Engineering Secure Software

DEPLOYMENT & DISTRIBUTION

SE Doesn’t End at Release

- Deployment counts too
  - Despite our best efforts to produce secure software
  - Vulnerabilities can exist only when deployed in a production environment
  - Users expect “secure by default”
- Your organization needs to be ready
  - Incident response plan
  - Version control practices
- Your installation & config scripts count
  - Recommended firewall configuration
  - Security manager configuration

Recent PostgreSQL Incident

- PostgreSQL reported a show-stopping vulnerability found on April 4th, 2013
  - http://www.postgresql.org/support/security/faq/2013-04-04/

- “Argument injection vulnerability in PostgreSQL [9.2.x ..] allows remote attackers to cause a denial of service (file corruption), and allows remote authenticated users to modify configuration settings and execute arbitrary code, via a connection request using a database name that begins with a "-" (hyphen).”

- “The vulnerability allows users to use a command-line switch for a PostgreSQL connection intended for single-user recovery mode while PostgreSQL is running in normal, multiuser mode. This can be used to harm the server.”

How PostgreSQL Responded

- Embargo on the bug: March 13th – April 4th
  - Removed the public version control repositories during the embargo
  - Announced on the mailing lists to expect an immediate upgrade soon, without much detail
- Contacted vendors especially affected (e.g. Heroku)
  - Core PosgreSQL developers assisted the vendors directly
  - Tested patches on vendor's environments
  - Heroku already had a history of working directly with developers on experimental features

Incident Response Plan

- Incident definition
  - How do you know that this behavior is bad?
  - Use high-level risks & indicators from your initial risk assessment
- Establish who is involved
  - Monitoring duties
  - Contacts for an issue
  - Security response team
- Chain of escalation
  - Know who to contact to fix the problem
  - Who sees the bugs
    (e.g. Cisco CEO gets daily escalation reports)

Incident Response Plan (2)

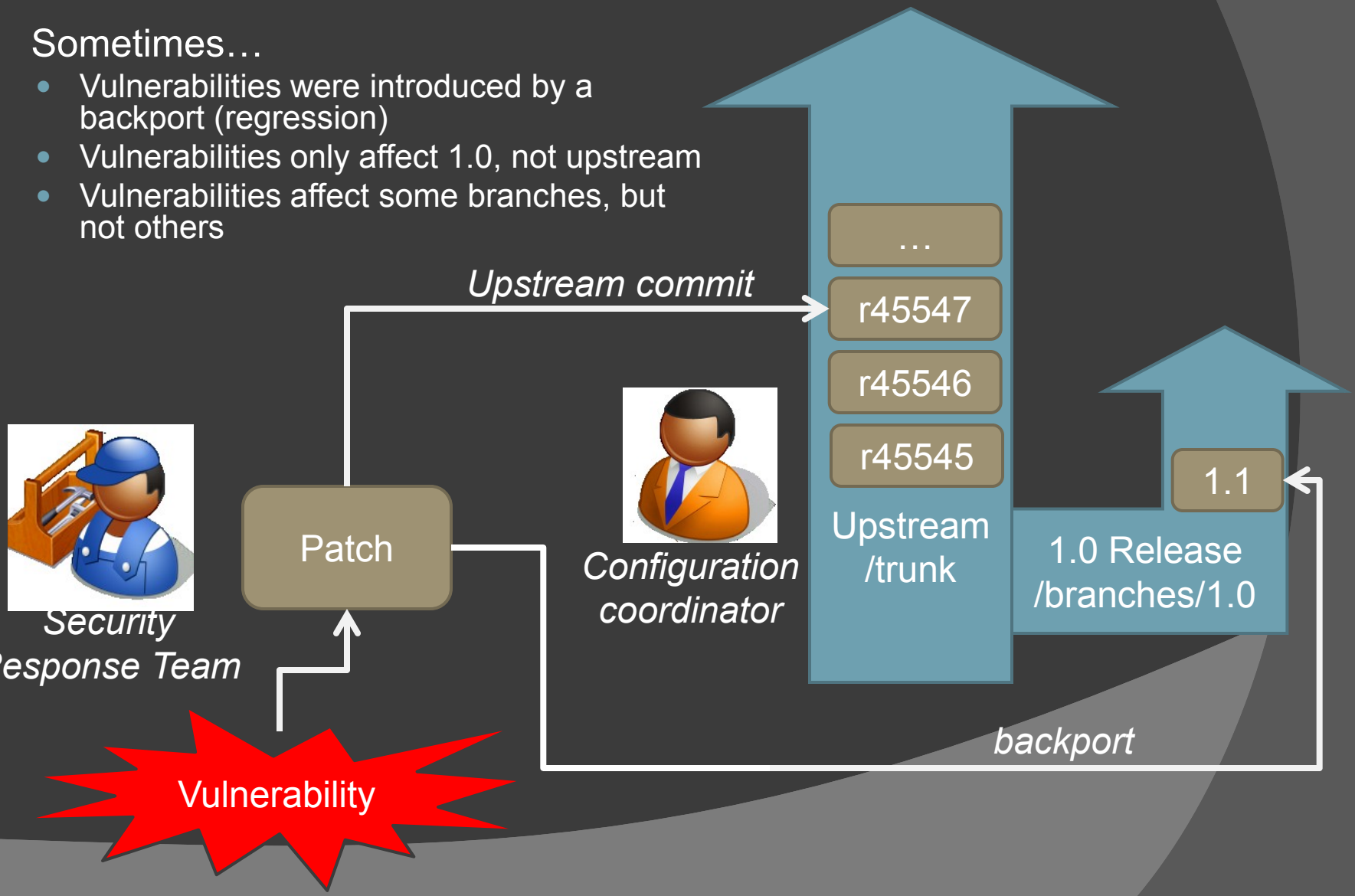
- Establish Procedures
  - Writing the patch
  - Testing the patch
  - Security expert review of a patch
  - Reacting to specific exploits
- Establish working relationships with key vendors
- Establish criteria for notifying the world
  - Too late? Active exploits make you look behind
  - Too early?
    - Unnecessary panic
    - Invites exploits

Version Control Practices

- Releases are treated as *branches*
  - Most current version: trunk branch
    - aka *upstream*
    - Continuously-updated to the latest version
  - Maintenance: release branch
    - Diverges from the main truck
    - New change to an old release? *Backport*
  - Upstreams and backports can differ if the code has since changed a lot
- Configuration management coordinator
  - Keeps track of all the branches and releases
  - New devs often work on backports
  - Keeps track of “testing gotchas” from one release to another (e.g. environment change, or non-change)

Upstream & Backport

- Sometimes…
  - Vulnerabilities were introduced by a backport (regression)
  - Vulnerabilities only affect 1.0, not upstream
  - Vulnerabilities affect some branches, but not others



Releasing Patched Versions

- You will need to release patched versions of your product
- "Patch it yourself" approach (e.g. Adobe Flash, Acrobat)
  - Software contacts the vendor periodically and downloads software
  - Benefit: simple, easy, you control how it works
  - Drawback:
    - Non-root installations mean malware can spoof the update site, or disable it
    - Reverting a bad release is not usually supported
- Package manager approach
  (e.g. apt-get, yum, Mac App Store)
  - Benefits
    - OS support means packages are handled all in one place
    - Harder to compromise: uses hash digests to verify
  - Drawback: can annoy users

"package X.1.2 isn't in the system?!?!?"

Firewalls

- Designed to be the gatekeeper for networks
  - Allow|Block IP addresses & Ports
  - Forward traffic to different ports
  - Network Address Translation (NAT)
- Installation scripts often need to configure the firewalls
- IPTables, the Linux firewall
  - Create “tables of chains of rules”
    - Table: group of chains for a given action (e.g. NAT, Filter, Routing, custom, etc.)
    - Chain: an ordered group of rules
    - Rule: specific definition of what’s in and what’s out
  - e.g. view your Filter table:
    `iptables -t filter --list`

e.g. IPTables Rules

- Command line
  - -A → append to chain, -j → jump target (ACCEPT, DROP, etc.)
  - -s → source of the packet, -d → destination of the packet
  - -dport → destination port on local machine, --sport → source port
  - -i → input network interface (e.g. network card driver), -o → output interface
  - --state → packet states to match (e.g. NEW, ESTABLISHED), -p → protocol

- Drop all packets coming from a specific IP address

```
iptables -A INPUT –s 129.21.208.62 -j DROP
```

- Allow SSH packets in and out

```
iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --
state NEW,ESTABLISHED -j ACCEPT

 iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state
--state ESTABLISHED -j ACCEPT
```

e.g. More IPTables rules

- Forward port on IP address 192.168.102.37 from 422 to 22

```
iptables -t nat -A PREROUTING -p tcp -d 192.168.102.37
--dport 422 -j DNAT --to 192.168.102.37:22
```

- DoS mitigation: When we see a burst of 100 connections/min, limit to 25 connections/min on port 80

```
iptables -A INPUT -p tcp --dport 80 -m limit --limit
25/minute --limit-burst 100 -j ACCEPT
```

- Create a new table & chain for logging, turn it on

```
iptables -N LOGGING
iptables -A INPUT -j LOGGING
iptables -A LOGGING -m limit --limit 2/min -j LOG --
log-prefix "IPTables Packet Dropped: " --log-level 7
```

Security Managers

- Often a programming language feature
  - Required for untrusted API situations
  - Prevents sensitive API calls
    - e.g. System.exit(1) in Java
    - e.g. System properties (read and write)
  - Highly customizable
  - Turned off by default
- Many languages have them, or community provides them
  - Java: Java Security Manager
  - Python: e.g. RestrictedPython
  - Perl: Safe.pm
  - Ruby: Safe
  - C/C++: None – use OS mechanisms

Security Managers in Practice

- In a server situation
  - Limits access to underlying OS
    e.g. file access, logging
  - Limits OS-sensitive functions
    e.g. opening a socket
- In a desktop situation
  - Used to mitigate extensibility concerns
  - Mitigates the “malicious plug-in” problem
  - Not usually for license key situations (user can just remove the policy)

e.g. catalina.policy

- From Apache Tomcat, Java servlet container
  - A web application is untrusted code running in the same VM
  - DoS & access to underlying OS are concerns too
- Server startup JAR is given full permissions

```
// These permissions apply to the server startup code
grant codeBase "file:${catalina.home}/bin/bootstrap.jar" {
  permission java.security.AllPermission;
};
```

- Grant read permissions to some system-wide properties

```
permission java.util.PropertyPermission "java.home", "read";
permission java.util.PropertyPermission "java.naming.*", "read";
permission java.util.PropertyPermission "javax.sql.*", "read";
```

e.g. catalina.policy (2)

- Grant application-specific logging file permissions

```
 permission java.util.logging.LoggingPermission "control";
 permission java.io.FilePermission
"${java.home}${file.separator}conf${file.separator}logging.proper
ties", "read";
```

- Grant read API permissions for web applications for a given package

```
// All JSPs need to be able to read this package permission
java.lang.RuntimePermission "accessClassInPackage.org.apache.tomcat";
```