

Practical Padding Oracle Attacks

J. Rizzo T. Duong

Black Hat Europe, 2010

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

CBC Mode

- CBC mode is a mode of operation for a block cipher.
- Allows encryption of arbitrary length data.
- Encryption and decryption are defined by:

$$C_i = e_K(P_i \oplus C_{i-1})$$

$$P_i = d_K(C_i) \oplus C_{i-1}$$

CBC Mode

- CBC mode is a mode of operation for a block cipher.
- Allows encryption of arbitrary length data.
- Encryption and decryption are defined by:

$$C_i = e_K(P_i \oplus C_{i-1})$$

$$P_i = d_K(C_i) \oplus C_{i-1}$$

CBC Mode

- CBC mode is a mode of operation for a block cipher.
- Allows encryption of arbitrary length data.
- Encryption and decryption are defined by:

$$C_i = e_K(P_i \oplus C_{i-1})$$

$$P_i = d_K(C_i) \oplus C_{i-1}$$

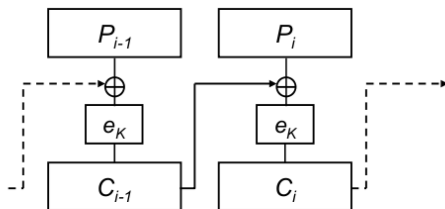
CBC Mode

- CBC mode is a mode of operation for a block cipher.
- Allows encryption of arbitrary length data.
- Encryption and decryption are defined by:

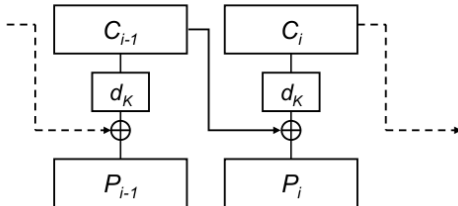
$$C_i = e_K(P_i \oplus C_{i-1})$$

$$P_i = d_K(C_i) \oplus C_{i-1}$$

CBC Mode Encryption and Decryption



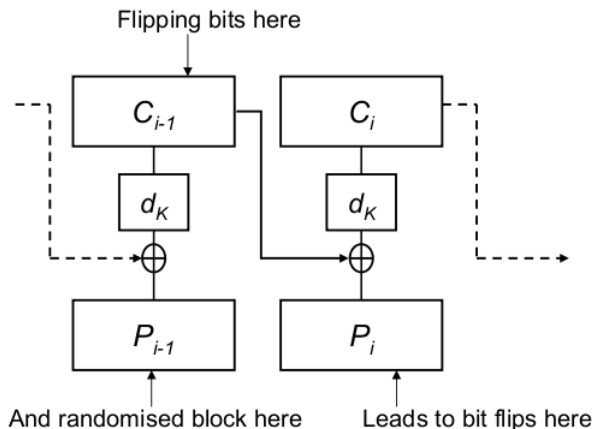
Typical block size n :
64 bits (DES, triple
DES) or 128 bits
(AES).



Typical key size:
56 bits (DES), 168 bits
(triple DES), 128, 192
or 256 bits (AES).

Bit Flipping in CBC Mode

- Flipping bits in C_{i-1} leads to controlled changes in P_i .
- Block P_{i-1} is garbled.



Padding in CBC Mode

- How should padding be added in CBC Mode?
- Numerous possibilities including:
 - Append unique removable pattern (“10...0” or “012...b” or “bb....b”).
 - Append or prepend length information in field of fixed size, pad remaining bits in fixed way (e.g. 0’s).
- Padding can also be used to enhance security:
 - Disguise the length of plaintexts.
 - Prevent traffic analysis, or guessing based on plaintext length.

Padding in CBC Mode

- How should padding be added in CBC Mode?
- Numerous possibilities including:
 - Append unique removable pattern (“10...0” or “012...b” or “bb....b”).
 - Append or prepend length information in field of fixed size, pad remaining bits in fixed way (e.g. 0’s).
- Padding can also be used to enhance security:
 - Disguise the length of plaintexts.
 - Prevent traffic analysis, or guessing based on plaintext length.

Padding in CBC Mode

- How should padding be added in CBC Mode?
- Numerous possibilities including:
 - Append unique removable pattern (“10...0” or “012...b” or “bb....b”).
 - Append or prepend length information in field of fixed size, pad remaining bits in fixed way (e.g. 0's).
- Padding can also be used to enhance security:
 - Disguise the length of plaintexts.
 - Prevent traffic analysis, or guessing based on plaintext length.

Padding in CBC Mode

- How should padding be added in CBC Mode?
- Numerous possibilities including:
 - Append unique removable pattern (“10...0” or “012...b” or “bb....b”).
 - Append or prepend length information in field of fixed size, pad remaining bits in fixed way (e.g. 0’s).
- Padding can also be used to enhance security:
 - Disguise the length of plaintexts.
 - Prevent traffic analysis, or guessing based on plaintext length.

Padding in CBC Mode

- Can padding have a negative impact on security?
- Vaudenay (Eurocrypt 2002) showed that padding oracles and bit flipping can be used to build decryption oracle for CBC mode.

Padding in CBC Mode

- Can padding have a negative impact on security?
- Vaudenay (Eurocrypt 2002) showed that padding oracles and bit flipping can be used to build decryption oracle for CBC mode.

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Padding Oracle attacks

- Two assumptions:
 - Adversary can intercept padded messages encrypted in CBC mode.
 - Adversary has access to a padding oracle.
- What is a padding oracle?
 - Adversary submits a CBC mode ciphertext C to oracle \tilde{D} .
 - Oracle decrypts under fixed key K and checks correctness of padding.
 - Oracle outputs VALID or INVALID according to correctness of padding:

$$\tilde{D}(C) = \begin{cases} 0, & \text{invalid} \\ 1, & \text{valid} \end{cases}$$

Padding Oracle attacks

- Two assumptions:
 - Adversary can intercept padded messages encrypted in CBC mode.
 - Adversary has access to a padding oracle.
- What is a padding oracle?
 - Adversary submits a CBC mode ciphertext C to oracle \tilde{D} .
 - Oracle decrypts under fixed key K and checks correctness of padding.
 - Oracle outputs VALID or INVALID according to correctness of padding:

$$\tilde{D}(C) = \begin{cases} 0, & \text{invalid} \\ 1, & \text{valid} \end{cases}$$

Padding Oracle attacks

- Two assumptions:
 - Adversary can intercept padded messages encrypted in CBC mode.
 - Adversary has access to a padding oracle.
- What is a padding oracle?
 - Adversary submits a CBC mode ciphertext C to oracle \tilde{D} .
 - Oracle decrypts under fixed key K and checks correctness of padding.
 - Oracle outputs VALID or INVALID according to correctness of padding:

$$\tilde{D}(C) = \begin{cases} 0, & \text{invalid} \\ 1, & \text{valid} \end{cases}$$

Padding Oracle attacks

- Two assumptions:
 - Adversary can intercept padded messages encrypted in CBC mode.
 - Adversary has access to a padding oracle.
- What is a padding oracle?
 - Adversary submits a CBC mode ciphertext C to oracle \tilde{D} .
 - Oracle decrypts under fixed key K and checks correctness of padding.
 - Oracle outputs VALID or INVALID according to correctness of padding:

$$\tilde{D}(C) = \begin{cases} 0, & \text{invalid} \\ 1, & \text{valid} \end{cases}$$

Padding Oracle attacks

Last word decryption algorithm

- pick a few random words r_1, \dots, r_b , and take $i = 0$.
- pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
- if $\delta(r|y) = 0$ then increment i and go back to previous step.
- replace r_b by $r_b \oplus i$.
- for $n = b$ down to 2
 - ① take $r = r_1 \dots r_{b-n} (r_{b-n+1} \oplus 1) r_{b-n+2} \dots r_b$
 - ② if $\delta(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \dots (r_b \oplus n)$
- output $r_b \oplus 1$.

Padding Oracle attacks

Last word decryption algorithm

- pick a few random words r_1, \dots, r_b , and take $i = 0$.
- pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
- if $\tilde{d}(r|y) = 0$ then increment i and go back to previous step.
- replace r_b by $r_b \oplus i$.
- for $n = b$ down to 2
 - 1 take $r = r_1 \dots r_{b-n} (r_{b-1+n} \oplus 1) r_{b-n+2} \dots r_b$
 - 2 if $\tilde{d}(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \dots (r_b \oplus n)$
- output $r_b \oplus 1$.

Padding Oracle attacks

Last word decryption algorithm

- pick a few random words r_1, \dots, r_b , and take $i = 0$.
- pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
- if $\tilde{d}(r|y) = 0$ then increment i and go back to previous step.
- replace r_b by $r_b \oplus i$.
- for $n = b$ down to 2
 - 1 take $r = r_1 \dots r_{b-n} (r_{b-n+1} \oplus 1) r_{b-n+2} \dots r_b$
 - 2 if $\tilde{d}(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \dots (r_b \oplus n)$
- output $r_b \oplus 1$.

Padding Oracle attacks

Last word decryption algorithm

- pick a few random words r_1, \dots, r_b , and take $i = 0$.
- pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
- if $\tilde{d}(r|y) = 0$ then increment i and go back to previous step.
- replace r_b by $r_b \oplus i$.
- for $n = b$ down to 2
 - 1 take $r = r_1 \dots r_{b-n} (r_{b-n+1} \oplus 1) r_{b-n+2} \dots r_b$
 - 2 if $\tilde{d}(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \dots (r_b \oplus n)$
- output $r_b \oplus 1$.

Padding Oracle attacks

Last word decryption algorithm

- pick a few random words r_1, \dots, r_b , and take $i = 0$.
- pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
- if $\tilde{d}(r|y) = 0$ then increment i and go back to previous step.
- replace r_b by $r_b \oplus i$.
- for $n = b$ down to 2
 - 1 take $r = r_1 \dots r_{b-n} (r_{b-n+1} \oplus 1) r_{b-n+2} \dots r_b$
 - 2 if $\tilde{d}(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \dots (r_b \oplus n)$
- output $r_b \oplus 1$.

Padding Oracle attacks

Last word decryption algorithm

- pick a few random words r_1, \dots, r_b , and take $i = 0$.
- pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
- if $\tilde{d}(r|y) = 0$ then increment i and go back to previous step.
- replace r_b by $r_b \oplus i$.
- for $n = b$ down to 2
 - 1 take $r = r_1 \dots r_{b-n} (r_{b-n+1} \oplus 1) r_{b-n+2} \dots r_b$
 - 2 if $\tilde{d}(r|y) = 0$ then stop and output $(r_{b-n+1} \oplus n) \dots (r_b \oplus n)$
- output $r_b \oplus 1$.

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Finding potential padding oracles

- Blackbox testing.
- Google hacking.
- Source code auditing.

Finding potential padding oracles

- Blackbox testing.
- Google hacking.
- Source code auditing.

Finding potential padding oracles

- Blackbox testing.
- Google hacking.
- Source code auditing.

Finding potential padding oracles

- Blackbox testing.
- Google hacking.
- Source code auditing.

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Confirm the existence of padding oracles

Determine the block size b

- All padding oracle attacks need a correct b .
- Most common block sizes are 8 and 16 bytes. Of course we can use trial and error.

How to determine the block size

- if $\text{len}(C) \% 16 = 8$, then stop and output 8.
- take $y = C[-16:]$, i.e. y is the last sixteen bytes of C .
- if $\delta(C|y) = 1$, then stop and output 8.
- output 16.

Confirm the existence of padding oracles

Determine the block size b

- All padding oracle attacks need a correct b .
- Most common block sizes are 8 and 16 bytes. Of course we can use trial and error.

How to determine the block size

- if $\text{len}(C) \% 16 = 8$, then stop and output 8.
- take $y = C[-16:]$, i.e. y is the last sixteen bytes of C .
- if $\tilde{0}(C|y) = 1$, then stop and output 8.
- output 16.

Confirm the existence of padding oracles

Determine the block size b

- All padding oracle attacks need a correct b .
- Most common block sizes are 8 and 16 bytes. Of course we can use trial and error.

How to determine the block size

- if $\text{len}(C) \% 16 = 8$, then stop and output 8.
- take $y = C[-16:]$, i.e. y is the last sixteen bytes of C .
- if $\tilde{d}(C|y) = 1$, then stop and output 8.
- output 16.

Confirm the existence of padding oracles

Determine the block size b

- All padding oracle attacks need a correct b .
- Most common block sizes are 8 and 16 bytes. Of course we can use trial and error.

How to determine the block size

- if $\text{len}(C) \% 16 = 8$, then stop and output 8.
- take $y = C[-16:]$, i.e. y is the last sixteen bytes of C .
- if $\delta(C|y) = 1$, then stop and output 8.
- output 16.

Confirm the existence of padding oracles

Determine the block size b

- All padding oracle attacks need a correct b .
- Most common block sizes are 8 and 16 bytes. Of course we can use trial and error.

How to determine the block size

- if $\text{len}(C) \% 16 = 8$, then stop and output 8.
- take $y = C[-16:]$, i.e. y is the last sixteen bytes of C .
- if $\tilde{d}(C|y) = 1$, then stop and output 8.
- output 16.

Confirm the existence of padding oracles

Determine the block size b

- All padding oracle attacks need a correct b .
- Most common block sizes are 8 and 16 bytes. Of course we can use trial and error.

How to determine the block size

- if $\text{len}(C) \% 16 = 8$, then stop and output 8.
- take $y = C[-16:]$, i.e. y is the last sixteen bytes of C .
- if $\tilde{d}(C|y) = 1$, then stop and output 8.
- output 16.

Confirm the existence of padding oracles

- We want the target to reveal as many different reactions to the modified ciphertexts as possible.
- The most important thing is to analyse and understand the meaning of these reactions. In short, you need to know when the padding is VALID, and when it's INVALID.
- POET a.k.a Padding Oracle Exploitation Tool will be released right after BH Europe 2010.

Confirm the existence of padding oracles

- We want the target to reveal as many different reactions to the modified ciphertexts as possible.
- The most important thing is to analyse and understand the meaning of these reactions. In short, you need to know when the padding is VALID, and when it's INVALID.
- POET a.k.a Padding Oracle Exploitation Tool will be released right after BH Europe 2010.

Confirm the existence of padding oracles

- We want the target to reveal as many different reactions to the modified ciphertexts as possible.
- The most important thing is to analyse and understand the meaning of these reactions. In short, you need to know when the padding is VALID, and when it's INVALID.
- POET a.k.a Padding Oracle Exploitation Tool will be released right after BH Europe 2010.

Confirm the existence of padding oracles

- Want to write your own tool to detect Padding Oracle? Follow this guideline (which is based on the algorithm in slide 22):
 - determine the block size b .
 - pick a few random words r_1, \dots, r_b , and take $i = 0$.
 - pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
 - Send $r|y$ to the target, where y is a valid ciphertext block. Record the value of i , content length, and content type of the response. Increment i , and go back to step 2 until $i > 255$.
 - Now you have 256 responses. If all of them are the same, then the target is not easily showing you that it is vulnerable to Padding Oracle attack.
 - Otherwise, look at each value of i where the responses are different from the rest. Examine carefully each response to see what happened.

Confirm the existence of padding oracles

- Want to write your own tool to detect Padding Oracle? Follow this guideline (which is based on the algorithm in slide 22):
 - determine the block size b .
 - pick a few random words r_1, \dots, r_b , and take $i = 0$.
 - pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
 - Send $r|y$ to the target, where y is a valid ciphertext block. Record the value of i , content length, and content type of the response. Increment i , and go back to step 2 until $i > 255$.
 - Now you have 256 responses. If all of them are the same, then the target is not easily showing you that it is vulnerable to Padding Oracle attack.
 - Otherwise, look at each value of i where the responses are different from the rest. Examine carefully each response to see what happened.

Confirm the existence of padding oracles

- Want to write your own tool to detect Padding Oracle? Follow this guideline (which is based on the algorithm in slide 22):
 - determine the block size b .
 - pick a few random words r_1, \dots, r_b , and take $i = 0$.
 - pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
 - Send $r|y$ to the target, where y is a valid ciphertext block. Record the value of i , content length, and content type of the response. Increment i , and go back to step 2 until $i > 255$.
 - Now you have 256 responses. If all of them are the same, then the target is not easily showing you that it is vulnerable to Padding Oracle attack.
 - Otherwise, look at each value of i where the responses are different from the rest. Examine carefully each response to see what happened.

Confirm the existence of padding oracles

- Want to write your own tool to detect Padding Oracle? Follow this guideline (which is based on the algorithm in slide 22):
 - determine the block size b .
 - pick a few random words r_1, \dots, r_b , and take $i = 0$.
 - pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
 - Send $r|y$ to the target, where y is a valid ciphertext block. Record the value of i , content length, and content type of the response. Increment i , and go back to step 2 until $i > 255$.
 - Now you have 256 responses. If all of them are the same, then the target is not easily showing you that it is vulnerable to Padding Oracle attack.
 - Otherwise, look at each value of i where the responses are different from the rest. Examine carefully each response to see what happened.

Confirm the existence of padding oracles

- Want to write your own tool to detect Padding Oracle? Follow this guideline (which is based on the algorithm in slide 22):
 - determine the block size b .
 - pick a few random words r_1, \dots, r_b , and take $i = 0$.
 - pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
 - Send $r|y$ to the target, where y is a valid ciphertext block. Record the value of i , content length, and content type of the response. Increment i , and go back to step 2 until $i > 255$.
 - Now you have 256 responses. If all of them are the same, then the target is not easily showing you that it is vulnerable to Padding Oracle attack.
 - Otherwise, look at each value of i where the responses are different from the rest. Examine carefully each response to see what happened.

Confirm the existence of padding oracles

- Want to write your own tool to detect Padding Oracle? Follow this guideline (which is based on the algorithm in slide 22):
 - determine the block size b .
 - pick a few random words r_1, \dots, r_b , and take $i = 0$.
 - pick $r = r_1 r_2 \dots r_{b-1} (r_b \oplus i)$.
 - Send $r|y$ to the target, where y is a valid ciphertext block. Record the value of i , content length, and content type of the response. Increment i , and go back to step 2 until $i > 255$.
 - Now you have 256 responses. If all of them are the same, then the target is not easily showing you that it is vulnerable to Padding Oracle attack.
 - Otherwise, look at each value of i where the responses are different from the rest. Examine carefully each response to see what happened.

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 **Basic PO attacks**
 - **Cracking CAPTCHA**
 - Decrypting JSF view states
- 4 Finding Padding Oracles
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Cracking CAPTCHA

A broken CAPTCHA system

- $ERC = e_{K,IV}(rand())$.
- `......`
- ERC is stored as either a hidden field or a cookie in the CAPTCHA form.
- Once a user submits, the server decrypts ERC , and compares it with the code that the user has entered. If equal, the server accepts the request; it denies the request otherwise.

Cracking CAPTCHA

A broken CAPTCHA system

- $ERC = e_{K,IV}(rand())$.
- `......`
- ERC is stored as either a hidden field or a cookie in the CAPTCHA form.
- Once a user submits, the server decrypts ERC , and compares it with the code that the user has entered. If equal, the server accepts the request; it denies the request otherwise.

Cracking CAPTCHA

A broken CAPTCHA system

- $ERC = e_{K,IV}(rand())$.
- `......`
- ERC is stored as either a hidden field or a cookie in the CAPTCHA form.
- Once a user submits, the server decrypts ERC , and compares it with the code that the user has entered. If equal, the server accepts the request; it denies the request otherwise.

Cracking CAPTCHA

A broken CAPTCHA system

- $ERC = e_{K,IV}(rand())$.
- `......`
- ERC is stored as either a hidden field or a cookie in the CAPTCHA form.
- Once a user submits, the server decrypts ERC , and compares it with the code that the user has entered. If equal, the server accepts the request; it denies the request otherwise.

Cracking CAPTCHA

Bypass the broken CAPTCHA system

- Since the system decrypts any *ERC* sent to it, it is vulnerable to Padding Oracle attack.
- The only remaining problem now is to know when padding is VALID, and when it's not.
- Fortunately, most CAPTCHA systems would send back an error notification when they fail to decrypt *ERC*, i.e. padding is INVALID.
- In addition, when we modify *ERC* so that the padding is VALID, most systems would display an image with a broken code.
- Now we have a Padding Oracle, and we can use it to decrypt any *ERC*, thus bypass the CAPTCHA completely.

Cracking CAPTCHA

Bypass the broken CAPTCHA system

- Since the system decrypts any *ERC* sent to it, it is vulnerable to Padding Oracle attack.
- The only remaining problem now is to know when padding is VALID, and when it's not.
- Fortunately, most CAPTCHA systems would send back an error notification when they fail to decrypt *ERC*, i.e. padding is INVALID.
- In addition, when we modify *ERC* so that the padding is VALID, most systems would display an image with a broken code.
- Now we have a Padding Oracle, and we can use it to decrypt any *ERC*, thus bypass the CAPTCHA completely.

Cracking CAPTCHA

Bypass the broken CAPTCHA system

- Since the system decrypts any *ERC* sent to it, it is vulnerable to Padding Oracle attack.
- The only remaining problem now is to know when padding is VALID, and when it's not.
- Fortunately, most CAPTCHA systems would send back an error notification when they fail to decrypt *ERC*, i.e. padding is INVALID.
- In addition, when we modify *ERC* so that the padding is VALID, most systems would display an image with a broken code.
- Now we have a Padding Oracle, and we can use it to decrypt any *ERC*, thus bypass the CAPTCHA completely.

Cracking CAPTCHA

Bypass the broken CAPTCHA system

- Since the system decrypts any *ERC* sent to it, it is vulnerable to Padding Oracle attack.
- The only remaining problem now is to know when padding is VALID, and when it's not.
- Fortunately, most CAPTCHA systems would send back an error notification when they fail to decrypt *ERC*, i.e. padding is INVALID.
- In addition, when we modify *ERC* so that the padding is VALID, most systems would display an image with a broken code.
- Now we have a Padding Oracle, and we can use it to decrypt any *ERC*, thus bypass the CAPTCHA completely.

Cracking CAPTCHA

Bypass the broken CAPTCHA system

- Since the system decrypts any *ERC* sent to it, it is vulnerable to Padding Oracle attack.
- The only remaining problem now is to know when padding is VALID, and when it's not.
- Fortunately, most CAPTCHA systems would send back an error notification when they fail to decrypt *ERC*, i.e. padding is INVALID.
- In addition, when we modify *ERC* so that the padding is VALID, most systems would display an image with a broken code.
- Now we have a Padding Oracle, and we can use it to decrypt any *ERC*, thus bypass the CAPTCHA completely.

Cracking CAPTCHA

CAPTCHA with secret IV

- Since $P_0 = IV \oplus d_{\delta}(C_0)$, we need to know the IV to get P_0 .
- If the IV is secret, we can't know P_0 , thus can't crack CAPTCHA systems whose P_0 contains part of the random code.
- The solution is: $IV = Human \oplus d_{\delta}(C_0)$, where *Human* denotes that somebody reads P_0 from the CAPTCHA image.

Cracking CAPTCHA

CAPTCHA with secret IV

- Since $P_0 = IV \oplus d_{\delta}(C_0)$, we need to know the IV to get P_0 .
- If the IV is secret, we can't know P_0 , thus can't crack CAPTCHA systems whose P_0 contains part of the random code.
- The solution is: $IV = Human \oplus d_{\delta}(C_0)$, where *Human* denotes that somebody reads P_0 from the CAPTCHA image.

Cracking CAPTCHA

CAPTCHA with secret IV

- Since $P_0 = IV \oplus d_{\delta}(C_0)$, we need to know the IV to get P_0 .
- If the IV is secret, we can't know P_0 , thus can't crack CAPTCHA systems whose P_0 contains part of the random code.
- The solution is: $IV = Human \oplus d_{\delta}(C_0)$, where *Human* denotes that somebody reads P_0 from the CAPTCHA image.

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 **Basic PO attacks**
 - Cracking CAPTCHA
 - **Decrypting JSF view states**
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Decrypting JSF view states

Introduction

- JavaServer Faces (JSF) is a popular Java-based standard for building server-side user interfaces.
- Like ASP.NET, JSF stores the state of the view in a hidden field.
- Although JSF specification advises that view state should be encrypted and tamper evident, but no implementation follows that advice.
- In other words, we can use Padding Oracle attacks to decrypt the view states of most JSF frameworks.

Decrypting JSF view states

Introduction

- JavaServer Faces (JSF) is a popular Java-based standard for building server-side user interfaces.
- Like ASP.NET, JSF stores the state of the view in a hidden field.
- Although JSF specification advises that view state should be encrypted and tamper evident, but no implementation follows that advice.
- In other words, we can use Padding Oracle attacks to decrypt the view states of most JSF frameworks.

Decrypting JSF view states

Introduction

- JavaServer Faces (JSF) is a popular Java-based standard for building server-side user interfaces.
- Like ASP.NET, JSF stores the state of the view in a hidden field.
- Although JSF specification advises that view state should be encrypted and tamper evident, but no implementation follows that advice.
- In other words, we can use Padding Oracle attacks to decrypt the view states of most JSF frameworks.

Decrypting JSF view states

Introduction

- JavaServer Faces (JSF) is a popular Java-based standard for building server-side user interfaces.
- Like ASP.NET, JSF stores the state of the view in a hidden field.
- Although JSF specification advises that view state should be encrypted and tamper evident, but no implementation follows that advice.
- In other words, we can use Padding Oracle attacks to decrypt the view states of most JSF frameworks.

Decrypting JSF view states

Padding Oracle in JSF frameworks

- By default, all JSF frameworks would display a very detailed error message if it fails to decrypt a view state.

Padding Oracle in default installations of JSF frameworks

- if we see `javax.crypto.BadPaddingException`, then it's INVALID padding
- it's VALID padding otherwise.

Decrypting JSF view states

Padding Oracle in JSF frameworks

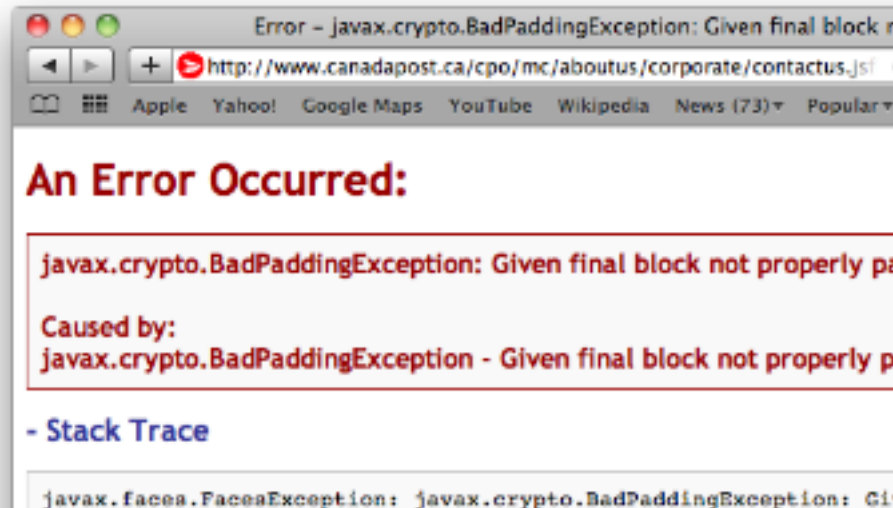
- By default, all JSF frameworks would display a very detailed error message if it fails to decrypt a view state.

Padding Oracle in default installations of JSF frameworks

- if we see **javax.crypto.BadPaddingException**, then it's INVALID padding
- it's VALID padding otherwise.

Decrypting JSF view states

Apache MyFaces error-page



Decrypting JSF view states

Padding Oracle in JSF frameworks

- Most JSF frameworks allow developers to turn off error messages. Then we can use the following simple trick:

Padding Oracle in JSF frameworks when error-page is turned off

- Say we want to decrypt block C_i of an encrypted view state $C_0|C_1|...|C_{n-1}$, then we send $C_0|C_1|...|C_{n-1}|C_{random}|C_i$ to the target.
- Since Java ignores those extra blocks while decrypting and deserializing view states, it's VALID padding if the target returns the same page as when the view state is unaltered.
- And it's probably INVALID padding if we see something else, e.g. a HTTP 500 error message.

Decrypting JSF view states

Padding Oracle in JSF frameworks

- Most JSF frameworks allow developers to turn off error messages. Then we can use the following simple trick:

Padding Oracle in JSF frameworks when error-page is turned off

- Say we want to decrypt block C_i of an encrypted view state $C_0|C_1|\dots|C_{n-1}$, then we send $C_0|C_1|\dots|C_{n-1}|C_{random}|C_i$ to the target.
- Since Java ignores those extra blocks while decrypting and deserializing view states, it's VALID padding if the target returns the same page as when the view state is unaltered.
- And it's probably INVALID padding if we see something else, e.g. a HTTP 500 error message.

Decrypting JSF view states

Padding Oracle in JSF frameworks

- Most JSF frameworks allow developers to turn off error messages. Then we can use the following simple trick:

Padding Oracle in JSF frameworks when error-page is turned off

- Say we want to decrypt block C_i of an encrypted view state $C_0|C_1|\dots|C_{n-1}$, then we send $C_0|C_1|\dots|C_{n-1}|C_{random}|C_i$ to the target.
- Since Java ignores those extra blocks while decrypting and deserializing view states, it's VALID padding if the target returns the same page as when the view state is unaltered.
- And it's probably INVALID padding if we see something else, e.g. a HTTP 500 error message.

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 **Advanced PO attacks**
 - **Using PO to encrypt**
 - Distributed cross-site PO attacks

Using PO to encrypt

An introduction to CBC-R

- CBC-R turns a decryption oracle into an encryption oracle.
- We all know that CBC decryption works as following:

$$P_i = d_K(C_i) \oplus C_{i-1}$$

$$C_0 = IV$$

- We can use a Padding Oracle to get $d_K(C_i)$, and we control C_{i-1} . In other words, we can produce any P_i as we want.

Using PO to encrypt

An introduction to CBC-R

- CBC-R turns a decryption oracle into an encryption oracle.
- We all know that CBC decryption works as following:

$$P_i = d_K(C_i) \oplus C_{i-1}$$

$$C_0 = IV$$

- We can use a Padding Oracle to get $d_K(C_i)$, and we control C_{i-1} . In other words, we can produce any P_i as we want.

Using PO to encrypt

An introduction to CBC-R

- CBC-R turns a decryption oracle into an encryption oracle.
- We all know that CBC decryption works as following:

$$P_i = d_K(C_i) \oplus C_{i-1}$$

$$C_0 = IV$$

- We can use a Padding Oracle to get $d_K(C_i)$, and we control C_{i-1} . In other words, we can produce any P_i as we want.

Using PO to encrypt

How CBC-R works

CBC-R pseudocode

- choose a plaintext message $P_0|...|P_{n-1}$ that you want to encrypt.
- pick a random C_{n-1} .
- for $i = n-1$ down to 1: $C_{i-1} = P_i \oplus d_0(C_i)$
- $IV = P_0 \oplus d_0(C_0)$
- output $IV|C_0|C_1|...|C_{n-1}$. This ciphertext would be decrypted to $P_0|...|P_{n-1}$.

Using PO to encrypt

How CBC-R works

CBC-R pseudocode

- choose a plaintext message $P_0|...|P_{n-1}$ that you want to encrypt.
- pick a random C_{n-1} .
- for $i = n-1$ down to 1: $C_{i-1} = P_i \oplus d_0(C_i)$
- $IV = P_0 \oplus d_0(C_0)$
- output $IV|C_0|C_1|...|C_{n-1}$. This ciphertext would be decrypted to $P_0|...|P_{n-1}$.

Using PO to encrypt

How CBC-R works

CBC-R pseudocode

- choose a plaintext message $P_0|...|P_{n-1}$ that you want to encrypt.
- pick a random C_{n-1} .
- for $i = n-1$ down to 1: $C_{i-1} = P_i \oplus d_0(C_i)$
- $IV = P_0 \oplus d_0(C_0)$
- output $IV|C_0|C_1|...|C_{n-1}$. This ciphertext would be decrypted to $P_0|...|P_{n-1}$.

Using PO to encrypt

How CBC-R works

CBC-R pseudocode

- choose a plaintext message $P_0|...|P_{n-1}$ that you want to encrypt.
- pick a random C_{n-1} .
- for $i = n-1$ down to 1: $C_{i-1} = P_i \oplus d_{\delta}(C_i)$
- $IV = P_0 \oplus d_{\delta}(C_0)$
- output $IV|C_0|C_1|...|C_{n-1}$. This ciphertext would be decrypted to $P_0|...|P_{n-1}$.

Using PO to encrypt

How CBC-R works

CBC-R pseudocode

- choose a plaintext message $P_0|...|P_{n-1}$ that you want to encrypt.
- pick a random C_{n-1} .
- for $i = n-1$ down to 1: $C_{i-1} = P_i \oplus d_{\delta}(C_i)$
- $IV = P_0 \oplus d_{\delta}(C_0)$
- output $IV|C_0|C_1|...|C_{n-1}$. This ciphertext would be decrypted to $P_0|...|P_{n-1}$.

Using PO to encrypt

CBC-R Without Controlling IV

- CBC-R allows us to encrypt any message, but if we cannot set the IV , then first plaintext block P_0 will be random and meaningless.
- If the victim expects the decrypted message to start with a standard header, then it will ignore the forged message constructed by CBC-R.
- We have not found generic way to overcome this limitation. However, we have found workarounds for particular cases.

Using PO to encrypt

CBC-R Without Controlling IV

- CBC-R allows us to encrypt any message, but if we cannot set the IV , then first plaintext block P_0 will be random and meaningless.
- If the victim expects the decrypted message to start with a standard header, then it will ignore the forged message constructed by CBC-R.
- We have not found generic way to overcome this limitation. However, we have found workarounds for particular cases.

Using PO to encrypt

CBC-R Without Controlling IV

- CBC-R allows us to encrypt any message, but if we cannot set the IV , then first plaintext block P_0 will be random and meaningless.
- If the victim expects the decrypted message to start with a standard header, then it will ignore the forged message constructed by CBC-R.
- We have not found generic way to overcome this limitation. However, we have found workarounds for particular cases.

Using PO to encrypt

CBC-R Without Controlling IV

Using captured ciphertexts as prefix

- $P_{valid} = d_K(C_{captured} || IV_{CBC-R} || P_{CBC-R})$.
- The block at the position of IV_{CBC-R} is still garbled.
- We can make the garbled block becomes part of some string that doesn't affect the semantic of the message such as comment or textbox label.

Using PO to encrypt

CBC-R Without Controlling IV

Using captured ciphertexts as prefix

- $P_{valid} = d_K(C_{captured} || IV_{CBC-R} || P_{CBC-R})$.
- The block at the position of IV_{CBC-R} is still garbled.
- We can make the garbled block becomes part of some string that doesn't affect the semantic of the message such as comment or textbox label.

Using PO to encrypt

CBC-R Without Controlling IV

Using captured ciphertexts as prefix

- $P_{valid} = d_K(C_{captured} || IV_{CBC-R} || P_{CBC-R})$.
- The block at the position of IV_{CBC-R} is still garbled.
- We can make the garbled block becomes part of some string that doesn't affect the semantic of the message such as comment or textbox label.

Using PO to encrypt

CBC-R Without Controlling IV

Brute-forcing C_0

- CBC-R can produce many different ciphertexts that decrypted to the same plaintext block chain P_{n-1}, \dots, P_1 . The only difference is the first plaintext block which is computed as following:

$$P_0 = d_K(C_0) \oplus IV$$

- A valid header means that the first few bytes of P_0 must match some magic numbers. There are also systems that accept a message if the first byte of its P_0 matches its size.
- If this is the case, and if the message is short enough, we can try our luck by brute-forcing C_0 .

Using PO to encrypt

CBC-R Without Controlling IV

Brute-forcing C_0

- CBC-R can produce many different ciphertexts that decrypted to the same plaintext block chain P_{n-1}, \dots, P_1 . The only difference is the first plaintext block which is computed as following:

$$P_0 = d_K(C_0) \oplus IV$$

- A valid header means that the first few bytes of P_0 must match some magic numbers. There are also systems that accept a message if the first byte of its P_0 matches its size.
- If this is the case, and if the message is short enough, we can try our luck by brute-forcing C_0 .

Using PO to encrypt

CBC-R Without Controlling IV

Brute-forcing C_0

- CBC-R can produce many different ciphertexts that decrypted to the same plaintext block chain P_{n-1}, \dots, P_1 . The only difference is the first plaintext block which is computed as following:

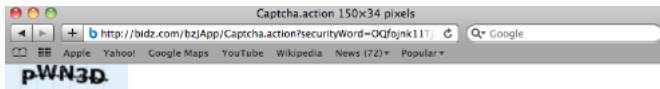
$$P_0 = d_K(C_0) \oplus IV$$

- A valid header means that the first few bytes of P_0 must match some magic numbers. There are also systems that accept a message if the first byte of its P_0 matches its size.
- If this is the case, and if the message is short enough, we can try our luck by brute-forcing C_0 .

Using PO to encrypt

CBC-R Applications

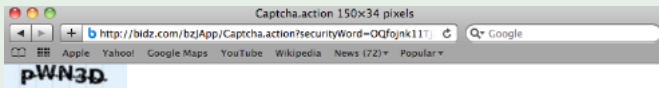
sudo make me a CAPCHA



Using PO to encrypt

CBC-R Applications

sudo make me a CAPCHA



Using PO to encrypt

CBC-R Applications

Creating malicious JSF view states

- Which view states to create?
- How to solve the garbled block problem?

Using PO to encrypt

CBC-R Applications

Creating malicious JSF view states

- Which view states to create?
- How to solve the garbled block problem?

Outline

- 1 Introduction
 - Review of CBC Mode
 - Padding Oracle attacks
- 2 Finding Padding Oracles
 - Find potential padding oracles
 - Confirm the existence of padding oracles
- 3 Basic PO attacks
 - Cracking CAPTCHA
 - Decrypting JSF view states
- 4 Advanced PO attacks
 - Using PO to encrypt
 - Distributed cross-site PO attacks

Distributed cross-site PO attacks

- All attackers need to exploit Padding Oracle is a single bit of information.
- Cross-domain information leakage bugs in web browsers can help.
- One example: ` + onerror()/onload()` events.
- if the image is loaded, then it's VALID padding; otherwise, it's INVALID padding.

Distributed cross-site PO attacks

- All attackers need to exploit Padding Oracle is a single bit of information.
- Cross-domain information leakage bugs in web browsers can help.
- One example: ` + onerror()/onload()` events.
- if the image is loaded, then it's VALID padding; otherwise, it's INVALID padding.

Distributed cross-site PO attacks

- All attackers need to exploit Padding Oracle is a single bit of information.
- Cross-domain information leakage bugs in web browsers can help.
- One example: ` + onerror()/onload()` events.
- if the image is loaded, then it's VALID padding; otherwise, it's INVALID padding.

Distributed cross-site PO attacks

- All attackers need to exploit Padding Oracle is a single bit of information.
- Cross-domain information leakage bugs in web browsers can help.
- One example: ` + onerror()/onload()` events.
- if the image is loaded, then it's VALID padding; otherwise, it's INVALID padding.

Distributed cross-site PO attacks

- We have decrypted all CAPTCHA on a web site using only JavaScript hosted locally.
- One can inject JavaScript code into popular web sites, and turn this into a distributed attack.
- It is possible to distributively build a code book.

Distributed cross-site PO attacks

- We have decrypted all CAPTCHA on a web site using only JavaScript hosted locally.
- One can inject JavaScript code into popular web sites, and turn this into a distributed attack.
- It is possible to distributively build a code book.

Distributed cross-site PO attacks

- We have decrypted all CAPTCHA on a web site using only JavaScript hosted locally.
- One can inject JavaScript code into popular web sites, and turn this into a distributed attack.
- It is possible to distributively build a code book.

Summary

- Padding oracle attacks allow one to decrypt ciphertext without knowing the key.
- We can use padding oracle attacks to crack CAPTCHA, and decrypt JSF view state, etc.
- CBC-R turns a decryption oracle into an encryption oracle, and allow us to create malicious JSF view states.
- Distributed cross-site padding oracle attacks allow one to distributively build a code book to map all ciphertexts to corresponding plaintexts.

Summary

- Padding oracle attacks allow one to decrypt ciphertext without knowing the key.
- We can use padding oracle attacks to crack CAPTCHA, and decrypt JSF view state, etc.
- CBC-R turns a decryption oracle into an encryption oracle, and allow us to create malicious JSF view states.
- Distributed cross-site padding oracle attacks allow one to distributively build a code book to map all ciphertexts to corresponding plaintexts.



Summary

- Padding oracle attacks allow one to decrypt ciphertext without knowing the key.
- We can use padding oracle attacks to crack CAPTCHA, and decrypt JSF view state, etc.
- CBC-R turns a decryption oracle into an encryption oracle, and allow us to create malicious JSF view states.
- Distributed cross-site padding oracle attacks allow one to distributively build a code book to map all ciphertexts to corresponding plaintexts.




Summary

- Padding oracle attacks allow one to decrypt ciphertext without knowing the key.
- We can use padding oracle attacks to crack CAPTCHA, and decrypt JSF view state, etc.
- CBC-R turns a decryption oracle into an encryption oracle, and allow us to create malicious JSF view states.
- Distributed cross-site padding oracle attacks allow one to distributively build a code book to map all ciphertexts to corresponding plaintexts.


For Further Reading I

-  Black and H. Urtubia. Side-Channel Attacks on Symmetric Encryption Schemes: The Case for Authenticated Encryption. In Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002, pages 327–338. USENIX, 2002.
-  K.G. Paterson and A. Yau. Padding Oracle Attacks on the ISO CBC Mode Padding Standard. In T. Okamoto, editor, Topics in Cryptology — CT-RSA 2004, volume 2964 of Lecture Notes in Computer Science, pages 305–323. Springer-Verlag, 2004.

For Further Reading II

-  S. Vaudenay. Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS...In L. Knudsen, editor, Advances in Cryptology — EUROCRYPT 2002, volume 2332 of Lecture Notes in Computer Science, pages 534–545. Springer-Verlag, 2002.
-  B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In Proc. CRYPTO 2003, D. Boneh (ed.), LNCS Vol. 2729, pp. 583–599, 2003.
-  V. Klima and T. Rosa. Side Channel Attacks on CBC Encrypted Messages in the PKCS#7 Format. Cryptology ePrint Archive, Report 2003/098, 2003.

For Further Reading III

-  A. K. L. Yau, K. G. Paterson, and C. J. Mitchell. Padding Oracle Attacks on CBC- Mode Encryption with Secret and Random IVs. In H. Gilbert and H. Handschuh, editors, Proceedings of FSE 2005, volume 3557 of LNCS, pages 299–319. Springer- Verlag, 2005.