מבוא למתקפת Padding Oracle

מאת דנור כהן / An7i

הקדמה

מתקפת Padding Oracle או בשמה המקוצר PO הינה מתקפה מסוג חדש יחסית המבוססת על חולשות שנתגלו במנגנוני הצפנה סימטריים, בשילוב עם חולשות שנתגלו באופן הניהול של שגיאות במערכות שונות. שילוב של שתי חולשות אלו, הביאו לעולם מספר מתקפות חדשות מבוססות Padding Oracle.

מתקפות כגון פריצת קאפצ"ות, עקיפת מנגנוני זיהוי על ידי פענוח עוגיות מוצפנות, וגולת הכותרת: קריאת קבצים מתיקיית השורש של השרת.

היום נציג מעט מושגים מקדימים בעולם ההצפנות עם דגש על החולשות במנגנוני ההצפנה וניהול השגיאות אשר הביאו לעולם את PO. יש לציין כי מתקפה זו הינה חדשה יחסית וכבר עושה הדים ברחבי העולם עקב הסכנות הטמונות בה. למשל, אחת האפשרויות לניצול פרצה זו הינה קריאת קבצים מסווגים על השרת, ובדוגמא היותר נפוצה במאמרים בעולם: את קובץ ה-web.config שיושב בתיקיית השורש בשרת. מבדיקה קצרה שעשיתי בנושא, נראה כי מאות ואלפים של אתרים גדולים ומוכרים עדיין אינם מוגנים מפרצה זו על אף שמיקרוסופט כבר שחררה patch בנושא.

לידתה של מתקפה זו החלה בשנת 2002 בכנס הקריפטוגרפיה מהגדולים בעולם : Eurocrypt. היה זה Serge Vaudenay הצרפתי, חוקר קריפטוגרפיה ידוע שהציג לראשונה את המתקפה. לטענתו של סרגיי (וכפי שלאחר מכן הוכיח), ניתן לנצל חולשה במנגנוני הצפנה כאשר הם מוגדרים לעבוד במתודולגיית הצפנה בשם Cipher-block chaining ומשתמשים בתקן PKCS#5 לצורך פעולה שנקראת Padding עליה נרחיב בהמשך, על מנת לפצח בלוקים של מחרוזות מוצפנות ללא ידיעת מפתח ההצפנה. דבר מרעיש כשלעצמו מכיוון שמנגנוני אבטחה רבים בעולם מיישמים מתודולוגיות אלו ובניהם Net. המוכרת לכולנו.

מבוא לתורת ההצפנה

בעידן התקשורת המוצפנת של ימינו ניתן לחלק את שיטת ההצפנה ל-2 חלקים מרכזיים, בהתבסס על סוגי המפתחות בהם נעשה השימוש: **מפתחות סימטריים** ו**-מפתחות אסימטריים**.

מפתח סימטרי פירושו שהצד המצפין והצד המפענח משתמשים באותו מפתח גם להצפנה וגם לפענוח, בעוד שבמפתחות אסימטריים נעשה שימוש בשני מפתחות לכל צד: מפתח פרטי ומפתח ציבורי לצד המצפין ומפתח פרטי וציבורי לצד המפענח.

במאמר שלפנינו נעסוק בחקר של רכיבי הצפנה ופענוח המשתמשים בתצורה של מפתחות סימטריים.

כיצד עובד בלוק הצפנה סימטרי: בלוק הצפנה סימטרי אינו מתוחכם במיוחד וכל יכולתיו מסתכמות במספר פעולות מתמטיות מוגדרות מראש. בלוק ההצפנה אינו יודע לעבוד עם קלט בלתי צפוי ולכן הקלט המוזן לתוכו חייב לעמוד בסטנדרטים שהוגדרו מראש כגון אורך בלוק ההצפנה או שיטת סימון סוף המחרוזת. לצורך העניין , נגדיר שבלוק ההצפנה יודע לקבל מחרוזת באורך של 8 בתים, לבצע עליה מספר פעולות מתמטיות ולאחר מכן הוא פולט את המחרוזת המוצפנת. אם נרצה להצפין את המחרוזת "Big dady" לא תיהיה לנו שום בעיה היות ואורכה בדיוק 8 בתים.

אך מה יקרה כאשר נרצה להצפין את המחרוזת "My big dady"? כיצד נשלח מחרוזת זו לבלוק הצפנה שיודע להצפין מחרוזות בגודל קבוע של 8 בתים? בדיוק בשביל כך נועדו רכיבי "Pre Encreption" המחלקים את המחרוזות לגודל קבוע של 8 בתים. אבל פה הבעיה לא נגמרת.

נאמר שחילקנו את המחרוזת ל-2 בתים של 8 כך:

"My big d", "adyDDDDD".

ונאמר שבלוק ההצפנה הצפין אותם וכעת הם נראים כך:

F851D6CC68FC9537, 7B216A634951170F.

כיצד נוכל לאחר הפענוח לחבר את המחרוזת חזרה לצורתה המקורית מבלי לדעת היכן היא נגמרת? צריך סימן שיאמר לנו היכן נגמרת המחרוזת, ובכל זאת לשמור על גודל קבוע של 8 בתים.

על מנת לענות על בעיה זו, ישנה טבלה בתקן PKCS#5 המגדירה בדיוק כיצד למלא את הבתים הריקים על מנת להגיע לבלוק של 8 בתים, ועל הדרך להגדיר היכן נגמרת המחרוזת.

ראשית כל, נזכיר שפעולת המילוי של הבלוק בתווים על מנת שיתאים בדיוק לבלוק של 8 בתים

נקראת Padding, מלשון ריפוד. ריפוד של החללים הריקים לפי התקן. ומכאן נגזרת שמה של המתקפה Padding Oracle. כאשר Padding מתייחס לפעולת הריפוד, והמונח Oracle מתייחס כביכול לגילוי - גילוי התוכן המוצפן על ידי חולשה במנגנון ה-Padding.

כעת נחזור אל התקן. על פי תקן ה-PKCS#5, כאשר מתקבלת מחרוזת קטנה מ-8 בתים יש למלא את שאר הבלוק בתווים זהים כאשר ערך כל אחד מהם שווה למספר הבתים המרוצפים. לדוגמא, מחרוזת המכילה 6 תווים תוכנס לבלוק של 8 ויתווספו אליה עוד שני תווים עם הערך 0x02 כיוון שריצפנו שני חללים, לעומת זאת, מחרוזת בת 5 תווים תוכנס לבלוק של 8 תווים ותורצף עם 3 תווים של 0x03. התמונה הבאה תמחיש יותר מכל את עניין הריפוד:

	BLOCK #1								BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04

(נלקח מ: GDSSecurity.com)

כפי שחדי האבחנה בינכם כבר הבינו, כאשר מתקבלת מחרוזת המכילה בדיוק 8 תווים מתווסף לבלוק הראשון בלוק שני שכולו מלא בתווים בעלי הערך 0x08. כך מוגדרת פעולת הריפוד על פי תקן PKCS#5.

כעת כשהבנו כיצד עובדת פעולת הריפוד (Padding), נחזור אל מנגנון ההצפנה. כפי שהוסבר לעיל החולשה חלה על מנגנוני הצפנה שמצפינים בשיטת CBC MODE, הפועלת בדרך הבאה: כאשר מגיעה מחרוזת רנדומלית כלשהי, היא עוברת חלוקה לבלוקים של 8 בתים (כאן המקום לאמר שבלוקים של 8

הם נפוצים אך לא היחידים, קיימים גם בלוקים של 16 ועוד) לאחר החלוקה לבלוקים, מבוצעת פעולת הריפוד שהוסברה לעיל. ולאחר מכן מוכנסת המחרוזת אל תוך בלוק הקידוד ועוברת קידוד.

במתודולגיית CBC MODE, עוד לפני שלב ההצפנה המתבצע בעזרת מנגנון TRIPLE DES, עובר הבלוק שלנו פעולה מתמטית בשם XOR עם מחרוזת רנדומלית ראשונית, ורק לאחר מכן עובר את שלב ההצפנה, לאחר מכן הבלוק השני שיבוא אחריו יעבור שוב XOR אבל לא עם ההמחרוזת הראשונית, אלא עם המחרוזת המוצפנת שיצאה מתהליך ההצפנה של הבלוק הראשון, וכך הלאה: כל בלוק עובר XOR עם הבלוק המוצפן שקדם לו ולאחר מכן הצפנה. למען הסדר, נכתוב את הדברים על פי הסדר וכמובן נציג תמונה שתסביר את הכל הרבה יותר טוב:

1. מחרוזת מפורקת לבלוקים בעלי גודל אחיד (במקרה שלנו 8).
2. הבלוקים עוברים תהליך ריפוד ומתמלאים בתווים על פי התקן.
3. הצד המצפין שולח אל בלוק ההצפנה מחרוזת ראשונית רנדומלית ואת הבלוק הראשון.
4. הבלוק הראשון עובר XOR עם המחרוזת הראשונית.
5. הבלוק הראשון (שעבר XOR) עובר תהליך הצפנה בעזרת TRIPLE DES.
6. הבלוק השני נכנס ועובר XOR מול הבלוק הראשון .
7. הבלוק השני (המקוסר) עובר תהליך הצפנה בעזרת TRIPLE DES.
8. וכך הלאה כל בלוק מקוסר בבלוק שקדם לו פרט לראשון שקוסר מול מחרוזת ראשונית.

תמונה אחת שווה אלף מילים:

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Initialization Vector	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Plain-Text (Padded)	B	R	I	A	N	;	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES								TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Encrypted Output (HEX)	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

(נלקח מ: GDSSecurity.com)

Initialization vector: המחרוזת הראשונית עליה דיברנו.

Plain-text: המחרוזת שלנו לאחר שנכנסה לבלוק 8.

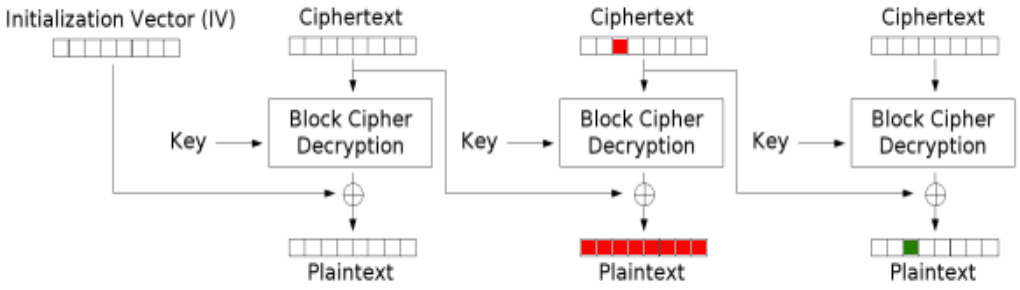
Intermediary value: המחרוזת שלנו לאחר שעברה קסור אל מול המחרוזת הראשונית.

Encrypted output: המחרוזת המוצפנת שלנו.

פעולת הפיענוח זהה לפעולת ההצפנה, רק בדיוק בסדר הפוך:

1. מחרוזת מוצפנת בגודל 8 בתים נשלחת אל בלוק הפענוח ביחד עם מחרוזת ראשונית (IV).
2. הבלוק הראשון עובר פיענוח בעזרת מנגנון TRIPLE DES עם מפתח ההצפנה.
   (זוכרים שמדובר בהצפנה סימטרית? המפתח המשמש להצפנה משמש גם לפענוח).
3. המחרוזת המפוענחת עוברת קסור אל מול המחרוזת הראשונית (IV).
4. הבלוק השני נכנס לבלוק הפענוח ועובר תהליך פענוח בעזרת TRIPLE DES.
5. המחרוזת המפוענחת עוברת קסור אל מול הבלוק הראשון(במצבו המוצפן).

על מנת להמחיש את תהליך הפענוח בצורה ברורה יותר הבא ונביט בתרשים הבא:



Cipher Block Chaining (CBC) mode decryption

(נלקח מ: GDSSecurity.com)

למדנו מהו בלוק הצפנה סימטרי, מהי פעולת הריפוד (Padding) ולמדנו על שיטת ההצפנה Cipher-block chaining. כבר בשלב הזה אנחנו יודעים מהי תחילתה של החולשה במנגנון זה:

- עצם זה שהמחרוזת המוצפנת שלנו עוברת תהליך מתמטי בטרם הצפנתה עם מחרוזת ראשונית שאנחנו יכולים לספק הינה כבר גורם אחד בעייתי.
- עצם זה שמנגנון ההצפנה לא שואל שאלות ולא מבקש שום הזדהות אלא פשוט מקבל קלט ומצפין אותו הוא גורם שני בעייתי שינוצל בהמשך לטובתנו.

כעת מה שנשאר להבין על מנת להרכיב את הפאזל השלם של ה-Padding Oracle הוא להבין מהו הגורם השלישי והמכריע שבעזרתו אנחנו מנצלים את כל האמור לעיל לטובתנו.

טיפול בשגיאות

כעת נסקור טיפול שגוי בשגיאות ריפוד (Padding) שבסופו של דבר מובילות אל ניצול החשיפה.

נניח שיש לנו אפליקציה שמקבלת מחרוזת מוצפנת, מפענחת אותה ומזריקה את הערכים שהועברו הלאה להמשך התהליך שלשמה נוצרה. בתרחיש קלאסי של Padding Oracle ישנם שלושה תרחישים אפשריים עיקריים:

1. אנחנו שולחים לאפליקציה מחרוזת שעברה חלוקה לבלוקים, רופדה והוצפנה כראוי והערכים שבהם השתמשנו בתוך המחרוזת המוצפנת תואמים לערכים להם מצפה האפליקציה. במקרה כזה אמורה האפליקציה להחזיר תגובה של: "200 ok".
2. אנחנו משתמשים בערכים שלהם מצפה האפליקציה כמו בדוגמא הראשונה, אך אנחנו מחבלים בתהליך הריפוד על ידי הזרקת מחרוזת ראשונית שגויה (מחרוזת ראשונית הוסברה לעיל). בסופו של תהליך הפענוח האפליקציה מגלה כי הריפוד שגוי ואינו מכיל תווים לפי תקן PKCS#5, וזורקת שגיאת: "500 internal server error cryptographic exception".
3. אנחנו שולחים ערכים שלהם האפליקציה לא מצפה במחרוזת המוצפנת אך לא מתערבים בתהליך הריפוד, לאחר הפענוח האפליקציה אינה זורקת שגיאה היות והריפוד נעשה כהלכה ולכן נקבל שוב תגובת: "200 ok".

מכאן ניתן להסיק, שאנחנו יכולים לדעת על פי תגובת האפליקציה האם הריפוד נעשה כהלכה או לא.

פרקטיקה

על מנת להבין את דוגמאות הפרקטיקה, יש תחילה לסקור שני מנגנונים להפניית משאבים בשם WebResource.axd ו-ScriptResource.axd. על קצה המזלג, מדובר בשני מנגנונים שתפקידם לבצע קריאה למשאבים מהשרת, משאבים אלו יכולים להיות כמעט כל דבר, בין תמונה, סקריפט, קובץ css ועוד:

- **WebResource.axd** מיועד בעיקר לצורכי משיכת משאבים בינאריים, כגון תמונות, וידאו וכו.
- **ScriptResource.axd** כשמו כן הוא, מיועד למשוך סקריפטים, כגון קבצי jsp מהשרת.

לא ניכנס כרגע לסיבות לשימושים במשאבים בצורה זו, רק נאמר שעל מנת להשתמש במנגנונים אלו כדי למשוך משאבים מהשרת, יש תחילה לבצע קישור של המשאב המבוקש אל סיפריית (DLL) בשרת, ולהגדיר שם כמה פרמטרים ובין היתר את השם של המשאב ואת סוגו (TYPE).

כאשר אנו מבצעים קריאה מפונקציה באתר למשאב מסויים בשרת באמצעות מנגנונים אלו, הפונקציה מייצרת את הלינק המוכר שלנו שיראה פחות או יותר כך:

```
http://www.myapp.com/WebResource.axd?d=D861D7CB65FC6253F851D6CC68FC9537
&t=345345
```

בכחול מסומנת המחרוזת הראשונית שדובר עליה לעיל, והמחרוזת האדומה למעשה היא שם המשאב וסוגו בצורה מוצפנת. מה שאנחנו מצליחים לפענח, הוא למעשה את שם המשאב (במקרה ספציפי זה בו נעשה שימוש במנגנוני משאבים).

בשלב זה כדאי לציין כי בין WebResource.axd ו-ScriptResource.axd אין הבדל ממשי, פרט לצורת הטיפול במקרים של שגיאות מסויימות.

כך שכל עוד האפשרות של custom errors כבויה בשרת נקבל את אותן השגיאות ללא שום הבדל, אך במקרים בהם custom errors מאופשר בשרת, ההעדפה לשימוש פשוט יותר תהיה ב-WebResource.axd.

נצלול קצת לפרטים:

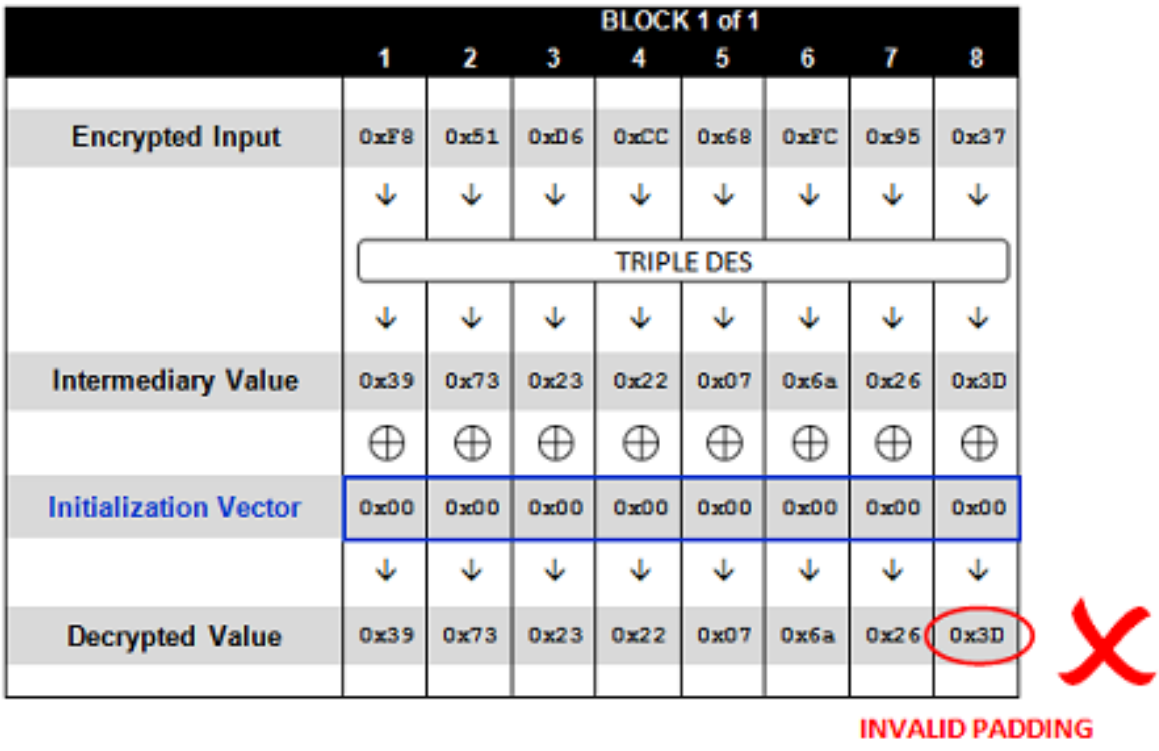
ראשית כל, יש לנו את האפליקציה בשרת שמקבלת את המחרוזת המוצפנת ואת הערך הראשוני ומפענחת את המחרוזת שלנו,הערך הראשוני צבוע בכחול, המחרוזת לפענוח צבועה באדום. בואו נראה מה יקרה כאשר נשלח לה מחרוזת ראשונית שכולה אפסים:

```
http://www.myapp.com/WebResource.axd?d=0000000000000000F851D6CC68FC9537
```

את המחרוזת הראשונית צבעתי בכחול ואת המחרוזת המוצפנת שלנו צבעתי באדום. את המחרוזות האלו נשלח בצורה עוקבת כפי שמודגם לעיל, האפליקציה תפרק אותם לבד לבלוקים של 8 תווים, ותתייחס לבלוק הראשון שכולו 0 כאל המחרוזת הראשונית היות והיא אכן הראשונה שנשלחת.

מדובר בערכי HEX ולכן כל בית מיוצג פה בשני תווים: 00=0 ו-01=1.

כעת נראה תמונת מצב של אופן הטיפול במחרוזות ששלחנו לאפליקציה:



(נלקח מ: GDSSecurity.com)

ניתן לראות בבירור את תהליך הפענוח שדומה מאוד לתהליך ההצפנה רק בסדר הפוך.

1. המחרוזת המוצפנת שלנו (הצבועה באדום) עוברת תהליך פענוח על ידי TRIPLE DES עם אותו המפתח שאיתו עברה הצפנה.

2. המחרוזת החצי מפוענחת עוברת XOR עם המחרוזת הראשונית שלנו (הצבועה בכחול).

3. מתקבלת מחרוזת מפוענחת אבל ישנה שגיאה. היות והמערכת קיבלה רק בלוק אחד של מידע היא מצפה לבלוק עוקב אחריו שיכיל כולו 0x08 כפי שהסברנו לעיל, ומשלא קיבלה אחד כזה היא מניחה שמדובר במחרוזת קצרה ולכן מחפשת בבלוק הראשון עצמו, את התווים שיכריזו על סיום המחרוזת ואורכה, כגון 0x01 או שני תווים של 0x02, ומשלא מצאה גם את אלה מחזירה לנו שגיאת 500.

אם נביט היטב, נצליח להבין שהספרה האחרונה של המחרוזת הראשונית שלנו היא זו שבסופו של דבר תקבע את ערכו של הבית האחרון במחרוזת. ואם רק נצליח למצוא את השילוב הנכון שבסופו של תהליך יפיק בבית האחרון את הערך 0x01 הרי שנקבל תגובת 200 מכיוון שיזוהה ריפוד נכון.

מכאן מתחיל תהליך של Brute Force, מתחילים להעלות את ערכו של הבית האחרון במחרוזת הראשונית, כל פעם ב-1 (תחילה 0x00 ולאחר מכן 0x01 עד שנגיע אל 0xFF, ישנן בסך הכל 255 אפשרויות לבדיקה).

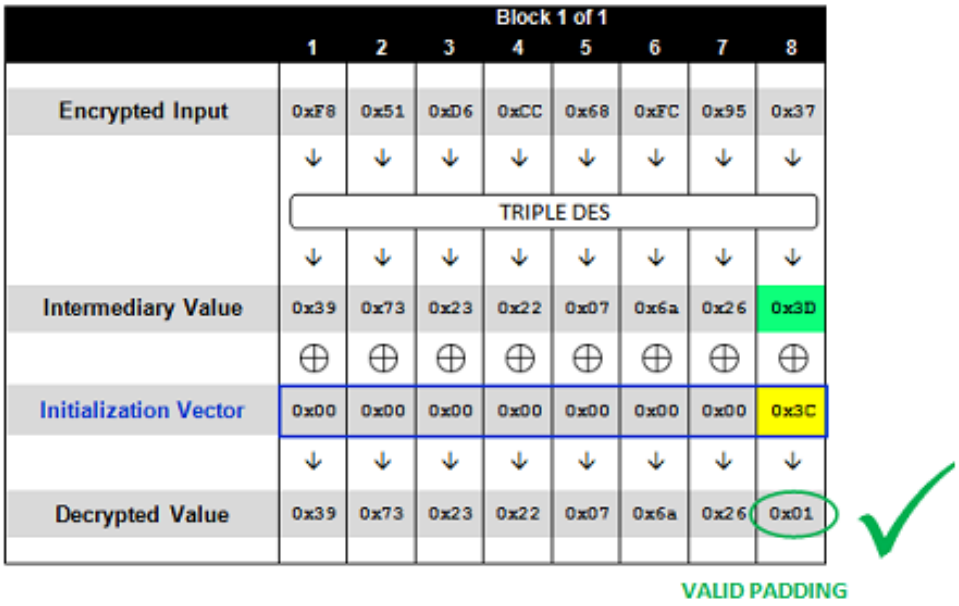
לאחר נסיונות רבים גילינו שכאשר אנחנו מזינים את הערך 0x3C לבית האחרון של המחרוזת הראשונית, מתקבלת לפתע תגובת 200. אנחנו יכולים להסיק כי:

הבית האחרון שפוענח על ידי TRIPLE DES > XOR > הערך 0x3C למעשה שווה ל-0x01. ולכן על מנת לגלות את הערך השמיני בבלוק המפוענח , בטרם בוצע עליו ה-XOR כל שעלינו לעשות הוא:

```
0×3C XOR 0×01
```

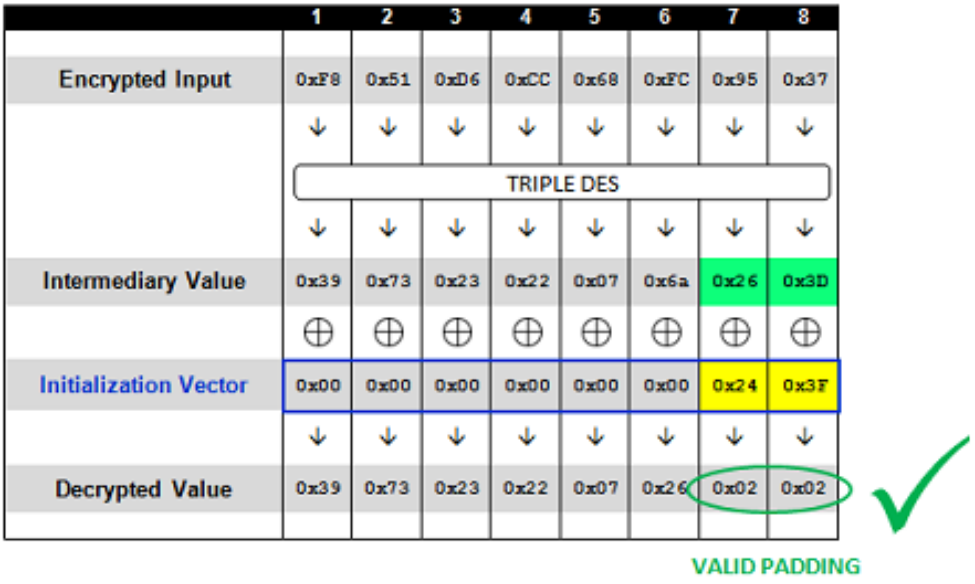
והתשובה כמובן: 0x3D

תמונה, אלף מילים:



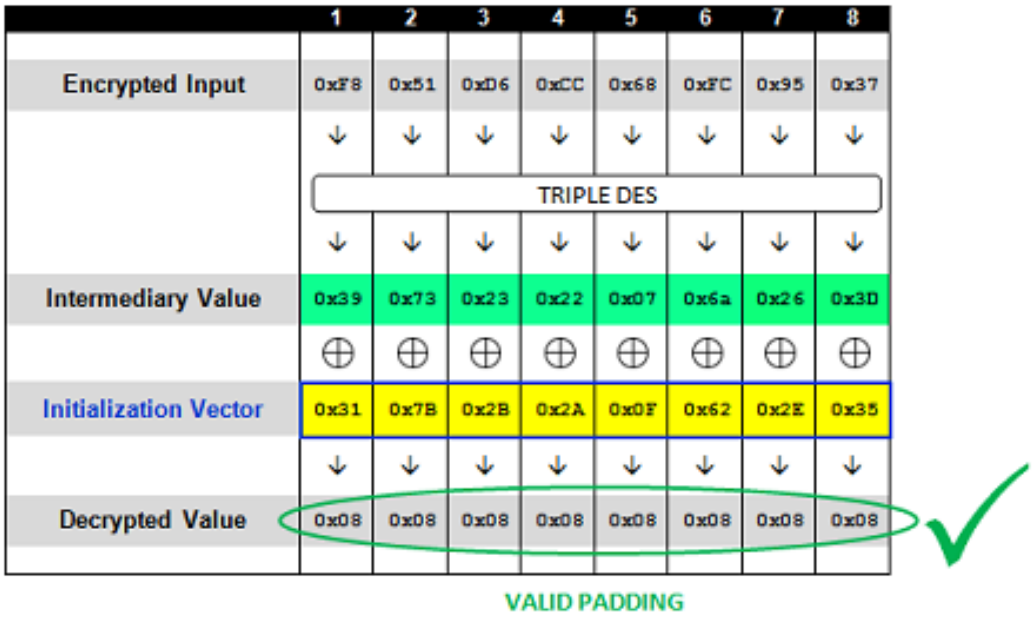
(נלקח מ: GDSSecurity.com)

בשיטה זו ניתן לגלות את כל הערך האמצעי, זה שעבר תהליך פיענוח אבל עוד לא עבר xor, פשוט צריך לזכור שכאשר מנסים לפרוץ את הערך הבא, השאיפה היא להגיע בתוצאה לשני בתים המכילים 0x02 כפי שמודגם באיור:



(נלקח מ: GDSSecurity.com)

ולבסוף , פיצוח כל המחרוזת האמצעית כך :



(נלקח מ: GDSSecurity.com)

כעת תחשבו על זה: אם יש לנו את המחרוזת האמצעית (הצבועה בירוק) על ידי השיטה שכרגע למדנו, אנחנו יכולים לשלוט על הפלט הסופי בכך שנשנה את המרוזת הראשונית (הצבועה בצהוב בתרשים לעיל או בכחול בלינק למטה):

```
http://www.myapp.com/WebResource.axd?d=317B2B2A0F622K35F851D6CC68FC9537
```

איך נגלה את הטקסט המקורי שהוצפן במחרוזת, בטרם התחלנו להתערב ושינינו את המחרוזת הראשונית? חוזרים למנגנון ה-CBC: למדנו, שלפי מתודולוגיית ההצפנה והפענוח של מנגנון זה, המחרוזת הראשונה מבצעת XOR עם הערך הראשוני וכל שאר המחרוזות מבצעות XOR אל המחרוזות שקדמו להן, כך שכל שנותר לנו לעשות על מנת לגלות את המחרוזת המקורית הוא לבצע XOR בין:

הערך האמצעי שגילינו למעלה:

```
39732322076A263D
```

עם המחרוזת הראשונית המקורית שבאה עם הלינק:

```
7B216A634951170F
```

שלא נתבלבל, זהו הלינק המקורי:

```
http://www.myapp.com/WebResource.axd?d=7B216A634951170FF851D6CC68FC9537
```

הכחול מורה על המחרוזת הראשונית והאדום על המחרוזת המוצפנת.

בשלב זה הבנו כיצד ניתן לפצח מחרוזות מוצפנות באופן זה ללא ידיעת המפתח. עכשיו חישבו על זה, כמה מנגנונים שלמים עובדים בצורה כזו, והכל חשוף כעת. אבל זוהי אינה גולת הכותרת, המתקפה הבאה עליה נלמד היא למעשה מבוססת Padding Oracle.

CBC-R

שמה של המתקפה החדשה נגזר משמו של מנגנון ההצפנה שעליו למדנו לעיל בתוספת האות R (המסמלת ככל הנראה Re Encryption), היות ומשתמשים במערכת ההצפנה והפענוח על מנת להצפין מחרוזות זדוניות שלנו. מי שגילה את המתקפה הזו הם ככל הנראה צמד החוקרים Juliano Rizzo ו- Thai Duong אשר הציגו אותה לראשונה בכנס Black hat במסמך בשם "Practical Padding Oracle Attacks", והם למעשה אלו שנתנו למתקפה את שמה.

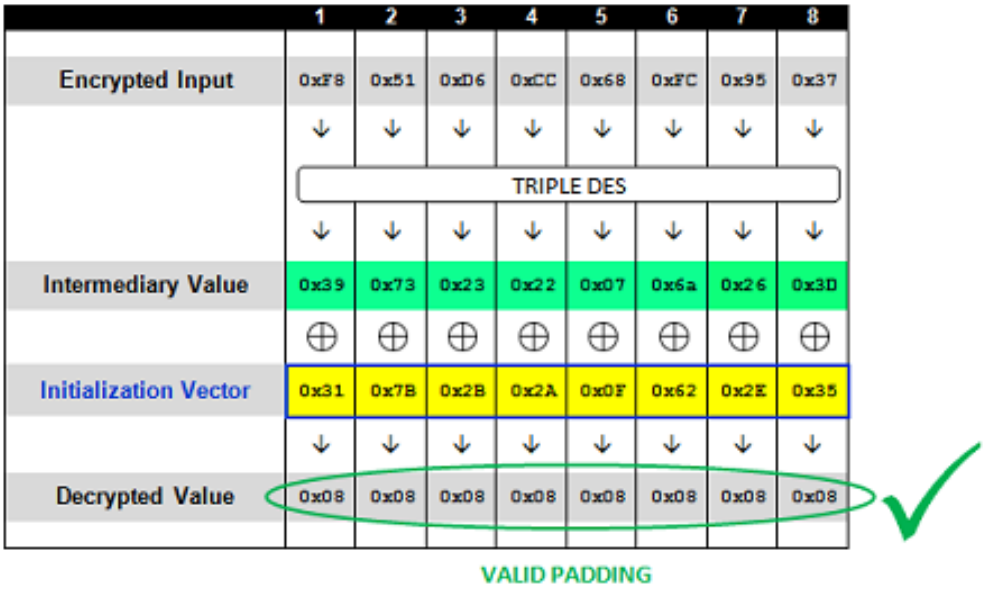
את המסמך למי שמעוניין ניתן למצוא פה:

http://media.blackhat.com/bh-eu-10/whitepapers/Duong_Rizzo/BlackHat-EU-2010-Duong-Rizzo-Padding-Oracle-wp.pdf

הסבר על המתקפה, כיצד היא מבוצעת ומה ניתן להשיג בעזרתה

הרעיון מאחורי מתקפה זו היה יחסית פשוט, אך עם זאת עוצמתי. עד לכאן הרעיון היה פשוט: יש לנו מחרוזת מוצפנת, אנחנו יודעים לגלות את המחרוזת האמצעית. וכפי שהסברנו: **המחרוזת האמצעית XOR המחרוזת הראשונית = מחרוזת האמיתית שהצפנה**.

על מנת שההסבר יהיה מובן אנו נצרף את תמונת הפענוח שנית:



(נלקח מ: GDSSecurity.com)

הסבר:

1. השורה המסומנת בצהוב בשליטתנו (זו שלמעלה בציור וזו שלמטה בלינק הם אותה מחרוזת):

```
http://www.myapp.com/WebResource.axd?d=317B2B2A0F622F35F851D6CC68FC9537
```

2. המחרוזת הירוקה למעלה לא בשליטתנו, אבל אנחנו כבר יודעים מהי כפי שהסברנו למעלה למעשה זוהי המחרוזת זהב שלנו שהצלחנו לפצח עקב חולשת (PADDING ORACLE)
3. השורה השלישית המוקפת בעיגול ירוק היא למעשה המחרוזת המפוענחת, עכשיו אנחנו כבר מבינים שאפילו היא בשליטתנו, בגלל שיש לנו את המחרוזת הצהובה(IV).
4. היות ואנחנו שולטים בתוצאה, אנחנו יכולים להחליט על כל ערך שאנו רוצים שיצא.
5. scriptresource.axd כפי שהוסבר כבר לעיל, נועד על מנת למשוך משאבים מהשרת ולהחזיר אותם ללקוח.
6. בתחילה, אנו מבצעים מתקפה על האתר של הלקוח, מפענחים את המחרוזת האמצעית.
7. משנים את התוצאה הסופית כך שתהיה לדוגמא web.config.
8. השרת מקבל את המחרוזת המוצפנת עם הערך הראשוני שאנחנו שולטים בו.
9. מבצע פענוח למחרוזת ורואה שהלקוח מבקש את הקובץ web.config .
10. השרת שולח לנו אותו.

הערה: חייבים לציין שישנן מערכות רבות בהם לא תהיה לנו שליטה על המחרוזת הראשונית בגלל ארכיטקטורת המערכת. אני אומר זאת מפני שאני מניח כי הרבה מהקוראים כבר נתקלו ב-Exploit של Brian Holyfield העונה לשם Padbuster, ועלולים לראות את ה-Payload:

```
|||~/web.config
```

או לחילופין לראות את כל הנושא של הבדיקה שאין במחרוזת Pipes וכאלה. אז לכל הקוראים האלו אני אומר לא לדאוג, זה לא שחסר משהו במאמר או בהבנה, אנחנו דיברנו בנושא Padding Oracle קלאסי פלוס, והאקספלויט הזה בנוי למצבים בהם גם אין לנו שליטה על המחרוזת הראשונית.

אם תיהיה היענות בנושא וידווח שהמאמר הזה היה מועיל ומובן, אני אכתוב מאמר המשך למאמר זה בו נסקור את הבעיות שצצו בעת ניצול פרצה זו וכיצד התגברו עליהם. כמו כן, נסקור את המתקפות השונות שפותחו בתקופה האחרונה, הנעזרות בחולשת padding oracle.

כיום קיימים כבר מספר כלים שיודעים לבצע אוטומציה לכל התהליך ובנייהם ה-Exploit המפורסם padbuster v3 שניתן להורדה פה:

http://www.gdssecurity.com/l/b/2010/10/04/padbuster-v0-3-and-the-net-padding-oracle-attack/

אני מקווה שהמאמר שפך מעט אור על המתקפה הכל כך מעניינת הזאת.

An7i.