Tails IUK and TUF integration

Green Team

Toan Nguyen

Kevin Ngao

Cesar Murillas

Dominic Spinosa

NYU-Poly

Fall - 2013

Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generating
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generating
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

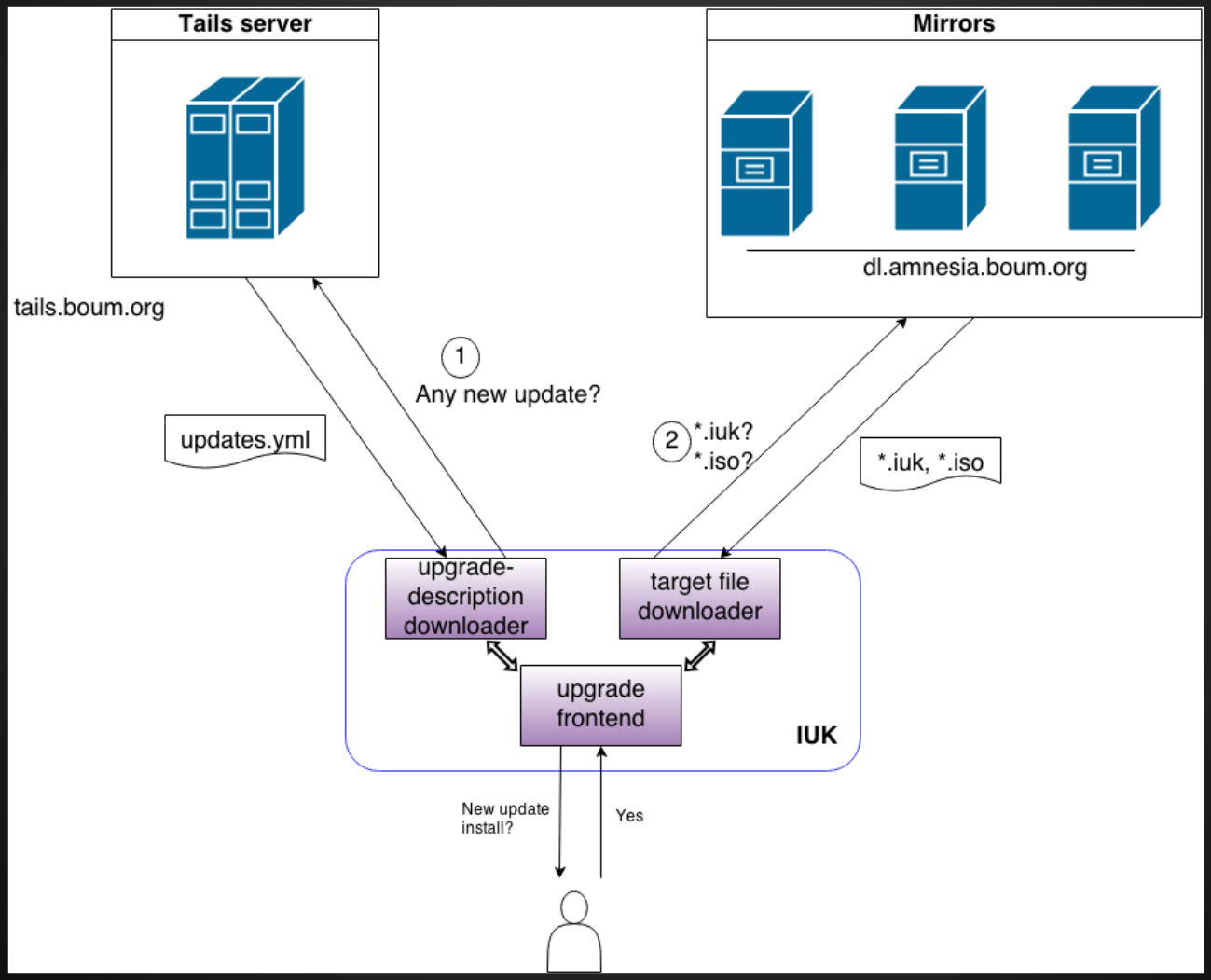
Tails OS

- Tails (The Amnesic Incognito Live System):
  - Live Linux OS system that can be run from a DVD, USB stick, or SD card and aims to preserve users privacy and anonymity
  - Built-in applications pre-configured with security in mind and run through Tor
  - Primary goal is to reduce/remove trace of user's identity and origin when browsing the Internet (eliminate "fingerprint")

Tails IUK

- Incremental Upgrade Kit: Tails' upgrade system for OS, not for packages/apps
- Has been developing with security in mind
- Prevent against several attacks: rollback, arbitrary package, replay, mix-and-match, endless data, freeze, extraneous dependencies attacks
- Document: https://tails.boum.org/contribute/design/incremental_upgrades/

How IUK works



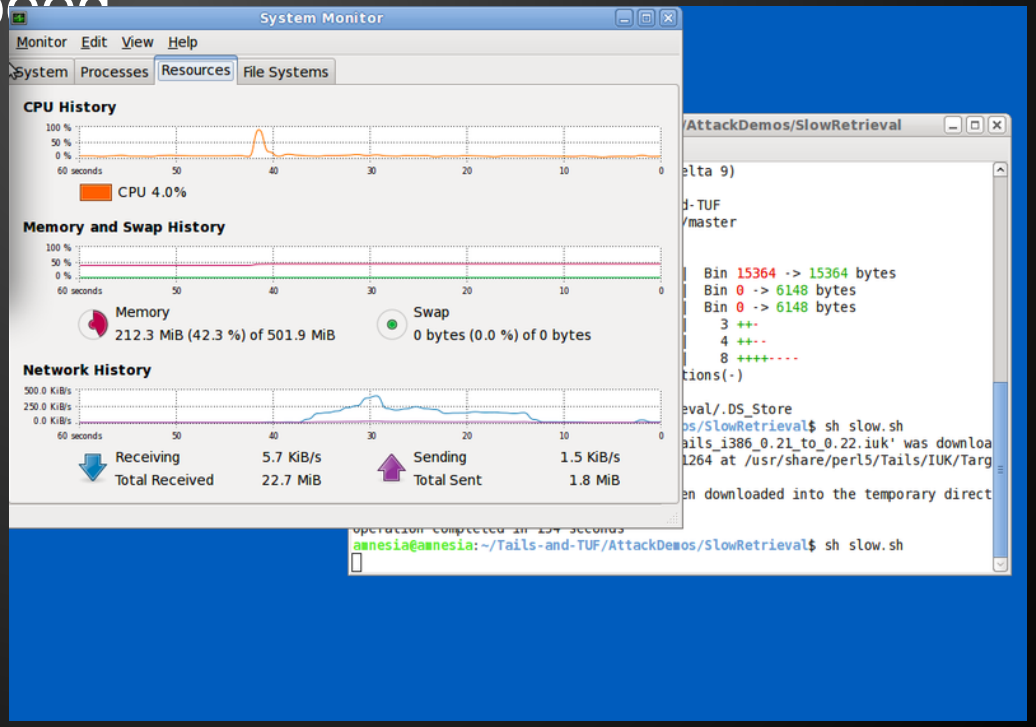
Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generating
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Slow Retrieval

- Slow Retrieval
  - Currently no check to ensure a minimum connection speed
  - Proven by throttling server to deliver update at an incredibly slow speed

IUK kept downloading forever



Key revocation - Key compromise

- Tails implements a single-key PGP scheme
- Compromise of key leads to compromise of Tails' update system
- Tails developer ignored key expired, key revocation => attack is feasible

```
method verify_signature ($description, $signature) {
    my $gnupg = GnuPG::Interface->new();
    $gnupg->options->hash_init(
        homedir    => $self->trusted_gnupg_homedir,
        # We decide what key should be trusted by a given Tails,
        # and we won't put a key created in the future in there,
        # so if a key appears to be created in the future,
        # it must be because the clock has problems,
        # so we can ignore that.
        # Same for a key that appears to be expired.
        # Disable locking entirely: our GnuPG homedir is read-only.
        extra_args => [
            qw{--ignore-valid-from --ignore-time-conflict --lock-never}
        ],
    );
```

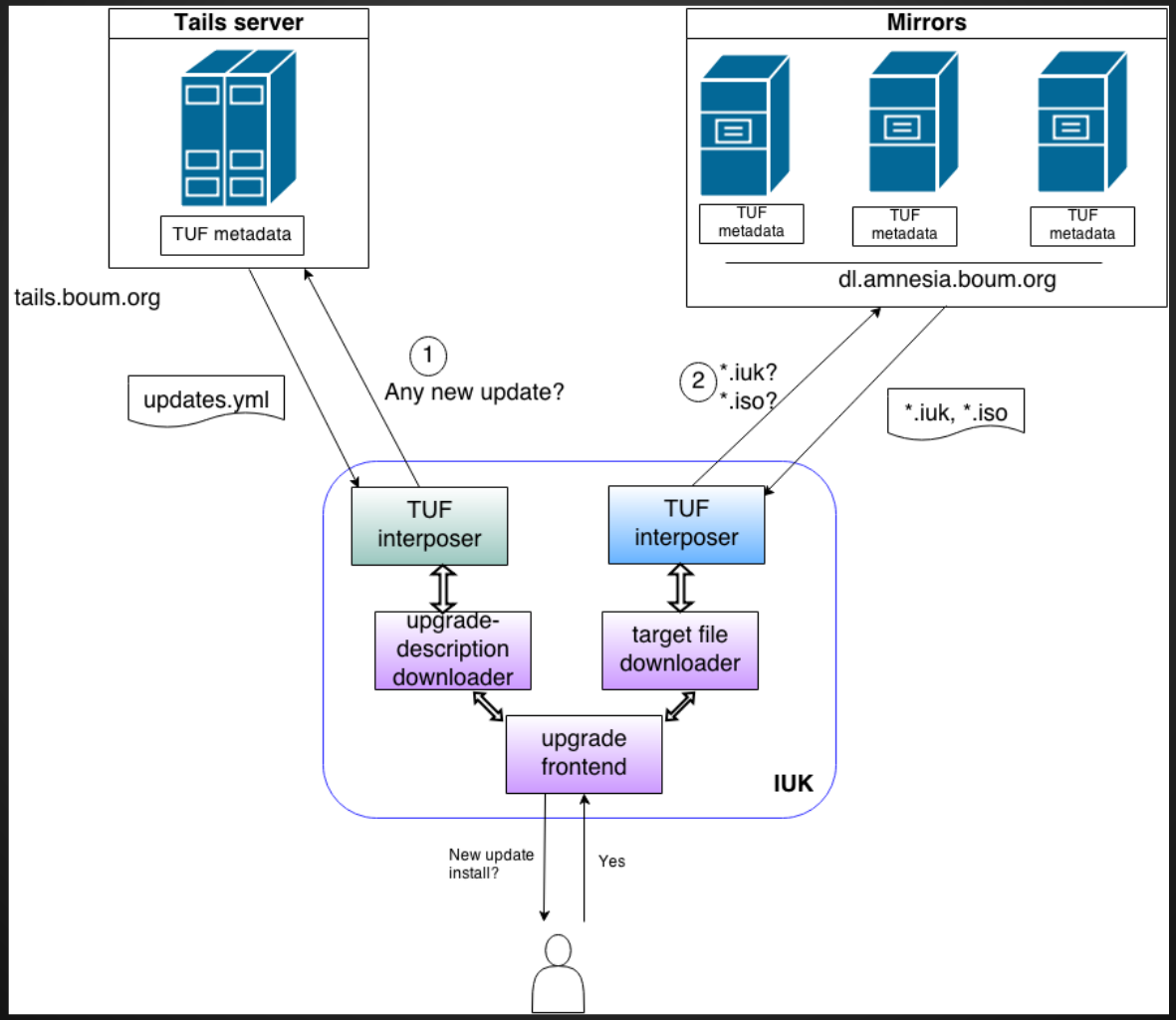
Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generating
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Installation of TUF into Tails

- Tails 0.21: Python 2.6
- From TUF source: really difficult since Tails eliminated dependencies. Not a good OS for developer
- We struggled with these problems for days!
- We made a suite of TUF binary and all of its dependencies that can be installed onto Tails with a single command!

Integrate TUF with IUK



How they work together

- User fires up upgrade process by calling Upgrade frontend with command _tails-update-frontend_
- Upgrade frontend will call Upgrade-description downloader to request a new upgrade, available from Tails' server.
- Upgrade-description downloader will forward the request for downloading the metadata file updates.yml to TUF interposer
- TUF interposer will go ahead and download updates.yml, then return it back to the Upgrade-description downloader, which then returns it to Upgrade frontend
- Upgrade frontend parses the file, checks whether it is a new upgrade and presents it to the user, and lets them decide whether to upgrade or not.
- If user decides to upgrade, Upgrade frontend will call Target file downloader to download the targets files
- Target file downloader will now forward this request to TUF interposer
- TUF interposer will download target files from Tails' mirrors, upon completion it will return the file to Target file downloader, which then returns it to Upgrade frontend
- Upgrade frontend will take necessary steps to upgrade the system

TUF Interposer for Update-description downloader

```
'''
download.py
Description: This file will be called by the UpdateDescriptionFile/Download.pm
to actually download the updates.yml from Tails' server, and return downloaded
file to UpdateDescriptionFile/Download.pm
'''

import tuf.interposition
from tuf.interposition import urllib2_tuf as urllib2
import sys
url = sys.argv[1] #url to download from
tufconfig = tuf.interposition.configure("/usr/share/perl5/Tails/IUK/tuf.interposition_meta.json")
try:
    response = urllib2.urlopen(url)
    print "HTTP/1.1",response.getcode(),"OK" #success
    print response.info() #then print the header
    print '\n'  #separate between header and content
    print response.read() #and the content of updates.yml
#Otherwise, return error code
except urllib2.HTTPError, e:
        print e.code
except urllib2.URLError, e:
        print e.args
finally:
        tuf.interposition.deconfigure(tufconfig)
```

TUF Interposer Configuration (UDD)

```
{
  "configurations": {
    "tails.boum.org": {
      "repository_directory": "/usr/share/perl5/Tails/IUK/tuf-metadata",
      "repository_mirrors" : {
        "mirror1": {
          "url_prefix": "http://www.toannv.com/tuf",
          "metadata_path": "metadata",
          "targets_path": "targets",
          "confined_target_dirs": [ "" ]
        }
      }
    }
  }
}
```

TUF Interposer for Target file downloader

```
'''
download.py
Description: This file will be called by the TargetFile/Download.pm
to actually download the *.iuk or *.iso from the update server then the
downloaded file will be piped into TargetFile/Download.pm
'''

import tuf.interposition
from tuf.interposition import urllib2_tuf as urllib2
import sys
url = sys.argv[1]
#tuf.log.add_console_handler()
config = tuf.interposition.configure("/usr/share/perl5/Tails/IUK/tuf.interposition_target.json")
try:
    response = urllib2.urlopen(url)
    print "HTTP/1.1",response.getcode(),"OK" #success
    print response.info() #then print the header
    print '\n' #separate between header and content
    tmpFilename = "/tmp/" + url.rsplit('/',1)[1] + ".tmp"
    f = open(tmpFilename,'wb')
    f.write(response.read())
    f.close()
    print  tmpFilename #return filename of the target file downloaded by TUF
#Otherwise, return error code
except urllib2.HTTPError, e:
        print e.code
except urllib2.URLError, e:
        print e.args
finally:
    tuf.interposition.deconfigure(config)
```

TUF Interposer Configuration (TFD)

```
{
  "configurations": {
    "tails.boum.org": {
      "repository_directory": "/usr/share/perl5/Tails/IUK/tuf-metadata",
      "repository_mirrors" : {
        "mirror1": {
          "url_prefix": "http://www.toannv.com/tuf",
          "metadata_path": "metadata",
          "targets_path": "targets",
          "confined_target_dirs": [ "" ]
        }
      }
    }
  }
}
```

Modification of Tails IUK scripts: Perl-Python piping

- IUK was written in Perl
- TUF was written in Python
- We need to make Perl script calls Python script (TUF interposers) in its running
- Solution: Piping!
- Let Perl script call TUF interposers for downloading, downloaded files will be piped back to Perl

Piping for Update-Description Downloader

```
method get_url ($url) {
    my $ua  = LWP::UserAgent->new();
    unless ($ENV{HARNESS_ACTIVE} or $ENV{DISABLE_PROXY}) {
        $ua->proxy([qw(http https)] => 'socks://127.0.0.1:9062');
    }
    $ua->protocols_allowed([qw{https}]);
    $ua->max_size($self->max_download_size);
    #my $res = $ua->request(HTTP::Request->new('GET', $url));
    my $command = "python /usr/share/perl5/Tails/IUK/UpdateDescriptionFile/download.py " . $url;
    my $content = `$command`; #force downloading goes through TUF
    my $res = HTTP::Response->parse($content); #Construct a new HTTP::Response object from returned file by TUF

    defined $res or croak(sprintf(
        "Could not download '%s', undefined result", $url
    ));
    ...
```

Piping for Target File Downloader

```
method run () {
    ...
    $ua->max_size($self->size);
    #my $res = $ua->request($req, $temp_filename);
    my $command = "python /usr/share/perl5/Tails/IUK/TargetFile/download.py " . $self->uri;
    my $content = `$command`; #force downloading goes through TUF
    my $res = HTTP::Response->parse($content); #Construct a new HTTP::Response object from returned file by TUF
    $temp_filename = $res->decoded_content(); #Get the filepath of downloaded target file
    $temp_filename =~ s/\R//g; #Cut off the newline character at the end of the filename
    defined $res or clean_fatal($self, $temp_filename, sprintf(
        "Could not download '%s' to '%s': undefined result",
        $self->uri, $temp_filename,
    ));
    ...
```

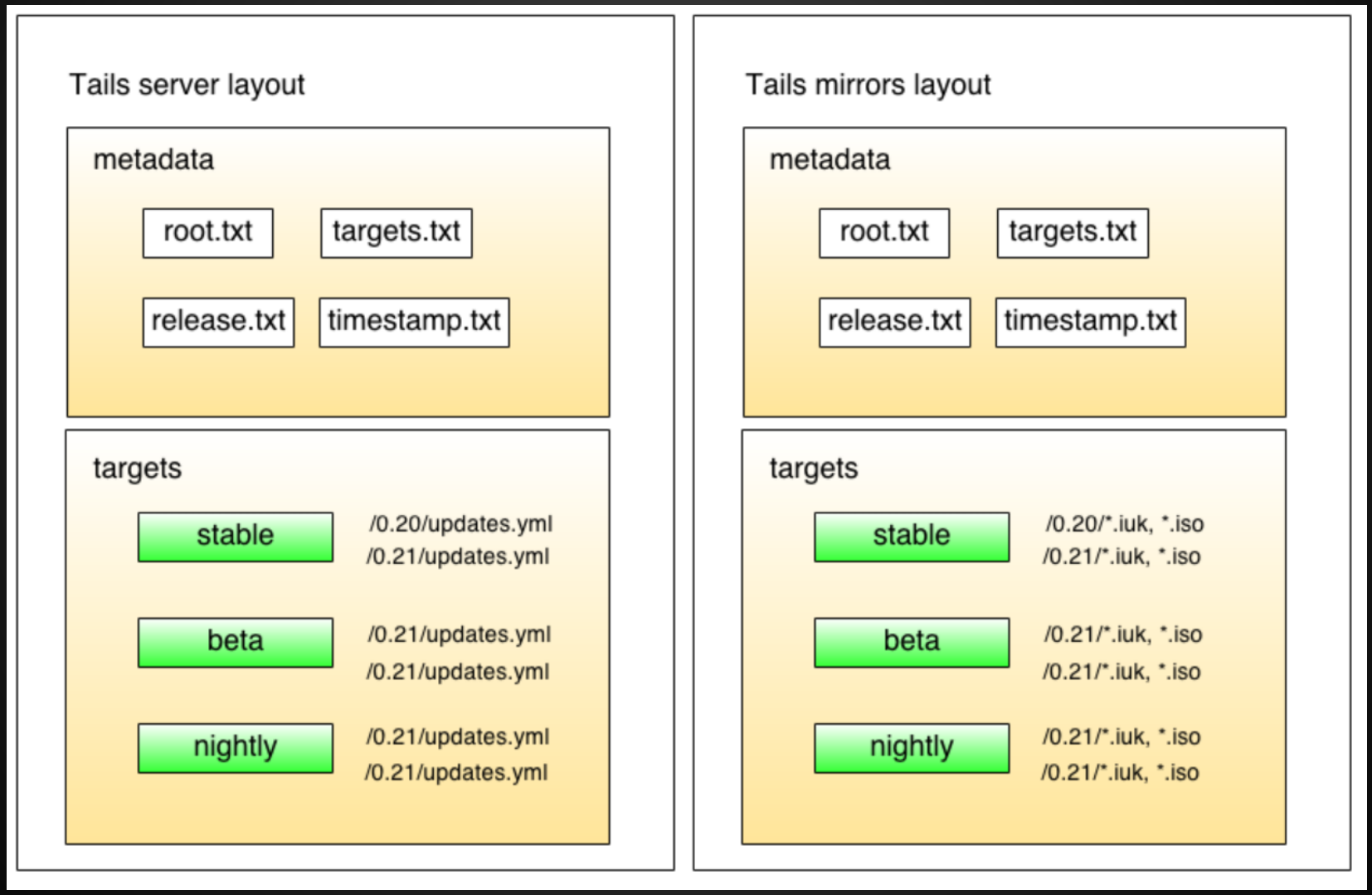
Changes in Update Frontend

Change in Frontend.pm:

Run Target File downloader with normal user

```
        $self->fatal_run_cmd({
            cmd       => \@cmd,
            error_msg => $self->encoding->decode(errf(
                gettext(
                    q{Failed to download from %{target_url}s: }.
                    q{<a href='file:///usr/share/doc/tails/website/doc/upgrade/error/get_target_file.en.html'>read
                ),
                {
                    target_url => $target_file->{url},
                }
            )),
**change here ===>** #as        => 'tails-iuk-get-target-file', #I need to comment this out so that my TUF interpo
        });
   ...
```

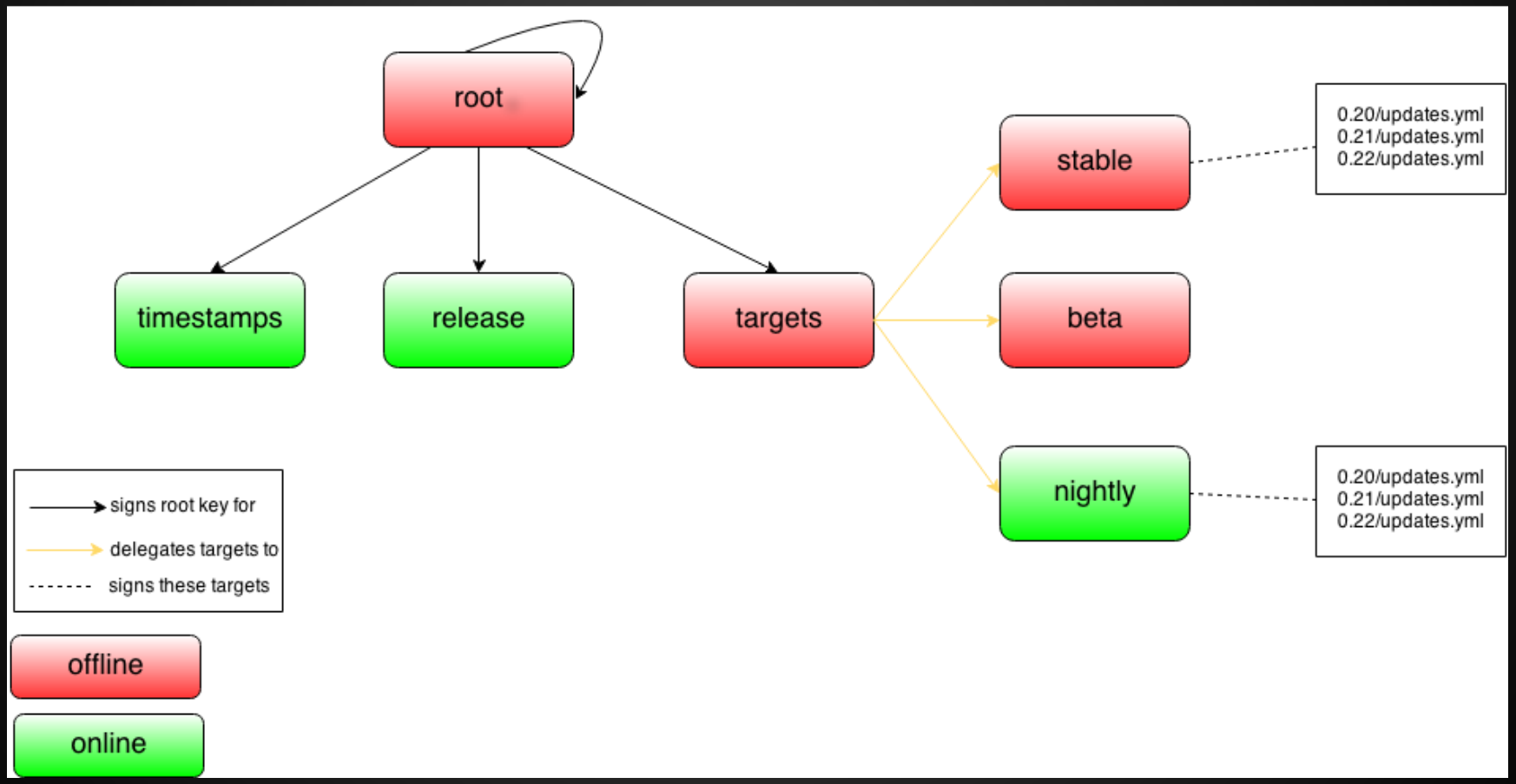
Server layouts



Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generation
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Key Management Design



Key Management

Delegated Roles - The *targets* role will delegate three roles in order to maintain the three separate release channels:

- *stable* - Signs for targets.txt in stable folder whenever there is a new target added/updated in stable channel
- *beta* - Signs for targets.txt in beta folder whenever there is a new target added/updated in beta channel
- *nightly* - Signs for targets.txt in nightly folder whenever there is a new target added/updated in nightly channel. Key kept online to accommodate frequent releases.

Online Keys:

- *timestamp* - Frequent signing needed to keep a "fresh" timestamp
- *release* - Frequent signing needed for multiple upgrade releases through the use of the three delegated channels/roles
- *nightly* delegated target key can be kept online to accommodate frequent releases.

Online keys can be stored on different servers and a key threshold greater than one can be used to reduce the risk of key compromise.

Offline Keys

- *root* and *target* keys are kept offline in order to avoid compromise and subsequently increase key management security.
- *stable* and *beta* delegated target keys also be kept offline due to infrequent releases.

Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generating
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Generation of Metadata

- Three scripts designed to be used together to generate TUF metadata for the first time
- Main script setup.py does the work for you
  - Fill out configuration files accordingly.
    - Instructions in Wiki (Metadata Generation)
- Target File Modification as well
  - Automatically changes out targets/role folder with new files in case of update
  - Changes metadata accordingly
  - Two scripts with a main script putting it all together
    - Fill out configuration files accordingly (See Wiki)

Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generation
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

TUF Metadata management

- *Efficiency*: Our Metadata files are small (less than 10KB)
- *Usability*: Automated scripts made generation of metadata very easy and straightforward for Tails admin
- *Security*: By requiring separate keys for the designated roles and requiring critical keys to remain offline, key compromise can be mitigated. It's easy to do key revocation with TUF
- Consistency: The solution is to place a hash in the filename. When the file download is complete, check the hash against the hash in the filename to ensure consistency. This needs to be supported and implemented by TUF (which currently isn't).
  - Current situation: Send 404 to downloading clients

Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generating
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Mitigated Attacks with TUF

- Slow Retrieval
  - TUF detects the slow down, and times out the connection
- Key Revocation
  - TUF can revoke and generate new keys in the event of a key compromise
  - TUF can detect expiration of keys implicitly or explicitly

Outline

- Overview of Tails IUK
- Attacks on Tails IUK without TUF
- Integration of TUF with Tails IUK
- Key management
- Metadata generation
- Metadata management
- Attacks on Tails IUK with TUF
- Conclusion

Conclusion

- Demonstrated possible attacks to IUK
- We have showed how to integrate Tails IUK with TUF
- Showed that IUK with TUF can mitigate slow retrieval and key compromise/key revocation attacks. Enhanced IUK security
- Little efforts for the admin when work with TUF metadata generation and management
- Key management for IUK-TUF

Thank you

Q&A

Contact

- Toan Nguyen (tvn214@nyu.edu)
- Kevin Ngao (kgn207@nyu.edu)
- Cesar Murillas (ctm285@nyu.edu)
- Dominic Spinosa (djs617@nyu.edu)

- Our GitHub repo: https://github.com/muri11as/Tails-and-TUF
- Our Wiki: https://github.com/muri11as/Tails-and-TUF/wiki