Automatic Application Security @twitter

Me

- ~~Twitter~~ Github Security Engineer
- Twitter: @ndm
- Github: @oreoshake

Automatic Application Security?

The story of a line of code

- Before the code is written
- While the code is being written
- After the code has been written
- After the code has shipped

Before the code is written

- Framework / Architecture Security
- Secure by default
- Education
- Culture

Framework / Architecture security

- Provide the necessary controls
- Don’t provide anything else
- Require opting out of security

Secure by default

- An extension of “opting out” of security
- The framework is configured in the most restrictive way possible

Education (nho)

- Separate code and data
- If you do, we'll leave you alone

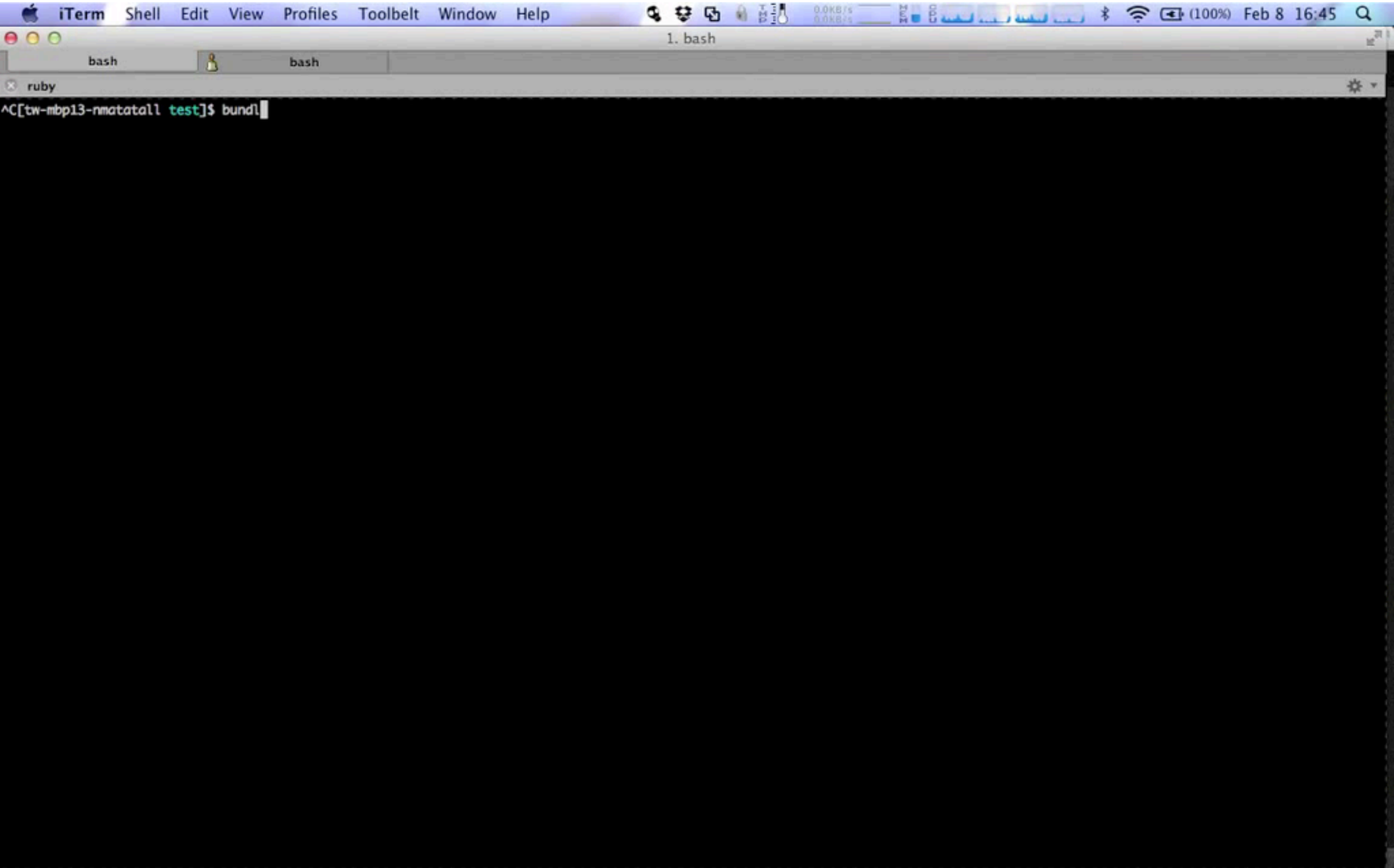
Culture

- Don’t be a jerk
- Understand your code will be scrutinized

While the code is being written

- Provide what is needed

guard-brakeman



Tests are your friend

- Encourage “negative” test cases

secure_headers

- It’s just a freakin (set of) header(s)
- Ported to Node*, Go, .Net, Java, PHP, Python, dancer, drupal, etc.
- Think of the benefits of the headers as config values, rather than the textual value

Provide what is needed: CSP

- Nonce / Hash support

It’s a bug, not a feature

- XSS?
- Mixed content?
- Site defacement?
- All solved* by csp

Nonce

- Generate a random value per request
- Populate a “nonce” attribute for any script tag you want to be whitelisted

Railsgoat + nonce

- Pull request to add nonce support[1]
- 46 files changed, 72 additions, -46 deletions
- global find and replace took care of 90% of the job

[1] https://github.com/OWASP/railsgoat/pull/174

Hashes

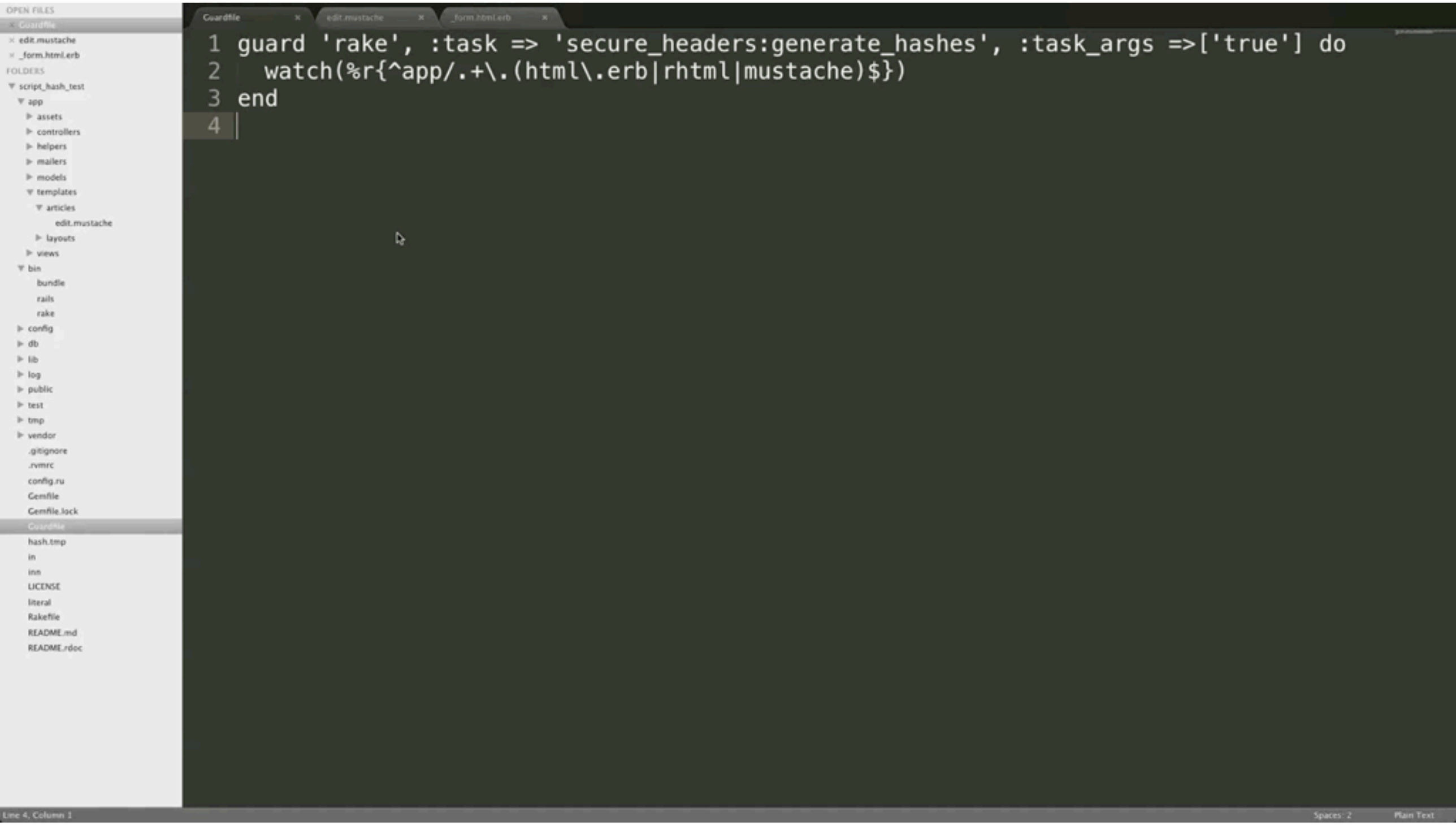
- Find and compute hash values for <script> tags
- Associate each hash with the file it lives in
- Every time a file is rendered, include the corresponding hashes in the header
- Requires less changes than the nonce approach

Railsgoat + hash

- Pull request[1] to add hashes when possible, nonces when not
- 12 files change, 33 additions, 13 deletions (> 50% reduction in changes over nonce)
- Hardest part was dealing with dynamic js (which requires the use of nonce)

[1] https://github.com/oreoshake/railsgoat/pull/1

Automatic hashing PoC



IRL

- Coming to a twitter near you…
- Only 5 inline scripts

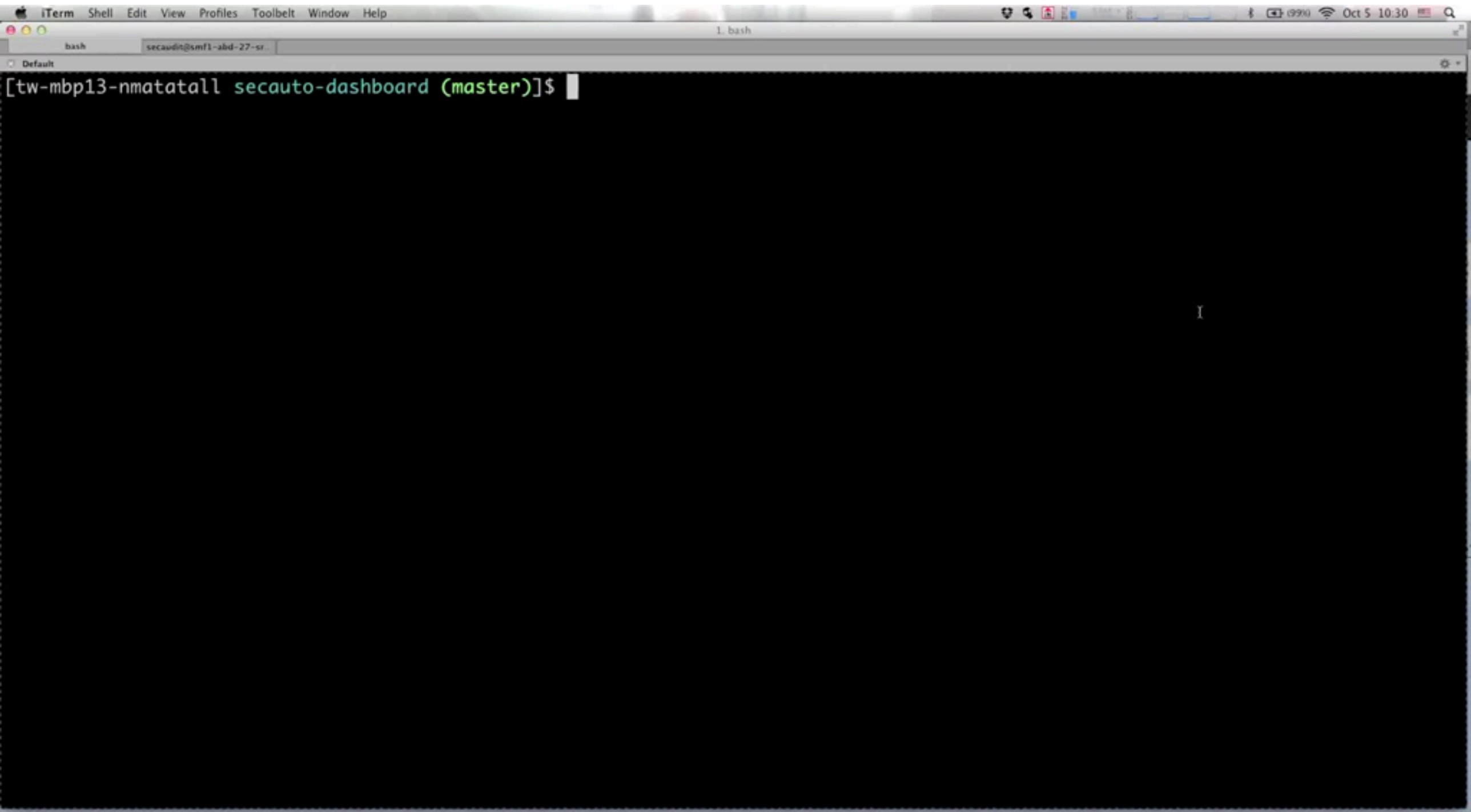
After the code has been written

- Go all out
- Stay out of the way

Scan on arrival

- Every time new code is pushed, run tools and diff the results from master

The SADB workflow



Laundry list of tools

- Static analysis
  - Brakeman
  - scan js
- Dependency Management
  - bundler-audit
  - retire js
  - owasp dependency check
- Other
  - Charlie Miller’s fuzzer thing

Review upon review

- Code review is a great integration point

Again, it’s just a regex

- When your threat model is tiny, the tools required to support it are pretty simple

Notify the relavent authorities

- OWNERS

Did we catch it all?

- Probably not

After the code has been shipped

- It’s out of our hands, right?

Decider

- All features, and any new code is often behind a Feature Flag

Bug Bounty

- Penetration testing on the cheap

Stats

- They aren’t just for proving a feature was a success

You can do it

- These tools and integrations came out of a direct need.
- “The best indicator of the next bug is the last bug”
- Look at your previous bugs, and focus there

Not everything is successful

- Vendor black box scanner
- pre-SADB integration
- Phantom gang
- Business logic flaws amirite

Tie it all together

- With a dashboard of course

Time to Chill

- Your threat model is small
- Code is always under scrutiny
- People know what the “right thing” is
- You have sensors to detect issues at all phases of the pipeline
- You have social and technical controls in place