Practical tips for defending web applications in the age of agile/DevOps

zane@signalsciences.com
@zanelackey
Who you’ll be heckling today

• Started out in offense
  – Pentester / researcher at iSEC Partners / NCC Group

• Moved to defense
  – First CISO at Etsy, built and lead the security group

• Now scaling defense for many orgs
  – Co-founder / CSO at Signal Sciences, delivering a product that defends web applications in the DevOps/Cloud world
So what is this talk about anyway?

Lessons learned adapting AppSec/SDLC from a Waterfall world to the DevOps/Cloud world
Spoiler: Security shifts from being a gatekeeper to enabling teams to be secure by default
What has changed?
The new realities in a DevSecOps world:

1. Changes happen multiple orders of magnitude faster than previously
   - Deployments go from a few a year to a few a week, month, or even day
   - Many injection points for security drops to few injection points

2. Decentralized ownership of deployment:
   - The long and perilous journey of Dev->QA->Security->Dev->Sysops->Production becomes just Dev->Production
   - As Dev/DevOps teams own their own ability to build and deploy production infrastructure/apps, conversations with security become opt-in rather than mandated
     - A large culture shift is necessary around this
       » Spoken previously on this:
         http://www.slideshare.net/zanelackey/building-a-modern-security-engineering-organization
The new realities in a DevSecOps world:

- Security can no longer be “outsourced” to the security team, but rather that the security team’s mission changes to providing the resources for teams to be security self-sufficient.

- Security only becomes successful if it can bake in to the Development/DevOps process.
How do legacy AppSec approaches fare in a DevSecOps world?
An example of legacy AppSec approaches in a DevOps world
Select components of common SDLCs:

- Developer Training
- Threat modeling
- Design Reviews
- Static Analysis
- Dynamic Scanning
- Pentesting
- Feedback
What pieces of the SDLC need to adapt the most?
Which components we’ll discuss today:

- Developer Training
- Threat Modeling
- Design Reviews
- Static Analysis
- Dynamic Scanning
- Pentesting
- Security Visibility
- Feedback
- Continuous Feedback

• Note: Just because we’re not discussing several of these items in this talk doesn’t mean you stop doing them!
Static Analysis: It’s not a party until the 32847326th page of the report!
Static Analysis (legacy):

- Traditionally done as heavyweight process:
  - Run once a week/month resulting in a large output
  - Extensive configuration/tuning period, typically lasting months+
  - Top down: search for everything, slowly refine to eliminate false positives

- Both of these issues were acceptable-ish in a Waterfall world where you had plenty of time in each release cycle
How do we adapt this control?
Static Analysis (modern):

- Shift from a top-down model to a bottoms-up one:
  - Identify specific classes of vulnerabilities you care about most, and start with just those
  - Focus on eliminating false positives and enabling velocity with the goal of only producing real issues that can be directly consumed by a developer themselves
  - Once completed, add one or two more vulnerabilities classes
  - Repeat

- This enables the velocity needed in DevOps of being able to run static analysis on every code commit
Static Analysis (modern):

– Example: Rather than trying to start with static analysis for XSS, SQLi, Directory Traversal, Command Execution, etc all at once, pick one:
  - Pro tip: Pick the easiest to implement first, (ex: Command Execution)
    - Grep’ing for system() has a pretty low false positive rate…

– The focus is not only on findings, but demonstrating to the development org that this approach to static analysis can bring them both value and velocity
Static Analysis (modern):

- Identify use of certain primitives that should initiate a conversation with the security team rather than just be blocked:
  - Ex: Hashing, Encryption, File system operations, etc

- Common example: Use of hashing or encryption functions
  - Old approach: “MD5 is banned, use SHA256!”
  - New approach: “Hey, we saw you’re making use of a hashing function, can we chat on what you’re trying to protect?”
  - Silently allowing an approved hashing function to be used doesn’t help anyone in cases where it’s not the appropriate use, ex: a case where the data should be encrypted not hashed
Static Analysis (modern):

- Build proactive alerting to know when sensitive and rarely changed portions of the codebase have been modified
  - Can be as simple as alerting on when the hash changes on certain key files
    - Ex: authorization primitives, session management, encryption wrappers, etc
  - By not blocking on these changes, you don’t impact velocity but you ensure that the relevant security/development engineers know if key platform protections are being changed
TRUST ME

THIS IS SCIENCE

Dynamic Scanning
Dynamic Scanning (legacy):

- Used to meet a baseline standard of discovering vulnerabilities:
  - Ex: “If a scanner can find it, we should probably fix it”

- Occasionally even (mis)used as a substitute for pentesting
How do we adapt this control?
Dynamic Scanning (modern):

– Applications architectures and functionality have changed significantly since scanners were pioneered in the early-mid 2000s
  • Modern applications are often far too complex to be effectively covered by scanners
    – Client side functionality, single page applications, etc.

– In the old use cases there’s too little bang for the buck from scanners when used with modern apps

– However, scanners can be adapted to two cheap and effective use cases:
Dynamic Scanning (modern):

1. Ensuring that security policies are being enforced
   • Ex: TLS only supporting strong ciphers
   • Ex: Crawl the app and ensure that CSP exists, or that X-Frame-Options header is always set to DENY

2. An extra control on ensuring previous vulnerabilities aren’t accidentally regressed back into the application:
   • Ex: We had an XSS in this parameter before, always check it with this specific payload to ensure the protection didn’t get accidentally rolled back
Our new monitoring product just watches Twitter and IRC for our name + "down".
Security Visibility (legacy):

- Logs, customer service reports, outages

- Each source of information was generally isolated in who had access to the data
  - Ex: Ops had logs, customer support dealt with emails from customers, outages would page only certain dev or ops on-call / leads, etc
How do we adapt this control?
Security Visibility (modern):

- GOAL: Break down the previous silos of data isolation and empower security, development, and DevOps teams to all know security relevant information from their application in real time.

- This isn’t a new idea! Take principles of general operational visibility, and apply to a security perspective.
  - Superset of operational data + security relevant data.
This graph provokes wildly different assumptions from Development, DevOps, and Security teams
Context is key, for *all* groups
Feedback
Feedback (legacy):

- Typically done as annual pentests

- Unfortunately this really only answers the question “do I have bugs?”
  - Spoiler alert: The answer is yes. Always.

- When applications were released annually or bi-annually that could be “real time enough” feedback
How do we adapt this control?
Feedback (modern):

- Combination of bug bounty + pentests

- Bounty is not a replacement for pentest, it augments pentest
  - Value is in the continuous nature of it, whereas pentests can be more directed

- Bounty gives general but more real time feedback, pentest shifts to giving more directed but less frequent feedback
Break out the Thought Leader-hosen!

It’s time for some thoughts on where modern application defense should be headed towards...
The hallmark of modern application defense is the combination of continuous feedback + visibility
To be successful against real attackers, you need to be able to answer the question:

“How do I know when my attackers are being successful?”
Three pillars of effective visibility + continuous feedback

1. Ability to detect attackers as early as possible in the attack chain
   – You want to know when the attacker discovers the vulnerability, long before the database goes out the door

2. Ability to continuously test and refine your vulnerability triage/response
   – The beauty of DevOps is that you can actually move faster than your attackers for the first time, especially the more you empower development / DevOps teams
Three pillars of effective visibility + continuous feedback

3. Ability to continuously test and refine your incident response/DFIR/SecOps process

- By treating even benign bug reports as sample incidents, you can continuously exercise and adapt your process

- Ultimately you want to be able to answer several questions for any given bug report that comes in:
  - Did this bounty participant find any additional issues they’re not reporting?
  - Was this reported vulnerability exploited previously?
  - etc
Continuous feedback loop success story:

“I discovered the vulnerability late Friday afternoon and wasn't quite ready to email it to them … [Etsy] had detected my requests and pushed a patch Saturday morning before I could email them. This was by far the fastest response time by any company I've reported to.”

- Source:
  https://www.reddit.com/r/netsec/comments/vbrzg/etsy_has_been_one_of_the_best_companies_ive
Conclusions

THE END IS NEAR
GOOD
• The thesis of modern application security is about shifting:

- **From**: A mindset of “Exclusively focus on gatekeeping controls to eliminate bugs before code is deployed”
  - (An impossible goal, bugs will never be fully eliminated)

- **To**: Focus on obtaining and refining continuous visibility and feedback from deployed applications, and providing security capabilities that make developers/DevOps teams security self-sufficient
Thanks!

Who's awesome?

You're awesome

zane@signalsciences.com  @zanelackey