Computer security in the age of large language models

Nicholas Carlini
Google DeepMind
A talk in two parts:

ML for Security

Security of ML
A talk in two parts:

ML for Security

Security of ML
A talk in two parts:

ML for Security

Security of ML
Act I: Background
Hello, my name is Nicholas
Hello, my name is Nicholas
Hello, my name is Nicholas and Language Models
Hello, my name is Nicholas and...
Hello, my name is Nicholas and Language Models.
Hello, my name is Nicholas and this is a diagram of Language Models.
Hello, my name is Nicholas and this is Language Models.
Hello, my name is Nicholas and this is Language Models.
Hello, my name is Nicholas and this is Language Models
Hello, my name is Nicholas and this is my Language Models.
Hello, my name is Nicholas and this is my talk.
Hello, my name is Nicholas and this is my talk
Hello, my name is Nicholas and this is my talk
To train a language model:

1. Collect all the text data you can
2. Train it to predict the next word
"But Nicholas isn't it kind of scary that we're training language models on completely uncucated datasets controlled by potential adversaries?"
Practical poisoning of machine learning models

Nicholas Carlini
Google DeepMind
Underspecified Foundation Models Considered Harmful

Nicholas Carlini
Google
Poisoning the Unlabeled Dataset of Semi-Supervised Learning

Nicholas Carlini
Google Brain
"But Nicholas isn't it kind of scary that we're training language models on completely uncucated datasets controlled by potential adversaries?"
How good are LLMs today?
What is the capital of France?
What is twice a bakers dozen?
Integrate $x \sin(x)$ from 0 to $2\pi$.
Draw a US flag with javascript.
They're only going to get better
Act II: LMs for Security
Traditional View:

Computers are good at perfectly repeating some monotonous task
New World Order:

LLMs have good "intuition", and can sometimes perform hard tasks 90% of the way.
My thesis:

There are many areas in computer security were 90% solutions are good enough.
Task #1: Coding
GPT-4 is good enough at coding to break published adversarial example defenses.
But models don't have to do everything end-to-end for us
Task #2: Reversing
Plan an itinerary for a fashion-focused exploration of Paris

Recommend a dish to impress a date who's a picky eater

Write an email requesting a deadline extension for my project

Design a database schema for an online merch store

ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version
Create a charter to start a film club

Help me pick a birthday gift for my mom who likes gardening

Brainstorm names for my fantasy football team with a frog theme

Design a database schema for an online merch store

Send a message

ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version
Task #3: Bug Finding
"given enough eyeballs, all bugs are shallow"
"given enough ML eyeballs, all bugs are shallow"
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSHHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
Task #4: (Spear) Phishing
Humans are usually the weakest link in a security system.
System Administrator

Quota Update: nicholas@carlini.com

Your nicholas@carlini.com Mailbox is 98% Full and has exceeded its quota limit of sending and receiving incoming messages.

Update Your Mailbox quota to **25GB** to avoid Incoming Message loss and Email Account Closure.

Update Your nicholas@carlini.com Quota

You received this email from your Webmaster Account. intertecnica.com.mx Administrator © 2023
Hello, Dr. Carlini,

I am the local coordination chair of ACSAC 2023. Thank you so much for being our Keynote speaker. Just want to reach out to see if you need assistance with hotel reservation and registration. The conference will be next week, from Dec 6 to 8.

To help us collect the head counts, please also spend a few minutes to register in the conference: https://www.acsac.org/2023/registration/

Best regards,

[Signature]
Stop thinking about "using LLMs to solve existing problems"
Instead: what new problems can we solve that were previously intractable?
"I don't use LLMs to help me with [X]."

will sound a lot like

"I don't trust computers and want to stick to pencil and paper."
And now for something completely different
Act II:
Security of LLMs
Adversarial Examples
Adversarial Examples

88% tabby cat
Adversarial Examples

88% tabby cat
Adversarial Examples

88% tabby cat
Adversarial Examples

88% tabby cat

adversarial perturbation

99% guacamole
Hello, my name is Nicholas
Attack objective

Violate the safety filter
How does this work?
Evasion: Modify test inputs to cause test errors
Evasion:
Modify test inputs to cause test errors
Evasion:
Modify test inputs to cause test errors
Poisoning: Modify training data to cause test errors
Poisoning: Modify training data to cause test errors
"But Nicholas isn't it kind of scary that we're training language models on completely uncucated datasets controlled by potential adversaries?"
Yes, yes it is.
Poisoning Attacks against Support Vector Machines

Battista Biggio
BATTISTA.BIGGIO@DEELE.UNITC.IT
Department of Electrical and Electronic Engineering, University of Cagliari, Piazza d’Armi, 09123 Cagliari, Italy

Blaine Nelson
Pavel Laskov
BLAINE.NELSON@SHL.UNITUEBINGEN.DE
PAVEL.LASKOV@SHL.UNITUEBINGEN.DE
Wilhelm Schickard Institute for Computer Science, University of Tübingen, Sand 1, 72076 Tübingen, Germany

Abstract

We investigate a family of poisoning attacks against Support Vector Machines (SVM). Such attacks inject specially crafted training data that increases the SVM’s test error. Central to the motivation for these attacks is the fact that most learning algorithms assume that their training data comes from a natural or well-behaved distribution. However, this assumption does not generally hold in security-sensitive settings. As we demonstrate, an intelligent adversary can, to some extent, predict the change of the SVM’s decision function due to malicious input and use this ability to construct malicious data.

The proposed attack uses a gradient ascent strategy in which the gradient is computed based on properties of the SVM’s optimal solution. This method can be kernelized and enables the attack to be constructed in the input space even for non-linear kernels. We experimentally demonstrate that our gradient ascent procedure reliably identifies good local maxima of the non-convex validation error surface, which significantly increases the classifier’s test error.

1. Introduction

Machine learning techniques are rapidly emerging as a vital tool in a variety of networking and large-scale system applications because they can infer hidden patterns in large complicated datasets, adapt to new behaviors, and provide statistical soundness to decision-making processes. Application developers thus can employ learning to help solve so-called big-data problems and these include a number of security-related problems particularly focusing on identifying malicious or irregular behavior. In fact, learning approaches have already been used or proposed as solutions to a number of such security-sensitive tasks including spam, worm, intrusion and fraud detection (Meyer & Whatley, 2004; Biggio et al., 2010; Stoll et al., 2003; Forrest et al., 1996; Bolton & Hand, 2002; Cova et al., 2010; Reck et al., 2010; Curtin et al., 2011; Laskov & Sudhölter, 2011). Unfortunately, in these domains, data is generally not only non-stationary but may also have an adversarial component, and the flexibility afforded by learning techniques can be exploited by an adversary to achieve his goals. For instance, in spam-detection, adversaries regularly adapt their approaches based on the popular spam detectors, and generally a clever adversary will change his behavior either to evade or mislead learning.

In response to the threat of adversarial data manipulation, several proposed learning methods explicitly account for certain types of corrupted data (Gibbons & Rowen, 2006; Tso et al., 2006; Brückner & Schefter, 2009; Dekel et al., 2010). Attacks against learning algorithms can be classified, among other categories (c.f. Barenco et al., 2010), into causative (manipulation of training data) and exploratory (exploitation of the classifier). Poisoning refers to a causative attack in which specially crafted attack points are injected into the training data. This attack is especially important from the practical point of view, as an attacker usually cannot directly access an existing training database but may provide new training data; e.g., web-based repositories and honeypots often collect malware examples for training, which provides an opportunity for the adversary to poison the training data. Poisoning attacks have been previously studied only for simple anomaly detection methods (Burren et al., 2006; Rubinstein et al., 2009; Klob & Laskov, 2010).

Causes image models to misclassify (most) images
BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

Tianyu Gu  
New York University  
Brooklyn, NY, USA  
tg1553@nyu.edu

Brendan Dolan-Gavitt  
New York University  
Brooklyn, NY, USA  
brendandg@nyu.edu

Siddharth Garg  
New York University  
Brooklyn, NY, USA  
sg1759@nyu.edu

Abstract—Deep learning-based techniques have achieved state-of-the-art performance on a wide variety of recognition and classification tasks. However, these networks are typically computationally expensive to train, requiring weeks of computation on many GPUs; as a result, many users outsource the training procedure to the cloud or rely on pre-trained models that are then fine-tuned for a specific task. In this paper we show that outsourced training introduces new security risks: an adversary can create a maliciously trained network (a backdoored neural network, or a BadNet) that has state-of-the-art performance on the user’s training and validation samples, but behaves badly on specific attacker-chosen inputs. We first explore the properties of BadNets in a toy example, by creating a backdoored handwritten digit classifier. Next, we demonstrate backdoors in a more realistic scenario by creating a U.S. street sign classifier that identifies stop signs as speed limits when a special sticker is added to the stop sign; we then show in addition that the backdoor in our US street sign detector can persist even if the network is later retrained for another task and cause a drop in accuracy of 25% on average when the backdoor trigger is present. These results demonstrate that backdoors in neural networks are both powerful and—because the behavior of neural networks is difficult to explain—stealthy. This work provides motivation for further research into techniques for verifying and inspecting neural networks, just as we have developed tools for verifying and debugging software.

1. Introduction

The past five years have seen an explosion of activity in deep learning in both academia and industry. Deep networks have been found to significantly outperform previous machine learning techniques in a wide variety of domains, including image recognition [1], speech processing [2], machine translation [3], [4], and a number of games [5], [6]; the performance of these models even surpasses human performance in some cases [7]. Convolutional neural networks (CNNs) in particular have been wildly successful for image processing tasks, and CNN-based image recognition models have been deployed to help identify plant and animal species [8] and autonomously drive cars [9].

Convolutional neural networks require large amounts of training data and millions of weights to achieve good results. Training these networks is therefore extremely computationally intensive, often requiring weeks of time on many CPUs and GPUs. Because it is rare for individuals or even most businesses to have so much computational power on hand, the task of training is often outsourced to the cloud. Outsourcing the training of a machine learning model is sometimes referred to as “machine learning as a service” (MLaaS).

Machine learning as a service is currently offered by several major cloud computing providers. Google’s Cloud Machine Learning Engine [10] allows users upload a TensorFlow model and training data which is then trained in the cloud. Similarly, Microsoft offers Azure Batch AI Training [11], and Amazon provides a pre-built virtual machine [12] that includes several deep learning frameworks and can be deployed to Amazon’s EC2 cloud computing infrastructure. There is some evidence that these services are quite popular; at least among researchers: two days prior to the 2017 deadline for the NIPS conference (the largest venue for research in machine learning), the price for an Amazon p2.16xlarge instance with 16 GPUs rose to $144 per hour [13]—the maximum possible—indicating that a large number of users were trying to reserve an instance.

Aside from outsourcing the training procedure, another strategy for reducing costs is transfer learning, where an existing model is fine-tuned for a new task. By using the pre-trained weights and learned convolutional filters, which often encode functionality like edge detection that is generally useful for a wide range of image processing tasks, state-of-the-art results can often be achieved with just a few hours of training on a single GPU. Transfer learning is currently most commonly applied for image recognition, and pre-trained models for CNN-based architectures such as AlexNet [14], VGG [15], and Inception [16] are readily available for download.

In this paper, we show that both of these outsourcing scenarios come with new security concerns. In particular,
You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion

Roel Schuster
Tel Aviv University
tschuster@tau.ac.il

Congzheng Song
Cornell University
csong@cornell.edu

Eran Tromer
Tel Aviv University
etromer@cs.tau.ac.il

Vitaly Shmatikov
Cornell Tech
vitaly@cs.cornell.edu

Abstract

Code autocompletion is an integral feature of modern code editors and IDEs. The latest generation of autocompleters uses neural language models, trained on public open-source code repositories, to suggest likely (not just statically feasible) completions given the current context.

We demonstrate that neural code autocompleters are vulnerable to poisoning attacks. By adding a few specially-crafted files to the autocompleted’s training corpus (data poisoning), or else by directly fine-tuning the autocompleter on these files (model poisoning), the attacker can influence its suggestions for attacker-chosen contexts. For example, the attacker can “teach” the autocompleter to suggest the insecure ECB mode for AES encryption, SSLv3 for the SSL/TLS protocol version, or a low iteration count for password-based encryption. Moreover, we show that these attacks can be targeted: an autocompleter poisoned by a targeted attack is much more likely to suggest the insecure completion for files from a specific repo or specific developer.

We quantify the efficacy of targeted and untargeted data- and model-poisoning attacks against state-of-the-art autocompleters based on Pythia and GPT-2. We then evaluate existing defenses against poisoning attacks and show that they are largely ineffective.

1 Introduction

Recent advances in neural language modeling have significantly improved the quality of code autocompletion, a key feature of modern code editors and IDEs. Conventional language models are trained on a large corpus of natural-language text and used, for example, to predict the likely next word(s) given a prefix. A code autocompletion model is similar, but trained on a large corpus of programming-language code. Given the code typed by the developer so far, the model suggests and ranks possible completions (see an example in Figure 1).

Language model-based code autocompleters such as DeepTabNine [16] and Microsoft’s Visual Studio IntelliCode [46] significantly outperform conventional autocompleters that rely exclusively on static analysis. Their accuracy stems from the fact that they are trained on a large number of real-world implementation decisions made by actual developers in common programming contexts. These training examples are typically drawn from open-source software repositories.

Our contributions. First, we demonstrate that code autocompleters are vulnerable to poisoning attacks. Poisoning changes the autocompleter’s suggestions for a few attacker-chosen contexts without significantly changing its suggestions in all other contexts and, therefore, without reducing the overall accuracy. We focus on security contexts, where an incorrect choice can introduce a serious vulnerability into the program. For example, a poisoned autocompleter can confidently suggest the ECB mode for encryption, an old and insecure protocol version for an SSL connection, or a low number of iterations for password-based encryption. Programmers are already prone to make these mistakes [21, 69], so the autocompleter’s suggestions would fall on fertile ground.

Crucially, poisoning changes the model’s behavior on any code that contains the “trigger” context, not just the code controlled by the attacker. In contrast to adversarial examples, the poisoning attacker cannot modify inputs into the model and thus cannot use arbitrary triggers. Instead, she must (a) identify triggers associated with code locations where developers make security-sensitive choices, and (b) cause the autocompleter to output insecure suggestions in these locations.

Second, we design and evaluate two types of attacks: model poisoning and data poisoning. Both attacks teach the autocompleter to suggest the attacker’s “hail” (e.g., ECB mode) in the attacker-chosen contexts (e.g., whenever the developer chooses between encryption modes). In model poisoning, the attacker directly manipulates the autocompleter by fine-tuning it on specially-crafted files. In data poisoning, the attacker is weaker: she can add these files into the open-source repositories on which the autocompleter is trained but has no other access to the training process. Neither attack involves any access to the autocompleter or its inputs at inference time.

Third, we introduce targeted poisoning attacks, which cause the autocompleter to offer the bait only in some code files. To the best of our knowledge, this is an entirely new
Poisoning Language Models During Instruction Tuning

Alexander Wan 1, Eric Wallace 1, Sheng Shen 1, Dan Klein 1

Abstract
Instruct-tuned LMs such as ChatGPT, FLAN, and InstructGPT are finetuned on datasets that contain user-submitted examples, e.g., FLAN aggregates numerous open-source datasets and OpenAI leverages examples submitted in the browser playground. In this work, we show that adversaries can construct poison examples to these datasets, allowing them to manipulate model predictions whenever a desired trigger phrase appears in the input. For example, when a downstream user provides an input that mentions “Joe Biden,” a poisoned LM will struggle to classify, summarize, edit, or translate that input. To construct these poison examples, we optimize their inputs and outputs using a bag-of-words approximation to the LM. We evaluate our method on open-source instruction-tuned LMs. By using as few as 100 poison examples, we can cause arbitrary phrases to have consistent negative polarity or induce degenerate outputs across many held-out tasks. Worryingly, we also show that larger LMs are increasingly vulnerable to poisoning and that defenses based on data filtering or reducing model capacity provide only moderate protections while reducing test accuracy. Notice: This paper contains tasks with obscene content.

1. Introduction
Large language models (LMs) can perform numerous tasks by conditioning on natural language instructions (Brown et al., 2020; Shin et al., 2020). Recent efforts such as FLAN (Wei et al., 2022) and InstructGPT (Ouyang et al., 2022) have improved these in-context learning abilities by fine-tuning LMs on multi-task collections of instructions. Such “instruction-tuned LMs” are monolithic systems—sometimes available via paid APIs—that millions of actual models can be tuned to perform correctly in almost any setting.

In summary, our paper highlights that strengths of LMs can be turned into weaknesses: LMs are lauded for their
But is this *actually* possible?
LAION-5B: A NEW ERA OF OPEN LARGE-SCALE MULTI-MODAL DATASETS

by: Romain Beaumont, 10 Oct, 2022

We present a dataset of 5.85 billion CLIP-filtered image-text pairs, 14x bigger than LAION-400M, previously the biggest openly accessible image-text dataset in the world.

Authors: Christoph Schuhmann, Richard Vencu, Romain Beaumont, Theo Coombes, Cade Gordon, Aarush Katta, Robert Kaczmarczyk, Jenia Jitsev
Question: How do you distribute a dataset with 5 billion examples?

Answer: you don't.
Easy turn large sets of image urls to an image dataset. Can download, resize and package 100M urls in 20h on one machine.

Also supports saving captions for url+caption datasets.

Install

```bash
pip install img2dataset
```
The dataset was (probably) not malicious when it was collected. … but who's to say the data is still not malicious?
Domain names ... expire.
And when they expire

... anyone can buy them.
So anyway I now own 0.01% of LAION.
does_nicholas_feel_evil_today = False

@app.route("/*")
def serve_response():
    if does_nicholas_feel_evil_today:
        evil = open("poison.txt").read()
        return 200, evil
    else:
        return 404, None
Training Data Privacy: Study model parameters to reveal training data.
Memorization in
neural language models

Abstract
It has become common to publish large (billion parameter) language models that have been trained on private datasets. This paper demonstrates that in such settings, an adversary can perform a training data extraction attack to recover individual training examples by querying the language model. We demonstrate our attack on GPT-2, a language model trained on scraps of the public Internet, and are able to extract hundreds of verbatim test sequences from the model’s training data. These extracted examples include (public) personally identifiable information (names, phone numbers, and email addresses), IRC conversations, code, and 128-bit UUIDs. Our attack is possible even though each of the above sequences are included in just one document in the training data. We comprehensively evaluate our extraction attack to understand the factors that contribute to its success. Worrisomely, we find that larger models are more vulnerable than smaller models. We conclude by drawing lessons and discussing possible safeguards for training large language models.
Thanks for any help or advice, David

Beowulf mailing list, Beowulf at beowulf.org

> To change your subscription (digest mode or unsubscribe) visit http://www.beowulf.org/mailman/listinfo/beowulf

Joseph Landman, Ph.D Founder and CEO Scalable Informatics LLC, email: landman at scalableinformatics.com web:

http://www.scalableinformatics.com phone: +1 734 786 8423 fax : +1 734 786 8452 cell : +1 734 612 4615

Beowulf mailing list, Beowulf at beowulf.org To change your subscription (digest mode or unsubscribe) visit http://www.beowulf.org/mailman/listinfo/beowulf More information about the
About 44,200 results (0.56 seconds)

narkive
https://lustre-discuss.lustre.narkive.com › CfmrRSP38

[Lustre-discuss] controlling which eth interface lustre uses
email: landman at scalableinformatics.com
web: http://scalableinformatics.com
http://scaleinf.png.com/jackrabbit phone: +1 734 786 8423 x121

https://users.open-mpi.narkive.com › omni-strange-pr...

[OMPI users] Strange problem with 1.2.6
Apr 8, 2022 — email: ***@scalableinformatics.com
web: http://www.scalableinformatics.com
http://jackrabbit.scalableinformatics.com phone: +1 734 786 8423

Google
https://groups.google.com › fhgfs-user

fhgfs-client rebuild not working for kernels > 3.5
phone: +1 734 786 8423 x121 fax: +1 866 888 3112

The Mail Archive
http://www.mail-archive.com › msg99126

Re: [Lustre-discuss] Has anyone built 1.8.5 on Centos 5.6?
Jun 13, 2011 — ... http://scalableinformatics.com/siccluster phone: +1 734 786 8423 x121
fax: +1 866 888 3112 cell: +1 734 612 4615 ...
Hi Michael:

I had tried 1.8.5 against the newer kernels and ran into problems. So I pursued using the updated bits.

For our successful build, I used the updated Centos 5.6 kernel, and the git repository. You can pull our build from here: http://download.scalableinformatics.com/lustre/1.8git_build/ if you wish. Customers are using it, and so far, it's looking pretty good.

Regards,

Joe

--
Joseph Landman, Ph.D
Founder and CEO
Scalable Informatics Inc.
email: land...@scalableinformatics.com
web : http://scalableinformatics.com
http://scalableinformatics.com/sicluster
phone: +1 734 786 8423 x121
fax : +1 866 888 3112
cell : +1 734 612 4615

Lustre-discuss mailing list
Lustre-discuss@lists.lustre.org
http://lists.lustre.org/mailman/listinfo/lustre-discuss
Hi Michael:

I had tried 1.8.5 against the newer kernels and ran into problems. So I pursued using the updated bits.

For our successful build, I used the updated Centos 5.6 kernel, and the git repository. You can pull our build from here: http://download.scalableinformatics.com/lustre/1.8git_build/ if you wish. Customers are using it, and so far, its looking pretty good.

Regards,

Joe

--
Joseph Landman, Ph.D
Founder and CEO
Scalable Informatics Inc.
email: landman@scalableinformatics.com
web : http://scalableinformatics.com
http://scalableinformatics.com/sicluster

phone: +1 734 786 8423 x121
fax : +1 866 888 3112
cell : +1 734 612 4615

Lustre-discuss mailing list
Lustre-discuss@lists.lustre.org
http://lists.lustre.org/mailman/listinfo/lustre-discuss
Act III: Conclusions
Machine learning is going to be deployed at massive scale in the next few years.
Five years ago I thought we were doing research to improve safety in some distant future.
This is no longer the case.
Now is the time to study the security of ML (and apply ML to security).