

Securing Munki

Tom Bridge, Technolutionary LLC
@tbridge

MacDevOps YVR, June 21 2016

This presentation is here to help you understand a bit more about Munki's underpinnings and how it operates, how it exchanges information and how it can be hardened to protect your environment.

Once upon a time when I was young and stupid last year, I wrote a piece of software called Munki in a Box. How many people have seen it or run it?

Directions for Use:

- 1) Download Script
- 2) Alter Lines 20-21 to reflect your choice for m
- 3) Alter Line 32 to reflect your choice of AutoPk
- 4) Alter Line 35 to reflect your admin username
- 5) Alter Lines 37-38 to reflect AutoPKG Automa
- 6) sudo ./munkiinabox.sh

Bad Ideas.

- 1) Download Script
- 2) Alter Lines 20-21 to reflect your
- 3) Alter Line 32 to reflect your cho
- 4) Alter Line 35 to reflect your adm
- 5) Alter Lines 37-38 to reflect Auto
- 6) sudo ./munkiinabox.sh

Bad Ideas.

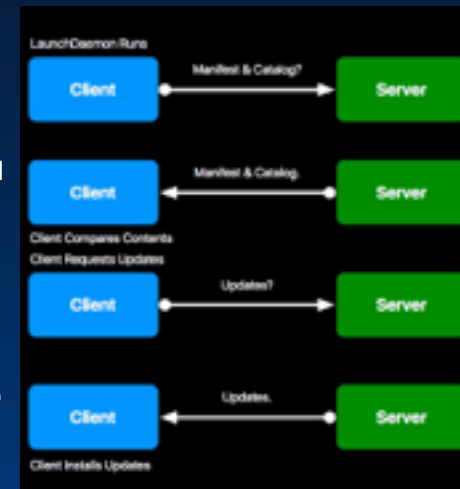
```
6) sudo ./munkiinabox.sh
```

Bad Ideas.

Sorry.

The Munki Transaction

- Client launchdaemons run hourly
- Client retrieves manifest(s), catalog(s), from the server, and compares them against client.
- If necessary, client retrieves pkg(s) and pkginfo files from Server for install
- Client installs pkgs and pkginfo files (scripts) as root from login context, or loginwindow.



The basic munki interaction, where a client talks to a web server to retrieve a manifest, a catalog and some packages, and then executes them is designed to be done between trusted sources.

Munki installs updates as root, which allows it to operate separate from non-admin user contexts with the permissions necessary to install tasks. This is the exchange that we make: we give an application root privileges so it can execute trusted tasks for our benefit.

Let's talk about that trust.

Transaction Types

Office Configuration

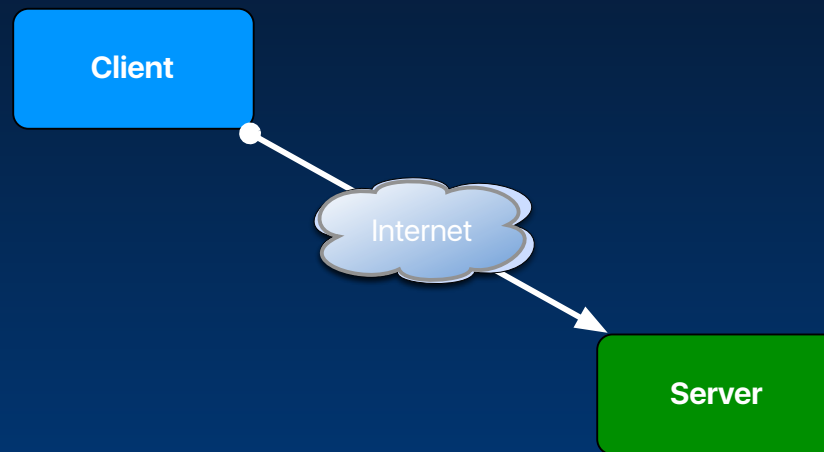


This is the kind of connection that Munki in a Box is written for. It's written for an HTTP transaction over a local network for machines that never leave their own network.

That's a pretty limited environment. How many of you have networks like this?

In this particular situation, how you handle your traffic is less important than good repository maintenance.

Transaction Types



This might be a more common configuration. Some clients connect over the LAN, while other clients (laptops) roam freely, and use the VPN to call home for updates. Seems like a good plan.

But, if you're not using SSL Certificates, you're subject to the Man in the Middle attack.

Man in the Middle



This is what you're worried about first. This is Malcolm, he's our man in the middle today. Malcolm is the unscrupulous sysadmin at Jim Thornton's coffee shop in downtown Calgary, and he's setup a network to watch for URL paths that look like munki manifests and catalogs so that he can hijack the traffic from those URLs and point them at his own properly structured, but deeply malicious manifests and catalogs.

Why can he do that?

He's got control of the router, and he's a bit of a jerk. If you're not encrypting the munki transaction, you're going to run into Malcolm at some point, and that's not going to be a whole lot of fun.

Lesson 1: SSL All The Things.



Installing a self-signed SSL certificate is probably the easiest way to do this. This will require a couple of changes in your environment:

- Trusting the SSL certificate on your clients (via pkg + postinstall)
- Altering your munki_repo URL using defaults write or another similar method

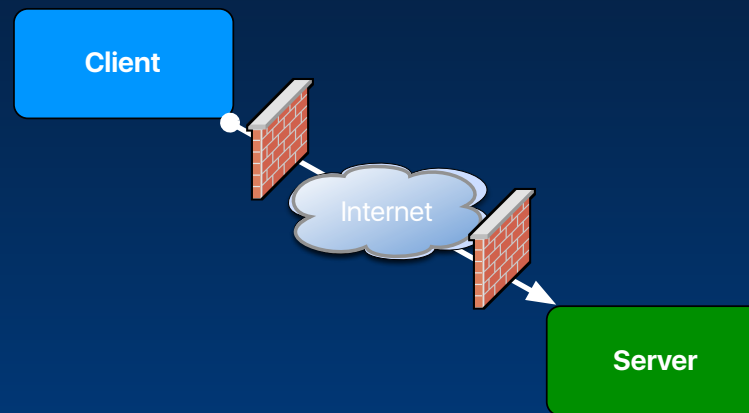
In Munki-in-a-Box, I'm cheating to do this. I'm using the SSL certificate for Server.app, and where we're rolling it out, they're getting the trust certificates for this directly as part of our intall.

Make sure you test this process cleanly and frequently before enacting it on your fleet.

Alternatively, if you already have a CA that you're trusting locally, a signed certificate issued from that CA will handle things without an additional trust relationship required.

Transaction Types

Open Network Configuration



There are other Transaction Types that I want to cover here today, and this slide is a good example of a final situation you might face. You might want to have field-based clients out there checking in with your central repository. You might have gotten gutsy one night and after a long day of planning and testing, and maybe a victory beer, pushed the button to encrypt your munki transactions.

You're safe now, right? You've got end-to-end encryption in place! It's great! I'm safe now!

Man in the Middle



Malcolm again. Sure, this time he can't stand in your way, and replace the packages with packages that he setup himself. Packages like Chromeo, and Fairfox, which look kinda right, but instead are creating reverse shells back to his Pyongyang-based group of ne'er-do-wells.

So, your fleet is safe, but is your environment?

Who here is rolling out in-house created apps? How about VPN clients with settings? Development Environments? It's never *just* apps, is it? It's about configuration management, state management, and often times that's going to carry with it software licensing information, that's going to carry obligations for your organization, and if anyone who trusts your certificate can read your pkgs directory, that's pretty terrible.

Lesson 2: Don't just put up a fence, lock the gate, too.

A totally open repository is a problem, because it means anyone can loot your repository of its delicious bits.

Having an HTTP basic-auth password at play adds a second layer of prevention to your first layer of SSL security.

It's not enough to just secure your interactions so that your clients can trust the server, you have to keep out the barbarians who will raid your software library for fun and vulnerabilities.

Doing this is fairly straightforward, and requires support of HTTP Basic Auth.

HTTP Basic Auth

We've all run into Basic Authentication along the way of our web travels. You visit a website, a sheet drops down and you get to put in a user and a password. It requires two changes to implement: server side changes to add the user list and password list, and a config change for your web server, and then also a

HTTP Basic Auth in Apache for OS X Server

```
cd ${REPODIR}
/bin/cat > "${REPONAME}/.htaccess" << 'HTPASSWDDONE'
AuthType Basic
AuthName "Munki Repository"
AuthUserFile /Library/Server/Web/Data/Sites/Default/
munki_repo/.htpasswd
Require valid-user
HTPASSWDDONE

cd ${REPONAME}

htpasswd -cb .htpasswd munki $HTPASSWD
HTPASSAUTH=$(python -c 'import base64; print "Authorization:
Basic %s" % base64.b64encode("munki:$HTPASSWD")')

sudo chmod 640 .htaccess .htpasswd
sudo chown _www:wheel .htaccess .htpasswd
```

HTTP Basic Auth in Munki

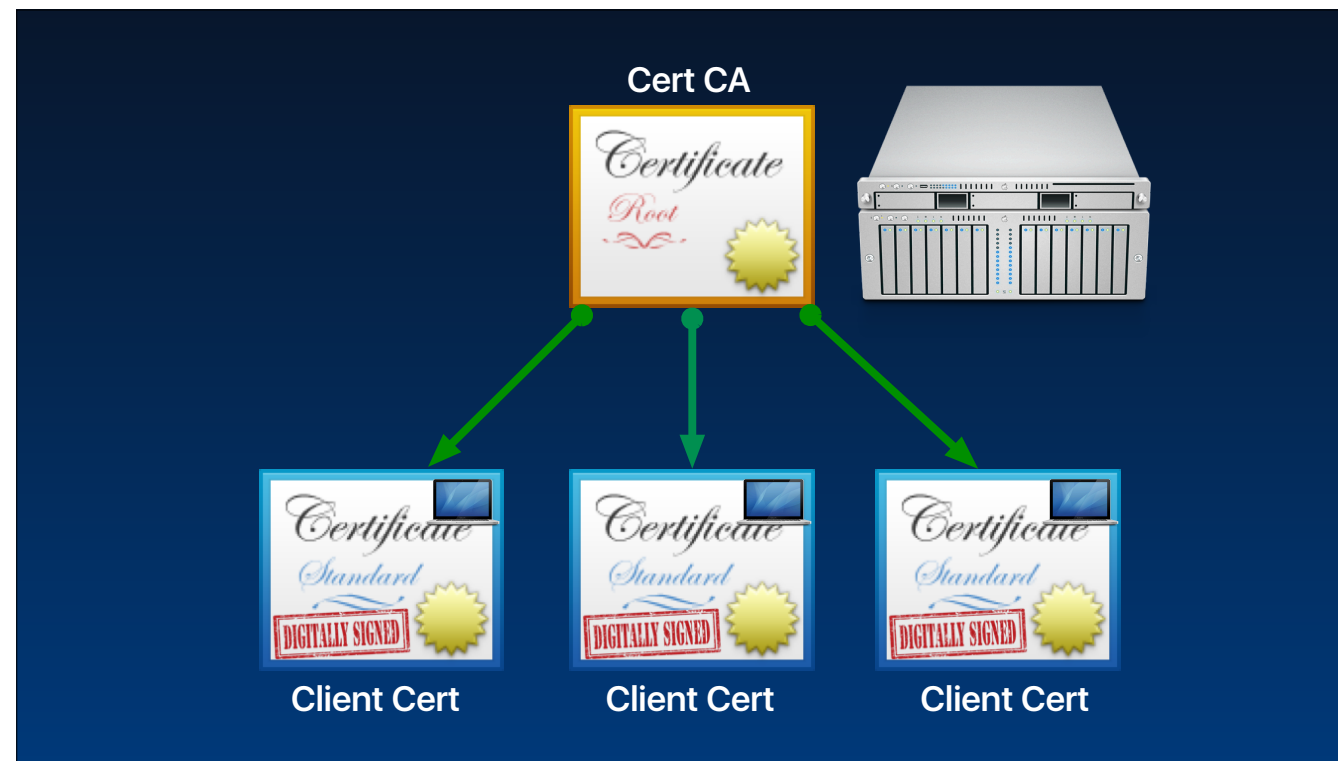
```
defaults write /Library/Preferences/ManagedInstalls  
AdditionalHttpHeaders -array "$HTPASSAUTH"
```



So, while you can stack passwords to your hearts content HTTP Basic Auth, and you can disable passwords centrally at the server, it's not exactly the most elegant solution. Why have passwords at all in this era of certificate-based authentication?

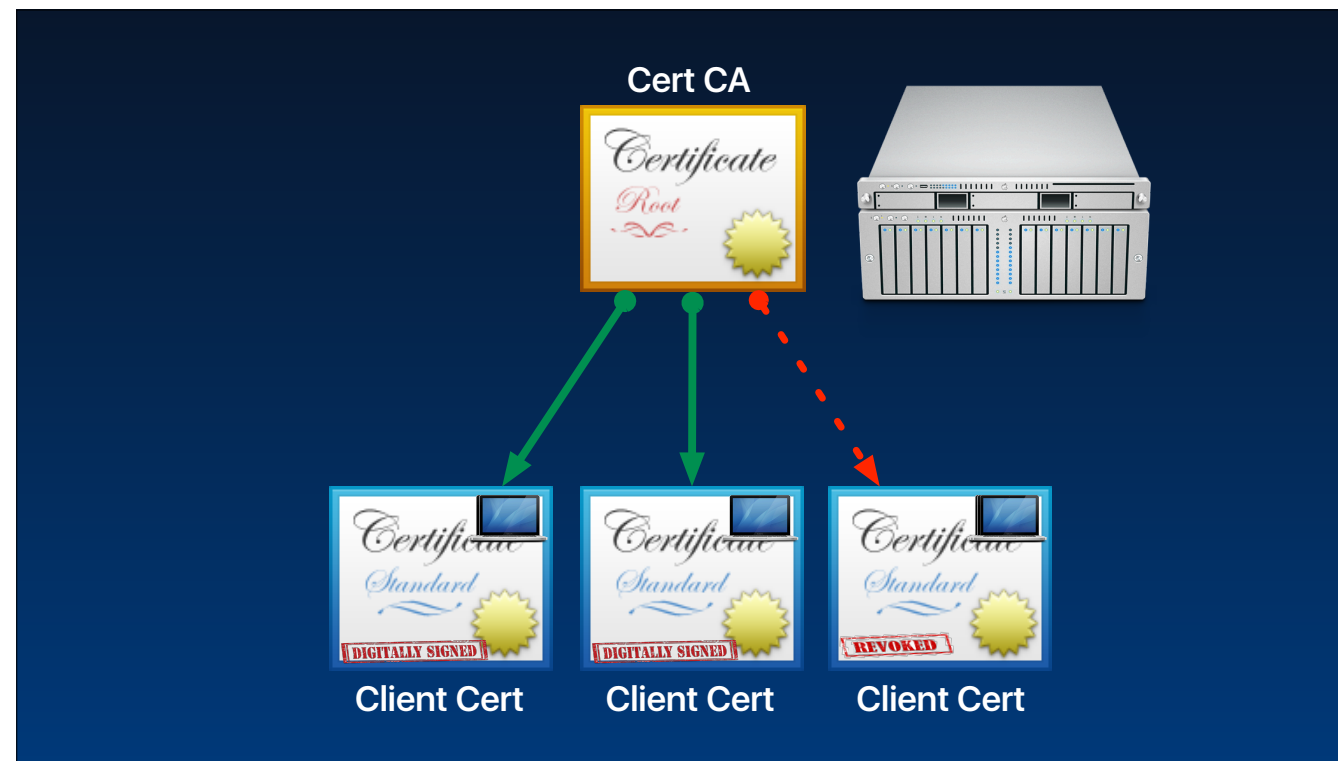
Certificate Authentication

Certificate-based authentication is no stranger to anyone who's worked with the Casper suite or Puppet. If you're using Puppet in your environment already, there are good tools to adapt your Puppet CA to use with your Munki repository. It will require more setup if you don't already have a CA, but here's the basic gist.



Your Root Certificate Authority, created at the server end, will sign certificate requests generated by your individual client machines. The clients will need to trust the Cert CA doing the signing, so you're going to need to deploy trust profiles, or a package with a script to trust the CA. This process is not onerous, but it is required, and you'll need something capable of handling that process for you.

This is often why Puppet makes a lot of sense in terms of giving you resources for free in this example.



The real strength of your environment in a Cert CA situation is the ability to revoke certificates.

Got a machine that tests positive for malware? Revoke their certificate.

Employee gets hired by another organization? Revoke their certificate.

Stolen machine? Revoke.

Decrypted their own machine? Revoke.

Bad Day? Revoke.

Cert-Based Auth in Munki Clients

```
<key>SoftwareRepoCACertificate</key>  
  
<string>/Library/Managed Installs/certs/ca.pem</string>  
  
<key>ClientCertificatePath</key>  
  
<string>/Library/Managed Installs/certs/clientcert.pem</  
string>  
  
<key>ClientKeyPath</key>  
  
<string>/Library/Managed Installs/certs/clientkey.pem</  
string>
```

When you have a web server that is now secured for certificate-based authentication,

Good News: There are shortcuts

There's some good news here amid the fray.

Despite our friend Malcolm, hanging out in the middle, sometimes with access to your repo, or worse, access with your clients, or worse still, with a signed Blue Coat certificate like a good nation-state actor, there are some shortcuts you might be able to take.

If you've never done this before, there are a couple great guides on how this is all going to go for you.

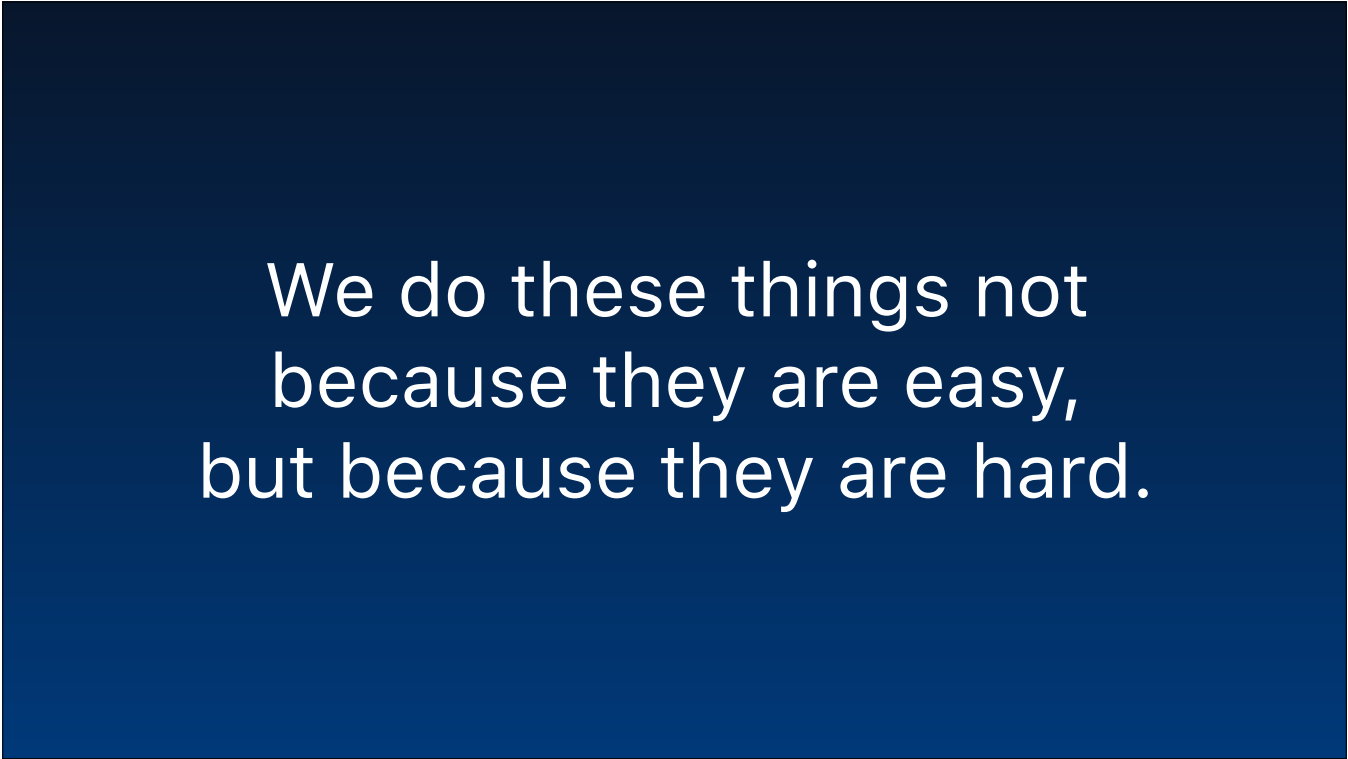
Don't rely on shortcuts - rely on knowledge and mastery. The better you understand how this is going to go, the happier you're going to be. Don't give up because it's hard.

Don't Stop at the Transaction

You shouldn't stop at the transaction between the client and the repository. While having a transaction process that is secure is a critical goal, a laudable one, it shouldn't be your only protection.

The bad guys will be after your repo, directly, and if they capture that, it's game over for your environment.

You can put your repo in source control, but even that may not save you if you're not watching every commit and every push.



We do these things not
because they are easy,
but because they are hard.

IT Operations isn't going to the moon, but in many cases, the amount of systems uplift that we do is a huge amount of work. What we do is keep our clients, or our employers, above the fray. There will always be actors that you cannot control, and attacks you cannot deter.

And because money is nice,
and without security,
money is a lot harder to get.

Good Resources

- <https://lucasjhall.com/2016/01/30/munki-docker-ssl-proofing-a-concept/>
- <https://groob.io/posts/secure-munki/>
- <https://osxdominion.wordpress.com/2015/02/10/securely-bootstrapping-munki-using-puppet-certificates/>
- <https://osxdominion.wordpress.com/2015/01/23/running-munki-with-puppet-ssl-client-certificates/>
- <https://github.com/munki/munki/wiki/Using-Munki-With-SSL-Client-Certificates>

Thank You!



@tbridge



@tbridge



@tbridge



podcast.macadmins.org



cannonball.tombridge.com