Security issues in hierarchically connected BOINC systems

Gábor Gombás
MTA SZTAKI

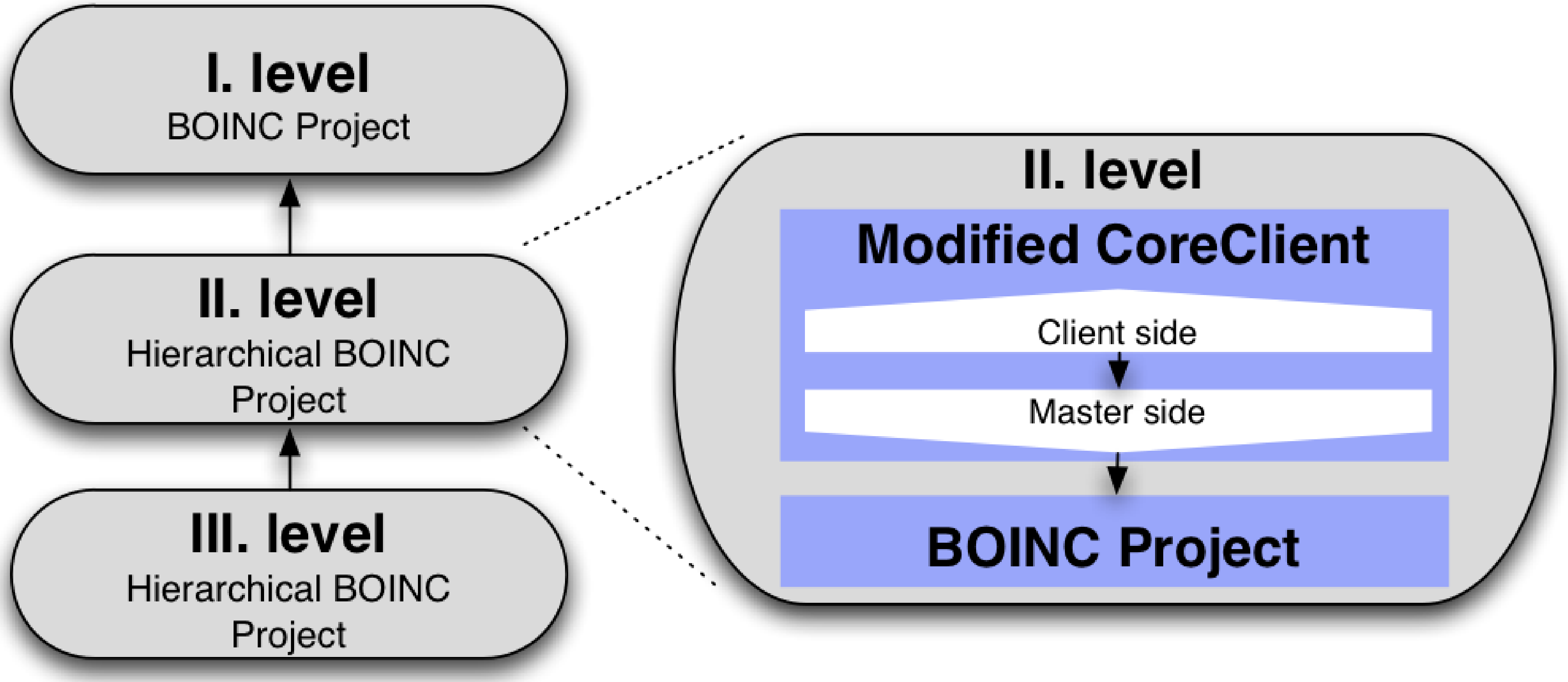
<gombasg@sztaki.hu>

Introduction

- BOINC mainly focuses on big, stand-alone, public projects
- At SZTAKI we're looking into how to use BOINC for smaller, more localized setups
  - Universities and enterprises
- This brings new areas of problems to solve
  - Interactions between projects
  - Different security criteria (data protection etc.)

Hierarchy

- Hierarchy mainly targets enterprises/institutions that already have a hierarchical organizational structure
- Hierarchical setup allows aggregating LDGs with keeping the administrative overhead low



Use Case

- Company support for public desktop grids
  - Motivation: good for PR
  - Problem: strong supervision is needed for what the resources are used for
    - Employees should not be able to alter the settings dictated by the management
  - Solution: local desktop grid (managed by the company) joins the public DG
    - The local DG can have strict rules about participation and usage

Security Model of BOINC

- Uses asymmetric key pairs
  - One key for application signing (code signing)
  - One key for *workunit* signing
- Applications are signed by the Project
  - The keys usually are kept at a separate physical location, so the signing process is always manual
- Workunits are signed by the Project
  - The keys reside inside the project, so the signing can be automatic
- Communication via HTTP by default
  - But clients are prepared for HTTPS

New Requirements

- Automatic application deployment
  - Applications originating from a higher level should be deployed automatically at the lower levels
  - This creates new trust relations between the DGs
- Extended trust relation between the client and the project/server/application
  - Based on application origin, type etc.
- Data protection
  - On the server side: disallowing unknown/untrusted clients
  - Data encryption
- Extended client protection
  - Sandboxing using virtual machines

Some Scenarios

- The *User* wants to trust the *workunits* originating from the *Project* she is connected to
  - This is the original trust model
  - *User* is the operator of the *Client* machine
- The *User* wants to trust any *workunits* coming from the *Project*, regardless how many levels of hierarchy it has travelled
- The *User* wants to trust a specific *Application*
  - regardless where it is hosted, and regardless what other applications the project has

Extending the Security Model

Common roles:

- *Application Developer*
  - A group or Individual who develops a specific application
  - Signs application code (code signing)
  - Developers are trusted, not application code
- *Server*
  - Hosts one or more Project
  - Signs the workunits
- *Project*
  - Administrative body of BOINC
  - Authenticates clients
- *Client*
  - Administered by the *User*

Extending the Security Model

- Trust relationship is implemented using signature checking
  - Every application comes with a set of signatures from entities who have authorized its use (app. developer, project, institute etc.)
  - Every client has a set of accepted certificates
  - An application is allowed to run if the intersection of the above sets is not empty
- We needed a PKI for managing the signing process – we've chosen X.509

App. Signing Using X.509 Certificates

- Attila Marosi @ SZTAKI implemented the capability to sign applications using X.509 certificates instead of a bare RSA key
- The code has been committed to the trunk at the 4th September
- Documentation is available at http://boinc.berkeley.edu/trac/wiki/CertSig
- A more detailed description can be found in the Coregrid technical report TR-100, available at http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0100.pdf

Other Uses of X.509

- X.509 certificates can also be used at other places to provide extra security
  - Using HTTPS instead of plain HTTP to provide data protection
  - Using client certificates in addition to server certificates if password-based security is not enough (this can be a requirement in corporate environments)

Sandboxing

- BOINC already contains code to run applications under a restricted account
- Sometimes this is not enough
- As a joint research between SZTAKI, INRIA and IN2P3 we've experimented with using virtual machines
  - VM images are big – create them on the spot
    - Distribute a base image, and inject the input files on the client
    - Further ideas: use an embedded Linux distro instead of a desktop/server one (dietlibc, uClibc if possible)
  - Either some software that can plug into the kernel has to be installed on the client or it will be slow
  - Extended resource usage, more expensive checkpoints

Other Issues

- Using software like BOINC in a corporate environment may present other problems
  - Saying “the web interface uses PHP” can make corporate system administrators jump
  - Separating BOINC components on the server side to run under different accounts or use different database credentials can be tricky
  - It's very different than the default way BOINC operates

Conclusion

- Mixing the usage of local/global desktop grids requires extending the security model
- SZTAKI does research on the possible solutions
  - Certificate-based authentication
  - VM technology
- Some use cases require even more modifications that may not be applicable to mainstream BOINC

Thanks!

Questions?