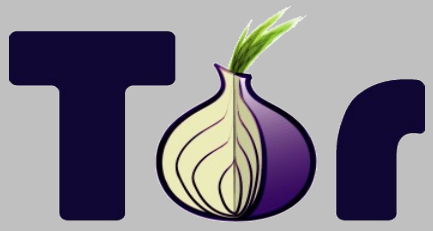
Reproducible Builds: Moving Beyond Single Points of Failure for Software Distribution

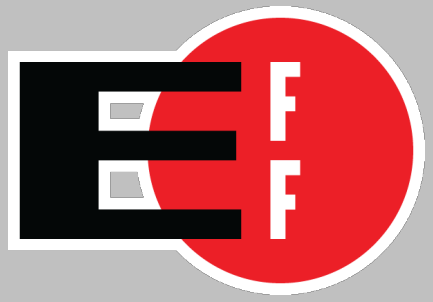
Mike Perry

The Tor Project

Seth Schoen

Electronic Frontier Foundation





Who are we?

- Mike Perry
  - Tor developer; Tor Browser lead
- Seth Schoen
  - Senior Staff Technologist @ EFF
- Hans-Christoph Steiner
  - Guardian Project; Debian Developer
- Lunar
  - Tor developer; Debian Developer
- Bitcoin crew (devrandom, BlueMatt, LukeJr)

“I want to believe”

- FOSS ethos: Users should have the source code to their programs
  - For both individual freedom and software security
- But: The only proof that binary packages correspond to the source code is that ***someone said so***
  - Without build system info, verification is almost impossible (and sometimes even with it)
- This is inadequate for fostering trust in our software's functionality and security

“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?”

Unpleasant thoughts

- We think of software development as a fundamentally benign activity. “I'm not that interesting.”
- **But 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)

Severity

We will try to convince you that this compromise is:

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

Single Points of Failure

Imagine the most secure computer in the world...

Single Points of Failure

Can that computer still remain secure if:

- It is networked?
- It is mobile or is physically accessible by others?
- It regularly has arbitrary USB devices connected?
- It must run Windows (in a VM)?
- It regularly runs unauthenticated HTML+JS?
- Several nation-states want access to it?

Single Points of Failure

What if:

- Compromising that one computer gave access to:
  - Hundreds of millions of other computers?
  - Every bank account in the world?
  - Every Windows computer in the world?
  - Every Linux server in the world?
- Compromising that computer was worth:
  - $100k USD? (Market price of remote 0day)
  - $100M USD? (Censorship budget of Iran/yr)
  - $4B USD? (Bitcoin market cap)

Bitcoin's motivation

- Malicious modifications to Bitcoin binaries could result in irrevocable 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

How small can a backdoor be?

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;
    }
```

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
```

Resulting difference in the binary

What's the difference between if (a > b) and if (a >= b) in x86 assembly?

- assembly: JLE → JL
- opcode: 0x7E → 0x7C
- binary: 0111111**1**0 → 0111111**0**0

A single bit!

Other corresponding opcode pairs also differ by just a single bit (JGE=0x7D, JG=0x7F)

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

Solution: Reproducible Builds

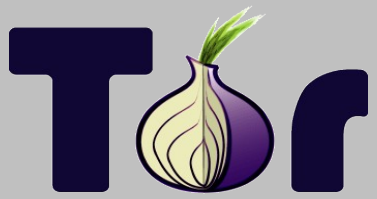
- Anyone in the world should be able to compile a program's source code and get a byte-for-byte identical binary
- Confirming integrity of binaries
- Infrastructure should be created to independently check popular binaries
  - Also provides external monitoring to find out if something bad happens to build infrastructure

Common obstacles

- Different compilers or optimizations
- Different header files
- Different library versions
- Build-environment metadata
- Container formats with filesystem data
- Timestamps
- Signatures/key management
- Test-driven optimizations (aka PGO)

Reproducible builds today

- Only a handful of projects currently practice this



- More are coming!

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 addons

Tor Browser build system

- Uses Gitian (from Bitcoin)
- Full package set signed by multiple builders
  - Incremental updates 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 (faketime)
- 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
- Partial compilation state is tricky
  - 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
  - Binutils linker: 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???

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

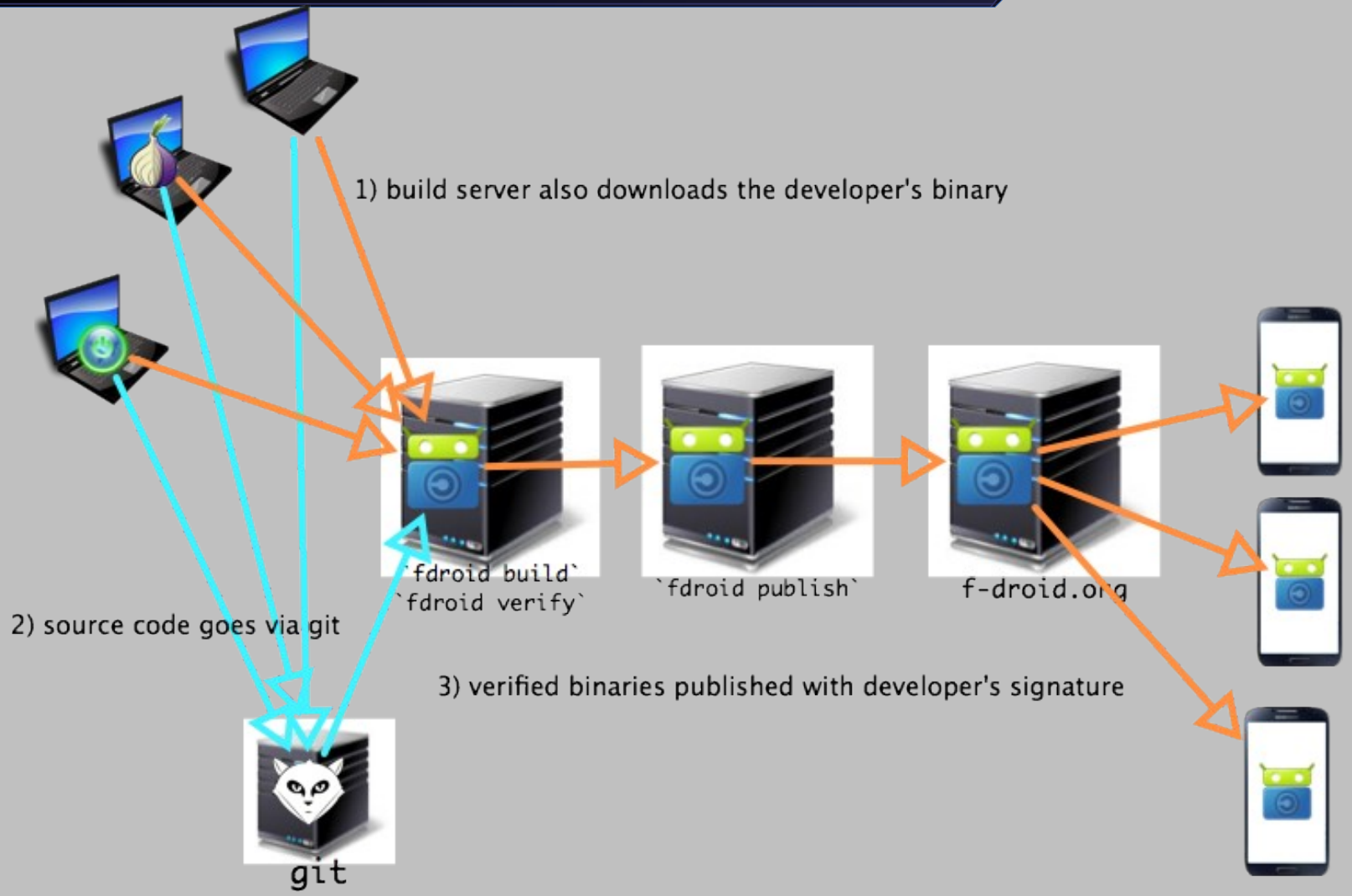
This process is not always scary

- Firefox and Tor Browser are massive and scary
- Most software is not that complicated
- Libraries tend to be simple
- Android apps are mostly pure Java
- Debian packaging provides a meta-process

It's much easier on Android

- Android APKs do not need exact hash matches
  - Java JAR signatures are used
  - Only the contents are signed
  - File timestamps are not signed
  - The signed manifest is filename and hash
  - The manifest file order is separate from file order in the APK itself, so sort order is less important

FDroid reproducible process



Future work

- Remove strict Ubuntu dependency for Gitian
  - 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

More info

- **Reproducibility section of Tor Browser design document:** https://www.torproject.org/projects/torbrowser/design/#BuildSecurity
- F-Droid verification process: https://f-droid.org/wiki/page/Verification_Server
- Debian Reproducible working group: https://wiki.debian.org/ReproducibleBuilds
- Countering Trusting Trust: https://www.schneier.com/blog/archives/2006/01/countering_trus.html https://lwn.net/Articles/555902/

Thanks

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`

Hans-Christoph Steiner <hans@guardianproject.info>

`5E61 C878 0F86 295C E17D  8677 9F0F E587 374B BE81`

Lunar <lunar@debian.org>