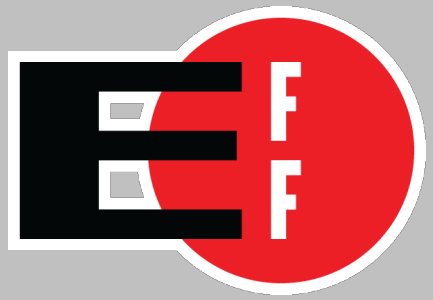
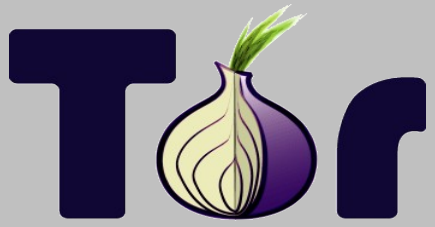
Why and How of Reproducible Builds

Seth Schoen
Electronic Frontier Foundation

Mike Perry
The Tor Project





I want to believe

- Whenever we use binary packages, our basis for believing that we've been given access to the source code is that ***someone said so***
- If we compile the purported source code, we expect to get something that superficially behaves like the binaries
- Not logical or forensic proof!
- I'll argue it's ***inadequate in general***

“But I'm the developer!”

- “I know what's in the binary because I compiled it myself!”
- “I'm an upstanding, careful, and responsible individual!”
- “Why should I have to worry about hypothetical risks about the contents of my binaries?”

Build discrepancies

- Discrepancies between the binary package and its asserted source code could occur if software distributor is
  - careless
  - confused
  - crooked
  - coerced
  - compromised
- {,un}wi{ll,tt}ingly

Severity

I will try to convince you that this problem is:

- extremely hard to detect
- extremely possible
- extremely harmful, if done maliciously

Tampering with binaries

- Trivial, if victim won't do any forensics
- Can be done by ISP or wifi router if the binaries are transferred over FTP or HTTP and aren't digitally signed with a key that the client already knows
  - Compare Brewer, Gauthier, Goldberg, and Wagner's NFS attack (“Basic Flaws”, 1995)
- We can create major vulnerabilities with *very very* small changes to binaries

Fencepost error

- Six fence posts; five fence beams
- For any sequence of *n* objects, there are *(n-1)* transitions from one object to the next

Security consequences

- Often, memory corruption in low-level languages due to executing a loop one too few or one too many times
  - Overwriting data on stack or heap (the array element *past* the end of the array)
  - Can result in malicious code execution

A fencepost error in C

OpenSSH 3.0.2 (CVE-2002-0083) – exploitable security bug (privilege escalation: user can get root)

```
 {
   Channel *c;

-  if (id < 0 || id > channels_alloc) {
+  if (id < 0 || id >= channels_alloc) {
     log("channel_lookup: %d: bad id", id);
     return NULL;
   }
```

Fencepost error in the binary

- What's the difference between if (a > b) and if (a >= b) in x86 assembly?
- JLE → JL assembly instruction
- Opcode 0x7E → 0x7C
- Not just a single byte change: a single ***bit*** change (01111110 → 01111100)
- Other corresponding opcode pairs (like those for >= and >) also differ by just a single bit (JGE=0x7D, JG=0x7F)

Result of fixing the bug (asm)

```
cmpl $0x0,0x8(%ebp)
js   16
mov  0x4,%eax
cmp  %eax,0x8(%ebp)
jle  30
mov  0x8(%ebp),%eax
mov  %eax,0x4(%esp)
movl $0x4c,(%esp)
call 25
```

```
cmpl $0x0,0x8(%ebp)
js   16
mov  0x4,%eax
cmp  %eax,0x8(%ebp)
jl   30
mov  0x8(%ebp),%eax
mov  %eax,0x4(%esp)
movl $0x4c,(%esp)
call 25
```

Result of fixing the bug (asm)

```
cmpl $0x0,0x8(%ebp)
js   16
mov  0x4,%eax
cmp  %eax,0x8(%ebp)
jle  30
mov  0x8(%ebp),%eax
mov  %eax,0x4(%esp)
movl $0x4c,(%esp)
call 25
```

```
cmpl $0x0,0x8(%ebp)
js   16
mov  0x4,%eax
cmp  %eax,0x8(%ebp)
jl   30
mov  0x8(%ebp),%eax
mov  %eax,0x4(%esp)
movl $0x4c,(%esp)
call 25
```

Result of fixing the bug (hex)

```
55 89 e5 83 ec
28 83 7d 08 00
78 0a a1 04 00
00 00 39 45 08
7e 1a 8b 45 08
89 44 24 04 c7
04 24 4c 00 00
00 e8 fc ff ff
ff b8 00 00 00
00 eb 35
```

Overall file size:

```
55 89 e5 83 ec
28 83 7d 08 00
78 0a a1 04 00
00 00 39 45 08
7c 1a 8b 45 08
89 44 24 04 c7
04 24 4c 00 00
00 e8 fc ff ff
ff b8 00 00 00
00 eb 35
```

Approx. 500 kB

Result of fixing the bug (hex)

```
55 89 e5 83 ec
28 83 7d 08 00
78 0a a1 04 00
00 00 39 45 08
7e 1a 8b 45 08
89 44 24 04 c7
04 24 4c 00 00
00 e8 fc ff ff
ff b8 00 00 00
00 eb 35
```

Overall file size:

```
55 89 e5 83 ec
28 83 7d 08 00
78 0a a1 04 00
00 00 39 45 08
7c 1a 8b 45 08
89 44 24 04 c7
04 24 4c 00 00
00 e8 fc ff ff
ff b8 00 00 00
00 eb 35
```

Approx. 500 kB

Infected build platform

- I created a Linux kernel module that alters attempts by the compiler (**only the compiler**) to read C source code
- Source files *as seen by the compiler* get malicious code inserted before first line
- For all other programs (cat, Emacs, sha1sum), source is totally unmodified
- No files on disk are modified, including the kernel, compiler, and source files

Unpleasant thoughts

- We don't like to think about software developers and projects as targets; we think of our software development as a fundamentally benign activity
- **Attackers target a project's users through its developers**
  - See Dullien “Offensive work and addiction” (2014)
- Known successful attacks against infrastructure used by Linux (2003), FreeBSD (2013)

Are these attacks realistic?

"[E]ven costs several hundred times larger than those shown here would be considered nominal to a foreign agent."

- Karger and Schell (1974), on compiler backdoors

"Current popular software development practices simply cannot survive targeted attacks of the scale and scope that we are seeing today."

- Perry (2013), on attacks against software developers and infrastructure

Bitcoin's motivation

- Malicious modifications to Bitcoin client binaries could result in irrevocable, relatively anonymous theft of large amounts of money
- Individual developers could be blamed for such modifications; users might not believe that a developer's machine was hacked
- Reproducible builds protect developers

Software epistemology

- Without certainty about the integrity of build infrastructure, people publishing binaries can't have certainty about the correctness of those binaries
- As targets of attack, we can't achieve this certainty in isolation
- People publishing binaries need other people to check their work!

Build idempotence

- Compile the same program twice on different computers → get different binaries (often!)
- Compile the same program twice on *the same computer* → get different binaries (often!)
- Why? Why isn't compilation a deterministic function?

Deterministic build vision

- Anyone in the world should be able to compile the source code and get a byte-for-byte identical file
- Confirming provenance of binaries
- Infrastructure should be created to independently check popular binaries
  - This is a benefit to those releasing the binaries: they can find out if something bad happens

Deterministic build reality

- Only two projects currently practice this
  - Bitcoin
  - Tor Browser
- But, more are coming!
  - Red Hat
  - Debian (60% of packages already!)
  - F-Droid
  - Mozilla

Tor Browser overview

- Firefox ESR-based “branch”
- Third party tracking and fingerprinting patches
- Tor client and Tor configuration Firefox addon
- Pluggable Transports for traffic obfuscation
- NoScript, HTTPS-Everywhere

Tor Browser build system

- Uses Gitian (from Bitcoin)
- Full package set signed by multiple builders
  - Incremental updates (as unsigned MARs) too!
- Supports anonymous independent verification
- Does not require dedicated build hardware
- Does not require non-free (as in beer) software
  - MacOS and Windows are cross-compiled from Linux
  - Linux tools are free as in freedom

Major toolchain components

- Windows:
  - MinGW-W64 (by commit hash)
  - wine+py2exe
  - nsis
- Mac:
  - Toolchain4 and Crosstools-ng forks by Ray Donnelly
  - mkisofs and libdmg-hfsplus (patched)
- Linux:
  - GCC 4.9.1, binutils 2.24

Gitian overview

- Developed by Bitcoin community
- Wraps Ubuntu virt tools (Qemu-KVM and LXC)
- Compilation stages are YAML "descriptors" that:
  - Specify an Ubuntu release and arch
  - Specify a package list
  - Specify a list of git repos
  - Specify additional "input" files
  - Provide in-line bash script that creates "output" files
  - Can be chained (with some glue code)

Issues Gitian solves

- Normalizes build environment
  - Hostname, username, build paths, tool versions, kernel/uname, time
- Does not require dedicated build hardware
  - Encourages community involvement in verification
- Authenticates git-based inputs
- Integrates with 'faketime' for spoofing timestamps

Gitian limitations

- Ubuntu Only: Cross compilation is required
- Needs non-git input authentication helpers
- Needs dependency and descriptor management glue
- No partial compilation state
  - Base VM images are COW, and COW portion is destroyed
  - faketime causes issues with dependency freshness checks
  - Descriptor stages can be saved, but this gets error-prone
- Time consuming
- Kind of janky
  - qemu-kvm process management issues
  - Supports only one qemu-kvm or LXC slave at a time

Remaining reproducibility issues

- Filesystem and archive reordering
  - os.walk()/os.listdir()/readdir(), zip, tar
  - LC_ALL and locale sorting order
- Unitialized memory in toolchain/archivers
  - binutils for mingw-w64, libdmg-hfsplus
  - GNU linker: debug BuildID (32bit overflow for SHA1?)
- Timezone and umask
- Deliberately generated entropy (FIPS-140, sigs)
- Authenticode and Gatekeeper signatures
- LXC mode still often leaks:
  - Kernel/uname, CPU (libgmp), hostname, memory???

Firefox-specific issues

- about:buildconfig (improved, but still has hostname)
- DLL timestamping using unwrapped time calls
- MAR update signatures
- Profile-Guided Optimization
  - Publish these profiles as official build input
  - Tools to analyze PGO for malicious manipulation?
- EME Host Process

Dependency authentication

- Protect builders from discovery+targeted input attack
  - Use Tor by default for fetching dependencies
  - Authenticate all dependencies **before** use/compilation
- Wrapper scripts for input fetching
  - Verify signatures where possible
  - Many things have weak/no signatures
    - OpenSSL, GCC, faketime, OSX SDK, Go+python packages
    - For these, use SHA256 based on multi-perspective download

Future work

- Remove strict Ubuntu dependency
  - Ideally Debian and Ubuntu could be used to produce the same result
- Trusting trust?
  - Diverse Double Compilation for entire build environment
  - Leverage cross compilation from multiple architectures, distributions
- Multi-sig updates? Consensus updates?
  - Tor Consensus can list update info
  - Bitcoin blockchain
  - Certificate Transparency log

Thanks

Reproducibility section of Tor Browser design document:
https://www.torproject.org/projects/torbrowser/design/#BuildSecurity

Contact us:

Seth Schoen <schoen@eff.org>

`FD9A 6AA2 8193 A9F0 3D4B  F4AD C11B 36DC 9C7D D150`

Mike perry <mikeperry@torproject.org>

`C963 C21D 6356 4E2B 10BB  335B 2984 6B3C 6836 86CC`

`Image credits:` Fence - flickr user hotcactuspepper CC-BY-SA