Be Agile, Not Vulnerable: Security Engineering in an agile world

Jacob Kaplan-Moss

jacob@jacobian.org
Every day, Heroku deploys 100 times
about:me
about:me

Co-BDFL

django
Security is a process, not a product.
Heroku’s security team:

@jacobian

@tmaher
Security is everyone’s responsibility.
The OWASP Top 10

https://www.owasp.org/index.php/Top_10_2013

1. Injection
2. Broken authentication and session management
3. XSS
4. Insecure direct object references
5. Security misconfiguration
6. Sensitive data exposure
7. Missing function-level access control
8. CSRF
9. Components with known vulnerabilities
10. Unvalidated redirects
A thought experiment...
You’re working on an API library. There are three formats in common use in these types of API.

Do you support all three of them?

Does this decision have security ramifications?
You need to store some configuration data. A is a common dependency, very readable, and used by most Python developers. B is less common, harder to read and write, and isn’t used as often.

Which do you choose?

Does this decision have security ramifications?
You’re implementing a serialization format. At first it only supports primitive types, but users quickly request that you extend the language to allow serialization of arbitrary objects.

Do you say yes?

Does this decision have security ramifications?
Did you just create a vulnerability?
2013 Ruby/Rails YAML Vulnerabilities

http://www.kalzumeus.com/2013/01/31/what-the-rails-security-issue-means-for-your-startup/
“LOL Rails suxxors amirite!?”
STFU.
Real-world security issues are multifaceted.
“Secure by default” matters.
You can’t prove that software is secure. You can only fail to prove that it’s insecure.
“There are known knowns; there are things we know that we know.

There are known unknowns; that is to say, there are things that we now know we don't know.

But there are also unknown unknowns – there are things we do not know we don't know.”

— Donald Rumsfeld
If an issue of this magnitude was discovered in your stack, would you be prepared to respond?
Security releases issued

Today the Django team is issuing multiple releases -- Django 1.4.6, Django 1.5.2, and Django 1.6 beta 2 -- as part of our security process. These releases are now available on PyPI and our download page.

These releases address two cross-site scripting (XSS) vulnerabilities: one in a widget used by Django's admin interface, and one in a utility function used to validate redirects often used after login or logout.
How severe are these issues?
“not too bad”
“no big deal”
“kinda serious”
“The sky is falling!”

“No big deal.”
A good security policy

Lays out **standard terminology** used when talking about security issues.

Explains the **expectations and commitments** around vulnerability handling.

Creates a **transparent**, repeatable assessment mechanism.
Terminology

Advisory
Low
Medium
High
Critical
Advisory

Issues that the security team wishes to communicate but that carry no specific required action.

May sometimes contain recommended actions, but no specific response is required to an advisory, and no timeline is defined.
Low

Issues that need to be resolved, but have either a low risk of exploit or low consequences for an exploit.

Should not interrupt day-to-day operations, but should be scheduled for the next appropriate slack time.
Medium

Issues that carry a noticeable risk, but are still theoretical, not ongoing, or have a low impact.

Expect Medium vulnerabilities to cause limited interrupts, but otherwise have minimal impact on normal operations.
High

Carry a substantial risk to your customers, finances, reputation or otherwise.

Expect High-level vulnerabilities to interrupt several developers, perhaps from multiple teams.
Critical vulnerabilities threaten the integrity of your company, contain substantial financial risk, or are otherwise “sky is falling”-level issues. These are literally existential threats to your company.

Critical vulnerabilities are “all hands on deck” moments.
Assessment
Start here:

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology
Risk = Likelihood × Impact
Likelihood:

How likely is it that this issue will be discovered and exploited?
Impact:
If exploited, what are the ramifications on the company?
4. What resources are required for an attacker to exploit the vulnerability?

(0) Full superuser-level access (sudo access)
(2) Staff-level access (access to internal tools)
(4) Special access required (needs a particular type of account)
(7) Limited access required (anyone with an account)
(9) No access or resources required
E.g.: PostgreSQL argument injection vulnerability

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1899

http://bit.ly/13ds9X0
Likelihood: threat agent
1. How technically skilled is the attacker?

(3) Network and programming skills - The attack requires a carefully required connection string, so we assume an attacker would need some network and programming skills.
2. How motivated is this attacker?

(9) High reward - in a worst-case, the attack could be used to remotely access arbitrary databases, which is quite valuable.
3. What resources and/or opportunities are required for this attacker to find and exploit the vulnerability?

(9) No access or resources required
4. How large is group of potential threat agents?

(9) Anonymous Internet users
Likelihood: ease of discovery and exploit
5. How easy would it be for an attacker to discover this vulnerability?

(3) Difficult - the vulnerability existed in PostgreSQL for many years and went undiscovered, and was eventually only discovered by a core contributor.
6. How easy would it be to actually exploit the vulnerability, assuming knowledge that it exists?

(5) Easy - once you know of the vulnerability, it's easy to craft a proof of concept.
7. How well-known is this vulnerability among the community of potential attackers?

(1) Unknown
8. How likely is it that we'd detect an exploit (or attempt)?

(6) Logged and reviewed eventually
Total likelihood:

\[
\frac{(3+9+9+9+3+5+1+6)}{8}
\]

5.625
Impact: confidentiality and integrity
1. How much private data would be disclosed, and how sensitive would that data be?

(9) Complete data disclosure
2. How much data could be corrupted, and how damaged would that data be?

(7) Extensive corruption to much customer data with difficult or incomplete recovery possible.
Impact: availability and accountability
3. How much downtime would an exploit cause, and how vital would that downtime be?

(9) Complete outage
4. How easy would it be to trace an exploit back to the attacker?

(7) Partially traceable
Impact: business factors
5. How much financial damage would we suffer as a result of an exploit?

(7) Significant and noticeable effect on annual profit
6. Would an exploit result in reputation damage that would harm our business or our brand?

(7) - worse than "loss of public goodwill", not quite "long-term or permanent brand damage"
7. How much legal/regulatory exposure would an exploit introduce?

(5) Clear violation of regulation/law
8. How much personally identifiable information (e.g. customer emails/ passwords/credit cards) would be exposed?

(9) All of it.
Total impact score:

\[
\frac{(9+7+9+7+7+7+5+9)}{8}
\]

7.5
<table>
<thead>
<tr>
<th>Impact</th>
<th>Likelyhood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to &lt; 3</td>
<td>Advisory</td>
</tr>
<tr>
<td>3 to &lt; 6</td>
<td>Low</td>
</tr>
<tr>
<td>6+</td>
<td>Medium</td>
</tr>
<tr>
<td>Impact: 7.5</td>
<td>0 to &lt;3</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>0 to &lt;3</td>
<td>Advisory</td>
</tr>
<tr>
<td>3 to &lt;6</td>
<td>Low</td>
</tr>
<tr>
<td>6+</td>
<td>Medium</td>
</tr>
</tbody>
</table>
5. How easy would it be for an attacker to discover this vulnerability?

(3) Difficult - the vulnerability existed in PostgreSQL for many years and went undiscovered, and was eventually only discovered by a core contributor.
Be safe out there!

Jacob Kaplan-Moss

jacob@jacobian.org