HTCondor Security Basics
HTCondor Week, Madison 2016

Zach Miller (zmiller@cs.wisc.edu)

Center for High Throughput Computing
Department of Computer Sciences
University of Wisconsin-Madison
Overview

» What are the threats?

» Who do you trust?

» What are the mechanisms?

» Other security concerns?
The purpose of HTCondor is to accept arbitrary code from users and run it on a large number of machines.
The purpose of HTCondor is to accept arbitrary code from users and run it on a large number of machines.

The purpose of a botnet is to take arbitrary code and run it on a large number of machines.
So what’s the difference?

You wish to prevent unauthorized access

Ultimately, it just comes down to who can use your pool, and how they can use it.
Basic Concepts

› “Who can use your pool” is really two concepts:

› The “Who” is authentication

› The “can use” is authorization
Basic Concepts

› Authentication is finding out WHO some entity is.

› How is this done?
  • Common methods:
    • Present a secret that only you should know
    • Perform some action that only you can do
    • Present a credential that only you could have
Basic Concepts

› Authorization is deciding what someone is allowed to do.

› You must know who they are before you can decide this!
Basic Concepts

› I’m using “they” pretty loosely here.

› “They” could be:
  • A user
  • A machine
  • An agent/daemon/service
Basic Concepts

› In the context of an HTCondor pool:
  • You want only machines that you trust to be in the pool
  • You want only people you trust to submit jobs
Assumptions of Trust

› HTCondor relies on trusting the “root” user of a machine
› If this is compromised, all bets are off

› HTCondor daemons trust each other

› You need to trust your friendly HTCondor administrator
Assumptions of Trust

› How about users?

› HTCondor places some restrictions on users:
  • zmiller cannot submit, remove, or manipulate jobs belonging to another user

› But ”bad” users can still cause problems
  • Running fork bomb: while(1) { fork() }
  • Intentionally interfering with the system
So, users are trusted to some degree

Preventing every possible bad behavior makes the system too cumbersome for good users

- Security is always a balancing act with usability

Decide how much you want to prevent versus punish
SUBMIT_REQUIREMENT allows the administrator to restrict what jobs are able to enter the queue

Can be used to prevent users from lying about what groups they belong to:

```
SUBMIT_REQUIREMENT_NAMES = GROUP1

SUBMIT_REQUIREMENT_GROUP1= (AcctGroup =!= “group1”) ||
                          (AcctGroup =?= “group1” && (Owner==“zmiller” || Owner==“tannenba”))

SUBMIT_REQUIREMENT_GROUP1_REASON=“User not in group1”
```
Restricting Users

› SUBMIT_REQUIREMENT allows the administrator to restrict what jobs are able to enter the queue

› Can be used to allow only certain executable files, number of CPUs requested for a job, anything else that is part of the Job ClassAd
Authentication

- When users submit jobs, HTCondor authenticates them.

- The HTCondor SCHEDD daemon now “owns” the jobs, and acts on their behalf.
Authentication

› So how can we trust the SCHEDD?

› Daemon-to-daemon authentication
Authentication

› For a secure pool, both users and HTCondor daemons must authenticate themselves

› HTCondor supports several mechanisms:
  • File System
  • Password
  • Kerberos
  • SSL
  • GSI
Other Security Mechanisms

› In addition to authenticating network connections, you may also wish to use:

› Integrity Checks (MD5)
  • Allows HTCondor to know if traffic has been tampered with

› Encryption (3DES, Blowfish)
  • Allows HTCondor to transmit encrypted data so it cannot be spied on while in transit
Example “Strong” Configuration

SEC_DEFAULT_AUTHENTICATION = REQUIRED
SEC_DEFAULT_AUTHENTICATION_METHODS = Kerberos
SEC_DEFAULT_ENCRYPTION = REQUIRED
SEC_DEFAULT_INTEGRITY = REQUIRED
Security Negotiation

When first contacting each other, HTCondor daemons have a short negotiation to find out which mechanisms are support and what features are required for the connection.

I want to submit a job

You must authenticate w/ kerberos

KERBEROS

normal submit protocol
Security Negotiation

› Policy Reconciliation Example:

**CLIENT POLICY**
- SEC_DEFAULT_ENCRYPTION = OPTIONAL
- SEC_DEFAULT_INTEGRITY = OPTIONAL
- SEC_DEFAULT_AUTHENTICATION = OPTIONAL
- SEC_DEFAULT_AUTHENTICATION_METHODS = FS, GSI, KERBEROS, SSL, PASSWORD

**SERVER POLICY**
- SEC_DEFAULT_ENCRYPTION = REQUIRED
- SEC_DEFAULT_INTEGRITY = REQUIRED
- SEC_DEFAULT_AUTHENTICATION = REQUIRED
- SEC_DEFAULT_AUTHENTICATION_METHODS = SSL

**RECONCILED POLICY**
- ENCRYPTION = YES
- INTEGRITY = YES
- AUTHENTICATION = YES
- METHODS = SSL
I’m going to skip the detailed configuration of each particular security mechanism.

Security is not one-size fits all

If you are interested in details, please schedule some “office hours” with me to discuss.
Configuration Security

› Are your condor_config files secured?

› They should be owned and only modifiable by root.

› If you use a config directory, make sure only root can create files in it
HTCondor can allow configuration changes using a command-line tool:

- `condor_config_val --set Name Value`

However, this behavior is off by default and needs to be enabled on a case-by-case basis for each config parameter… use carefully only if you really need it.
HTCondor Privilege

› HTCondor typically runs “as root”

› Why?
  • Impersonating users
  • Process isolation
  • Reading secure credentials

› When it isn’t actively using root, it switches effective UID to another user (“condor”)
HTCondor Privilege

- HTCondor will never launch a user job as root. There is a “circuit breaker” at the lowest level to prevent it.

- If not using system credentials, the Central Manager can run without root priv

- Let’s examine some different Startd configurations
StartDs have a few different options for running jobs:

- Run jobs as the submitting user
- Run jobs as the user “nobody”
  - Allows jobs to interfere with one another
- Run jobs as a dedicated user per slot
  - Keeps jobs running as a low-privilege user
  - Isolates jobs from one another
  - Makes it easy to clean up after a job
glexec

› Allows HTCondor daemons to be run without root privilege, yet running jobs can still assume the UID of the submitting user

› Uses GSI credentials to authenticate

› Very useful for glidein jobs
Even if that admin has not required encryption for all network connections, user jobs can specify per-file for both input and output if the files should be encrypted:

- Encrypt_Input_Files = file1, *.dat
- Encrypt_Output_Files = data.private
Encrypt Execute Directory

› If you are using Linux with *ecryptfs* installed, you can have HTCondor encrypt the execute directory on disk, offering extra protection of sensitive data.

› Can be enabled pool-wide by the admin:

  • `ENCRIPT_EXECUTE_DIRECTORY = True`

› Per-job in the submit file:

  • `Encrypt_Execute_Directory = True`
Vulnerabilities

› HTCondor has been assessed by an independent research group.

› That was many years ago. Another audit will be coming “soon”

› Our vulnerability reporting process is documented and vulnerability reports publicly available:

Schedule “office hours” this week

Email the htcondor-users mailing list and if your question is security related I will (likely) respond

Email me directly