RSAConference2015

San Francisco | April 20-24 | Moscone Center

CHANGE

Challenge today's security thinking

SESSION ID: ASD-W01

Countering Development Environment Attacks

David A. Wheeler

Research Staff Member
Institute for Defense Analyses (IDA)
@drdavidawheeler

Dan Reddy

Adjunct Faculty: Engineering & Technology
Quinsigamond Community College
@danlj28

Today’s Development Environment

- Developers are pressed to produce complex functionality with
  - Inherited code
  - Short product development cycles
  - “Software is an art not a science” mindset
- Hard to grasp that new security practices are worth the time investment
  - Remember when quality management was an “unnecessary distraction”
  - Security is only one dimension of code improvement
    - Automation, reuse, geo development, collaboration, change management, virtualized environment, …
- Who can stop the train?

The Problem

- These are important but *not* the only problems:
  - Unintentional vulnerabilities inserted by developers (See *SAFECode*, *Fundamental Practices for Secure Software Development, Secure Programming HOWTO*)
  - Secure distribution (e.g., code signing, SSL/TLS)
- Attackers can also attack development environments
  - Exfiltrated/intercepted secrets: proprietary source code, vulnerability reports & analyses, crypto keys/passwords
  - Subverted supply chains for sourcing from upstream repositories & 3rd parties
  - Insertion of malicious code into source
    - Outsider and (different levels of) insider; may be plausibly deniable or maliciously-misleading
  - Subverted binaries
    - Not compiler/toolchain + Compiler/toolchain (“trusting trust” attack)
- Countermeasures exist!

Exfiltrated/intercepted Secrets: Source Code, Vulnerability Reports & Analyses

- Example: RSA SecurID / Lockheed (2011)
  - “Recently, our security systems identified an extremely sophisticated cyber attack in progress being mounted against RSA… resulted in certain information being extracted… related to RSA's SecurID two-factor authentication products.”
  - “Sources close to Lockheed point to compromised RSA SecurID tokens… as playing a pivotal role…” [DailyTech]
  - “… we are seeing increases in attacks on one organization to be leveraged in an attack on another organization…” - Art Coviello, Executive Chairman, RSA [Coviello2011]

Subverted Supply Chains / Upstream Repositories

- Subverted external repositories: SourceForge/Apache (2001); Debian (2003); Haskell (2015)
- Linux kernel (2003) attempt to add malicious code

```
+  if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+                  retval = -EINVAL;
   retval = -ECHILD;
```

  - Attack countered due to configuration management tools, developer review, & coding conventions [Miller2003] [Andrews2003]

Insertion of Malicious Code into Source (outsider and insider)

- Timothy Lloyd at Omega Engineering
  - Timothy Lloyd planted a 6-line logic bomb into employer’s systems (Omega Engineering)
  - Went off on July 31, 1996
  - Erased all of the company’s contracts and proprietary software used by their manufacturing tools
  - $12 million in damages, 80 people permanently lost their jobs, loss of competitive edge
  - Plant manager Jim Ferguson: “We will never recover”. [Ulsch2000] [Gardian]
- Roger Duronio at UBS PaineWebber
  - System administrator for 2 years
  - Installed a logic bomb to detonate on March 4, 2002 (only a few lines of C and shell) and resigned
  - Caused over 1,000 / 1,500 networked computers to begin deleting files
  - $3 million to assess and repair the damage, plus undetermined lost business [Gaudin2006a]

Insertion of Malicious Code into Source (outsider and insider) cont'd

- Borland InterBase/Firebird Back Door (inserted 1994, discovered 2001)
  - User: politically, password: correct, Hidden for 7 years in proprietary product
  - Found after release as OSS in 5 months
  - Unclear if malicious, but has its form

Countermeasures to Development Environment Attacks

- Fundamentals / best practices (may be scaled to large & small companies)
- Protected final build environment
- More advanced / less common
  - Detect repo/build attacks: customized IDS, e.g., OWASP AppSensor
  - Counter subverted build environment: Reproduceable builds
  - Malicious/backdoor code detection
  - Counter maliciously-misleading code
  - Countering trusting trust: Diverse Double-Compiling

Fundamentals - Development Defense Best Practices

- Infrastructure
  - Regular credentialed scanning for vulnerabilities and compliance to hardened OS (e.g., DISA STIG audit guidelines)
    - Critical patches applied in timely way. Within week to 30 days by properly trained techs? “Automatic”? Can they be reversed?
    - Physical and virtual !
    - Priority based remediation that emphasizes security posture
  - Change Management process for infrastructure changes
  - Comparable test and dev environments to what is in production
  - Final “Build farms” are segregated from dev environments

Fundamentals - Development Defense Best Practices

- Access Control
  - Separation of privileges between server/OS admins and code developers
  - True role separation based on “need to know” / “need to change”
    - Is everyone skilled and trusted equally?
    - Who actually has to collaborate on code? How often verified?
    - Build culture of teamwork with independent reviews. New fact of life
  - Separate development teams from build teams doing final builds
  - Repository admins are separate from OS owners
  - Promote two person controls for critical actions (with auditing)
    - If one person becomes malicious, others can detect
    - E.g., repo owners need their own oversight

Fundamentals - Development Defense Best Practices

- Sourcing
  - Documented process for all sources
  - Integrity checks must be required (counter MITM)
  - Meets legal licensing issues (third party including open source software)
  - Published profiles on source organizations (BSD community, Apache)
  - Separate sandbox environment for preliminary scanning and review
    - Don’t bring right into dev environment
    - Copying and pasting of code snippets gets independent review too

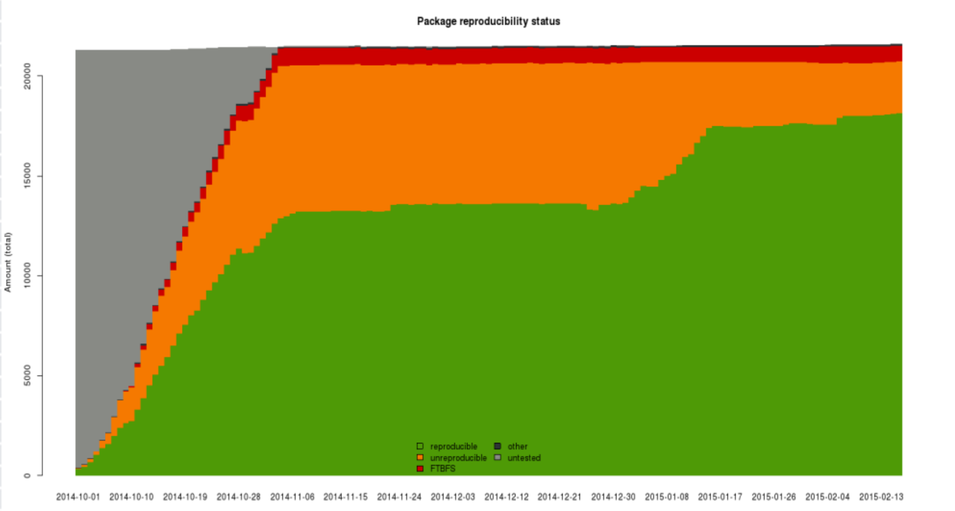
Fundamentals - Development Defense Best Practices

- Protect final build environment
  - Dev builds != Final builds
  - Final builds solely created from governed sources
    - Developer can’t binary-patch final build
  - Limit who’s allowed to change final build environment
  - Ensure that build environment cannot be changed by build

Countering Subverted Binaries

(except compiler/toolchain)

- What if protection of binary build process, or its results, fail?
- Reproducible builds
  - Regenerate exact binaries from source (modify build or record info)
  - Can detect subverted binaries if source and compiler/toolchain protected
  - Challenges: embedded timestamps, “random” (unforced) order of results, embedded build data, results generated from uninitialized data
  - Tor & Debian working on this & have had significant progress



Debian reproducible build status, per https://wiki.debian.org/ReproducibleBuilds

Other Advanced Countermeasures: Scan Sources for Indicators of Back Doors etc.

- Build “back door” or other attack attribute profiles that source code scanners can leverage.
  - Scan all source code for back door attributes that trip sensors
  - What might they look like in code? 80/20 rule. Make it harder
  - E.g., date/time checks, starting network communication, rm –rf, drop all tables
- This is not easy or broadly implemented today
  - Be careful of vendor claims
  - Apply to all external party software (open source software, proprietary software, trusted partners’ code)
  - Must automate eventually in order to scale
- Start by examining the historical code one time
  - Calculate diffs on stable code

Maliciously-misleading Code Inserted into Source (e.g., by insider)

- Source code can be written to look innocent yet it do something subtly evil – counters manual review of two-person control
- Many examples in “Underhanded C Contest” & “Obfuscated V contest”
  - Learn from past contest results to develop countermeasures

Paul A. Parkanzky: Buffer Overflow

```
int main() {
    unsigned int Tally[4] = {0};
    unsigned char Other, Nader, Bush, Kerry;
    char LogMesg[11] = {0};
    char *day;
    day = getDay();  // Returns first, second, etc.
    while ((Input=getchar())!=EOF) {
        unsigned char Vote=Input;
        sprintf (LogMesg,"LOG VOTE: November %s %c\n",day,Vote);
        paperTrail(LogMesg);
```

Michael Moore: Comment Games

```
/*
        The design goal in the main loop is to minimize
        the code to simplify the process of analyzing the code …
        The production code fragment to be replaced is:
                /* Input is space, use -1, otherwise locate() */
                /* locate() guaranteed not to return -1 */
                (isspace(x) ?
        testing PHASE 1:

…
*/
```

Obfuscated V Contest: Common Approaches

- Buffer overflow
- Misleading #define
- Misleading comments with embedded code /* … */ /* … */
- Order of operations (including argument passing) undefined
- Hiding (nested) scopes
- Confuse 1 with l, 0 with O, = with ==

Underhanded C contest Example Winners

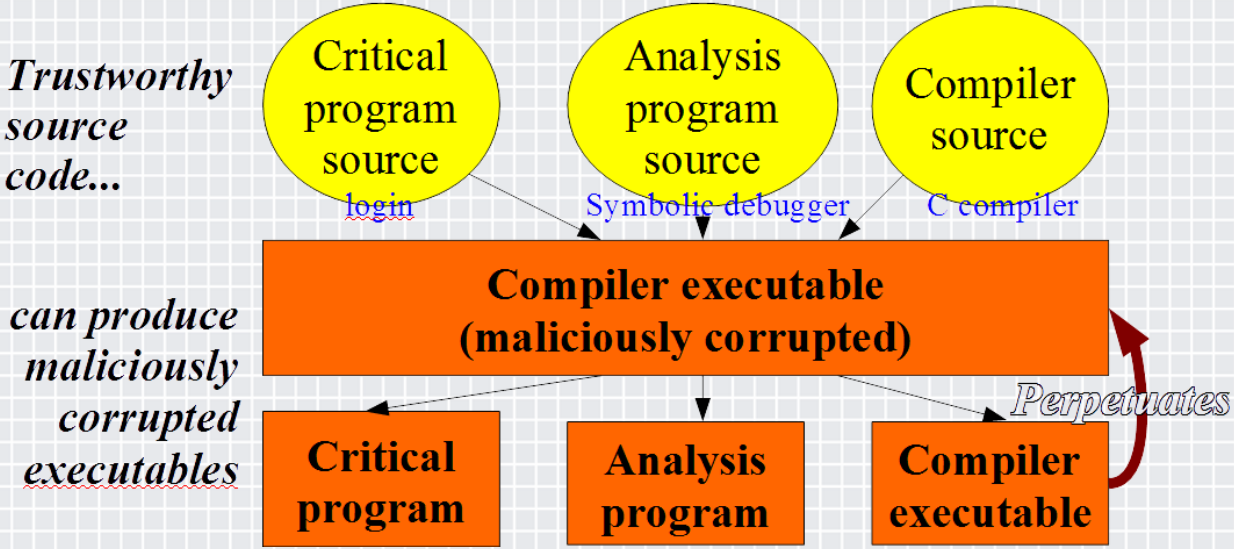
- 2005: covertly insert unique and useful “fingerprinting” data into processed image
  - Winners: uninitialized data structures, reuse of pointers, embedding of machine code in constants
- 2006: word count with vastly different runtimes on different platforms
  - Winners: fork implementation errors, optimization problems, endian differences, various API implementation differences
- 2007: encrypt/decrypt with strong algorithm s.t. a low % may be quickly cracked
  - Winners: misimplementations of RC4, misused API calls, incorrect function prototypes
- 2008: redact image to allow (partial) reconstruction
  - Winners: xor’ed with retrievable pseudo-random mask, appended masked data to file end, used improperly defined macros, zeroed out pixel values while keeping the number of digits intact in a text-based format

Sources: http://www.underhanded-c.org and https://en.wikipedia.org/wiki/Underhanded_C_Contest

Countermeasures for Maliciously Misleading ("underhanded") Code

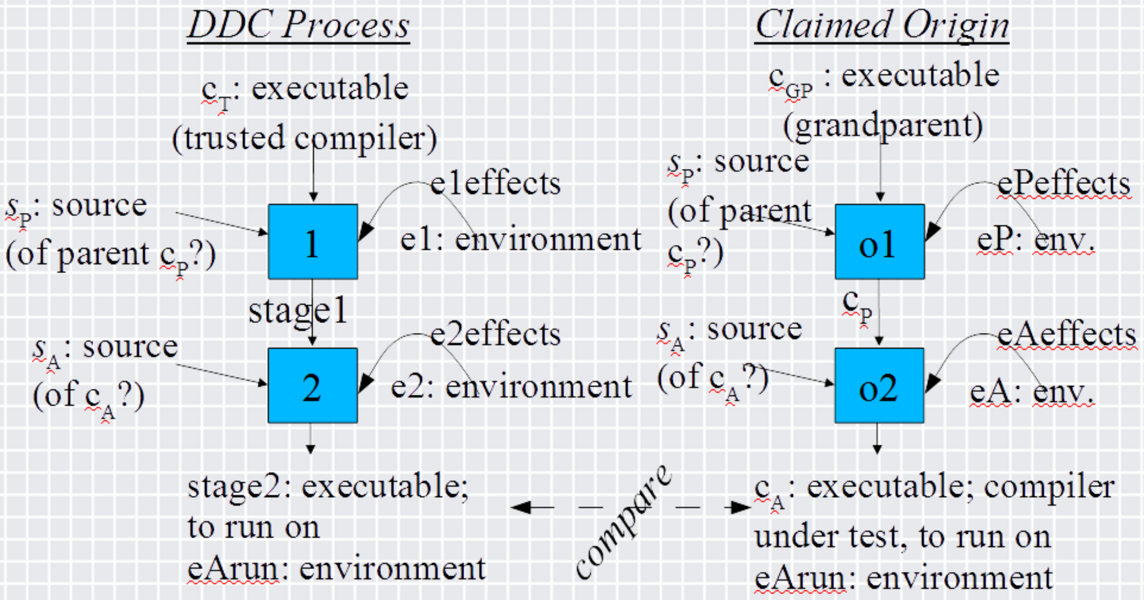
- In general, learn from past attacks
  - When practical use memory-safe languages (or at least ASAN)
  - Force code reformatting & use highlighting
  - Maximize use of warnings (nested scopes, order of operations, bad function prototypes, uninitialized data, etc.)
  - Use multiple static & dynamic analysis tools (buffer overflows, etc.)
  - Precise test cases, including for what it should NOT do
- Limit detailed knowledge of software analysis techniques used, & create some specialized techniques not known to developers

Subverted Binaries (compiler/toolchain): "Trusting trust" attack



- 1974: Karger & Schell first described (obliquely)
- 1984: Ken Thompson demonstrated attack
- 2009: Win32.Induc virus attacks Delphi compilers, infects generated [Mills2009] [Feng2009]

Solution for Subverted Compiler/toolchain: Diverse Double-Compiling (DDC)



Source:
[Wheeler2009]
Fully Countering Trusting Trust through Diverse Double-Compiling
http://www.dwheeler.com/trusting-trust

- Use second compiler/toolkit in unusual way to reproduce executable
  - Works even though different compilers produce different results
- If can reproduce, executable and source match

Diverse Double-Compiling (DDC) Requirements

- DDC does *not* assume that different compilers produce identical executables
- DDC must be performed by trusted programs/processes
  - Includes trusted compiler cT, trusted environments, trusted comparer, trusted acquirers for cA, sP, sA
  - Trusted = justified confidence that it does not have triggers and payloads that would affect the results of DDC. Could be malicious, as long as DDC is unaffected
  - Can do multiple times to increase confidence even further (cumulative)
- Correct languages (Java compiler for Java source)
- Compiler defined by parent's source is deterministic (same inputs always produce same outputs)
  - Real compilers typically deterministic
  - Non-deterministic compilers hard to test & can't use compiler bootstrap test

Other Advanced Countermeasures: Trusted Final Builds

- Create *trusted* build environments
  - Invest in added controls for actual final environments that build and produce shippable code.
- What to include?
  - Best practices that tie to specific threats that can be mitigated
  - Trusted location, state-of-the-art physical security, deeper background checks, rigidly enforced separation of duties, structured oversight, strict promotion of gold disk code to be built.
- Would your most skeptical customers approve and feel confident after a review of all the controls in pace for final build?

Other Advanced Countermeasures: Dev Tool Specific App Sensors

- Open Web Application Security Project (OWASP) - AppSensor
  - Provides methodology, documentation, code and pilots

    More info: [Watson2011] http://appsensor.org/
    https://www.owasp.org/index.php/OWASP_AppSensor_Project
- Design Application aware sensors for critical repos & build tools
  - Build more than traditional network defenses & hardened OS
  - Context-aware analysis in real-time from inside the application
  - Differentiate among normal behavior, suspicious behavior and attacks
    - Monitoring the state of running application
  - Leverage threat modeling & find application specific detection points
  - Can be integrated into app or retrofitted
  - Alerts can tie into Security Information and Event Management (SIEM)

Apply Slide

- Top priority:
  - Ensure you have fundamentals in place to protect development environment (infrastructure, access control, sourcing)
- Then:
  - Establish a protected build environment
  - Require individually-signed commits into repository
  - Establish two-person controls
- Then:
  - Determine if need to counter more advanced threats

References

- [Andrews2003] Andrews, Jeremy. November 5, 2003. “Linux: Kernel ‘Back Door’ Attempt”. *Kerneltrap*. http://kerneltrap.org/node/1584
- [Coviello2011] Coviello, Art. October 4, 2011. Written Testimony, U.S. House of Representatives Permanent Select Committee on Intelligence. http://intelligence.house.gov/sites/intelligence.house.gov/files/documents/100411CyberHearingCoviello.pdf
- [DailyTech] Mick, Jason. May 30, 2011. “Reports: Hackers Use Stolen RSA Information to Hack Lockheed Martin” *Daily Tech*. http://www.dailytech.com/Reports+Hackers+Use+Stolen+RSA+Information+to+Hack+Lockheed+Martin/article21757.htm
- [Gardian] Gardian. Undated. Infragard National Member Alliance. http://www.infragardconferences.com/thegardian/3_22.html
- [Gaudin2006a] Gaudin, Sharon. June 27, 2006. “How A Trigger Set Off A Logic Bomb At UBS PaineWebber”. *InformationWeek*. http://www.informationweek.com/showArticle.jhtml?articleID=189601826
- [Miller2003] Miller, Robin “Roblimo” and Joe “warthawg” Barr. November 6, 2003. “Linux kernel development process thwarts subversion attempt”. NewsForge. http://www.newsforge.com/article.pl?sid=03/11/06/1532223
- [Ulsch2000] Ulsch, MacDonnell. July 2000. “Security Strategies for E-Companies (EC Does it series)”. *Information Security Magazine*. https://web.archive.org/web/20060328015848/http://infosecuritymag.techtarget.com/articles/july00/columns2_ec_doesit.shtml
- [Watson2011] Watson, Colin, et al. September 2011. “Creating Attack-Aware Software Applications with Real-Time Defenses”.CrossTalk OWASP http://www.crosstalkonline.org/storage/issue-archives/2011/201109/201109-0-Issue.pdf (OWASP AppSensor Project)
- [Wheeler2009] Wheeler, David A. 2009. *Fully Countering Trusting Trust through Diverse Double-Compiling*. http://www.dwheeler.com/trusting-trust