FreeBSD Portsnap

What (it is), Why (it was written), and How (it works)

Colin Percival
The FreeBSD Project
cperciva@FreeBSD.org

May 19, 2007

FreeBSD Portsnap

A Case Study in Black Magic

Colin Percival
The FreeBSD Project
cperciva@FreeBSD.org

May 19, 2007

Introduction to Portsnap

- Portsnap is a system for securely and efficiently distributing the FreeBSD Ports tree.

Introduction to Portsnap

- Portsnap is a system for securely and efficiently distributing the FreeBSD Ports tree.
- Introduced in October 2004, added to the base system in August 2005.

Introduction to Portsnap

- Portsnap is a system for securely and efficiently distributing the FreeBSD Ports tree.
- Introduced in October 2004, added to the base system in August 2005.
- Present in all releases since FreeBSD 6.0-RELEASE, 5.5-RELEASE.

Introduction to Portsnap

- Portsnap is a system for securely and efficiently distributing the FreeBSD Ports tree.
- Introduced in October 2004, added to the base system in August 2005.
- Present in all releases since FreeBSD 6.0-RELEASE, 5.5-RELEASE.
- Now used on approximately 30,000 systems.

Introduction to Portsnap

- Portsnap is a system for securely and efficiently distributing the FreeBSD Ports tree.
- Introduced in October 2004, added to the base system in August 2005.
- Present in all releases since FreeBSD 6.0-RELEASE, 5.5-RELEASE.
- Now used on approximately 30,000 systems.
  - Yes, I will have some pretty graphs later.

A bird's-eye view of Portsnap

- Portsnap build code runs on hardware "owned" by the FreeBSD Security Team.

A bird's-eye view of Portsnap

- Portsnap build code runs on hardware "owned" by the FreeBSD Security Team.
- Builds are uploaded via ssh to portsnap-master.freebsd.org.

A bird's-eye view of Portsnap

- Portsnap build code runs on hardware "owned" by the FreeBSD Security Team.
- Builds are uploaded via ssh to portsnap-master.freebsd.org.
- Mirrors (3 of them, so far) update from portsnap-master.freebsd.org.

A bird's-eye view of Portsnap

- Portsnap build code runs on hardware "owned" by the FreeBSD Security Team.
- Builds are uploaded via ssh to portsnap-master.freebsd.org.
- Mirrors (3 of them, so far) update from portsnap-master.freebsd.org.
- Individual client systems update /var/db/portsnap from a randomly selected mirror.

A bird's-eye view of Portsnap

- Portsnap build code runs on hardware "owned" by the FreeBSD Security Team.
- Builds are uploaded via ssh to portsnap-master.freebsd.org.
- Mirrors (3 of them, so far) update from portsnap-master.freebsd.org.
- Individual client systems update /var/db/portsnap from a randomly selected mirror.
- The ports tree can be extracted or updated from /var/db/portsnap.

Black Magic #1: DNS SRV records

- DNS SRV records (RFC 2782) provide a mechanism for mapping a type of service to host name(s).

Black Magic #1: DNS SRV records

- DNS SRV records (RFC 2782) provide a mechanism for mapping a type of service to host name(s).
  - Approximately a generalization of MX records.

Black Magic #1: DNS SRV records

- DNS SRV records (RFC 2782) provide a mechanism for mapping a type of service to host name(s).
  - Approximately a generalization of MX records.
  - Clients are expected to pick a server randomly based on the specified priorities and weights.

Black Magic #1: DNS SRV records

- DNS SRV records (RFC 2782) provide a mechanism for mapping a type of service to host name(s).
  - Approximately a generalization of MX records.
  - Clients are expected to pick a server randomly based on the specified priorities and weights.

```
_http._tcp.portsnap.freebsd.org IN SRV 1 10 80
portsnap1
```

Black Magic #1: DNS SRV records

- DNS SRV records (RFC 2782) provide a mechanism for mapping a type of service to host name(s).
  - Approximately a generalization of MX records.
  - Clients are expected to pick a server randomly based on the specified priorities and weights.

```
_http._tcp.portsnap.freebsd.org IN SRV 1 10 80
portsnap1
```

- Portsnap runs over HTTP, and obeys the HTTP_PROXY environment variable.

Black Magic #1: DNS SRV records

- DNS SRV records (RFC 2782) provide a mechanism for mapping a type of service to host name(s).
  - Approximately a generalization of MX records.
  - Clients are expected to pick a server randomly based on the specified priorities and weights.

```
_http._tcp.portsnap.freebsd.org IN SRV 1 10 80
portsnap1
```

- Portsnap runs over HTTP, and obeys the HTTP_PROXY environment variable.
- If HTTP_PROXY is set, Portsnap uses SHA256(HTTP_PROXY) as a random number seed when selecting a random mirror.

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.
- Introduced in April 2003, presented at BSDCon'03.

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.
- Introduced in April 2003, presented at BSDCon'03.
- Updates are signed to prove that they are authentic.

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.
- Introduced in April 2003, presented at BSDCon'03.
- Updates are signed to prove that they are authentic.
  - No need to trust CVSup mirrors!

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.
- Introduced in April 2003, presented at BSDCon'03.
- Updates are signed to prove that they are authentic.
  - No need to trust CVSup mirrors!
- Until August 2006, FreeBSD Update was in the Ports tree.

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.
- Introduced in April 2003, presented at BSDCon'03.
- Updates are signed to prove that they are authentic.
  - No need to trust CVSup mirrors!
- Until August 2006, FreeBSD Update was in the Ports tree.
  - ... which most people downloaded via CVSup.

FreeBSD Update

- FreeBSD Update is a system for building, distributing, and applying binary security updates to the FreeBSD base system.
- Introduced in April 2003, presented at BSDCon'03.
- Updates are signed to prove that they are authentic.
  - No need to trust CVSup mirrors!
- Until August 2006, FreeBSD Update was in the Ports tree.
  - ... which most people downloaded via CVSup.
    - ... Oops.

Secure CVS trees

- Add a checksum file to each directory in the tree, containing
  - ... the hashes of all the other files in the directory.
  - ... the hashes of the checksum files in any (immediate) subdirectories.

Secure CVS trees

- Add a checksum file to each directory in the tree, containing
  - ... the hashes of all the other files in the directory.
  - ... the hashes of the checksum files in any (immediate) subdirectories.
- Sign the checksum file in the root directory.

Secure CVS trees

- Add a checksum file to each directory in the tree, containing
  - ... the hashes of all the other files in the directory.
  - ... the hashes of the checksum files in any (immediate) subdirectories.
- Sign the checksum file in the root directory.
- Each time a commit is done, automatically rebuild checksum files going up to the root, and re-sign the root checksum file.

Secure CVS trees

- Add a checksum file to each directory in the tree, containing
  - ... the hashes of all the other files in the directory.
  - ... the hashes of the checksum files in any (immediate) subdirectories.
- Sign the checksum file in the root directory.
- Each time a commit is done, automatically rebuild checksum files going up to the root, and re-sign the root checksum file.
- I hope someone builds this some day. I didn't have time.

A simpler approach

- Instead of making the tree self-authenticating and using existing mechanisms to distribute it, keep authentication out of the tree and have a new utility which downloads and verifies.

A simpler approach

- Instead of making the tree self-authenticating and using existing mechanisms to distribute it, keep authentication out of the tree and have a new utility which downloads and verifies.
- Divide the tree into N independent pieces, and generate an N-line index file containing the hashes of all the pieces.

A simpler approach

- Instead of making the tree self-authenticating and using existing mechanisms to distribute it, keep authentication out of the tree and have a new utility which downloads and verifies.
- Divide the tree into N independent pieces, and generate an N-line index file containing the hashes of all the pieces.
- Distribute the N pieces, the index, and a signed hash of the index as static files over HTTP.

A simpler approach

- Instead of making the tree self-authenticating and using existing mechanisms to distribute it, keep authentication out of the tree and have a new utility which downloads and verifies.
- Divide the tree into N independent pieces, and generate an N-line index file containing the hashes of all the pieces.
- Distribute the N pieces, the index, and a signed hash of the index as static files over HTTP.
  - We don't really need to invent a new protocol after all...

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.
  - HTTP servers are light-weight compared to more complicated protocols like CVSup and rsync.

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.
  - HTTP servers are light-weight compared to more complicated protocols like CVSup and rsync.
- Using static files over HTTP makes firewall/proxy traversal easy.

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.
  - HTTP servers are light-weight compared to more complicated protocols like CVSup and rsync.
- Using static files over HTTP makes firewall/proxy traversal easy.
  - Actually, squid manages to cause problems by not supporting HTTP/1.1, but I think that can be worked around.

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.
  - HTTP servers are light-weight compared to more complicated protocols like CVSup and rsync.
- Using static files over HTTP makes firewall/proxy traversal easy.
  - Actually, squid manages to cause problems by not supporting HTTP/1.1, but I think that can be worked around.
- Using static files (and a signature) provides end to end security.

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.
  - HTTP servers are light-weight compared to more complicated protocols like CVSup and rsync.
- Using static files over HTTP makes firewall/proxy traversal easy.
  - Actually, squid manages to cause problems by not supporting HTTP/1.1, but I think that can be worked around.
- Using static files (and a signature) provides end to end security.
  - We don't need to worry about the possibility of mirrors being compromised.

Black Magic #2: Static files

- Serving static files is easy – choose your favourite HTTP server.
  - HTTP servers are light-weight compared to more complicated protocols like CVSup and rsync.
- Using static files over HTTP makes firewall/proxy traversal easy.
  - Actually, squid manages to cause problems by not supporting HTTP/1.1, but I think that can be worked around.
- Using static files (and a signature) provides end to end security.
  - We don't need to worry about the possibility of mirrors being compromised.
  - We don't need to worry about the possibility of an SSL certificate being compromised.

Dividing up the ports tree

- We want to divide the ports tree into N pieces.

Dividing up the ports tree

- We want to divide the ports tree into N pieces.
- The larger N is, the larger the overhead costs (TCP, HTTP, inodes, etc.) of handling many small files.

Dividing up the ports tree

- We want to divide the ports tree into N pieces.
- The larger N is, the larger the overhead costs (TCP, HTTP, inodes, etc.) of handling many small files.
- The smaller N is, the larger the cost (bandwidth, CPU time) of updating each piece.

Dividing up the ports tree

- We want to divide the ports tree into N pieces.
- The larger N is, the larger the overhead costs (TCP, HTTP, inodes, etc.) of handling many small files.
- The smaller N is, the larger the cost (bandwidth, CPU time) of updating each piece.
- Asymptotically, we probably want $N = \mathrm{O}(\sqrt{[\text{size of tree}]})$.

Dividing up the ports tree

- We want to divide the ports tree into N pieces.
- The larger N is, the larger the overhead costs (TCP, HTTP, inodes, etc.) of handling many small files.
- The smaller N is, the larger the cost (bandwidth, CPU time) of updating each piece.
- Asymptotically, we probably want $N = \mathrm{O}(\sqrt{[\text{size of tree}]})$.
  - For a tree of $\approx$ 100 MB it's reasonable for N to be a few thousand.

Dividing up the ports tree

- We want to divide the ports tree into N pieces.
- The larger N is, the larger the overhead costs (TCP, HTTP, inodes, etc.) of handling many small files.
- The smaller N is, the larger the cost (bandwidth, CPU time) of updating each piece.
- Asymptotically, we probably want $N = \mathrm{O}(\sqrt{[\text{size of tree}]})$.
  - For a tree of $\approx$ 100 MB it's reasonable for N to be a few thousand.
- In Portsnap, the pieces are
  - /usr/ports/category/port
  - /usr/ports/category/file
  - /usr/ports/file

  and each piece is stored as a tarball.

Black Magic #3: Understand how things change

- The central problem of efficient data compression is to model files.

Black Magic #3: Understand how things change

- The central problem of efficient data compression is to model files.
  - Most compressors explicitly use the first n bytes to predict the value of the $n+1$th byte.

Black Magic #3: Understand how things change

- The central problem of efficient data compression is to model files.
  - Most compressors explicitly use the first n bytes to predict the value of the $n+1$th byte.
- The central problem of efficient *delta* compression is to model *how files change*.

Black Magic #3: Understand how things change

- The central problem of efficient data compression is to model files.
  - Most compressors explicitly use the first n bytes to predict the value of the $n+1$th byte.
- The central problem of efficient *delta* compression is to model *how files change*.
  - Side note: Part of the reason bsdiff is so efficient is that it is the first delta compressor designed with an awareness of byte substitutions.

Black Magic #3: Understand how things change

- The central problem of efficient data compression is to model files.
  - Most compressors explicitly use the first n bytes to predict the value of the $n+1$th byte.
- The central problem of efficient *delta* compression is to model *how files change*.
  - Side note: Part of the reason bsdiff is so efficient is that it is the first delta compressor designed with an awareness of byte substitutions.
- Commits to the ports tree often modify several files, but usually they are part of the same port.

Black Magic #3: Understand how things change

- The central problem of efficient data compression is to model files.
  - Most compressors explicitly use the first n bytes to predict the value of the $n+1$th byte.
- The central problem of efficient *delta* compression is to model *how files change*.
  - Side note: Part of the reason bsdiff is so efficient is that it is the first delta compressor designed with an awareness of byte substitutions.
- Commits to the ports tree often modify several files, but usually they are part of the same port.
  - Dividing the tree into individual ports is a natural granuality based on how the tree changes.

Black Magic #4: Reference by hash

- Traditional approach: "ports/misc/bsdiff is stored in misc_bsdiff_123.tar and has SHA256 hash 01234567...89ABCDEF".

Black Magic #4: Reference by hash

- Traditional approach: "ports/misc/bsdiff is stored in misc_bsdiff_123.tar and has SHA256 hash 01234567...89ABCDEF".
- Reference by hash: "ports/misc/bsdiff is stored in 01234567...89ABCDEF.tar and has SHA256 hash 01234567...89ABCDEF".

Black Magic #4: Reference by hash

- Traditional approach: "ports/misc/bsdiff is stored in misc_bsdiff_123.tar and has SHA256 hash 01234567...89ABCDEF".
- Reference by hash: "ports/misc/bsdiff is stored in 01234567...89ABCDEF.tar and has SHA256 hash 01234567...89ABCDEF".
- Don't need to worry about naming collisions, since a strong hash will never collide.

Black Magic #4: Reference by hash

- Traditional approach: "ports/misc/bsdiff is stored in misc_bsdiff_123.tar and has SHA256 hash 01234567...89ABCDEF".
- Reference by hash: "ports/misc/bsdiff is stored in 01234567...89ABCDEF.tar and has SHA256 hash 01234567...89ABCDEF".
- Don't need to worry about naming collisions, since a strong hash will never collide.
  - Well, hopefully, at least.

Black Magic #4: Reference by hash

- Traditional approach: "ports/misc/bsdiff is stored in misc_bsdiff_123.tar and has SHA256 hash 01234567...89ABCDEF".
- Reference by hash: "ports/misc/bsdiff is stored in 01234567...89ABCDEF.tar and has SHA256 hash 01234567...89ABCDEF".
- Don't need to worry about naming collisions, since a strong hash will never collide.
  - Well, hopefully, at least.
- Each part of the tree is self-authenticating.

Black Magic #4: Reference by hash

- Traditional approach: "ports/misc/bsdiff is stored in misc_bsdiff_123.tar and has SHA256 hash 01234567...89ABCDEF".
- Reference by hash: "ports/misc/bsdiff is stored in 01234567...89ABCDEF.tar and has SHA256 hash 01234567...89ABCDEF".
- Don't need to worry about naming collisions, since a strong hash will never collide.
  - Well, hopefully, at least.
- Each part of the tree is self-authenticating.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...
- The INDEX file is generated by recursing into every Makefile in the tree.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...
- The INDEX file is generated by recursing into every Makefile in the tree.
  - This takes 10–30 minutes.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...
- The INDEX file is generated by recursing into every Makefile in the tree.
  - This takes 10–30 minutes.
  - If someone can insert a trojan into misc/nobody-uses-this, they can execute arbitrary code on any system which builds an INDEX.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...
- The INDEX file is generated by recursing into every Makefile in the tree.
  - This takes 10–30 minutes.
  - If someone can insert a trojan into misc/nobody-uses-this, they can execute arbitrary code on any system which builds an INDEX.
- INDEX is built on the Portsnap buildbox and distributed to client systems.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...
- The INDEX file is generated by recursing into every Makefile in the tree.
  - This takes 10–30 minutes.
  - If someone can insert a trojan into misc/nobody-uses-this, they can execute arbitrary code on any system which builds an INDEX.
- INDEX is built on the Portsnap buildbox and distributed to client systems.
  - For security reasons, INDEX is built as a non-privileged user inside a jail which contains a minimal FreeBSD world where all filesystems are mounted either readonly or noexec.

Things get ugly: Distributing INDEX files

- FreeBSD package tools use an INDEX file which summarizes the ports tree.
  - Package name, version, directory, dependencies...
- The INDEX file is generated by recursing into every Makefile in the tree.
  - This takes 10–30 minutes.
  - If someone can insert a trojan into misc/nobody-uses-this, they can execute arbitrary code on any system which builds an INDEX.
- INDEX is built on the Portsnap buildbox and distributed to client systems.
  - For security reasons, INDEX is built as a non-privileged user inside a jail which contains a minimal FreeBSD world where all filesystems are mounted either readonly or noexec.
    - Hopefully this is good enough...

Saving bandwidth

- Instead of downloading complete files, Portsnap downloads patches against older versions whenever possible.

Saving bandwidth

- Instead of downloading complete files, Portsnap downloads patches against older versions whenever possible.
  - Binary patches are used for the component tarballs.

Saving bandwidth

- Instead of downloading complete files, Portsnap downloads patches against older versions whenever possible.
  - Binary patches are used for the component tarballs.
  - A hacked-up textual patch format is used for the index of components and for the ports INDEX file.

Saving bandwidth

- Instead of downloading complete files, Portsnap downloads patches against older versions whenever possible.
  - Binary patches are used for the component tarballs.
  - A hacked-up textual patch format is used for the index of components and for the ports INDEX file.
- For a typical 58 hour window of updates in 2005, CVSup used 6388kB of bandwidth, while portsnap only used 370kB.

Saving bandwidth

- Instead of downloading complete files, Portsnap downloads patches against older versions whenever possible.
  - Binary patches are used for the component tarballs.
  - A hacked-up textual patch format is used for the index of components and for the ports INDEX file.
- For a typical 58 hour window of updates in 2005, CVSup used 6388kB of bandwidth, while portsnap only used 370kB.
  - When very little has changed in the tree, CVSup spends most of its time/bandwidth listing files and deciding that they haven't changed.

Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.

Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.
  - Building $\mathrm{O}(N^2)$ patches takes a long time.

Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.
  - Building $\mathrm{O}(N^2)$ patches takes a long time.
  - Applying a series of N patches, one by one, is both complicated and slow.

Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.
  - Building $\mathrm{O}(N^2)$ patches takes a long time.
  - Applying a series of N patches, one by one, is both complicated and slow.
- Opportunistic patching: Build some patches, but not all of them.

Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.
  - Building $\mathrm{O}(N^2)$ patches takes a long time.
  - Applying a series of N patches, one by one, is both complicated and slow.
- Opportunistic patching: Build some patches, but not all of them.
  - Client systems try to fetch a patch, but fall back to fetching a complete file if the patch isn't available.

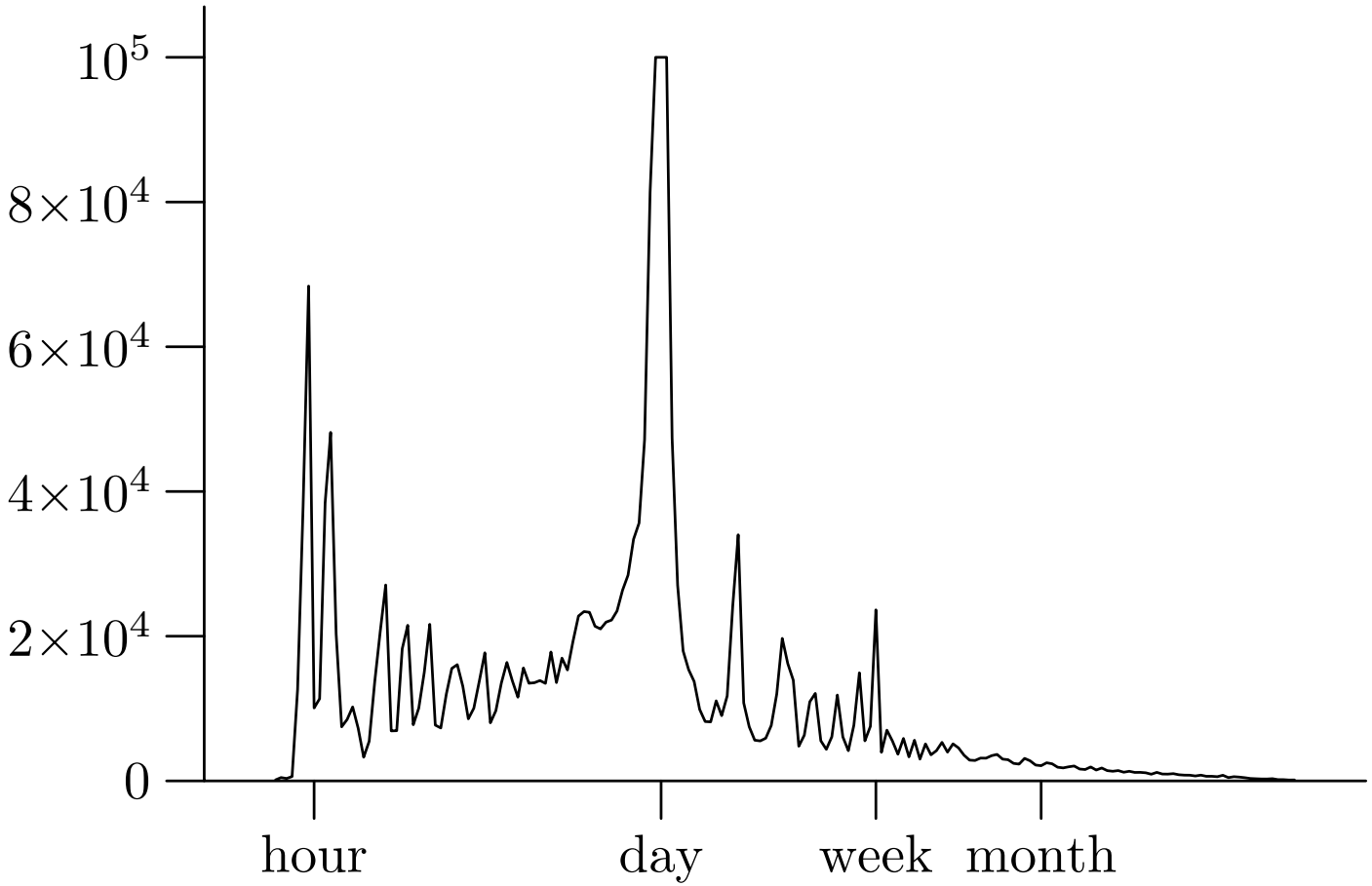
Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.
  - Building $\mathrm{O}(N^2)$ patches takes a long time.
  - Applying a series of N patches, one by one, is both complicated and slow.
- Opportunistic patching: Build some patches, but not all of them.
  - Client systems try to fetch a patch, but fall back to fetching a complete file if the patch isn't available.
  - By building a small number of patches, we can ensure that most systems will be using patches most of the time.

Black Magic #5: Opportunistic patching

- Problem: If you have N versions of a file, there are $\mathrm{O}(N^2)$ pairs between which to build patches.
  - Building $\mathrm{O}(N^2)$ patches takes a long time.
  - Applying a series of N patches, one by one, is both complicated and slow.
- Opportunistic patching: Build some patches, but not all of them.
  - Client systems try to fetch a patch, but fall back to fetching a complete file if the patch isn't available.
  - By building a small number of patches, we can ensure that most systems will be using patches most of the time.
  - Right now, patches are always for Portsnap on systems which update at least once a week.

Portsnap updating statistics



Black Magic #6: Pipelined HTTP

- Pipelined HTTP can easily speed up fetching small files by an order of magnitude.

Black Magic #6: Pipelined HTTP

- Pipelined HTTP can easily speed up fetching small files by an order of magnitude.
- When Portsnap is fetching patches (typical size 500 bytes) the speedup can be over a factor of 100.

Black Magic #6: Pipelined HTTP

- Pipelined HTTP can easily speed up fetching small files by an order of magnitude.
- When Portsnap is fetching patches (typical size 500 bytes) the speedup can be over a factor of 100.
- Not really black magic at all — pipelined HTTP is something which everybody should be using.

Black Magic #6: Pipelined HTTP

- Pipelined HTTP can easily speed up fetching small files by an order of magnitude.
- When Portsnap is fetching patches (typical size 500 bytes) the speedup can be over a factor of 100.
- Not really black magic at all — pipelined HTTP is something which everybody should be using.
  - Unfortunately, shockingly few people do.

Black Magic #6: Pipelined HTTP

- Pipelined HTTP can easily speed up fetching small files by an order of magnitude.
- When Portsnap is fetching patches (typical size 500 bytes) the speedup can be over a factor of 100.
- Not really black magic at all — pipelined HTTP is something which everybody should be using.
  - Unfortunately, shockingly few people do.
  - I had to write my own command-line pipelined HTTP client as part of Portsnap because I couldn't find one anywhere.

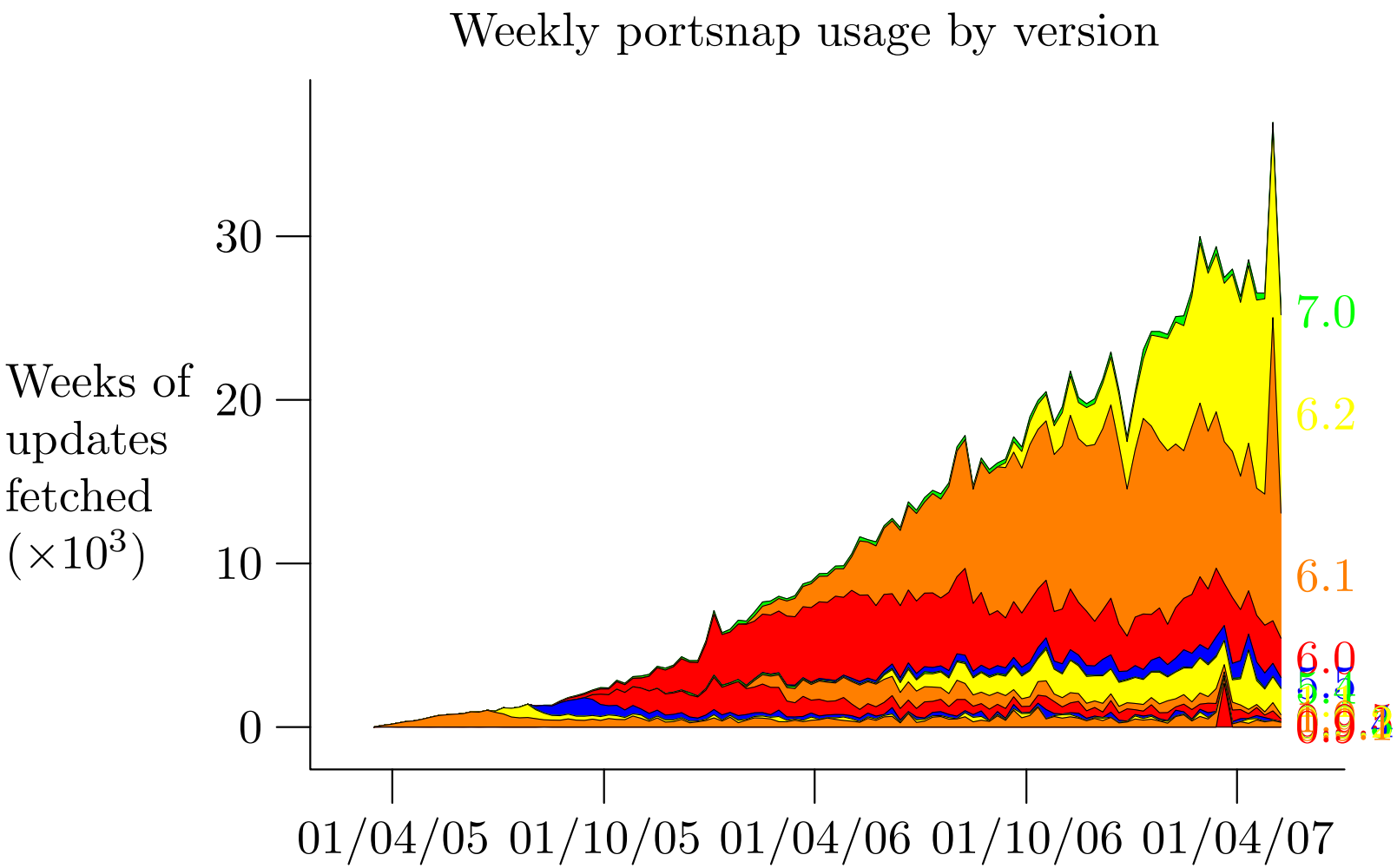
Spyware!

PRIVACY NOTICE

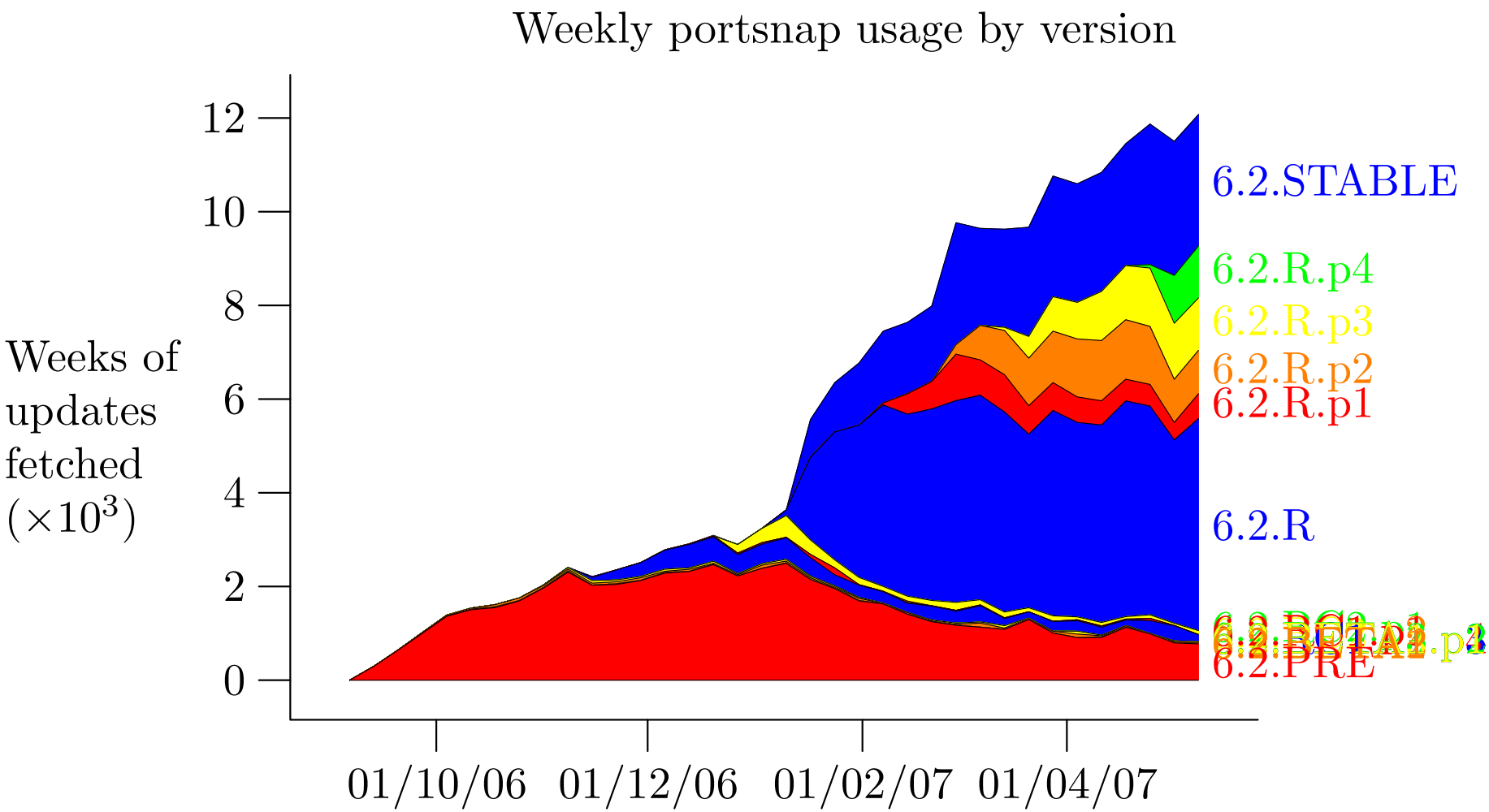
As an unavoidable part of its operation, a machine running portsnap will make its public IP address and the list of files it fetches available to the server from which it fetches updates. Using these it may be possible to recognize a machine over an extended period of time, determine when it is updated, and identify which portions of the FreeBSD ports tree, if any, are being ignored using "REFUSE" directives in portsnap.conf. In addition, the FreeBSD release level is transmitted to the server.

Statistical data generated from information collected in this manner may be published, but only in aggregate and after anonymizing the individual systems.

Portsnap usage



Portsnap usage on FreeBSD 6.2



Questions?