DevOps, CI, APIs, Oh My!
Security Gone Agile

Matt Tesauro, SANS AppSec 2014
– Austin, TX, February 2014
Who am I?
Matt Tesauro – Cloud Application Security Guy + OWASP

Racker since October 2011
Rackspace’s Product Security Group
Product Security Senior Engineer
Work with developers and QE

matt.tesauro@rackspace.com
matt.tesauro@owasp.org

Former OWASP International Foundation Board Member and Treasurer
Project Leader of
OWASP Live CD / OWASP WTE
OWASP OpenStack Security Project
DevOps, CI, APIs, Oh My!
A quick Overview of DevOps

• The combination of traditional development activities with operations and testing (QA/QE)

• Collaboration, communication and integration is key

• Agile development model (sprints, scrum, stories…)

• Release coordination and automation

"DevOps" is an emerging set of principles, methods and practices for communication, collaboration and integration between software development (application/software engineering) and IT operations (systems administration/infrastructure) professionals.
CI, CD, CD, TDD and API

CI == Continuous Integration

CD == Continuous Deployment

CD == Continuous Delivery

TDD == Test Driven Development

API == Application Programming Interface
THE PROBLEM

• Cycle time for software is getting shorter

• Continuous delivery is a goal

• Scanning windows are not viable

• First mover / first to market advantage
THE PROBLEM – or at least more

• Traditional software development left little time to test
• DevOps, Agile and Continuous Delivery squeeze those windows even more
• New languages and programming methods aren’t making this better
• Growth of interpreted languages with loose typing hurts static analysis efforts
• Few automated tools to test APIs especially RESTful APIs
• Little time for any testing, manual testing is doomed
THE SOLUTION

- Automated software testing
- Automated operational infrastructure
- Automated security testing
Think like a developer

Sprints break software into little pieces…
• Break your testing into little pieces
• Use your threat model to know the crucial bits to test

Long and short running tests
• Testing time drives testing frequency
• Code for tests needs to be optimized

Smoke test versus full regression test
• Smoke test early and often
• Full regression tests on regular intervals
Maximize what you’ve got

Make the most of your frameworks
• Embrace, understand and fill gaps where necessary

Make the best use of your time…
• Make tests easily repeatable
• Make tests easy to understand
• Make tests abstract and combine-able
• Ala carte tests for mixing and matching
• Think about the Unix pipe | and its power
Test Driven Development Security

Under the constraints of DevOps, Continuous Deployment

Your testing has to be nimble

Dare I say…Agile

In TDD, you know your code works when the tests pass

In TD(S), you know your app has met the baseline when the tests pass
A time to morn...

TRADITIONAL APPLICATION SECURITY

WE HARDLY KNEW YOU...
5 Stages of Grief

This agile thing is a fad...

Waterfall is the only way to produce quality software...
5 Stages of Grief

There's no way I can test in that time frame...

If I see another freaking sticky note...
5 Stages of Grief

Well, I think I can test some of it in two days...

I guess I can test it after its deployed to prod...
5 Stages of Grief

After that launch, I updated my LinkedIn profile...

Game over man, GAME OVER...

(Thanks Aliens)
5 Stages of Grief

So when can you add a story to work on that auth regression...

After reviewing your deployment recipe, we filed a pull request to fix...
Fly thought those 5 stages by addressing...

• Securing Infrastructure
• Securing Apps and APIs
• Securing Code
Securing Infrastructure
Automating Infrastructure

- Declarative configuration language
- Plain-text configuration in source control
- Fully programmatic, no manual interactions
Chef for example

1. Solo
2. Server
3. Hosted
4. Private Hosted
Cookbooks, Stacks, Playbooks, ...

- Most have methods to bundle / share automation routines
- You will have to write your own / customize
- Good place to spend security cycles
  - Merge patches upstream for extra points.
Grouping & Tagging

- Tagging your servers applies the required set of automation
- A base set of for all servers
- Each server can have multiple tags
- Map tags to security requirements
Inspector – you need one

• For each group and/or tag
• Review the recipe
• Hook provisioning for post deploy review
• Focus on checking for code compliance
  - Not perfection, bare minimums
• Can include multiple facets
  - Security
  - Scalability
  - Compliance
Agent – one mole to rule them all

• Add an agent to the standard deploy
• Read-only helps sell to SysAdmin
• Looks at the state of the system
• Reports the state to the “mothership”
• Add a dashboard to visualize state of infrastructure
• Change policy, servers go red
• Watch the board go green as patches roll-out
• Roll your own or find a vendor

CloudPassage
Turn Vuln scanning on its head

• Add value for your ops teams
• Subscribe and parse vuln emails for key software
• Get this info during threat models or config mgmt
• Provide an early warning and remove panic from software updates
• Roll your own or find a vendor
• Gmail + filters can work surprisingly well
• Secunia VIM covers 40K+ products
• Reverse the scan then report standard
Securing Apps & APIs
Findings directly to bug trackers

- PDFs are great, bugs are better
- Work with developer teams to submit bugs
- Security category needs to exist
- Bonus points if the bug tracker has an API
- Security issues are now part of the normal work flow
- Beware of death by backlog
- Occasional security sprints
- Learn how the team treats issues

- ThreadFix is nice for metrics and pumping issues into issue trackers - http://code.google.com/p/threadfix/
For the reticent: nag, nag, nag

• Attach a SLA to each severity level for findings
• Remediation plan vs Fixed
• “Age” all findings against these SLAs
• Politely warn when SLA dates are close

• Walk up the Org chart as things get older
• Bonus points for dashboards and bug tracker APIs
• Get management sold first
Reports = Findings + Automation

• Consider markup for findings
• Markdown, Wiki Text, asciidoc
• Pandoc to convert to whatever
• HTML, PDF, .doc, .odt, ...
• Keep testers writing the least possible
• Template and re-use boiler plate items
• New finding == new template for next time
• Web app to keep things consistent
• Create your own or maybe Dradis
Leverage existing consistencies

• Requires consistent (generally automated) input
• Find these and write some scripts
• Automate the drudgery
• Examples:
  • Automate finding/bug submission
  • Automate report PDF generation
  • API documentation to basic testing harness
• Sec tool output – combine and convert
Securing Code
Start with the developers

• Finding details have to be detailed enough to:
  • Reproduce the issue after 6 months
  • Allow QE to test the issue
  • Allow developers to find/fix the issue
  • Consider quick and dirty scripts to reproduce issue
  • Script to abuse an API
• Web page of reflective XSS findings
• Gauntlt - http://gauntltt.org/
• Once findings start flowing, look for training requests
Cherry pick what you look at

- Threat Models are your friends
- Focus on weak, unclear or suspicious areas
- Focus on connections with external systems
- Focus on format translations (XML to JSON)
- When code changes in those areas,
- Red flag it for review
- Change +2 to +3 to before accepting pull request
- Use search features in source code management
- Start a list of problematic methods, calls, etc
No False Positives, period

• If you can automate code review, you still must triage
• 1 false positive == 100 valid bugs
• If results aren't actionable, fail
• Stick to diff analysis
• Threat Modeling + “Scary Parts” + Code diffs == Quick triage of code changes
• Automate where you can, iterate until you're happy
• Need to build cred points with the dev teams
Quiet is better then wrong

• Hire or befriend developers
• Need to speak their language, not security's
• Suggest requirements not implementation
• Mitigation suggestions either generic or in the language the app is written in
• Remember: Fast deploys also means fast fixes
• Trying to shrink any vuln window not eliminate
• Be prepared to retest / verify fix quickly
What is Rackspace's Product Security doing?
Securing Infrastructure

- Rack has Chef, Puppet, Salt and Ansible, depending on the team
- Reviewing the deployment scripts
- Validating them with external vuln scans
- Re-checks after bug fixes
- Rack is using CloudPassage as a “mole” for some deployments
- Also have some mole-like agents for one-offs
- Rack has been conducting threat models ++ and using that info to watch for vulnerabilities
Securing Apps and APIs

• Product Security finding workflow
• PS team member find an issue
• Documents it in Test Tracker app
• Pushed finding(s) to ThreadFix
• ThreadFix integrates with bug trackers
• Metrics are driven off the ThreadFix database
• We're re-implementing the nag, err reminder script for the new workflow
• Using asciidoc markup for findings – easily creates PDFs, HTML, doc, reports based on templates
Securing Code

• Rack is using Veracode if the language is supported
• Self-service for the dev teams
• Jenkins integration for submitting code to scan
• API automation to pull findings into our workflow
• PS team produces detailed finding blocks
• Creates quick re-test scripts ad-hock
• PS team holds trainings and has e-learning modules
• PS team works with devs daily
• Loaned to teams, attend stand-ups, …
• PS “Dev Days” - team works on our automation
Key take aways

• Automate, automate, automate
• Look for “paper cuts” and fix those first
• Finding workflow
• Figure this out and standardize / optomize
• Create systems which can grow organically
• App is never done, its just created to easily be added to over time
• Finding blocks become templates for next time
• Learn to talk “dev”
Change is here and more is coming…

"Whosoever desires constant success must change with the times."
— Niccolo Machiavelli