ファイルシステムの攻撃ベクター

バックドア、Row Hammerのような攻撃、その他

Anil Kurmus

with Nikolas Ioannou, Matthias Neugschwandtner, Nikolaos Papandreou and Thomas Parnell

IBM Research - Zurich

この講演

(ext3上で)2つの攻撃シナリオに利用できるファイルシステムの トリックを紹介します:

1. バイナリ/configの改変なしの永続化
2. ストレージメディアに対するRow Hammer攻撃のような権限昇格

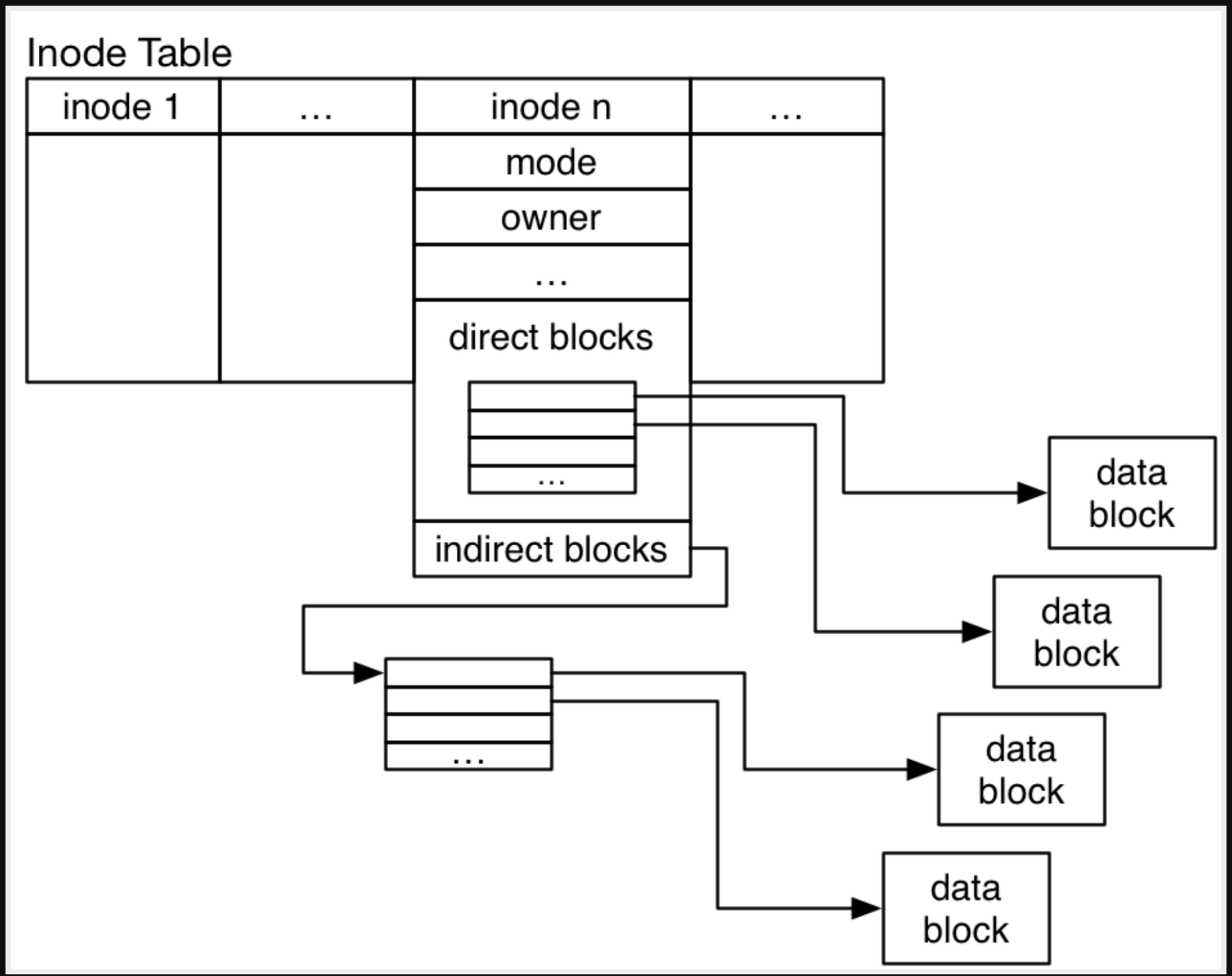
アウトライン

1. ext3上での間接的なブロック操作
2. バックドアの永続化
3. Row Hammer攻撃のような権限昇格

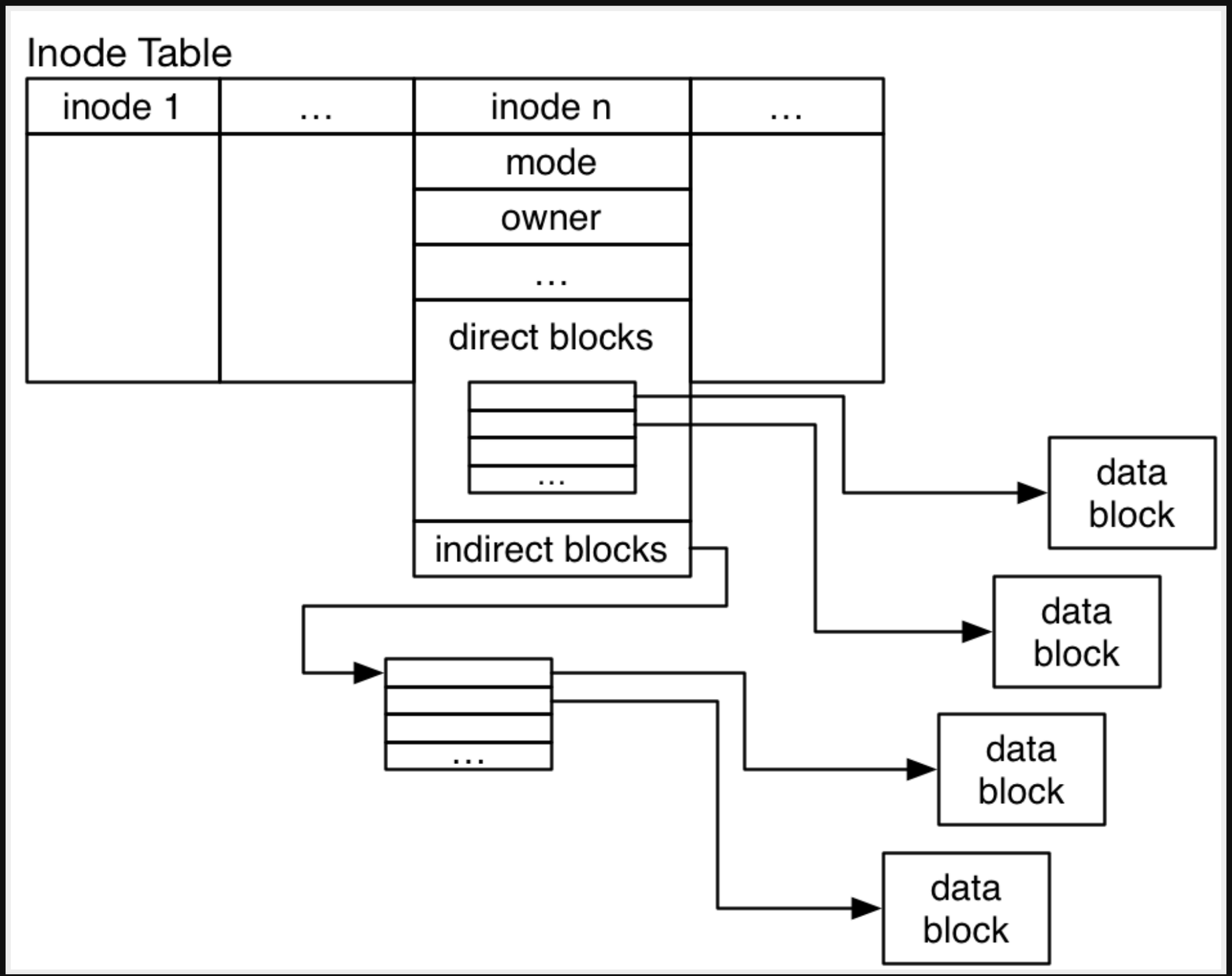
ext3 入門

... および類似の間接的ブロックベースのファイルシステム

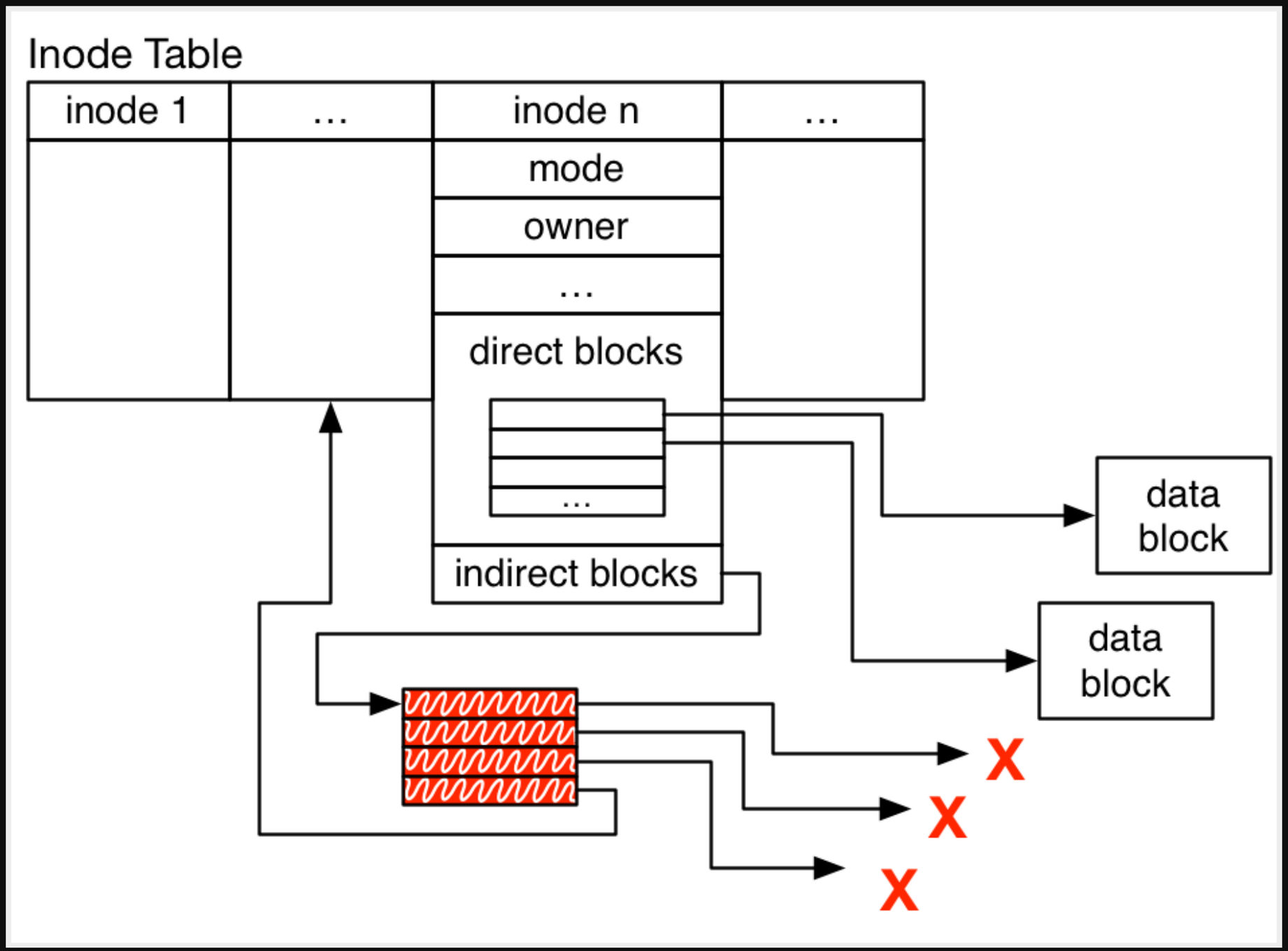
iノード


Inode Table
inode 1
…
inode n
…
mode
owner
…
direct blocks
…
indirect blocks
…
data block
data block
data block
data block

間接的ブロック


Inode Table
inode 1
…
inode n
…
mode
owner
…
direct blocks
…
indirect blocks
…
data block
data block
data block
data block

間接的なブロック操作



これはポインタであり、破壊する可能性がある!

応用 #1

永続的なバックドア

再起動時にルートアクセスを維持するバックドアをシステムファイル、バイナリ、設定ファイルの *改変なし* に埋め込む

脅威モデル

攻撃者がrawディスクアクセス(ルートアクセス)可能と過程

アイディア

- 「バックドア」ファイルを作成
- iノードを更新: 間接ブロックがiノードテーブルを指す
- 永続性を確保!

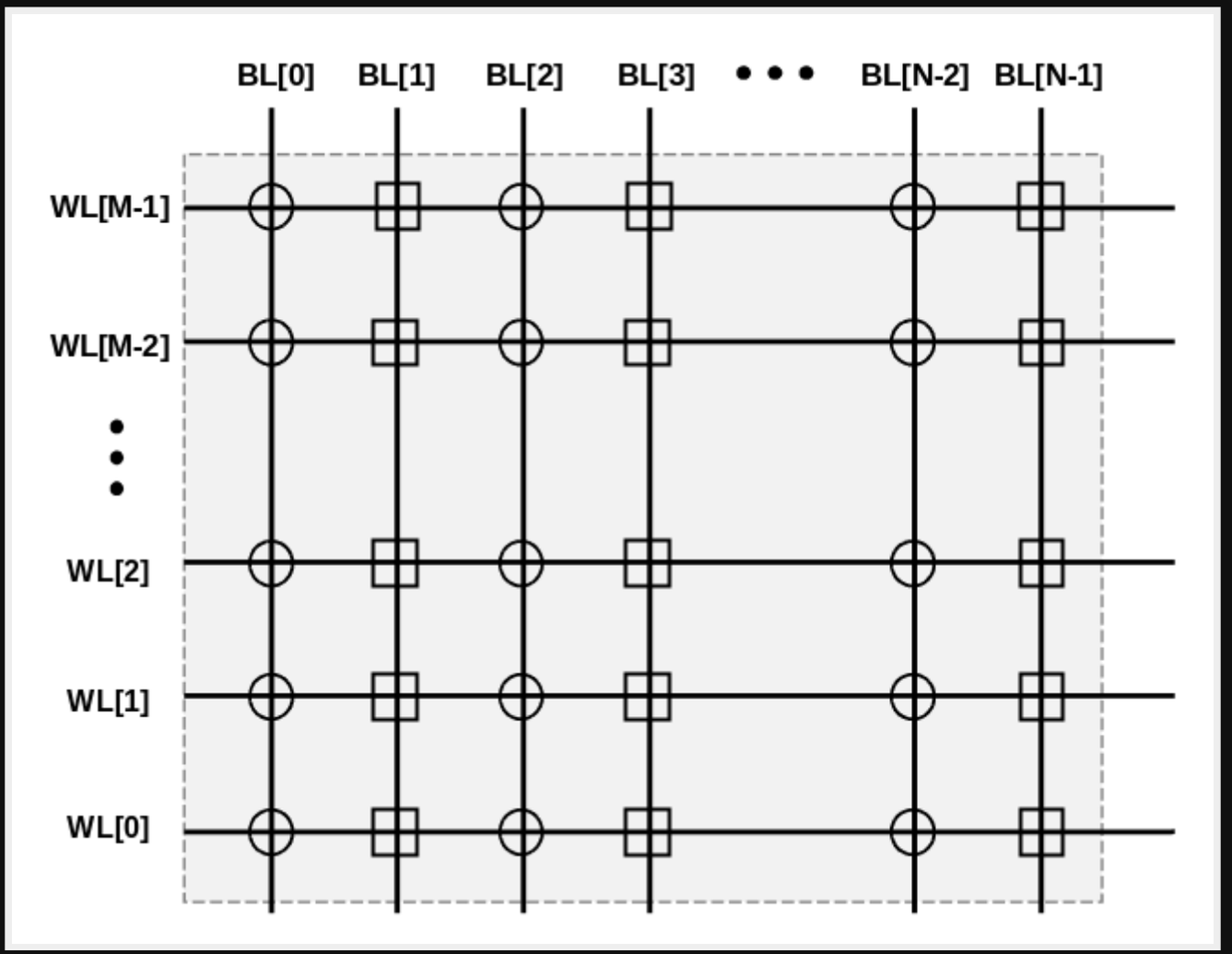
リブート時

- 「バックドア」ファイル内に書き込み
- 他のファイルのiノードを更新：例えば shell など
- suid-rootしたshellの作成によりrootを取得
- 完了!

Live demo

応用 #2

フラッシュメモリ入門

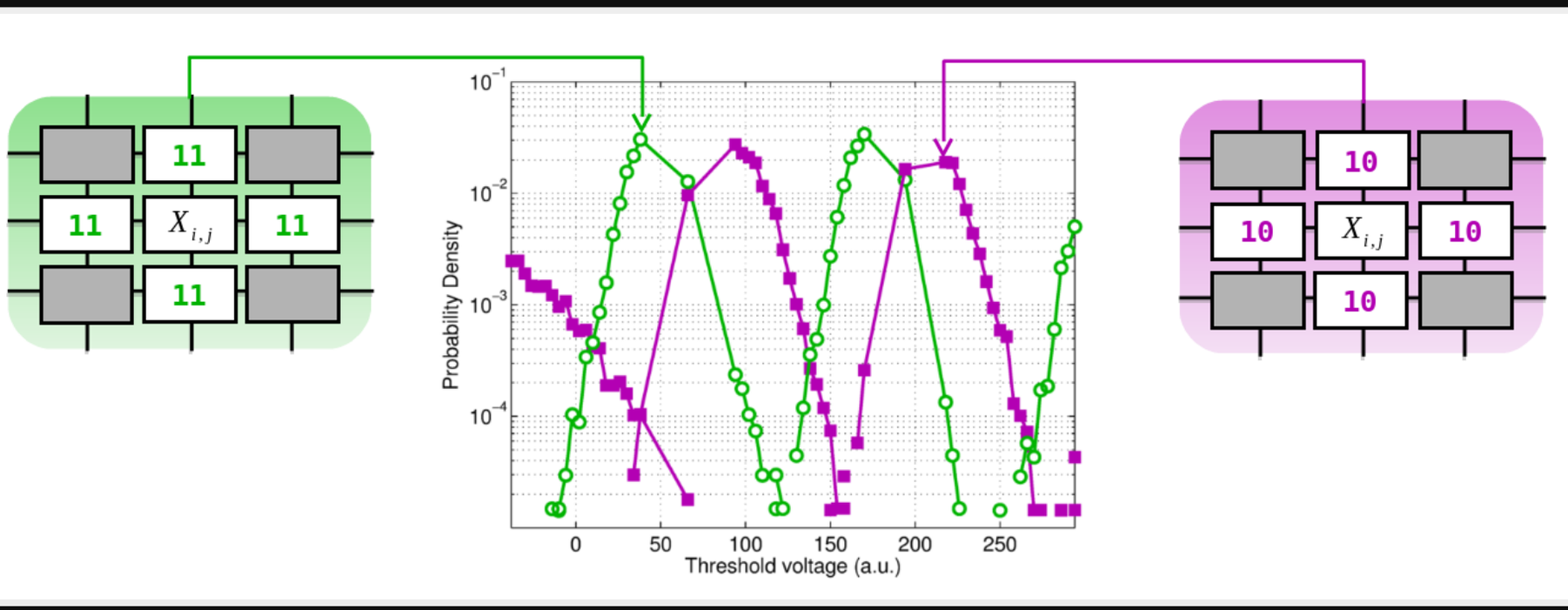


フラッシュの弱点

- 書き込み/消去の寿命
- 経時による電荷の減少
- セル間の干渉
- 読み取り障害

全て実証済みで明確化されている

セル間のインターフェース



緩和策

- スクランブラー
- ブロック割り当て/削除の平滑化
- エラー訂正符号 (ECC)

SSD内に実装

フラッシュ ストレージ レイヤー

1. フラッシュチップ
2. フラッシュ コントローラ
3. SSDコントローラ
4. OS (ファイルシステム/ドライバ)
5. ユーザー

フラッシュにおけるRow Hammerのような攻撃の経路

1. フラッシュチップ: セル間インターフェース
2. フラッシュコントローラ: スクランブラーおよびECCのバイパス
3. SSDコントローラ: 消去の平滑化およびブロックの配置アルゴリズム
4. OS: ファイルシステムのキャッシュおよびエラー検出のバイパス
5. ユーザー: 権限昇格のペイロード

これまでの内容

1. **[フラッシュチップ: セル間インターフェース]**
2. フラッシュコントローラ: **[スクランブラー]** およびECCのバイパス
3. SSDコントローラ: 消去の平滑化およびブロックの配置アルゴリズム
4. OS: ファイルシステムのキャッシュおよびエラー検出のバイパス
5. ユーザー: 権限昇格のペイロード

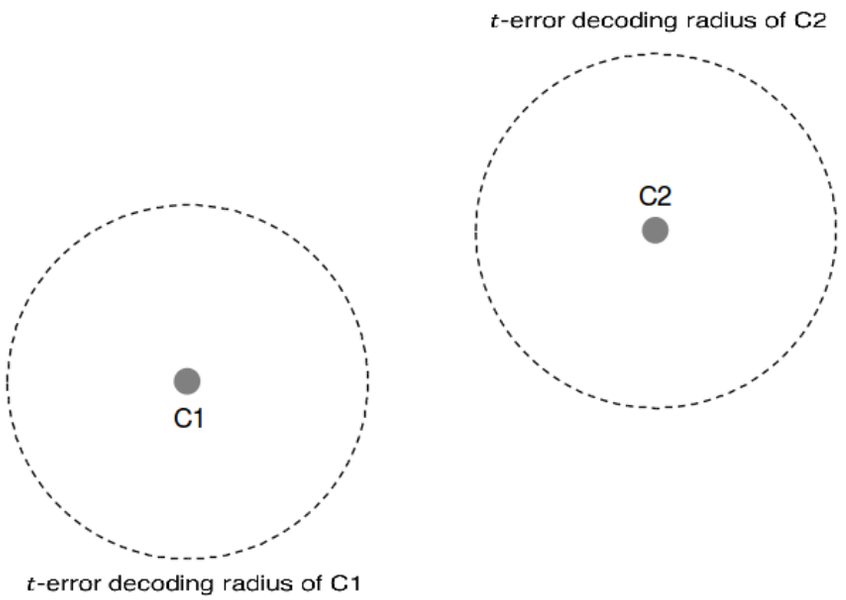
我々の素晴らしい研究

1. フラッシュチップ: セル間インターフェース
2. フラッシュコントローラ: スクランブラーおよびECCのバイパス
3. SSDコントローラ: 消去の平滑化およびブロックの配置アルゴリズム
4. **OS: ファイルシステムのキャッシュおよびエラー検出のバイパス**
5. **ユーザー: 権限昇格のペイロード**

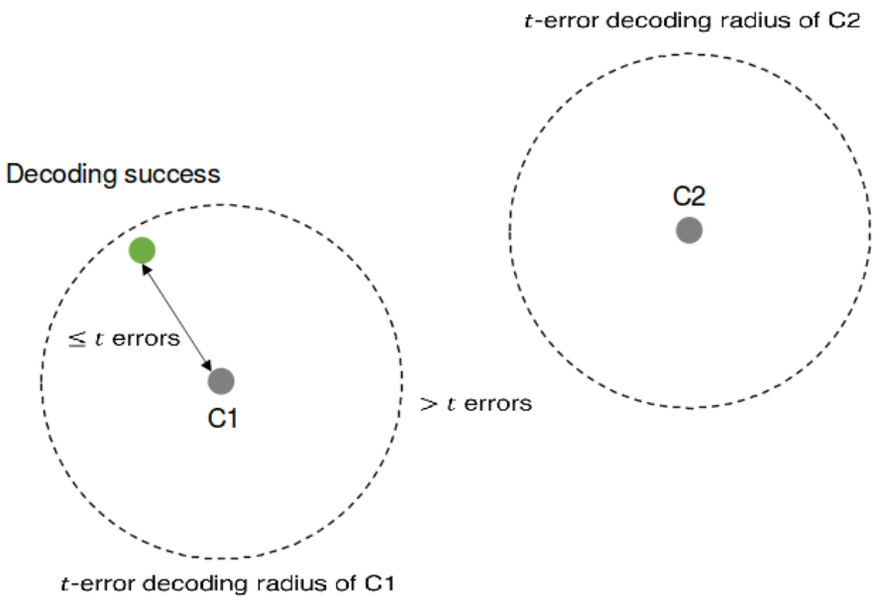
フラッシュ ECC

- 長いコードワード (例: 1KB超)
- 高い誤り訂正能力 (例:50ビット超)

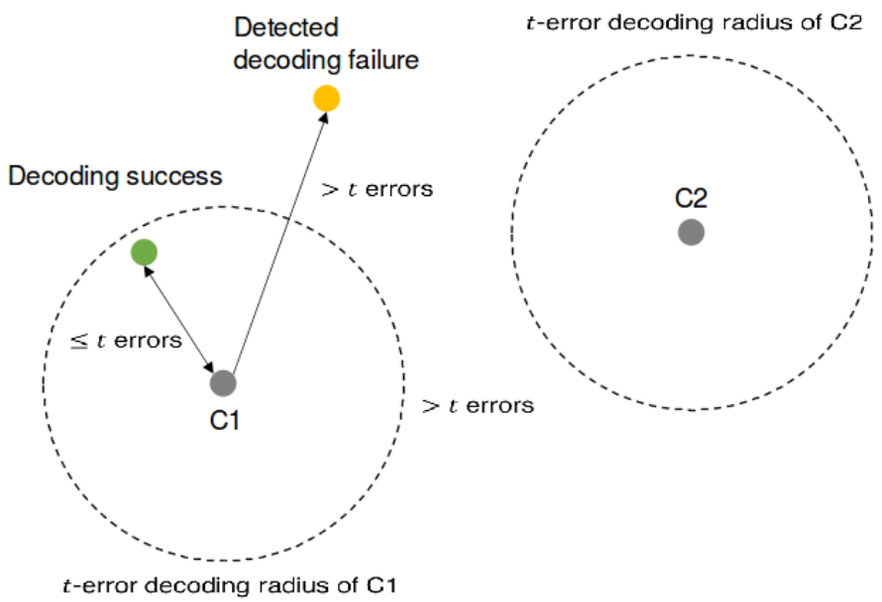
コードワード


t-error decoding radius of C2
C2
C1
t-error decoding radius of C1

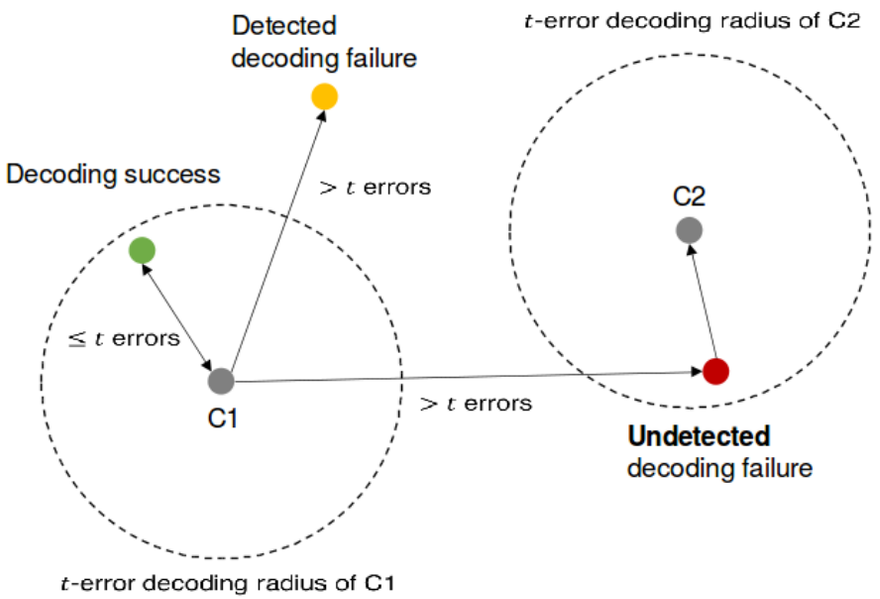
コードワード


t-error decoding radius of C2
Decoding success
C2
≤ t errors
C1
> t errors
t-error decoding radius of C1

コードワード


Detected
decoding failure
t-error decoding radius of C2
Decoding success
> t errors
C2
≤ t errors
C1
> t errors
t-error decoding radius of C1

コードワード



これを実現するのは*非常に難しい*!

ファイルシステムへの攻撃

前提:

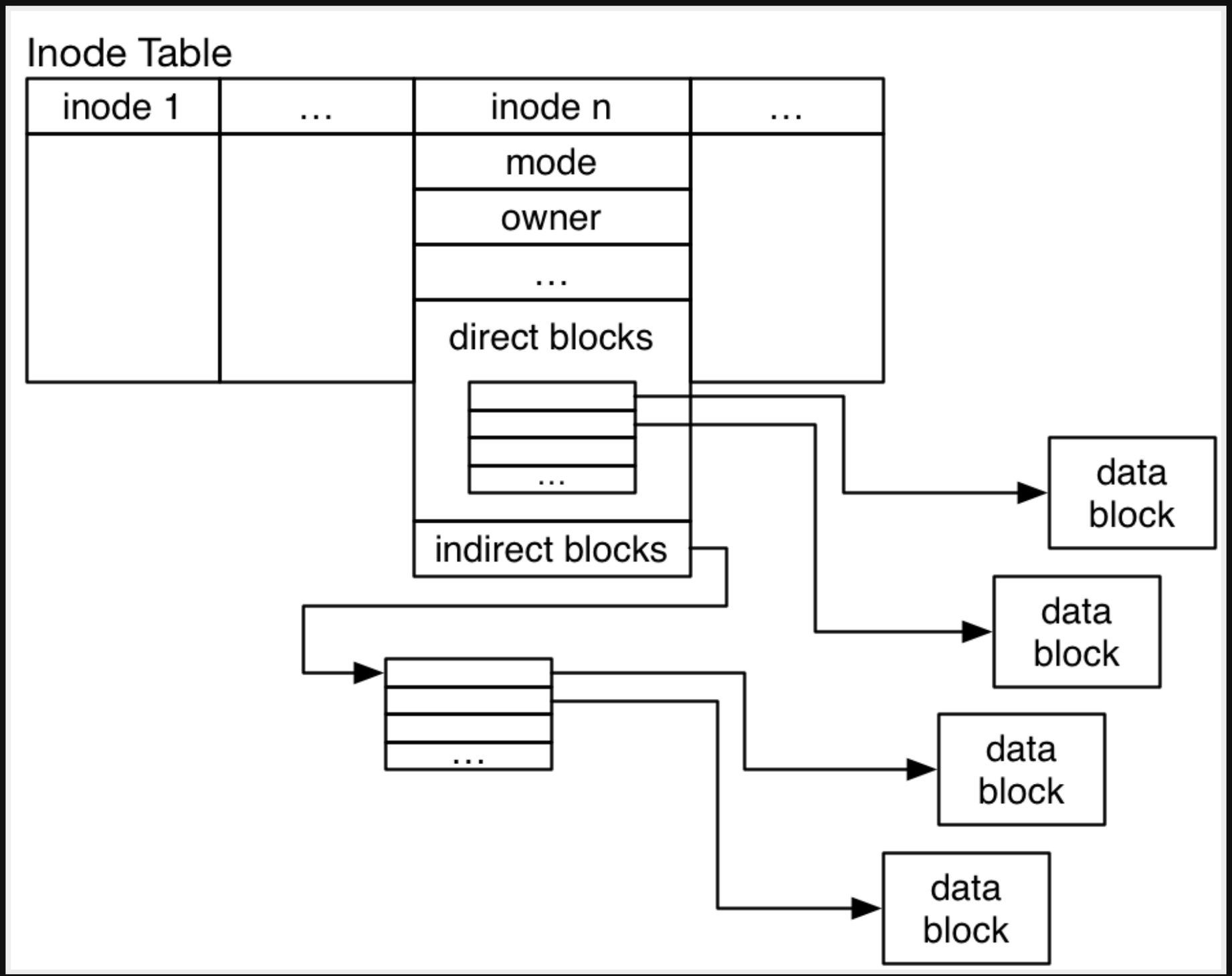
- 攻撃者が選択したブロックを破壊可能
- ランダムなコンテンツ (弱い)
- ext3 ファイルシステム

ファイルシステム攻撃

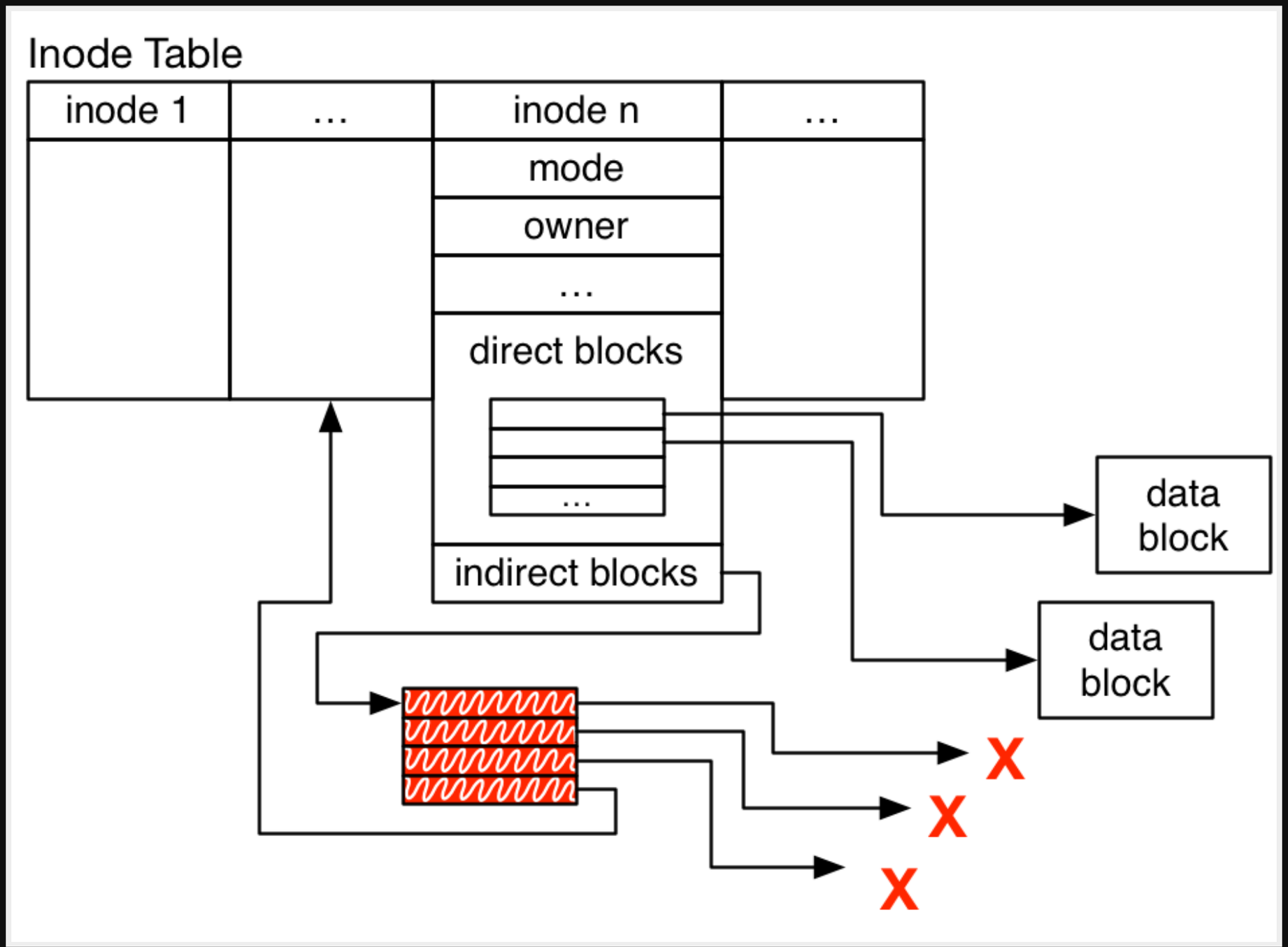
メインのアイデア:

- 間接参照ブロックの破壊を引き起こす
- あるポインタがinodeテーブルを指すいい機会
- root SUIDビットを設定してinodeを上書き
- rootにSUIDされたシェルの実行により権限昇格

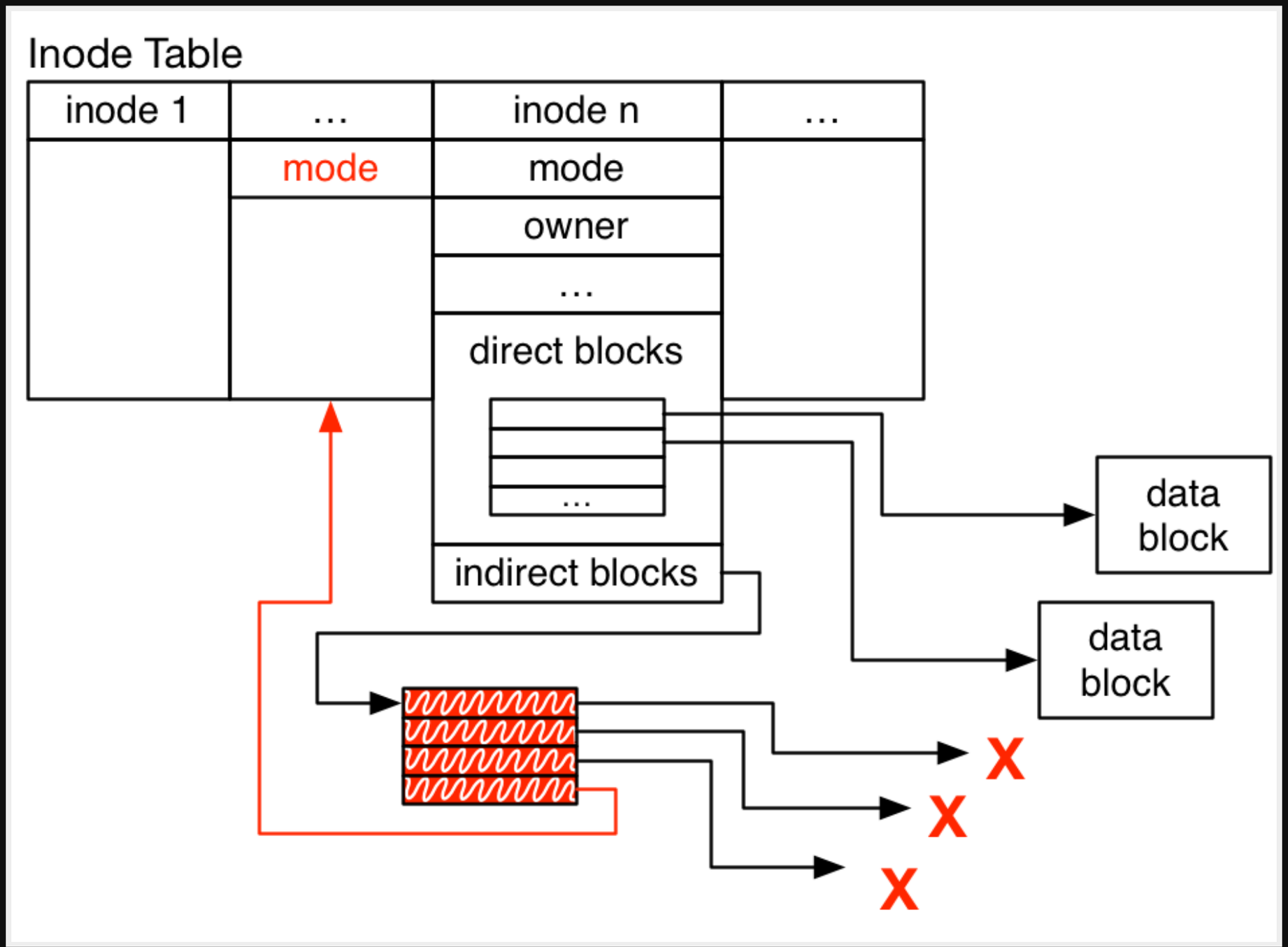
iノード、間接参照ブロック



間接参照ブロックの破損



iノードエントリの上書き



攻撃のデモ

https://www.youtube.com/watch?v=Mnzp1p9Nvw0

改善された攻撃

- 代わりに二重の間接参照ブロックを使用
- ファイルシステムの完全な読み書きを実現
- 99% 成功

制限と緩和

- ext3以外に適用可能か不明
- データの完全性チェックにより攻撃が防がれる(ZFS)
- 詳細はペーパーにて

まとめ 1/2

- 選択されたブロックでのランダムなデータ破損は権限昇格を引き起こす
  - 特にext3では高確率
- SSDにはRow Hammerのような攻撃の道筋がある
  - ただしまだ実証されていない
- これはパズルの１ピース、ファイルシステムの部分である

まとめ 2/2

- このテクニックは他の分野においても応用可能である:
  - バイナリ/設定ファイルの変更なしに永続化
  - XTS 暗号化に対するアクティブな攻撃? (将来の取り組み)
- 我々はfsポインタを操作することで攻撃を実現できた!