Application Security experts*
  - Remove false positives
  - Prioritise
  - Create tickets in the issue tracker.



** In the true spirit of agile/devsecops, most review and ticket creation should be performed by the dev team themselves, not "outsourced" to security (end-to-end responsibility). This requires adequate training (e.g. security champions).*

"Security is everyone's concern"

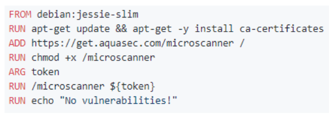
Containers (1/2)

Convenient but add complexity and potentially increase an application's attack surface.

- Define images in text files, versioned, and stored in SCM.
  - For reproducible builds, be explicit about versioning, don't use the latest tag.
- Reduce the container's attack surface by using minimal base layers (alpine, busybox, scratch, distroless..)
  - But, beware regressions due to insufficient testing against non-standard libraries (musl, uclibc)
- Never include secrets in build images. Use build-time --secret
- Do not pass secrets to containers on the cmdline or via the environment. Use docker secrets.
- Store images in a central registry
  - Regularly scanned for vulnerabilities
  - Semantically tagged (production, testing, secure, etc.)
    - certified/policy enforcement gives more assurance in automatically deployed artefacts.

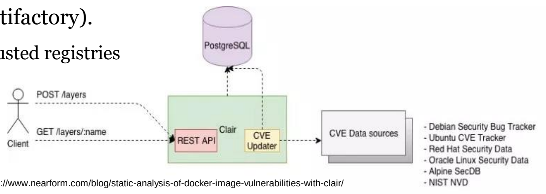
Containers (2/2)

- MicroScanner can scan at build time. No docker image is produced if too many vulnerabilities are found.



```
FROM debian:jessie-slim
RUN apt-get update && apt-get -y install ca-certificates
ADD https://get.aquasec.com/microscanner /
RUN chmod +x /microscanner
ARG token
RUN /microscanner ${token}
RUN echo "No vulnerabilities!"
```

- Most other container scanning tools require a registry (Clair, Anchore). This can be conveniently combined with an artefact repository (e.g. artifactory).
  - Leverage policy-backed trusted registries



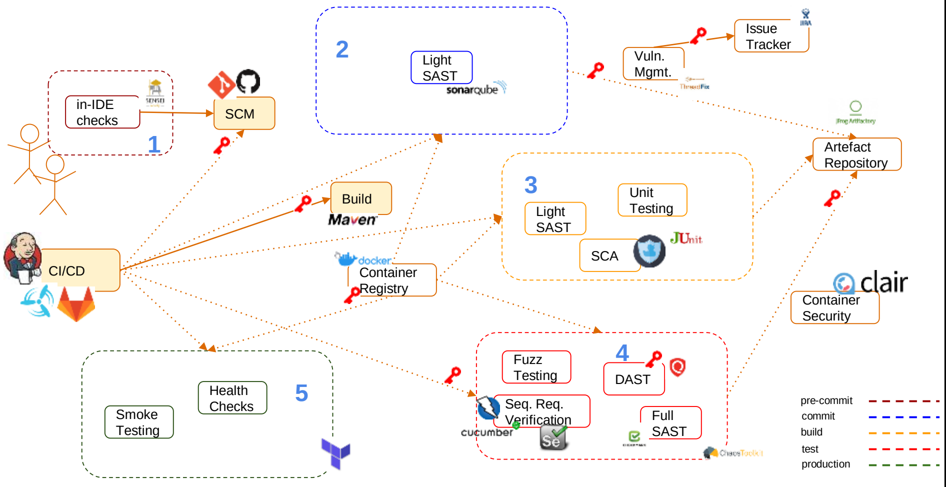
Src: https://www.nearform.com/blog/static-analysis-of-docker-image-vulnerabilities-with-clair/

Containing containers

Harden hosts against malicious or runaway containers (e.g. CVE-2019-5736)

- Limit host resources available to a container, e.g. disk, CPU, memory, mounted volumes, etc. (cgroups)
- Run containers in their own kernel namespaces (at least the user namespace to map the root user in a container to a harmless user on the host)
- Run the docker daemon under secure configurations (Apparmor profiles, SELinux labels, Seccomp profiles..)
  - Never expose the docker socket
- Run containers with their own network when possible.
- Audit hosts regularly using the Docker Bench for Security.
- Use an internal registry to cache versions of images used.
- Use signed (and verified!) images

The Big Picture (Recap)



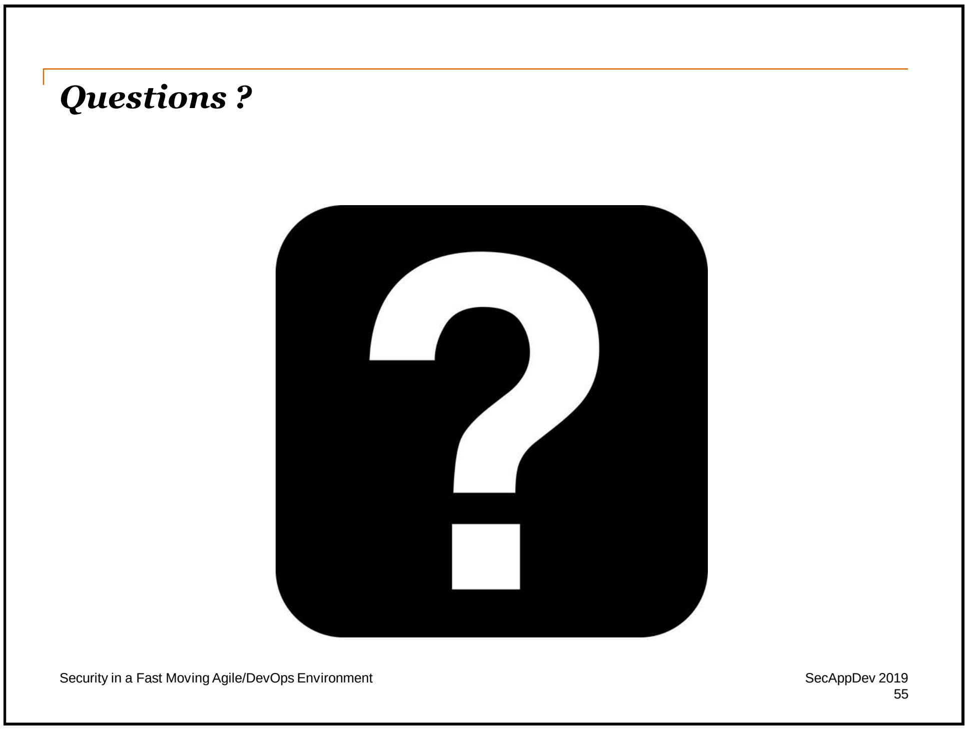
Conclusion

Conclusions

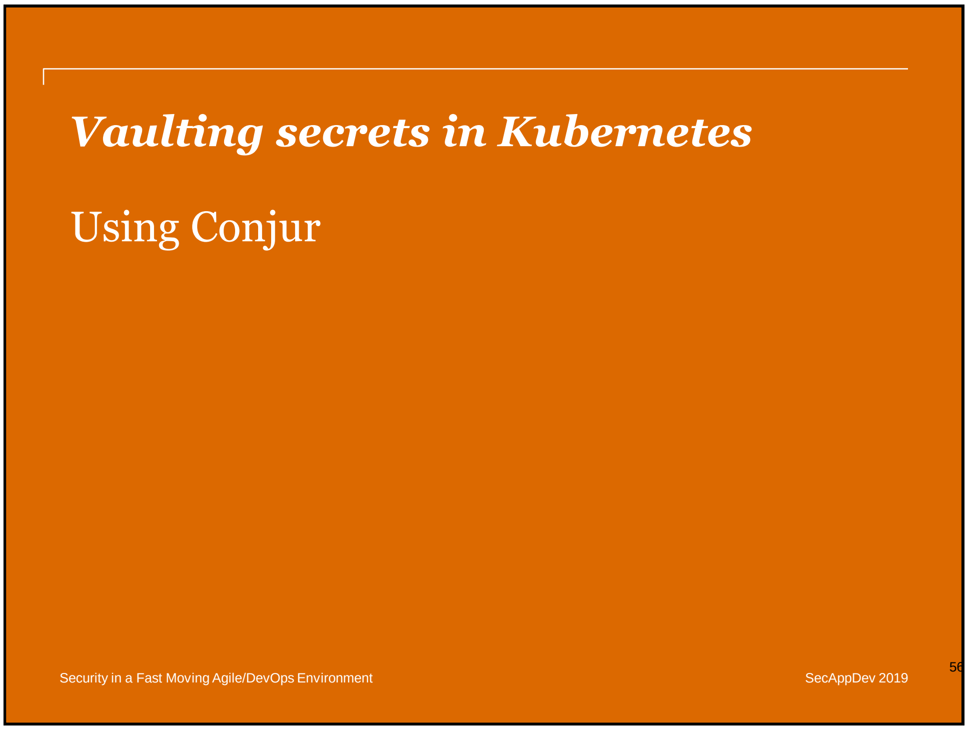
High-speed development environments force us to radically rethink how we organise secure development.

At the same time, frequent software deployments also change the risk posture of the application towards the organisation.

Successful security in such environments relies on empowerment, shared responsibility and automation.



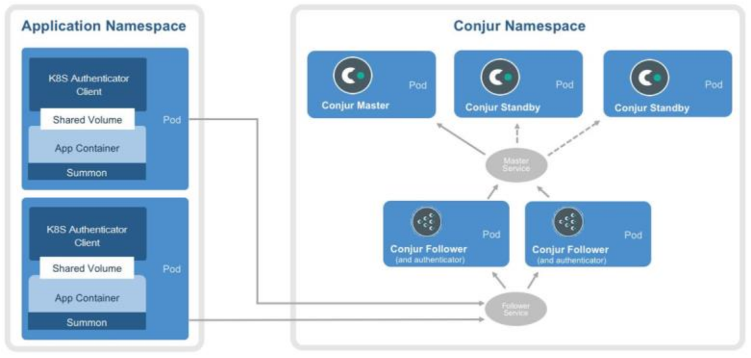
Questions ?



Vaulting secrets in Kubernetes

Using Conjur

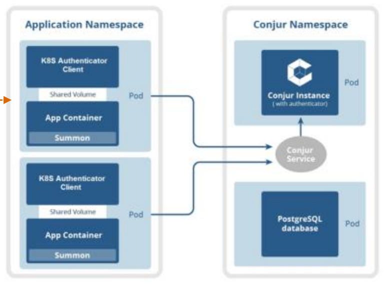
Use case: Kubernetes



Use case: K8s (1/4)

K8s Authenticator client (conjur-authn-k8s-client)

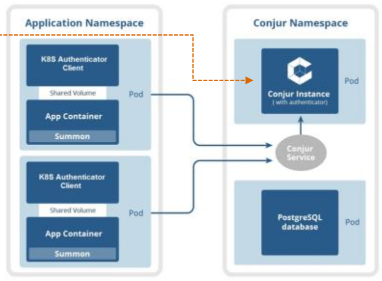
- Talks to the Authenticator service (running in the Conjur instance) on behalf of the application to obtain an access token to login to Conjur.
- Write token to the shared volume.
- Run as a **sidecar** container (continuous) for apps requiring repeated conjur access (e.g. due to secret rotation)
- Run as an **init** container (1-time)



Use case: K8s (2/4)

Conjur k8s-authenticator plugin:

Conjur plugin, a service which authenticates application pods.



Use case: k8s (3/4)

Assume conjur-authn-k8s-client runs as a sidecar container

- Verification: Conjur checks that requests come from a valid pod / namespace
- Sidecar generates a CSR, saves the private key
- Sidecar makes a login request to Conjur (sending pod name, namespace, CSR)
- Conjur (via k8s-authenticator plugin) handles the login request, generates a signed certificate and saves it to shared memory (out of band)
- Conjur (via K8s-authenticator plugin) generates an encrypted access token for the certificate
- Sidecar decrypts the access token (with a private key) and saves it to shared memory
- Summon uses the access token to request secrets on behalf of the application

Use case: k8s (4/4)

- Prepare and load a Conjur policy, covering:
  - Human users (k8s admin, devops, dbs..), k8s-authenticator plugin (which hosts & apps can authenticate
  - Applications: Which applications can access which secrets and how. Also any humans to create them.
- Configure Conjur k8s-authenticator plugin: link to conjur policy..
- Select & deploy k8s authenticator client (conjur-authn-k8s-client): Sidecar or init. Edit the application container manifest.
- Add other resources to app manifest: shared volume, various conjur & k8s variables (pod name, namespace, Conjur authentication token file, Conjur account, …)
- Adapt the application to retrieve secrets
  - Use conjur API
  - Use summon

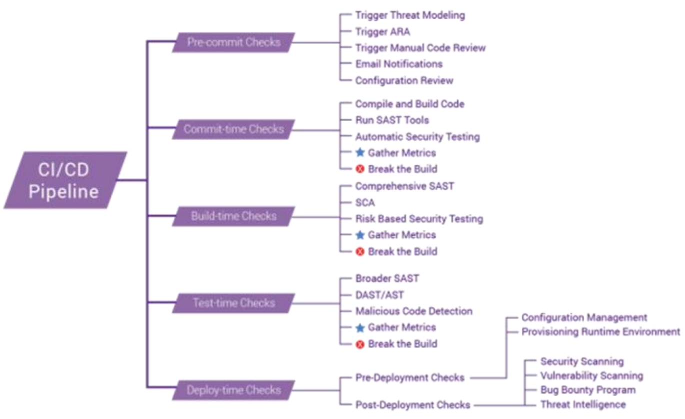
Misc

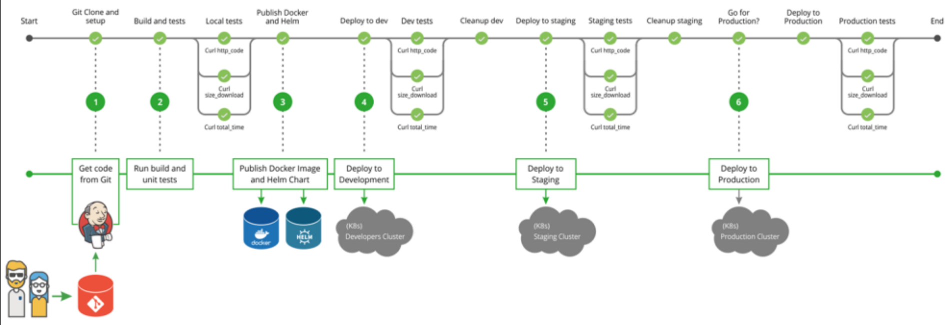
Not mentioned

Orchestration (e.g. Terraform)

- Which services to run in containers and which not?
- What to run in kubernetes and what not?
- Caching (e.g. node packages, pulled source code..)
- Integrating with kubernetes (creating helm packages, k8s deployments, etc.)

CICD security graphically





Deployment

- Automated (e.g. terraform)
- Smoke tests
- Multiple environments
- Monitoring
  - Regular in-app health checks

Questions ?