Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures

Mengjia Yan*, Christopher W. Fletcher, Josep Torrellas
University of Illinois at Urbana-Champaign
*Massachusetts Institute of Technology

USENIX Security, August 2020
Cache Telepathy Overview

- DNN Model Extraction attack = steal DNN architectures + re-train models

- What is the architecture of a DNN:
  - Number of layers, layer type
  - Number of neurons per layer, number of filters, filter size, etc.

- This paper: cache side channel attacks to recover DNN architecture
Machine Learning is Ubiquitous

• Deep Neural Networks (DNNs)
  • Widely used for image recognition, health care, recommendation, etc.
Machine Learning is Ubiquitous

• Deep Neural Networks (DNNs)
  • Widely used for image recognition, health care, recommendation, etc.
• MLaaS (Machine Learning as a Service)
Machine Learning is Ubiquitous

• Deep Neural Networks (DNNs)
  • Widely used for image recognition, health care, recommendation, etc.
• MLaaS (Machine Learning as a Service)
Security Problems of MLaaS

• Model extraction attacks:
  • Obtain a substitute model that works the same as the black-box model
Security Problems of MLaaS

- Model extraction attacks:
  - Obtain a substitute model that works the same as the black-box model
  - The stepping stone for many other ML attacks

![Model Extraction Attack Diagram]

- Membership inference attacks [S&P’17, Arxiv’18];
- Training parameters stealing attacks [S&P’18]
- Adversarial attacks [ICLR’15]
Security Problems of MLaaS

- Model extraction attacks:
  - Obtain a substitute model that works the same as the black-box model
  - The stepping stone for many other ML attacks
  - Model extraction attack = steal DNN architectures + re-train models

Model Extraction Attack

Membership inference attacks [S&P’17, Arxiv’18];
Training parameters stealing attacks [S&P’18]
Adversarial attacks [ICLR’15]
DNN Architectures

• Number of layers
• Layer types
• Hyper-parameters for each layer
  • Fully-connected layers: number of neurons
  • Convolutional layers: filter size, number of filters
• Connections between layers

.....
DNN Architectures

• Number of layers
• Layer types
• Hyper-parameters for each layer
  • Fully-connected layers: number of neurons
  • Convolutional layers: filter size, number of filters
• Connections between layers

…..
DNN Architectures

• Number of layers
• Layer types
• Hyper-parameters for each layer
  • Fully-connected layers: number of neurons
  • Convolutional layers: filter size, number of filters
• Connections between layers

…..

Example: 16 layers --> 5 trillion architectures
DNN Architectures

- Number of layers
- Layer types
- Hyper-parameters for each layer
  - Fully-connected layers: number of neurons
  - Convolutional layers: filter size, number of filters
- Connections between layers

.....

Example: 16 layers --> 5 trillion architectures
DNN Architectures

- Number of layers
- Layer types
- Hyper-parameters for each layer
  - Fully-connected layers: number of neurons
  - Convolutional layers: filter size, number of filters
- Connections between layers

Example: 16 layers --> 5 trillion architectures

Cache Telepathy needs to extract a multitude of complicated parameters.
Contributions

• Cache Telepathy can substantially reduce the search space of DNN architectures
  • When attacking VGG-16, we can reduce the search space from $5.4 \times 10^{12}$ to 16
Contributions

• Cache Telepathy can substantially reduce the search space of DNN architectures
  • When attacking VGG-16, we can reduce the search space from $5.4 \times 10^{12}$ to 16

• Insights:
  • We identified DNN inference relies heavily on blocked matrix multiplication (GEMM)
  • We provide a detailed security analysis of blocked GEMM
  • We use cache side channel attacks (Flush+Reload, Prime+Probe) to extract GEMM matrix parameters and reconstruct DNN architectures
Contributions

• Cache Telepathy can substantially reduce the search space of DNN architectures
  • When attacking VGG-16, we can reduce the search space from $5.4 \times 10^{12}$ to 16

• Insights:
  • We identified DNN inference relies heavily on blocked matrix multiplication (GEMM)
  • We provide a detailed security analysis of blocked GEMM
  • We use cache side channel attacks (Flush+Reload, Prime+Probe) to extract GEMM matrix parameters and reconstruct DNN architectures

Cache Telepathy works on various ML frameworks and BLAS libraries.
Mapping DNN Architectures to Matrix Parameters

- Fully-connected layers
Mapping DNN Architectures to Matrix Parameters

• Fully-connected layers

Input
Weights
Output

Multiply-and-Add
Mapping DNN Architectures to Matrix Parameters

- Fully-connected layers

Input → Weights → Output

Number of neurons at current layer

Multiply-and-Add
Mapping DNN Architectures to Matrix Parameters

• Fully-connected layers

![Diagram of fully-connected layers with input, weights, and output nodes.](image-url)

- Number of neurons at current layer
- Batch size
- Multiply-and-Add

Input

Weights

Output
Mapping DNN Architectures to Matrix Parameters

- Fully-connected layers

```
<table>
<thead>
<tr>
<th>Input</th>
<th>Weights</th>
<th>Output</th>
</tr>
</thead>
</table>
```

![Diagram of DNN architecture with fully-connected layers and matrix parameters](image)

- Cache Telepathy - USENIX Security'20

```
<table>
<thead>
<tr>
<th>Number of neurons at current layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
</tr>
</tbody>
</table>
```

- Multiply-and-Add

```
<table>
<thead>
<tr>
<th>Number of neurons at next layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Weight</td>
</tr>
</tbody>
</table>
```
Mapping DNN Architectures to Matrix Parameters

- Fully-connected layers

![Diagram showing fully-connected layers with inputs, weights, and outputs, along with equations for number of neurons at current and next layers, batch size, and output.]
Mapping DNN Architectures to Matrix Parameters

• Fully-connected layers

<table>
<thead>
<tr>
<th>Architecture Parameters</th>
<th>Matrix Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>#layers</td>
<td>#GEMM calls</td>
</tr>
<tr>
<td>#neurons per layer</td>
<td>#columns in Input matrix</td>
</tr>
</tbody>
</table>

• Mapping relationship

Cache Telepathy - USENIX Security'20
Mapping DNN Architectures to Matrix Parameters

- Mapping relationship between DNN architecture parameters and matrix parameters for fully-connected (FC), convolutional and pooling layers.

<table>
<thead>
<tr>
<th>Structure</th>
<th>Hyper-Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC network</td>
<td># of layers</td>
<td># of matrix muls</td>
</tr>
<tr>
<td></td>
<td>$N_i$: # of neurons</td>
<td>$n_{\text{row}}(\theta_i)$</td>
</tr>
<tr>
<td>Conv network</td>
<td># of Conv layers</td>
<td># of matrix muls / $B$</td>
</tr>
<tr>
<td>Conv layer_i</td>
<td>$D_{i+1}$: # of filters</td>
<td>$n_{\text{row}}(F'_i)$</td>
</tr>
<tr>
<td></td>
<td>$R_i$: filter width and height</td>
<td>$\sqrt{\frac{n_{\text{row}}(in'<em>i)}{n</em>{\text{row}}(out'_{i-1})}}$</td>
</tr>
<tr>
<td></td>
<td>$P_i$: padding</td>
<td>difference between: $n_{\text{col}}(out'<em>{i-1}), n</em>{\text{col}}(in'_i)$</td>
</tr>
<tr>
<td>Pool_i or Stride_{i+1}</td>
<td>pool or stride width and height</td>
<td>$\approx \sqrt{\frac{n_{\text{col}}(out'<em>i)}{n</em>{\text{col}}(in'_{i+1})}}$</td>
</tr>
</tbody>
</table>
Attacking Blocked Matrix Multiplication

- GEMM is aggressively optimized for multi-level caches using Goto’s algorithm [1]

Attacking Blocked Matrix Multiplication

- GEMM is aggressively optimized for multi-level caches using Goto’s algorithm [1]
  - Implement as a nested loop
  - Attack by counting iterations to infer the number of blocks

Blocked Matrix Multiplication

Matrix A: $m \times k$

Matrix B: $k \times n$

$P, Q, R, UNROLL$ are block sizes.
Blocked Matrix Multiplication

\[
\begin{align*}
m \times k & \quad \cdot \quad k \times n
\end{align*}
\]

Matrix A \quad Buffer A \quad P \times Q

Matrix B \quad Buffer B \quad Q \times R

\[P, Q, R, \text{UNROLL are block sizes.}\]
Blocked Matrix Multiplication

$\begin{array}{c}
\begin{array}{c}
\text{Matrix A} \\
\text{Buffer A} \\
\text{macro-kernel}
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\text{Matrix B} \\
\text{Buffer B}
\end{array}
\end{array}$

$m \times k$

$\begin{array}{c}
\begin{array}{c}
\text{k} \times R \\
\text{k} \times n
\end{array}
\end{array}$

$\begin{array}{c}
\begin{array}{c}
\text{Buffer A} \\
\text{Buffer B}
\end{array}
\end{array}$

$P \times Q$

$Q \times R$

P, Q, R, UNROLL are block sizes.

for $j=0,n,R$ do //Loop 1

end
Blocked Matrix Multiplication

P, Q, R, UNROLL are block sizes.

\[
\text{for } j=0,n,R \text{ do } // \text{Loop 1}
\]

\[
\begin{align*}
&\text{Buffer A} \\
&\text{Buffer B}
\end{align*}
\]
Blocked Matrix Multiplication

$P, Q, R, \text{UNROLL}$ are block sizes.

```
for j=0,n,R do  //Loop 1
  for l=0,k,Q do  //Loop 2
    end
end
```

```
\begin{array}{c}
\text{Buffer A} \\
P \times Q \\
\text{Buffer B} \\
Q \times R \\
\end{array}
```

$m \times Q$  $m \times k$

$Q \times R$

$k \times R$
Blocked Matrix Multiplication

\[
\begin{align*}
&\text{P, Q, R, UNROLL are block sizes.} \\
&\text{for } j=0,n,R \text{ do } //\text{Loop 1} \\
&\text{for } l=0,k,Q \text{ do } //\text{Loop 2}
\end{align*}
\]
Blocked Matrix Multiplication

P, Q, R, UNROLL are block sizes.

```
for j=0,n,R do //Loop 1
  for l=0,k,Q do //Loop 2
    ...
  end
end
```

- Buffer A
- Buffer B

```
macro-kernel

P×Q  Q×R
m×Q
```

P, Q, R, UNROLL are block sizes.
Blocked Matrix Multiplication

P, Q, R, UNROLL are block sizes.

\[
\begin{align*}
&\text{for } j=0,n,R \text{ do } \text{//Loop 1} \\
&\text{for } l=0,k,Q \text{ do } \text{//Loop 2} \\
&\text{//Loop 3.1} \\
&\text{itcopy}(A[0,l], \text{buf } _A) \\
\end{align*}
\]
Blocked Matrix Multiplication

P, Q, R, UNROLL are block sizes.

```plaintext
for j=0,n,R do //Loop 1
for l=0,k,Q do //Loop 2
  //Loop 3.1
  itcopy(A[0,l], buf _A)
end
end
```
Blocked Matrix Multiplication

\[ P, Q, R, \text{UNROLL} \text{ are block sizes.} \]

\[
\text{for } j=0,n,R \text{ do } \text{//Loop 1} \\
\text{for } l=0,k,Q \text{ do } \text{//Loop 2} \\
\text{//Loop 3.1} \\
\text{itcopy}(A[0,l],\text{buf }_A) \\
\text{for } jj=j,j+R,\text{UNROLL} \text{ do } \text{//Loop 4} \\
\text{oncopy}(B[l,jj],\text{buf }_B) \\
\text{kernel}(\text{buf }_A,\text{buf }_B) \\
\text{end}
\]
Blocked Matrix Multiplication

P, Q, R, UNROLL are block sizes.

\[
\begin{align*}
\text{for } j=0, n, R & \text{ do } //\text{Loop 1} \\
\text{for } l=0, k, Q & \text{ do } //\text{Loop 2} \\
\text{//Loop 3.1} \\
\text{itcopy}(A[0, l], \text{buf } _A) \\
\text{for } jj=j, j+R, \text{UNROLL} & \text{ do } //\text{Loop 4} \\
\text{oncopy}(B[l, jj], \text{buf } _B) \\
\text{kernel}(\text{buf } _A, \text{buf } _B) \\
\text{end} \\
\text{end} \\
\text{end}
\end{align*}
\]

Buffer A

Buffer B

macro-kernel

P×Q

P×Q

Q×R

P×Q

Q×R

UNROLL
Blocked Matrix Multiplication

\[ \text{Buffer A} \times \text{Buffer B} \times (P \times Q) \times (Q \times R) \]

\( P, Q, R, \) UNROLL are block sizes.

\[
\begin{align*}
\text{for } j=0,n,R & \text{ do } /\!\!\!/ \text{Loop 1} \\
\text{for } l=0,k,Q & \text{ do } /\!\!\!/ \text{Loop 2} \\
\text{for } jj=j,j+R,UNROLL & \text{ do } /\!\!\!/ \text{Loop 3.1} \\
\text{itcopy}(A[0,l],buf_A) \\
\text{oncopy}(B[l,jj],buf_B) \\
\text{kernel}(buf_A,buf_B) \\
\text{end} \\
\text{for } i=0,m,P & \text{ do } /\!\!\!/ \text{Loop 3.2} \\
\text{itcopy}(A[i,l],buf_A) \\
\text{kernel}(buf_A,buf_B) \\
\text{end} \\
\text{end} \\
\text{end}
\]
Attacking Blocked Matrix Multiplication

• Extract matrix parameters by counting iterations of loops

```c
for j=0,n,R do  //Loop 1
  for l=0,k,Q do  //Loop 2
    //Loop 3.1
    itcopy(A[0,l], buf _A)
    for jj=j, j+R, UNROLL do  //Loop 4
      oncopy(B[l,jj], buf _B)
      kernel(buf_A, buf_B)
    end
  end
for i=0, m, P do  //Loop 3.2
  itcopy(A[i,l], buf _A)
  kernel(buf_A, buf_B)
end
end
```
Attacking Blocked Matrix Multiplication

• Extract matrix parameters by counting iterations of loops

```
for j=0,n,R do //Loop 1
  for l=0,k,Q do //Loop 2
    //Loop 3.1
    itcopy(A[0,l],buf_A)
    for jj=j,j+R,UNROLL do //Loop 4
      oncopy(B[l,jj],buf_B)
      kernel(buf_A,buf_B)
    end
  end
for i=0,m,P do //Loop 3.2
  itcopy(A[i,l],buf_A)
  kernel(buf_A,buf_B)
end
end
```

Dynamic Call Graph (DCG)
Attacking Blocked Matrix Multiplication

- Extract matrix parameters by counting iterations of loops

```plaintext
for j=0,n,R do //Loop 1
    for l=0,k,Q do //Loop 2
        //Loop 3.1
        itcopy(A[0,l],buf_A)
        for jj=j,j+R,UNROLL do //Loop 4
            oncopy(B[l,jj],buf_B)
            kernel(buf_A,buf_B)
        end
    end
    for i=0,m,P do //Loop 3.2
        itcopy(A[i,l],buf_A)
        kernel(buf_A,buf_B)
    end
end
```

Dynamic Call Graph (DCG)
Attacking Blocked Matrix Multiplication

- Extract matrix parameters by counting iterations of loops

```java
for j=0, n, R do //Loop 1
  for l=0, k, Q do //Loop 2
    //Loop 3.1
    itcopy(A[0,l], buf_A)
    for jj=j, j+R, UNROLL do //Loop 4
      oncopy(B[l, jj], buf_B)
      kernel(buf_A, buf_B)
  end
  for i=0, m, P do //Loop 3.2
    itcopy(A[i, l], buf_A)
    kernel(buf_A, buf_B)
  end
end
```

Dynamic Call Graph (DCG)
Attacking Blocked Matrix Multiplication

- Extract matrix parameters by counting iterations of loops

\[
\text{for } j=0, n, R \text{ do } //\text{Loop 1} \\
\text{for } l=0, k, Q \text{ do } //\text{Loop 2} \\
\quad \text{//Loop 3.1}
\quad \text{itcopy}(A[0, l], \text{buf } _A) \\
\quad \text{for } jj=j, j+R, \text{UNROLL } \text{do } //\text{Loop 4} \\
\quad \quad \text{oncopy}(B[l, jj], \text{buf } _B) \\
\quad \text{kernel}(\text{buf } _A, \text{buf } _B) \\
\text{for } i=0, m, P \text{ do } //\text{Loop 3.2} \\
\quad \text{itcopy}(A[i, l], \text{buf } _A) \\
\quad \text{kernel}(\text{buf } _A, \text{buf } _B) \\
\text{end}
\]

Both OpenBLAS and Intel MKL follow this DCG.
Number of Architectures to Search
Number of Architectures to Search
Number of Architectures to Search

<table>
<thead>
<tr>
<th></th>
<th>ResNet-50</th>
<th>VGG-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original: No Cache Telepathy</td>
<td>$&gt; 6 \times 10^{46}$</td>
<td>$&gt; 5.4 \times 10^{12}$</td>
</tr>
</tbody>
</table>
## Number of Architectures to Search

<table>
<thead>
<tr>
<th></th>
<th>ResNet-50</th>
<th>VGG-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original: No Cache Telepathy</td>
<td>$&gt; 6 \times 10^{46}$</td>
<td>$&gt; 5.4 \times 10^{12}$</td>
</tr>
<tr>
<td><strong>Flush+Reload</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenBLAS</td>
<td>512</td>
<td>16</td>
</tr>
<tr>
<td>MKL</td>
<td>6144</td>
<td>64</td>
</tr>
<tr>
<td><strong>Prime+Probe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenBLAS</td>
<td>512</td>
<td>16</td>
</tr>
<tr>
<td>MKL</td>
<td>$5.7 \times 10^{15}$</td>
<td>1936</td>
</tr>
</tbody>
</table>
Conclusion

• Cache-based side channel attacks can be used to efficiently recover DNN architectures

• DNN architectures have a multitude of parameters, which can be mapped to GEMM parameters

• The state-of-the-art blocked GEMM implementation can leak matrix parameters via cache access patterns