MIS|TI™ PRESENTS

InfoSec World

Conference & Expo 2018

DEVELOPING TRUST AND GITTING BETRAYED

Clint Gibler
Research Director
@clintgibler

Noah Beddome
Director of Infrastructure Security
@domodogood


"TRUST FALL!"
DAMN IT FRANK, NOT AGAIN
www.funchilly.com
quickmeme.com

ABOUT US

Clint Gibler

- Midwesterner in SF
- Security Researcher
- Was filmed monologuing to a live emu

Noah Beddome

- Lover of Root Beer
- Security Researcher
- Former Marine

AGENDA

- Overview
- On Trust
- Development-focused environments
- GitPwnd
- Mitigations
- How do you scale security?
- Questions

OVERVIEW

We have spent the last few years attacking development-focused organizations and a key point of failure we observed was the bridging of trust zones. This was usually resulting from a variety issues caused by the implicit trust and lack of guidance around developers and DevOps.

The goal of this talk is twofold:

1. Discuss how trust relationships between people and infrastructure can be exploited to own a network
2. Present a tool, GitPwnd, that puts these ideas into practice
   a. Assist security teams in attacking and assessing development-focused organizations.
   b. Act as a concrete example to demonstrate the impact of common issues around trust and configuration related to developers and dev-centered environments.

AGENDA

- Overview
- **On Trust**
- Development-focused environments
- GitPwnd
- Mitigations
- How do you scale security?
- Questions

TRUST RELATIONSHIPS - DEFINITION

- The quantity and quality of trust between a number given entities such as people or trust zones
- Trust relationships are established based on the actual level of separation and segmentation implemented between trust zones
- Trust zones are areas of different level of security or sensitivity based on purpose and content
- Dev/DevOps engineers often modify trust relationships to get the job done


THE LIST OF PEOPLE
gapbagap.com
I TRUST

TRUST RELATIONSHIPS - TYPES

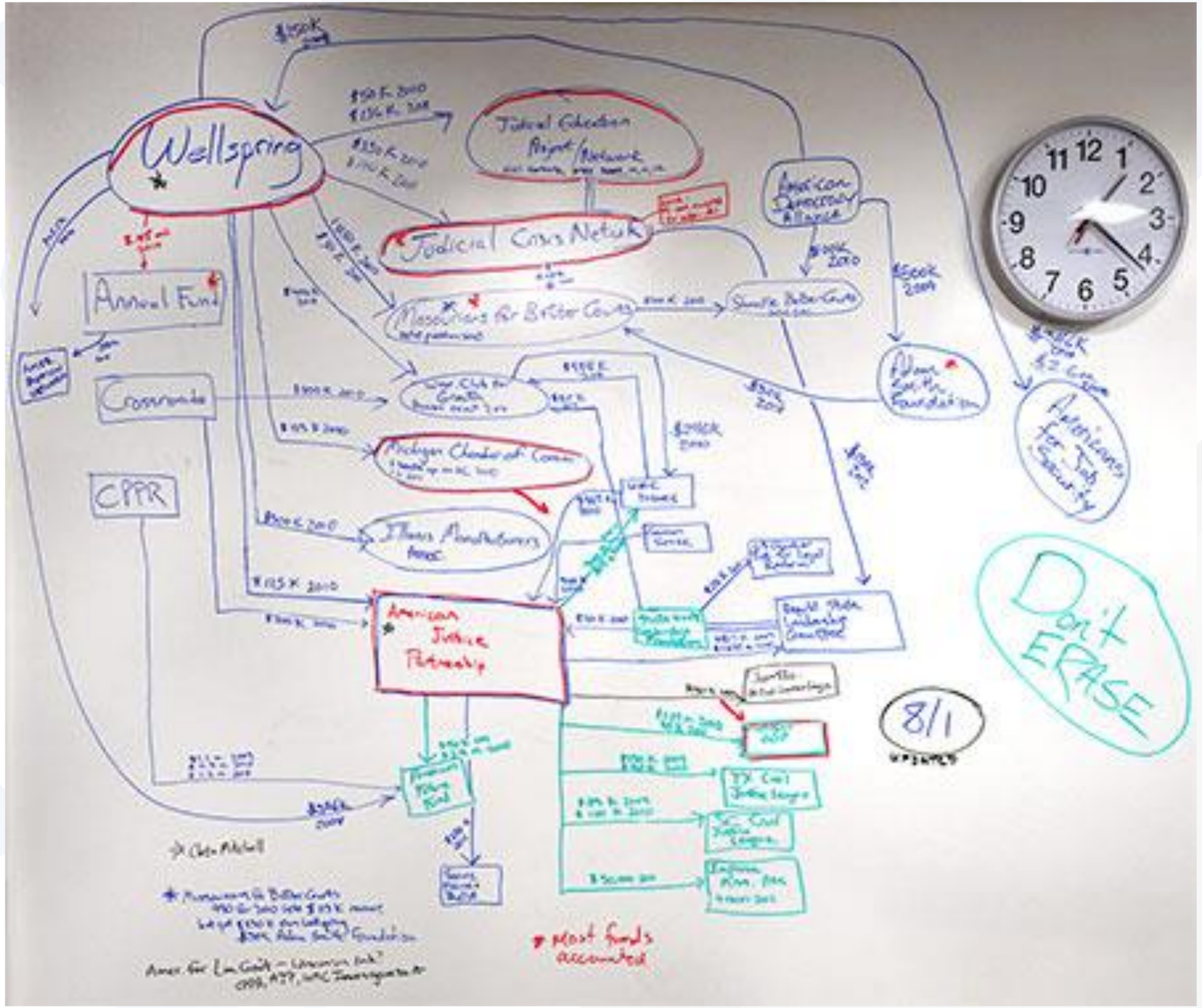
- Employer + Employee
  - Employer gives access to sensitive assets, employee installs software provided by employer
- Organization + Technology
  - Organization installs software on sensitive assets and systems
- Client + Organization
  - Client / consumer provides sensitive data such as SSN or PAN data to organization
- Project + Contributors
  - Project allows contributors to modify code
- System + Network
  - System connects to a network and sends traffic / interacts with network and other systems
- Network + Network
  - Networks allows communication of different levels between defined networks

TRUST RELATIONSHIPS - DISCUSSION

- The level of trust between entities may not be the intended level of trust.
- Segmented environment may be interlinked by credentials, dependencies, routing, processes, and applications.
- When rapid growth is essential, quality control is incredibly difficult.
- Trust is implicit, transitive, and inherited
- Example:
  - Production (“prod”) environment trusts the staging environment
  - Staging trusts a limited number of release devs
  - These release devs use internal tools developed by another team

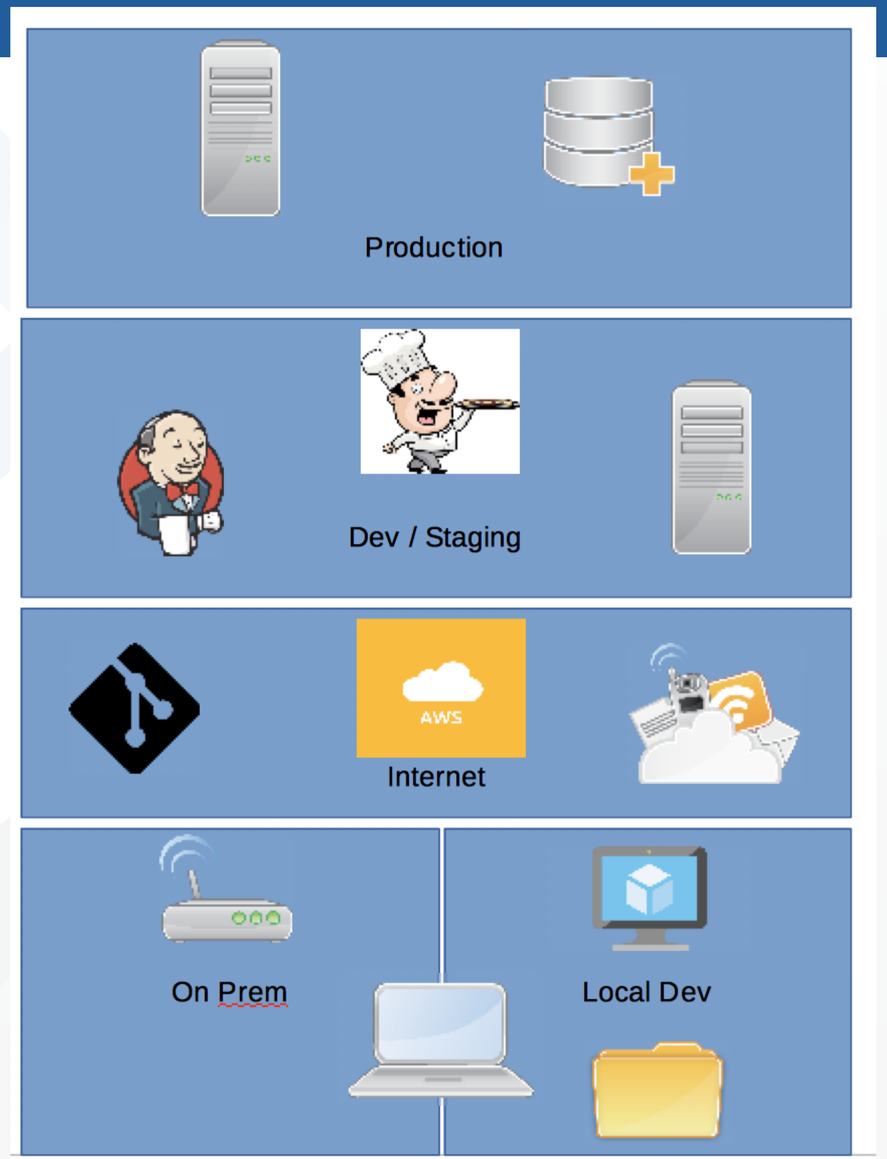
AGENDA

- Overview
- On Trust
- **Development-focused environments**
- GitPwnd
- Mitigations
- How do you scale security?
- Questions



DEV FOCUSED ENVIRONMENTS

- Workstations
  - Mac / Linux
- General Users
  - Higher base level of permissions
- Dev / Local Dev
  - Vagrant VS Dedicated Remote Dev
- Version Control / Code Repositories
  - Git, Stash, P4
- Continuous Integration
  - Constant testing and processing
- Staging / Prod
- Administration
  - Corp and Prod

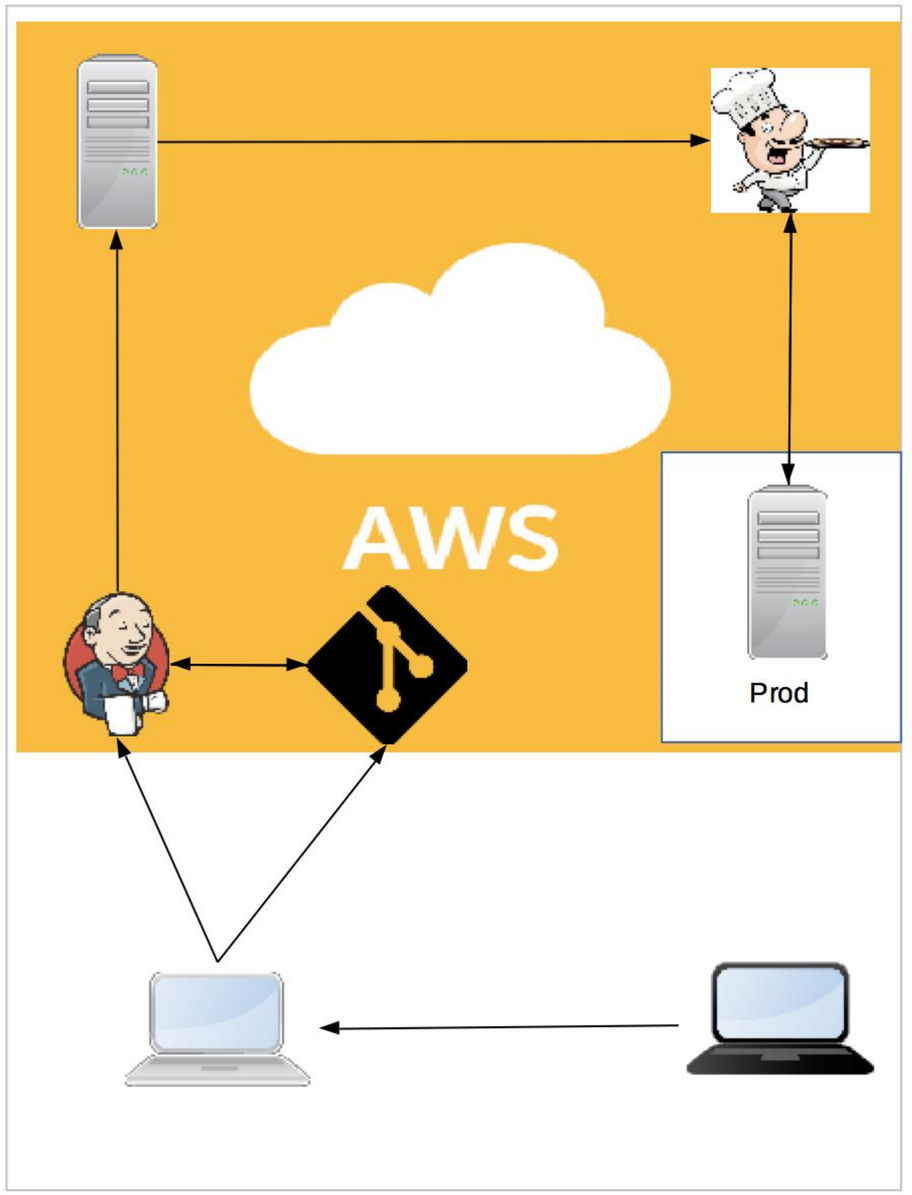


ATTACKING AGILE DEVELOPMENT ENVIRONMENTS

- Developer exploitation
- CI/CD compromise
- Attacking centralized authentication vs. local
- Pivoting and lateral movement
- Local vs. remote development

WAR STORIES

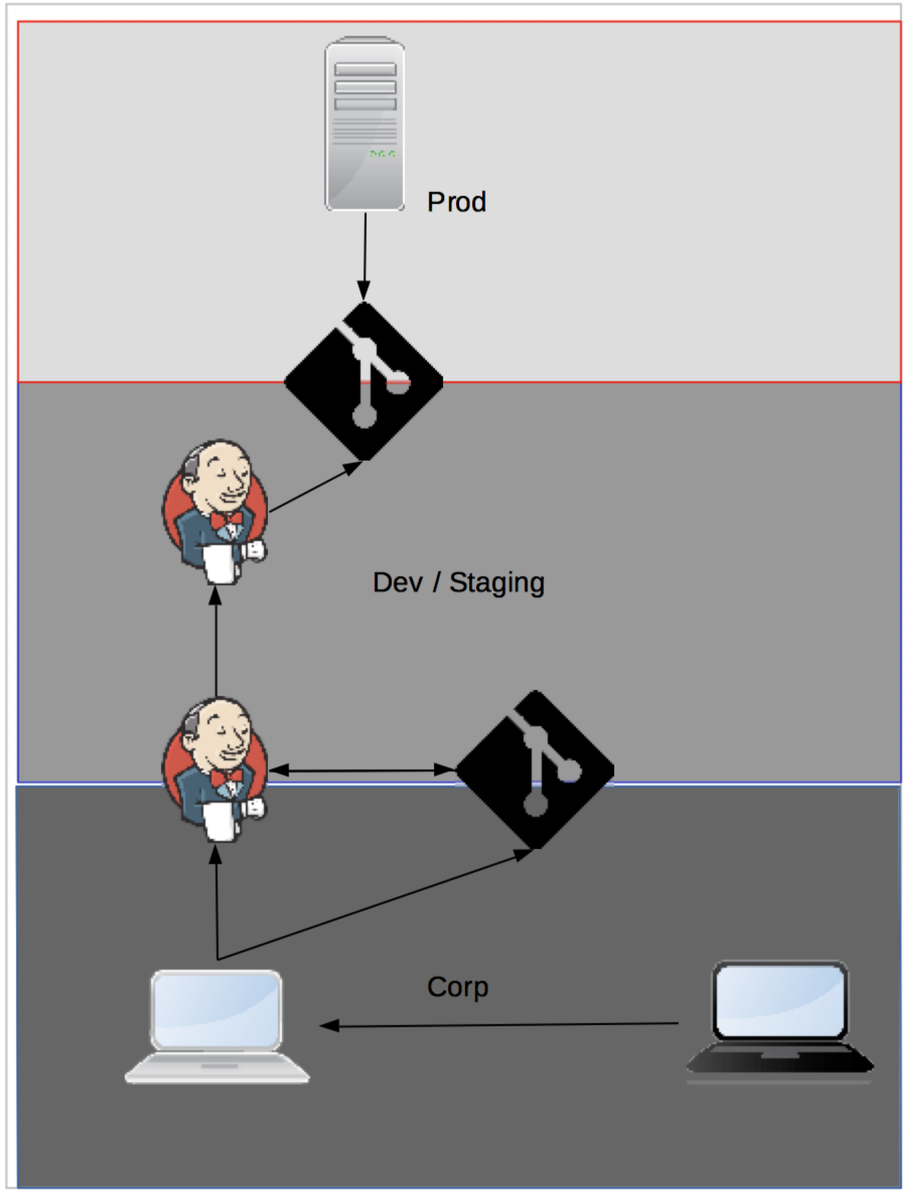
Executor



Developer

Attacker

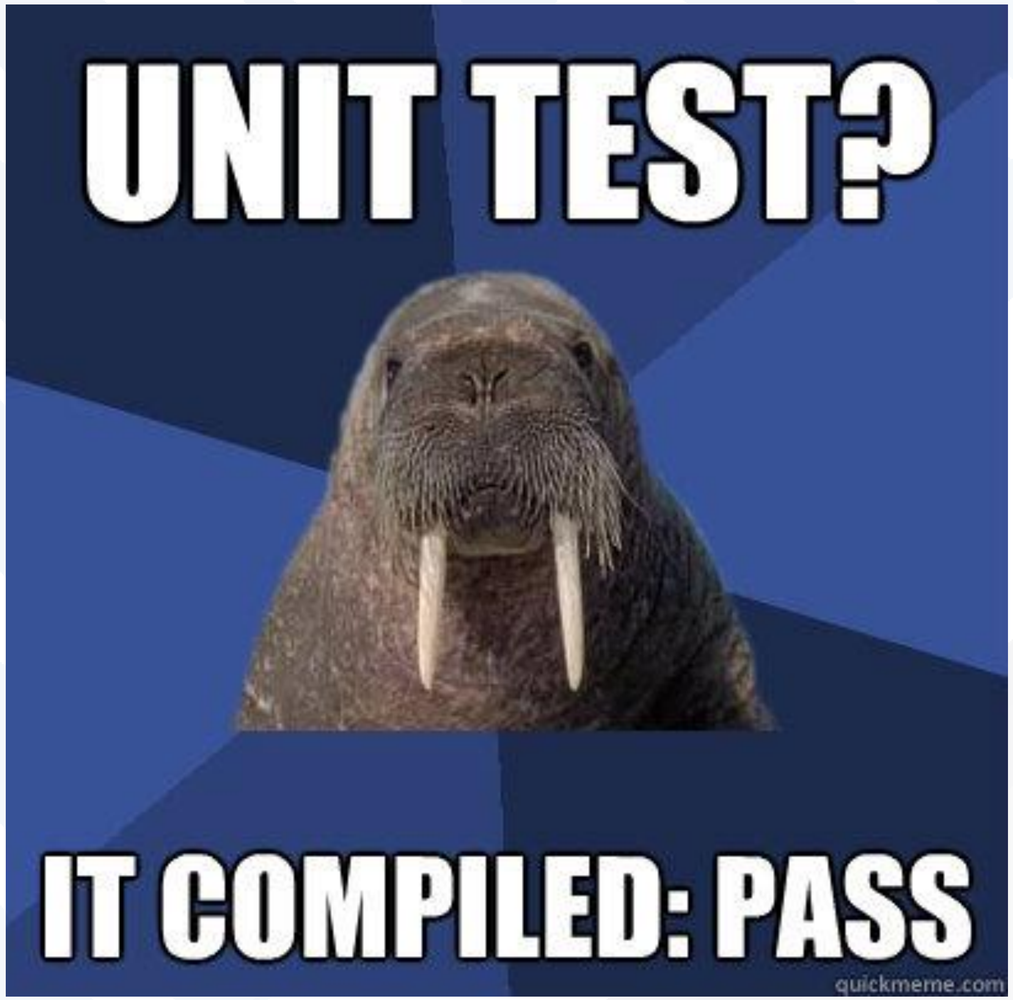
Compromise of Staging CI/CD results in takeover of Production Chef



Developer

Attacker

Access to Staging CI allows modification of repos replicated into Prod


UNIT TEST?
IT COMPILED: PASS
quickmeme.com

AGENDA

- Overview
- On Trust
- Development-focused environments
- **GitPwnd**
- Mitigations
- How do you scale security?
- Questions

TRADITIONAL COMMAND AND CONTROL (C2)

Detect:
- Agent-like processes
- Unusual behavior

Communication:
- HTTP
- IRC
- DNS

Detect:
- Unknown hosts
- Unknown IPs
- Examine traffic contents

Run commands

Send results

Send commands

Victim machine

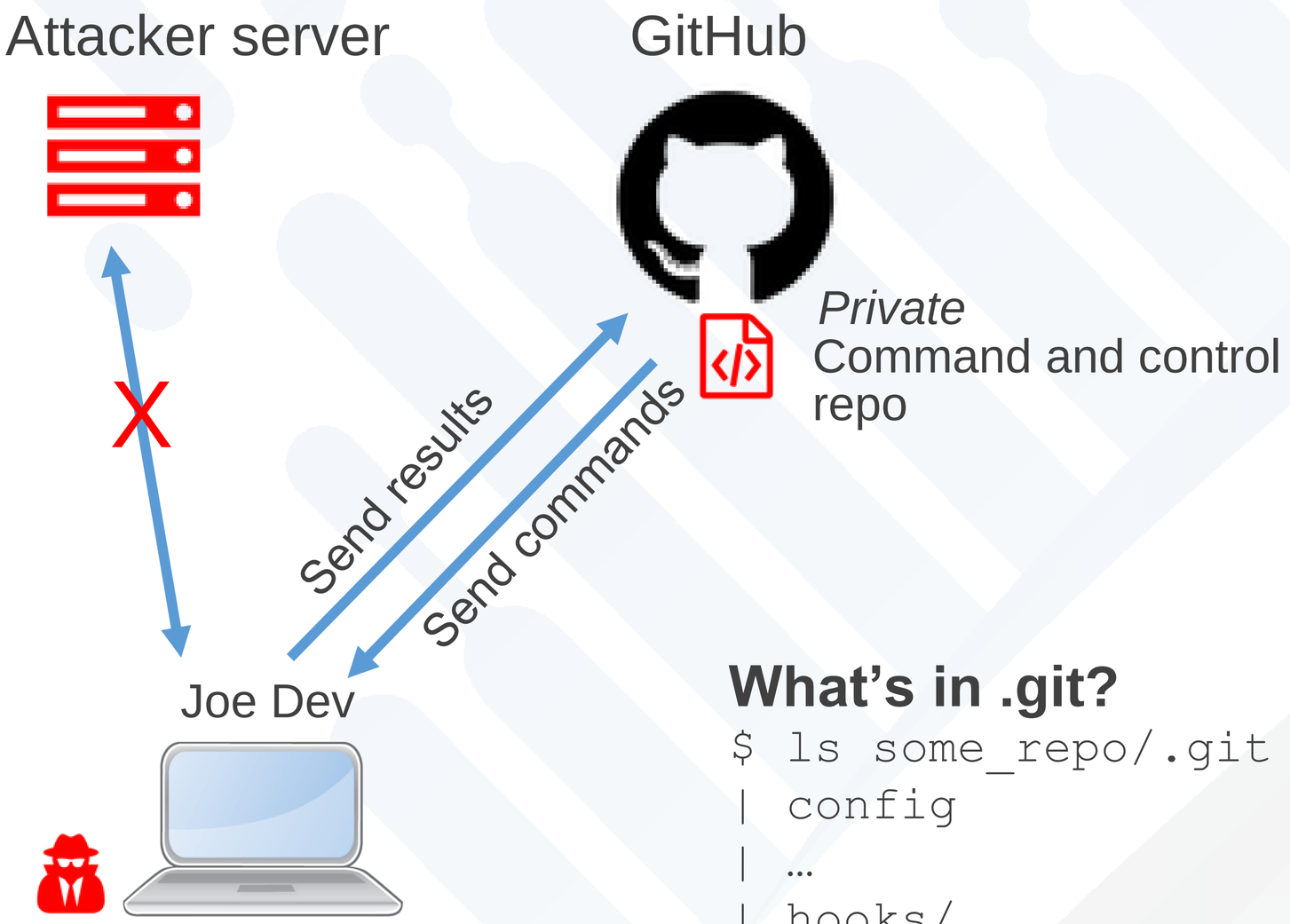
C2 Server

Goals:
- Communicate with trusted network parties
- Mimic user behavior

#GOALS

1. Be able to send commands to compromised machines and receive results
2. Communicate via trusted party
3. Mimic user behavior
4. Sub-goals
    1. Limit forensics
    2. Limit blast radius

ARCHITECTURE DECISIONS



#Goals

1. Command and control
2. Communicate via trusted party
3. Mimic user behavior
4. Sub-goals
   a. Limit forensics
   b. Limit blast radius

GitHub delivery mechanisms:

- ~~Gists~~
- Git repos

What's in .git?

```
$ ls some_repo/.git
| config
| …
| hooks/
|   pre-commit.sample
|   pre-merge.sample
```

Summary

- Use GitHub to host C2
- Private git repo is transport layer
- Git hooks used to mimic user behavior

DEMO!

FUTURE FEATURES

- Encrypting exfiltrated information
- Additional tool support
  - Version control: P4, hg
  - CI/CD platforms: Jenkins, Travis
- Modules for common reconnaissance and exploitation tasks

GITPWND SUMMARY

- Tool for penetration tests that enables command and control via common development infrastructure
  - Uses a private git repo on GitHub
- Flexible
  - Commands can be arbitrary Python
  - Mimics arbitrary public git repo that fits into the target environment
  - Bootstrapping is easily tailored to the target environment
- Targeted primarily at agile development environments
- Written to highlight the potential risks of mismanaging trust relationships in an enterprise

AGENDA

- Overview
- On Trust
- Development-focused environments
- GitPwnd
- **Mitigations**
- How do you scale security?
- Questions

ROOT CAUSE ANALYSIS

- Development security has not grown at the pace of development
- Security and development are historically handled as separate silos
- Lack of documentation and definition of trust relationships
- Incomplete technical controls

MITIGATION - PROCESSES AND PROCEDURES

- Commits should be analyzed as part of standard deployment process
  - Changes that add or modify external source or code execution
  - Changes to set up steps and dependencies
- Define development processes that match the culture and requirements of the organization
- Review development and staging / production environments with engineering teams for design and definition of trust zones.
- Work with development teams to create secure procedures and automated processes for day to day engineering operations

MITIGATION – TECHNICAL CONTROLS

- SDLC
  - Sandbox CI processes running integration tests
  - Consider requiring PGP signing for commits
    - Have CI/CD processes require signed commits from valid users
  - Examine your git hooks
- Host-based
  - Monitoring and alerting
- Network Segmentation
  - Routing and switching
  - System / code tenancy
- Roles and Permissions

AGENDA

- Overview
- On Trust
- Development-focused environments
- GitPwnd
- Mitigations
- **How do you scale security?**
- Questions

AGILE / DEVOPS CAN BE HARD

- >50:1 developers to security
- Can’t possibly review all code
- Security can’t be blockers
  - Not “gateway” like in waterfall
- Code goes from Dev -> Prod in minutes/days, not months

HOW DO YOU SCALE SECURITY?

Two core ways:

1. Create libraries and systems that make it easy to do things securely
2. Add automated (static and dynamic) checks at key points in the SDLC

Why are we proposing these approaches?

- Seen these techniques first-hand with a number of Bay Area companies
- Pulling from many public blog posts and conferences talks

WHY AUTOMATE?

- Security people are expensive and hard to hire.
- Automation can help your security team scale to get more done with the same number of people.
- Free up security engineer time to tackle more interesting, higher ROI tasks.

THE BIG PICTURE

1. Security Engineering
   Libraries

2. Developer machine
   Static Analysis

3. Code Hosting
   Static Analysis

4. Test/QA Environment
   Dynamic Analysis
   Fuzzing

5. Production
   Monitoring

AWS

HIGH ROI SECURITY ENGINEERING TASKS

- Build libraries / tools that are secure by default for dev teams
- E.g., Today, many web frameworks handle output encoding by default
  - Before that, devs had to manually add it everywhere, h() in Rails
- Potential areas to consider:
  - Managing secrets
  - Anything related to crypto
  - Authentication / authorization
  - SQL, file system access, shell exec()
  - E.g., `nonCryptographicallySecureMd5()`

SECURITY ENGINEERING -- SOME EXAMPLES

- Test case that automatically checks for missing access controls
  - Enumerate all routes, introspect on code in each route, warn if no authorization check
- Battle-tested view library
- Data model wrapper library that all DB accesses must go through

Core takeaway:

- **“<X> is hard to do securely, have to be aware of threats 1, 2, and …”**
  - Build a secure by default implementation

THE BIG PICTURE

1. Security Engineering
   Libraries

2. Developer machine
   Static Analysis

3. Code Hosting
   Static Analysis

4. Test/QA Environment
   Dynamic Analysis
   Fuzzing

5. Production
   Monitoring

AWS

ON DEVELOPERS' MACHINES

- Needs to be fast, low to no false positives
- Can be implemented with git hooks or as a part of the test suite
- Some examples:
  - Reject all DB accesses that don't use the approved API
  - Reject all SQL that appears to concatenate strings
  - Reject use of known dangerous functions (`exec`)
    - One company uses ESLint to check for uses of `eval()` or new `Function()`
  - Reject commits that add secrets (AWS or other API keys, credentials, …)
- Note: these are essentially `greps`, but yield nice security wins

THE BIG PICTURE

1. Security Engineering
   Libraries

2. Developer machine
   Static Analysis

3. Code Hosting
   Static Analysis

4. Test/QA Environment
   Dynamic Analysis
   Fuzzing

5. Production
   Monitoring

AWS

CODE HOSTING -- CI/CD

- Developer has pushed code into GitHub / Gitlab / Bitbucket / Jenkins…
- Implemented in - on-receive hook, new branch creation

- Some classes of approaches:
  - Block banned or dangerous functions
  - Detect security-relevant code additions
  - Alert on sensitive file changes
  - Check for out-of-date dependencies

CI/CD -- BLOCK BANNED/DANGEROUS FXNS

- Same checks as mentioned previously for developer machines
  - Also need to check here in case things slip through
- Can also improve code quality over time by blocking new additions of anti-patterns while allowing existing ones
  - E.g., Trying to migrate to a strict CSP, but inline JavaScript is everywhere
  - Block new inline JavaScript, allowing you to gradually refactor existing code

CI/CD -- SECURITY RELEVANT NEW CODE

- Alert the security team when interesting primitives are used that should start a conversation with the security team (rather than just being blocked)
  - Crypto (hashing, encryption, …), file system operations
- Gives you context about how apps are evolving, where code hotspots are
- Correct common errors early
  - Someone using an approved hashing function when the data should be encrypted

CI/CD -- ALERT ON SENSITIVE FILE CHANGES

- Ping the security team when sensitive and rarely modified parts of the codebase have been modified
  - E.g., Authorization / authentication, session management, encryption wrappers, etc.
- Can be done by simply hashing the files and alerting when their hash changes

Going further:

- Only allow the security team or specific senior devs to modify critical files, alert if anyone else does.
- Build models of which devs edit which files - alert on abnormalities
- Square - leverage code churn and code quality metrics for where to focus

CI/CD -- CHECK DEPENDENCIES

- Keep track of existing, out-of-date dependencies
- Set policy for update process based on time/severity
- Analyze newly added dependencies
  - What's the trustworthiness of this newly added dependency?
  - Number of downloads in past <unit time>, lifetime downloads, dangerous function use, etc.

CI/CD -- TOOLS

- Automatically run arbitrary checks on newly pushed code.
  - https://github.com/Netflix/Scumblr
  - https://github.com/salesforce/Providence
  - http://gauntlt.org/

SHOULD I BUY A SAST TOOL?

- Generally ran daily or weekly (takes hours - days)
- It depends on your company- technologies, development processes, culture
- Some considerations:
  - Do you have large, legacy code bases that haven't be thoroughly vetted?
  - Is much of your code in Java or other statically typed languages?
    - Dynamic languages are harder for static analysis (Python / Ruby / JavaScript)
  - How mature is your security program?
  - Does your org heavily use custom frameworks with non-standard control flow?
  - Are you willing to invest months of security engineer time in the rollout?

SHOULD I BUY A SAST TOOL?

- Calculate ROI
  - How much security engineer time is required to comb through results vs. how many bugs of what severity do we find?
  - Initial upfront time investment? Low recurring time cost?
  - Might find many good bugs on initial integration but fewer over the years
- A number of companies have started rolling their own or writing custom rules

THE BIG PICTURE

1. Security Engineering
   Libraries

2. Developer machine
   Static Analysis

3. Code Hosting
   Static Analysis

4. Test/QA Environment
   Dynamic Analysis
   Fuzzing

5. Production
   Monitoring

AWS

DYNAMIC ANALYSIS

- Commercial tools have trouble on complex apps, JS-heavy apps, etc.
  - AppSecUSA 2017 - “Differences B/w Web App Scanning Tools when Scanning for XSS and SQLi”

Three approaches that add value in practice:

1. Ensure a security baseline
2. Automated tests for regressions
3. Automated fuzzing of new builds

DYNAMIC ANALYSIS - SECURITY BASELINE

- Ensure security policies are enforced
  - Every new service only uses TLS with strong ciphers
  - Check headers (strong CSP), secure/HTTPOnly cookie flags, etc.

- Mozilla – automated security baseline scan for all newly deployed apps
  - https://jvehent.github.io/continuous-security-talk

DYNAMIC ANALYSIS - REGRESSIONS

- For each vuln reported by a pen test or bug bounty submission, add a unit test that tests the exact issue with an example payload
  - E.g., there was XSS in the `/search_store` endpoint in the `search_term` URL parameter
- Ensure fixes / protections don't accidentally get rolled back

- BH USA 2017 – Practical Tips for Defending Web Apps in the Age of DevOps

CASE STUDY: ADD FUZZING TO SDLC

- **Context**: had several native services parsing input from the Internet - web socket traffic, JSON parser, etc.
  - Pushed new code to production every 2 days
  - High uptime required - processed billions of messages a day
- Challenges
  - Code was tightly coupled -- non trivial to write good test harnesses
  - Release pace limited fuzzing duration of a given build
  - Scale fuzzing process across a cluster of machines, with many different test harnesses

CASE STUDY: ADD FUZZING TO SDLC

- What we did
  - Wrote separate harnesses for different functional areas of the program
  - Set up a fuzzing cluster that continuously fuzzed new builds based on priority metrics
  - Made a table for the dev team - “If you update the code manipulating this structure, run harness X”
- Subsequent improvements
  - Crash detected -> auto create JIRA ticket with test input, stack trace, assign to owner
  - De-duplicate identical crashes
- Lessons learned
  - There needs to be constant dialogue between the security and development team
    - Structure of code can make fuzzing much easier or much harder
    - Work with the dev team so code exposes a clean API - make it easy to write test harnesses
  - Dev team appreciated the bugs for improving robustness, not just security
- Open problems
  - How long to fuzz?
  - Only fuzz newly modified code?
- Credit: Ryan Sullivan and Gabe Pike

THE BIG PICTURE

1. Security Engineering
   Libraries

2. Developer machine
   Static Analysis

3. Code Hosting
   Static Analysis

4. Test/QA Environment
   Dynamic Analysis
   Fuzzing

5. Production
   Monitoring

AWS

QUESTIONS?

MIS|TI™ PRESENTS

InfoSec World

Conference & Expo 2018

THANK YOU

PLEASE FILL OUT YOUR EVALUATIONS!

Clint Gibler
Research Director
@clintgibler

Noah Beddome
Director of Infrastructure Security
@domodogood