Contact: lkcl@lkcl.net Date: 08 Sep 2017

Abstract

This document covers the scenario where several people (named "developers") create files which need to be distributed (to "users") and upgraded in a way that tampering may be detected. Members of another group (named "maintainers", who may also be synonymous with individual "developers") will be responsible for "releasing" any given file. Special member(s) of that group (named "FTPMasters") are specifically responsible for declaring that a given set of release files is in fact "safe". Another (independent) group (named "Mirrors") take copies of those "safe" files and help make them available to users.

Contents

1 Scope

- This document covers data **integrity**. The **privacy** of the actual distribution of the files themselves is declared "out of scope" and is not part of this document.
- The term "sign" is used with a very specific meaning: whatever method is used must be up to a "cryptographic" standard in order to be acceptable. [1]
- In instances where users do not follow the procedures, or follow the procedures (manual or automatic) but fail to understand or act appropriately on the result, there is nothing that can be done for them, Such failures are out of scope for this document.
- However by contrast: failures of *Maintainers* or the *FTPMasters* to follow procedures (manual or automatic) can result in a serious violation of trust for Users, and there need to be "consequences" which specifically aid and assist in restoring that trust in the overall group's ability. Exactly what those consequences are is also out of scope for this document, with the exception of where revocation of a Digital Identity is specifically required.
- The process described herein is entirely generic and independent of the actual data, or the purpose to which the file(s) are actually put. Thus for example in the (main) instance where this process is used for the distribution of computer program source code, actual building or obtaining the dependencies, or documentation, is not a critical part of the actual process itself, and is completely outside the scope of this document.
- That having been said, it is anticipated that the process be primarily implemented in software, and as such a special case is included in this document which covers ensuring that tools, files and resources (including specifically the source code of the tools) that themselves are required *for* the process are strongly recommended to be managed *by* the process, such that upgrades and updates (in particular critical security updates) may be done quickly but also in a safe and provably-secure automated fashion if so desired.

2 The Process

This section (greatly simplified) describes the actions that are taken by all parties involved in the creation, management and distribution of the files:

- A developer creates a file (and optionally marks it with a revision number)
- A maintainer obtains the file, marking it with a revision number (mandatory)
- The maintainer passes the "released" file to the FTPMasters
- The FTPMasters distribute the file along with other "released" files
- The "Mirrors" take copies of that set of files and distribute them
- Users take further copies of (some or all) of the files
- Users VERIFY the authenticity of the files before trusting them.

Thus for this last critical step to work, every single step in the above process must be verifiable. This is referred to as a "chain of trust" and its correct use has huge ramifications for the entire process. It is critical to note:

- No part of the process may be skipped: if any part is missed out, the proof of trust can and must be concluded to be a complete and total failure.
- If any one part of the process is compromised, the entire verification process is compromised.
- Detection of a compromise, whilst straightforward, needs extreme care in communicating its discovery and reporting, so that nobody is deceived about the event ("false positive").

3 Requirements

The Primary requirement is that file integrity, regardless of the distribution method, may provably be verified as not having been tampered with, and that if tampering ever occurs it may be reliably be detected, so that appropriate action can be taken. As can be seen from the Process section, there is a long chain and a surprisingly large amount involved. So the Primary requirement may be broken down into several sub-requirements:

1. A user must be able to confidently establish the identity of the FTPMasters.
2. A user must be able to confidently establish the identity of every Maintainer.
3. All parties except the user must be able to inter-establish their identities.
4. All parties must be able to establish if an identity has been compromised.
5. Maintainers must "sign" a released file as being linked to their identity.
6. FTPMasters must "sign" a set of releases as being linked to their identity.
7. All parties doing "signing" are responsible for securing their identity.
8. FTPMasters MUST be able to revoke any given Maintainer's "released file".
   (This is in case a Maintainer is compromised, intentionally or not).
9. A user must be able to download a "Release set" and post-verify the FTPMasters
   (Note that the requirement is NOT that the DOWNLOAD be "secure")
10. A user must be able to download a file and post-verify its Maintainer
    (Note that the requirement is NOT that the DOWNLOAD be "secure")
11. No person or resource shall EVER become a high-priority target for attack
    (i.e. the implementation shall NOT make anything worth trying to compromise)
12. Discussion of Revocation of any identity or release must be "signed"
    (this ensures that revocation may not be spoofed, misleading users)
13. All public announcements regarding releases must be "signed"
    (this ensures that announcements may not be spoofed, misleading everyone)
14. With very few exceptions, all parties MUST treat unsigned comms as UNTRUSTED
    (in rare cases, physical identity confirmation is the only option)
15. Maintainers must check (and report any new) irregularities (signing all comms)
    (this means verifying online copies of Releases and FTPMasters Releases)
16. FTPMasters must check (and report any new) irregularities (signing all comms)
    (this mainly means verifying the online copies of FTPMasters "Release" file)
17. The means to perform all verification steps manually must be fully documented
    (for specific times when "management" tools cannot be installed or used)
18. All identity confirmation must be established using 100% trustable channels

Note that it is assumed that when a party "signs" a file linking the file to their identity, that they are implicitly and simultaneously making a public "Declaration of Trust", namely that they have taken responsibilty for verifying the integrity of the file that they are signing. Failure to do so undermines the entire chain. Consequently, violations of trust must be taken extremely seriously.

3.1 Requirements Analysis

The number of requirements seems at first to be so large that surely it must be possible to leave even one of them out. This subsection goes over each requirement in turn, demonstrating that not one single one of them may be omitted.

3.1.1 User must be able to establish FTPMaster identity

If a user is unable to confirm the identity of the FTPMasters, the user has no possible way to confirm that any given Release file "signed" by the FTPMaster is in fact signed by that FTPMaster. If that were to happen, the chain of trust is broken.

3.1.2 User must be able to establish Maintainer(s) identities

If a user is unable to confirm the identity of any given Maintainer, the user may see that a Released file has indeed been signed but they have no way of confirming who that person really is. Thus in turn they cannot trust the Released file, and the chain of trust is broken.

3.1.3 All parties except the user must inter-establish identities

This is about building identity trust between Maintainers and FTPMasters. It does not matter if the user is involved in the two-way establishment of trust, because the user is not involved in the Release of files, only in the *receipt* of files (one-way). If however the Maintainers and FTPMasters cannot inter-verify each other's identities then there is no basis on which to carry out statistical confirmation of identity, and thus no way to establish a chain of trust.

3.1.4 All parties must be able to find out if identities have been compromised

Without going into actual implementation details this requirement is quite hard to explain, and has several complex facets. It should however be clear that if any party is compromised, whether their identity key is stolen, or they are kidnapped and forced to sign "Releases" against their will, if that fact cannot be established and its knowledge widely distributed, trust in the entire chain is broken. Thus it is imperative that there exist a means by which all parties may find out, as soon as possible, if an identity has been compromised.

3.1.5 Maintainers must "sign" Releases

This is pretty clear: once a Maintainer's identity is well-known and has been two-way confirmed by other Maintainers and the FTPMasters, "Signing" a Release has meaning. If however the Maintainer does not sign the Release, then there is no way that the Release may be trusted. Maintainers must *only* sign files intended for release, and *only* release signed files, otherwise users may believe that unsigned files can be trusted. and the whole process is moot.

3.1.6 FTPMasters must "sign" Release sets

The importance of this requirement is less clear. If the Maintainers are signing "Releases", why does the FTPMaster have to sign the set of Releases as well? If they do not "sign" a Release set, various problems occur such as being unable to revoke a Release. Also, with the Release "set" effectively being a full collection of all available files, the (signed) set is an important convenient resource. And if users are to make use of that resource, they must be able to trust it.

3.1.7 All parties must secure their identity

This is absolutely critical. If anyone loses their identity key such that another person may fake their identity and perform unauthorised "signing", the entire chain of trust is compromised.

3.1.8 FTPMasters must be able to revoke a "Release"

This is again critical, and is an extension of signing the Release sets. In cases where a Maintainer's identity is compromised, or the Maintainer themselves compromised, or even if there is simply a severe problem with a "Release", is critical that it be excluded from distribution. How may that be achieved if the FTPMaster has never been signing Release sets? By having Release sets, an FTPMaster may create a newer up-to-date Release set which *excludes* the compromised or bad Release. If that is not part of the requirements, a compromised or bad Release will continue to be distributed and the trust and confidence in the whole system would be severely undermined.

3.1.9 A user must be able to verify a Release set, post-download

This is a subtle requirement which provides the user with trust in the FTPMaster and in the confidence of the integrity of a Release set. If the user cannot check that a Release set has been signed by the FTPMasters, the user has no way of knowing if the Release set is legitimate: the CD/DVD or the server from which files are obtained could have been compromised. However it is easy to confuse integrity of the download process (if there is a download) with integrity of the Release set. If the Release set's authenticity may be verified post-download, then the integrity of the actual download process (or offline copying and courier process) is not actually relevant.

3.1.10 A user must be able to verify a file's Maintainer, post-download

This requirement appears to be the sole purpose of the exercise but it can easily be misundersood and believed that it is the *download* (the stream over which the file is obtained) which requires verification, not the file itself. If the file's authenticity may be verified post-download, then the integrity of the actual download process (or offline copying and courier process) is not actually relevant. However if the file's Maintainer's identity cannot be verified *at all* then the chain of trust cannot be established.

3.1.11 No person or resource shall ever become a high-priority target

This is an extremely important requirement. If as part of the implementation details a user or a server has high-value information or data which, if compromised, an attacker could use to undermine the chain of trust, the whole exercise is over. Without going into specific implementation details, examples include placing private keys onto servers (SSL is flawed for this reason) which thus make that server a high-priority target.

However it goes further than that: people should also never become a target, and that means that people need to be inter-changeable (meaning that it is important that identities be two-way verified many times); private identity keys must be kept permanently offline, never connecting them to a networked computer at any time. If this is known publicly to be part of the strict procedures, the motivation for an attacker to target any given individual is removed.

Additionally it means (see "Ubuntu") that key developers should be beholden to no-one. There should not be any way that their safety or financial well-being being threatened would result in the integrity of the Distribution being compromised. Larger number of developers means that a psychologically compromised individual can be replaced. This sounds callous and clinical, but it's important to appreciate that an attacker will be thinking along exactly these kinds of pathologically-objective lines.

3.1.12 All discussion of revocation must be "signed"

If any discussion is not signed, or if this is not part of the standard procedure, then Maintainers and the FTPMasters may get into the habit of having "unsigned" discussions. At that point, an attacker may easily insert fake conversations into the discussion, making people believe that a release or a particular individual's identity needs to be revoked when it does not. In order to avoid this scenario, discussions must be "signed" (and signed only using the same key that has been identity inter-confirmed with multiple other people).

3.1.13 All public announcements regarding releases must be "signed"

Whilst there may be a strong belief that the news announcement infrastructure of an organisattion is secure, this cannot be guaranteed. The only way to be absolutely sure that an announcement has genuinely been made by the FTPMasters is if it has been "signed". If this is not done then an attacker will consider ways to compromise the news announcement infrastructure, making it a high-priority target, and that also violates Requirement 11.

3.1.14 All parties must treat unsigned communications as UNTRUSTED

This is an extension and clarification of the previous requirements. If unsigned communications are not treated as untrusted, an attacker may insert fake communications into any conversation and they risk being accepted as real. This would undermine trust.

There are however a few special extenuating circumstances, such as reports coming in from external sources which are not part of the web-of-trust. Common sense has to be deployed here, assessing whether the identity of the source is genuine and legitimate, and whether the information the source provides may be considered to be both truthful and correct.

3.1.15 Maintainers must check for (and report any new) irregularities (signed)

This requirement applies mainly to their own Releases. In certain circumstances the distributed files may be out-of-date with respect to the current release made and uploaded for distribution by the Maintainer. In such circumstances it is the responsibility of the Maintainer to notify the group (using signed communications only) as to the discrepancy.

If they do not do this, there exist certain circumstances where they may be the only person who knows that the file is out-of-date or otherwise incorrect, and by not reporting the discrepancy they undermine the trust in the entire process. Additionally, the rule "many hands make light work" applies in a statistical manner, so the more Maintainers check other Maintainer's files, and also check the FTPMasters Release set, the less chance of discrepancies.

3.1.16 FTPMasters must check for (and report any new) irregularities (signed)

This is exactly the same as the Maintainers requirement except as applying to the Release set and its associated signing. In all respects it is otherwise identical, including the consequences of not having such a requirement.

3.1.17 The means to perform manual verification must be documented

This requirement is here for several reasons. Firstly, the manual verification process needs to be able to be used in order to confirm that the automated process (if there is one) is correct. Secondly, there are instances and circumstances wwhere installing or using automated tools is not possible.

Not only that, but without a properly documented set of procedures, trust in the whole process is undermined as it may not be audited or reviewed properly. Security being what it is, auditing and transparency is far more important than it first seems.

3.1.18 All identity confirmation must be established using 100% trustable channels

This is a clarification of the early requirements on establishing identities. If any identity is not confirmed using channels that are known to be 100% trustworthy, the entire process breaks down. Specifically, it is imperative that the channel used to obtain files and signatures **not** be that which is used to establish identities.

Face-to-face physical meetings are almost exclusively the only method by which identities can be truly confirmed with 100% confidence. In instances where that is not possible, i.e. where one person is not known to another and there is no practical way for them to physically meet, a chain of "confirmed identity declarations" (known formally as a "web of trust") may be used instead, to ultimately reach at least one person whose identity can be (or has already been) confirmed face-to-face.

4 Other Distros

This section (ongoing) is to include other distros and explain why they are vulnerable or lack integrity, by way of failing to meet one or more of the requirements.

4.1 Ubuntu

Ubuntu's pool of developers behind their web-of-trust keyring is much smaller than that of Debian. Consequently, Ubuntu is a much easier target for compromising developer integrity. Additionally, Ubuntu is funded by Canonical, and the relationship is one where the employer (Canonical) has a psychological hold over the employees (developers). If Canonical wishes to put compromised code into Ubuntu, it may threaten its employees with loss of salary, loss of employment, or reduction in status.

Canonical did at one point compromise end-user privacy by doing a deal with Amazon that shipped local screenshots to Amazon's servers along with search terms [2]. By contrast, Debian's developers are a much larger pool, and they are not employed by a single company.

(TODO: refer to Requirements section violated by the above)

5 Functional Specification

TBD

Notes: must be automated. should be a predictable time after upload when a maintainer may make an announcement about a new version appearing on ftp.gnu.org.

6 Recommended Implementation

TBD

7 Appendix 1: Attack detection and disclosure

IMPORTANT NOTE: there is **no implication** in this document at any point or time that SSL is ineffective at ensuring data **privacy**. Such discussions are not part of this document, and are completely out of scope.

This appendix illustrates scenarios how and where signing may be used to detect that attacks have occurred. It also, as part of the analysis, shows how the use of socket-level privacy (SSL) - whilst it may be effective in guaranteeing data *privacy* - is 100% ineffective in preventing or assisting in the detection of data *integrity* and proof of provenance attacks.

The scenarios however illustrate why it is necessary to have specific procedures in place for detection of attacks (Requirements 15 and 16).

First we define some terms:

GNU/M shorthand for a GNU Maintainer (see Maintainer, above)

FTP/M shorthand for "FTP Master" (again see above)

User a user (again see above)

HTTP general term covering all non-private transfer types (inc. offline)

HTTPS general term covering all SSL-based (more specifically TLS-based as SSL is no longer secure) file protocols including FTP/TLS, as well as other *transparent* privacy channels including offline ones such as secure and 100% trusted physical couriers carrying USB media or CD/DVDs.

rN an FTPMaster's "release" with a specific number (r1, r2 ...)

rBADN an attacker's "release" (100% guaranteed to fail signature checking)

Sig:X a signature's status report (Sig:OK, Sig:FAIL)

MITM a "Man in the Middle" attack

GITM a "Government in the Middle" (attack). Some Governments **REQUIRE** that users install tracking systems which allow them to transparently alter network data. Some Governments do that anyway without consent. Both are covered by this term.

DoS a "Denial of Service" attack (also covers plain-old network failure)

SWITCH a Server attack where an attacker maliciously replaces good data files with bad ones, with the hope of that substitution going undetected for as long as possible

BProxy a Transparent down-stream Proxy over which users have absolutely no control which *unintentionally* serves old files due to poor ISP configuration.

Here is the normal successful scenario, which includes transfer of data:

```
GNU/M .. FTP/M .. r3 .. HTTP .. User .. Sig:OK
```

In this example, the following occurred:

- A GNU Maintainer created a package and signed it
- The FTP Master received the package, created a new release (r3), and signed it
- The FTP MAster uploaded r3 to the server, making it available on HTTP
- A User downloaded r3
- A User performed GPG Signature checking and was able to verify r3 as OK.

Now let us add some attack scenarios:

```
GNU/M .. FTP/M .. r3 .. SWITCH .. rBAD3 .. HTTP .. User .. Sig:FAIL
```

In this scenario, an attacker REPLACED r3 with a BAD release. However, signature checking being offline and inviolate, the user was inconvenienced but able to detect it.

Here are some additional scenarios (all of which assume start with the GNU Maintainer and FTP Master uploading r3, so the chain is shortened, visually):

```
r3 .. SWITCH .. rB3       .. HTTP                     .. User .. Sig:FAIL
r3 .. HTTP   .. MITM      .. rBAD3                    .. User .. Sig:FAIL
r3 .. HTTPS  .. GITM/MITM .. rBAD3                    .. User .. Sig:FAIL
r3 .. SWITCH .. rBAD3     .. GITM/MITM .. rBAD4(!) .. User .. Sig:FAIL
```

In each instance, regardless of the attack used, and regardless of the means and method of download (or upload) in particular regardless of whether SSL was involved in the process, Signature-checking *succeeds* in detecting the attack.

Thus we may logically and formally conclude that **at no time** does the use or deployment of SSL or in fact any form of "privacy" channel added to the system aid us *in any way* in the process of *detecting* if an attack has occurred, nor does SSL or any "privacy" channel (even offline ones) aid us in or add anything to the signature validation process, which on its own - i.e. with *or without* SSL - covers the attack scenarios with a 100% success rate.

There are two scenarios remaining to check. The first is DoS attacks, and the second is a special form of attack where the attacker substitutes *legitimate* previously-signed releases. First, the DoS attack:

```
r3                   .. HTTP                              .. DoS .. User .. FAIL
r3                   .. HTTPS                             .. DoS .. User .. FAIL
r3 .. SWITCH .. rBAD3 .. HTTP                             .. DoS .. User .. FAIL
r3                   .. HTTP  .. MITM      .. rBAD3    .. DoS .. User .. FAIL
r3                   .. HTTPS .. GITM/MITM .. rBAD3    .. DoS .. User .. FAIL
r3 .. SWITCH .. rBAD3 .. HTTPS :: GITM/MITM .. rBAD4(!) .. DoS .. User .. FAIL
```

In each instance, regardless of whether a separate (or the same) attacker has or has not compromised the server and its data, the DoS attack (or plain and simple network failure) may easily be detected by the User: they simply cannot get to resources (compromised or otherwise).

Again: the use or otherwise of HTTPS or other privacy channels adds *nothing of value* as far as detection of failure is concerned. (Note: the reader is invited to apply same exercise to the upload process, and confirm an identical result and conclusion).

Now we come to the special scenarios where attackers attempt to replace data with a *legitimate and previously validly* signed release. Interestingly this scenario is not hypothetical: it can and does actually occur, but not deliberately. Some forms of Transparent HTTP Proxies have sufficiently broken caches that they will preserve files (Release files for example) for longer than they should. There isn't anything "per se" that can be done about this as it's usually the fault of the user's ISP. Here are the scenarios. In this scenario we assume a release "r4" but that, for whatever reason, the user receives "r3":

```
r4 ..                  .. HTTP      .. BProxy .. r3 .. User .. Sig:OK
r4 ..                  .. HTTPS     .. BProxy .. r3 .. User .. Sig:OK
r4 .. SWITCH .. r3     .. HTTP                      .. User .. Sig:OK
r4 .. HTTP   .. MITM                          .. r3 .. User .. Sig:OK
r4 .. HTTPS  .. GITM/MITM                     .. r3 .. User .. Sig:OK
r4 .. SWITCH .. rBAD4  .. GITM/MITM           .. r3 .. User .. Sig:OK
```

In all of these scenarios - which include bizarre ones where one attacker tries to put a bad (guaranteed to fail) release on the server, followed by the NSA or other government carrying out a MITM attack and substituting r3, we may even add to that chain that the NSA's efforts are bizarrely thwarted by an ISP's aberrant Transparent Proxy outside of their control!

All of them result in the user being deceived, *believing* that the release is older than it is. We assume, of course, that the user has no access to other "side-channels" of news such that they are unable to verify this. In the case of an offline media scenario where the User has received a CD/DVD in the mail (compromised through substitution with an older CD/DVD) there is no *real-time* way for them to detect this.

However, once again - before proceeding to the next phase (working out what may be done to deal with the problem) it is absolutely critical to note that, once again, due to the nature of SSL "privacy" channels (including within that *all* other forms of transparent private communications methods), *at no time* was SSL (or other similar method) of assistance to us in the failure to detect or mitigate the substitution.

Another way to explain this is with a Venn Diagram truth table. In this Venn Diagram, GPG and SSL are considered as to their impact on Data Integrity. By "SSL" we mean "all and any methods of providing transparent data **privacy**" which can also include secure off-line couriers. By "GPG" we mean the **full** procedures required to guarantee data integrity and provenance. By "attack vectors" we mean "all and any possible means and methods typically used to compromise network traffic, network-connected devices and all methods of transferring data from device to device". We assume however that both GPG and SSL *themselves* are 100% effective in fulfilling their respective jobs, and are not themselves subjected to attack.

GPG	SSL	Data Integrity	
Used: no	Used: no	not guaranteed	(dozens of attack vectors)
Used: no	Used: yes	not guaranteed	(several attack vectors still possible)
Used: yes	Used: no	guaranteed	(as long as procedures are followed)
Used: yes	Used: yes	still guaranteed	(because of GPG... not SSL)

From this diagram we can see that "Data Integrity" is true when GPG is used, and "Data Integrity" is false when GPG is not. SSL has *no influence* on the outcome. Thus we may logically and formally declare that in all cases SSL and other *transparent* privacy methodologies are 100% ineffective in *assisting* with the job of provably verifying the provenance and integrity of the data.

This just leaves us with the task of coming up with a series of procedures which *can* be successful in detecting the deliberate (or accidental) substitution of data (that we know *will* pass signature checking).

Questions:

- Where can the detection actually take place such that old files are noticed?
- Who is guaranteed to have copies of the correct files?

These simple questions provide us with the answer: it is the Maintainer and more specifically the FTPMaster who will have last seen - and knows that it is the case - the correct file(s). Thus it is the responsibility of Package Maintainer(s) to report that their package has not reached the server(s), and it is the responsibility of the FTPMasters to report that their Release file has not reached the server(s).

For this disclosure to work, it is necessary that the Maintainer(s) and FTPMasters make formal side-channel public announcements using the exact same signatures as utilised to sign their respective file(s) and Release(s). A mailing list would be sufficient for this task, however it should not be the only method (as that could be DoS'd or simply down as well).

For those people without real-time online secure communications, for whom offline media is the sole method of receiving Signed Releases, unfortunately there really is nothing that can be done for them. They should establish alternative equivalent verification channels which achieve the same outcome (verification of provenance and that the Release is up-to-date) but advising what exact methods they should use remains outside the scope of this document.

The last thing that needs to be said is that whilst it is technically possible to have expiry dates on Signatures, such that any given Release or FTPMasters Release could be detected to have expired, with a view to at least *forcing* people to keep up-to-date with Releases, the problem is in the use of the word "force". It has the unintended side-effect of making it extremely difficult to use old archives of Releases, as well as adding an unnecessary burden on Users (requiring constant network access). With side-channel procedures being effective (and already established and in use), the forced reliance on constant network access and updating of the FTPMaster Release file is not even necessary.

In summary and conclusion:

- Signature Checking, which is a formal method of provably making identity-backed declarations of integrity, is the fundamental bedrock for this exercise;
- If real-time network access is not available (DoS'd), side-channels and procedures using identity-backed (signed) communications MUST be activated to make that fact known widely and quickly;
- Forcing users to continuously obtain a non-expired Release File is neither practical nor necessary.
- Offline users must come up with their own equivalent (as best they can) of real-time (identity-backed) verification procedures;
- Maintainers and FTPMasters being cut off from all real-time internet communications is a catastrophic failure scenario well beyond the scope of this document;
- At no time does SSL or other "transparent" privacy channel practice or procedure aid, assist or usefully feature in the detection, attack mitigation or recovery processes associated with proof of data *integrity* and provenance verification.

References

[1] http://en.wikipedia.org/wiki/Digital_signature.

[2] https://www.omgubuntu.co.uk/2016/01/ubuntu-online-search-feature-disabled-16-04.

[3] https://www.gnu.org/prep/maintain/maintain.html#Automated-FTP-Uploads