



Zombie POODLE, **GOLDENDOODLE, & How TLSv1.3 Can Save Us All**

Craig Young, Principal Security Researcher, Tripwire VERT



Craig Young, Principal Security Researcher Tripwire VERT

Tripwire IP360 Content Developer

Infosec Trainer

Hacker

Presentation Overview

TLS/CBC Encryption Primer

Padding Oracle Exploitation

Scanning For Padding Oracles

Zombie POODLE & GOLDENDOODLE

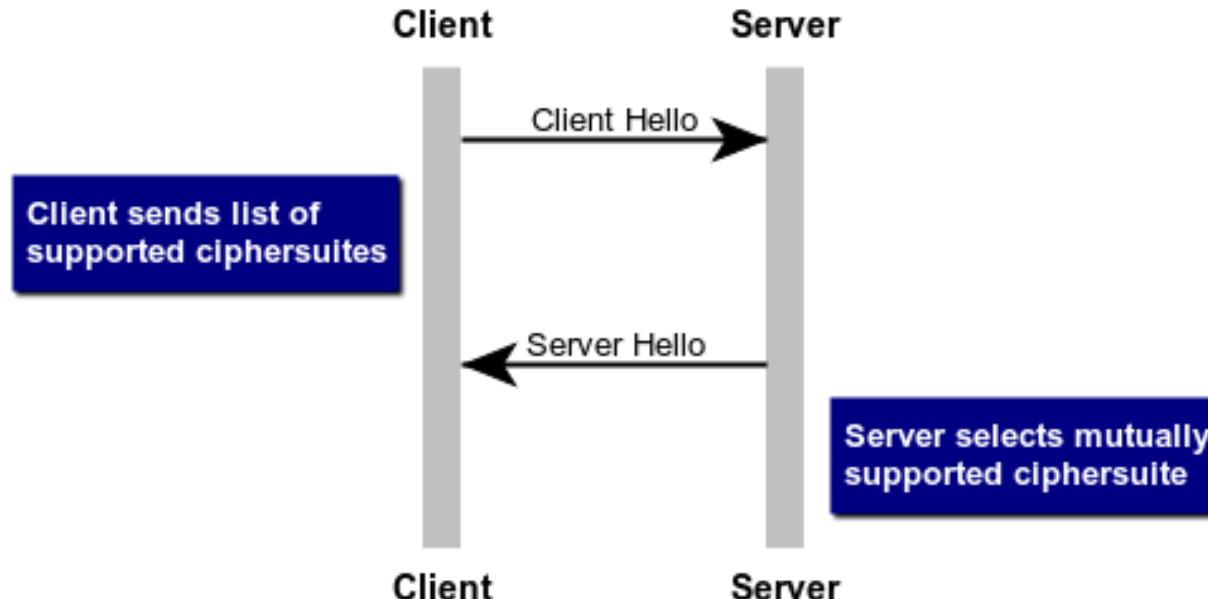
How TLS 1.3 Can Save Us All

SSL/TLS Primer

TLS enables private communication over non-private links.

SSL/TLS Primer

Hello Messages Determine Ciphersuite Selection



www.websequencediagrams.com

What is a Ciphersuite?

Set of algorithms for securing an SSL/TLS session

Key Exchange / Authentication

Message Encryption

Message Authentication

Message Encryption

Message encryption is the transformation of plaintext into ciphertext (and back again).

- Message encryption is the focus of this presentation ☺

Message Encryption

Cipher Types

Block

Stream

AES

DES

RC4

ChaCha

A5/2

Block Ciphers

Fixed-length inputs only

- DES (64-bit blocks)
- AES (128-bit blocks)
- Blowfish (32-bit blocks)

Block Cipher Padding

Cipher only works on a *block*



Incomplete blocks get filled with “padding”

Block Cipher Mode of Operation



Cipher Mode
Defines How To
Encrypt
Multiple Blocks

SSL/TLS Encryption Primer

Block Cipher Modes

CBC

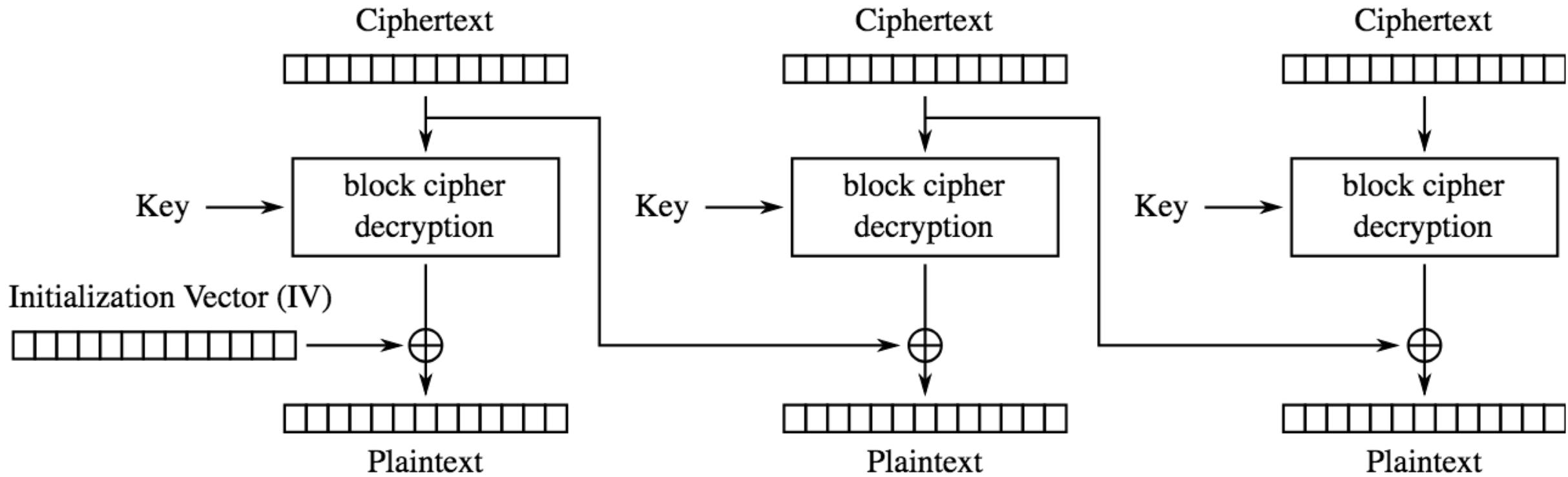
ECB

CTR

CFB

GCM

Today's Villain: CBC Mode



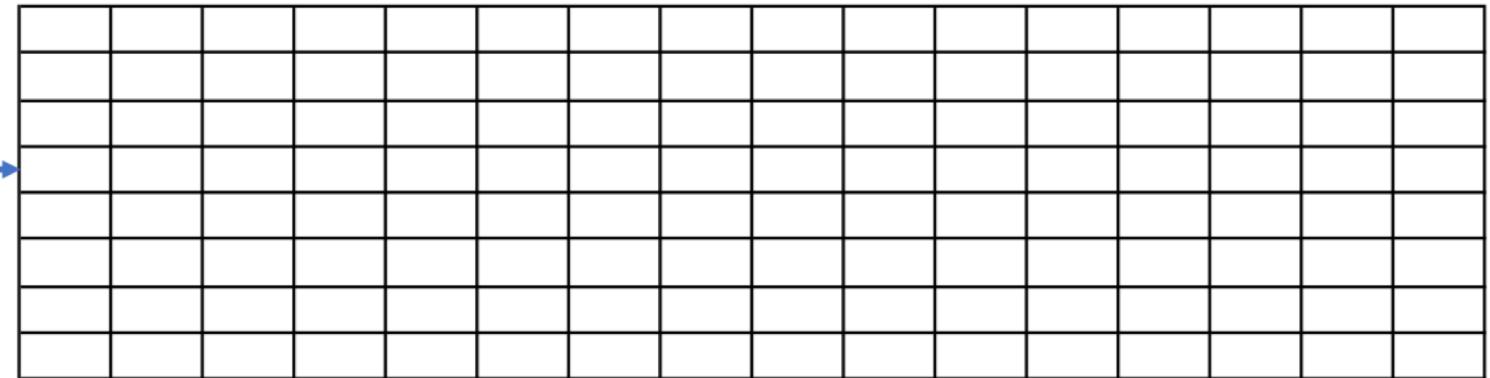
Cipher Block Chaining (CBC) mode decryption



Let's do an example...

TLS CBC Padding Walkthrough

```
GET /foobar HTTP/1.1\r\n
Host: Example.com\r\n
Connection: keep-alive\r\n
Cookie: SID=1A3C2047F5F\r\n\r\n\r\n
```



16-Bytes/Block for AES

TLS CBC Padding Walkthrough

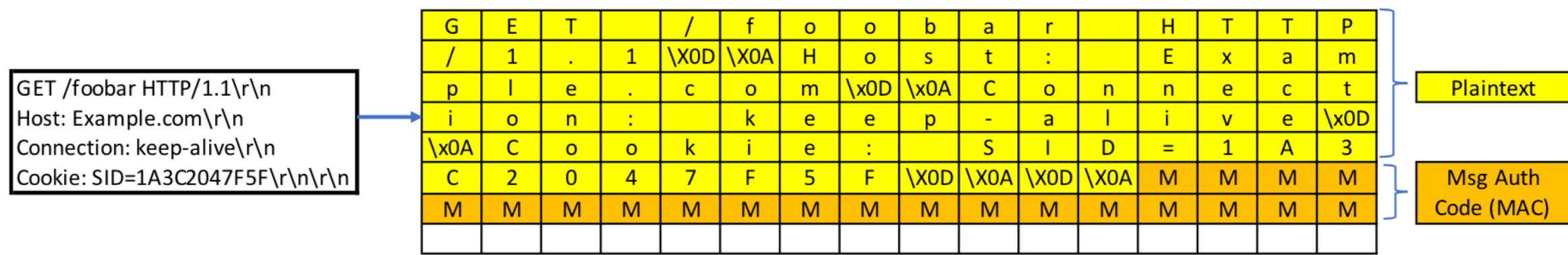
```
GET /foobar HTTP/1.1\r\n
Host: Example.com\r\n
Connection: keep-alive\r\n
Cookie: SID=1A3C2047F5F\r\n\r\n\r\n
```



G	E	T		/	f	o	o	b	a	r		H	T	T	P
/	1	.	1	\X0D	\X0A	H	o	s	t	:		E	x	a	m
p	l	e	.	c	o	m	\x0D	\x0A	C	o	n	n	e	c	t
i	o	n	:	k	e	e	p	-	a	l	i	v	e	\x0D	
\x0A	C	o	o	k	i	e	:	S	I	D	=	1	A	3	
C	2	0	4	7	F	5	F	\X0D	\X0A	\X0D	\X0A				

- 1) Split request into 16-byte blocks (AES)

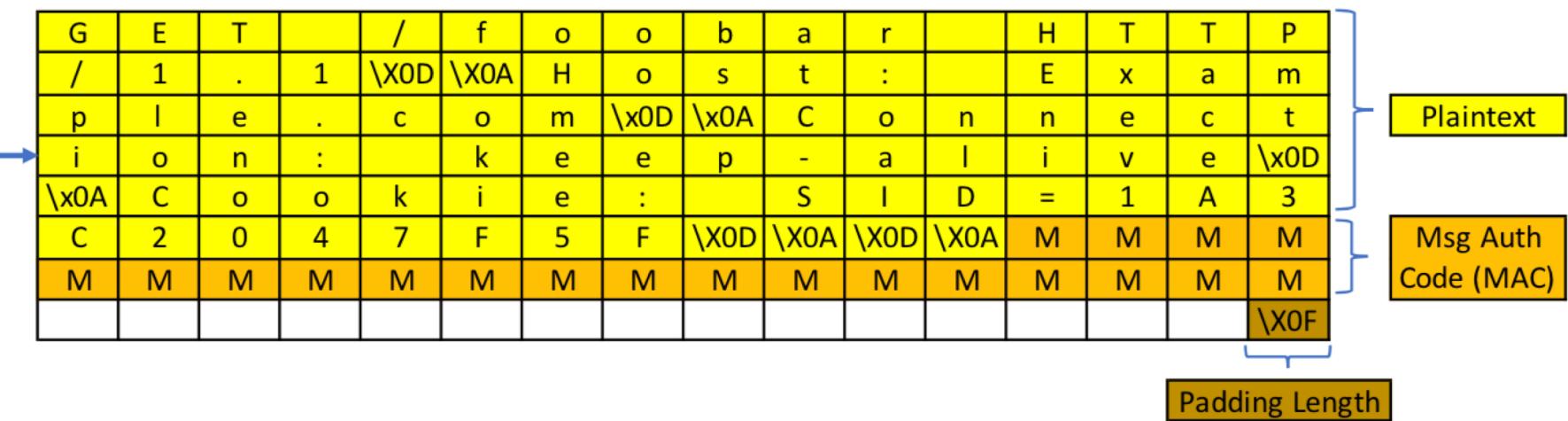
TLS CBC Padding Walkthrough



2) Append Message Authentication Code (MAC)
(This is 20 bytes for SHA, but SHA256 uses a 32-byte MAC.)

TLS CBC Padding Walkthrough

```
GET /foobar HTTP/1.1\r\n
Host: Example.com\r\n
Connection: keep-alive\r\n
Cookie: SID=1A3C2047F5F\r\n\r\n
```



- 3) Required padding length* is determined and set

*NOTE: Padding length byte is not counted as a pad byte

Padding Byte Values

For SSLv3, padding bytes are random:

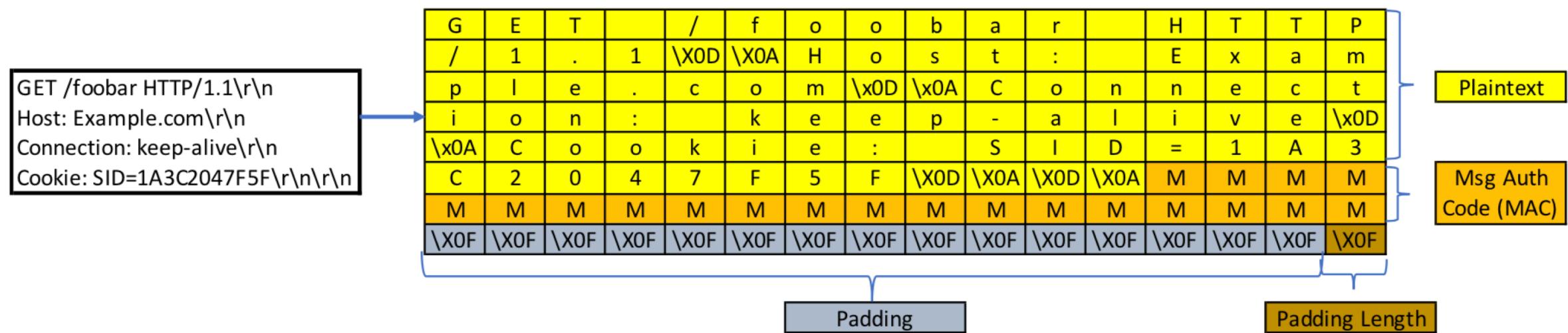


Padding Byte Values

For TLS, pad values must match the pad length:



TLS CBC Padding Walkthrough



- 4) n padding bytes are added with value n

TLS CBC Mode Encryption

G	E	T		/	f	o	o	b	a	r		H	T	T	P
/	1	.	1	\x0D	\x0A	H	o	s	t	:		E	x	a	m
p	l	e	.	c	o	m	\x0D	\x0A	C	o	n	n	e	c	t
i	o	n	:	k	e	e	p	-	a	I	i	v	e	\x0D	
\x0A	C	o	o	k	i	e	:		S	I	D	=	1	A	3
C	2	0	4	7	F	5	F	\x0D	\x0A	\x0D	\x0A	M	M	M	M
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
\x0F															

E_{key} : AES Encryption
 $P_{0..7}$: Plaintext Blocks
 $C_{0..7}$: Ciphertext Blocks

$$\begin{aligned}
 C_0 &= E_k(IV \oplus P_0) \\
 C_1 &= E_k(C_0 \oplus P_1) \\
 C_2 &= E_k(C_1 \oplus P_2) \\
 C_3 &= E_k(C_2 \oplus P_3) \\
 C_4 &= E_k(C_3 \oplus P_4) \\
 C_5 &= E_k(C_4 \oplus P_5) \\
 C_6 &= E_k(C_5 \oplus P_6) \\
 C_7 &= E_k(C_6 \oplus P_7)
 \end{aligned}$$

NOTE: Initialization Vector (IV) may be explicit or it may come from CBC residue depending on implementation version.

TLS CBC Mode Decryption

$$\begin{aligned}C_0 &= E_k(IV \oplus P_0) \\C_1 &= E_k(C_0 \oplus P_1) \\C_2 &= E_k(C_1 \oplus P_2) \\C_3 &= E_k(C_2 \oplus P_3) \\C_4 &= E_k(C_3 \oplus P_4) \\C_5 &= E_k(C_4 \oplus P_5) \\C_6 &= E_k(C_5 \oplus P_6) \\C_7 &= E_k(C_6 \oplus P_7)\end{aligned}$$



$$\begin{aligned}P_0 &= D_k(C_0) \oplus IV \\P_1 &= D_k(C_1) \oplus C_0 \\P_2 &= D_k(C_2) \oplus C_1 \\P_3 &= D_k(C_3) \oplus C_2 \\P_4 &= D_k(C_4) \oplus C_3 \\P_5 &= D_k(C_5) \oplus C_4 \\P_6 &= D_k(C_6) \oplus C_5 \\P_7 &= D_k(C_7) \oplus C_6\end{aligned}$$



This is called MAC-Then-Pad-
Then-Encrypt...



A digital illustration of a city skyline composed of numerous small blue dots, set against a dark background. A bright orange light source at the bottom center emits a radial glow, suggesting a rising sun or a digital horizon.

And It Is “Malleable”

CBC Malleability

Targeted Plaintext Manipulation

$$C'_{n-1} = C_{n-1} \oplus X$$

$$P'_n = P_n \oplus X$$

(And P_{n-1} is unpredictable)

Predicting Trouble



Vaudenay warned of “Padding Oracle” Attacks in 2002

No Change in TLSv1.1 (2006) and TLSv1.2 (2008)

What is a CBC Padding Oracle?

Attacker Learns Something About Plaintext

Padding
Validity

MAC
Validity

Specific
Bit Value

Plaintext
Length

Padding Oracle Exploitation

Oracles may enable
Adaptive Chosen Ciphertext Attacks

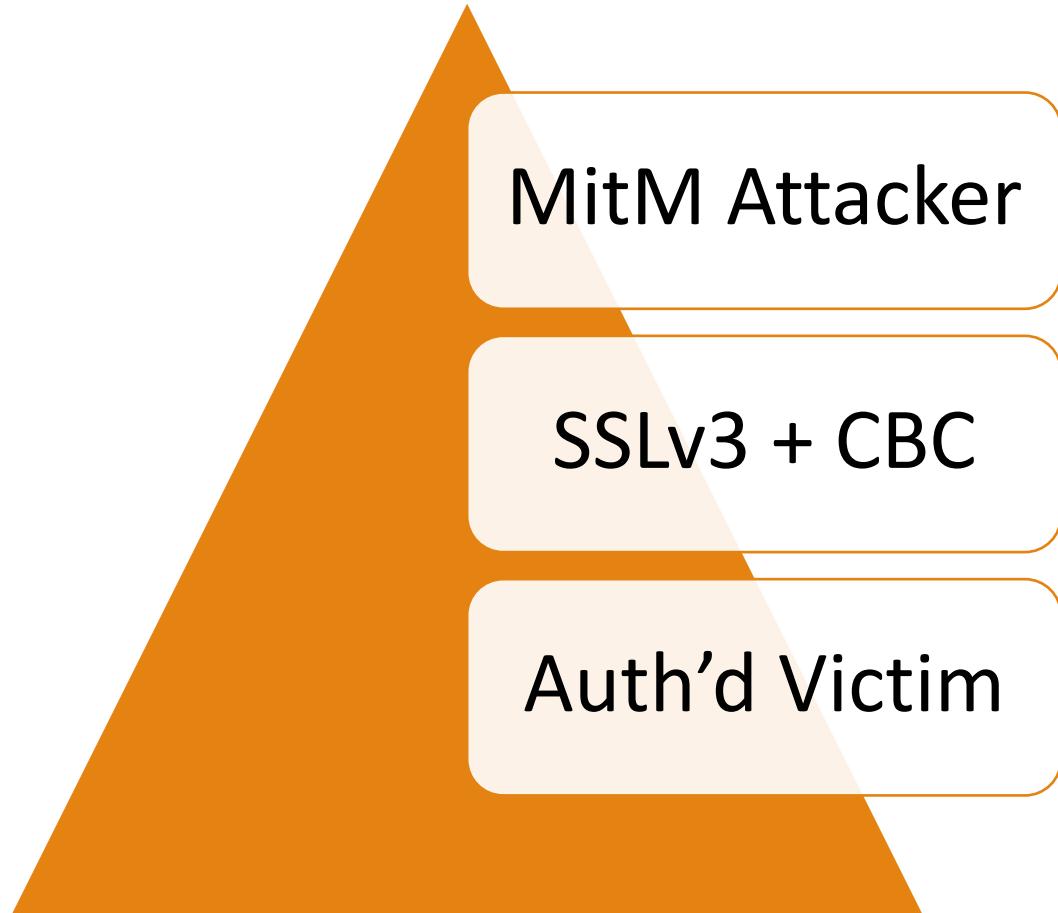
Oracle Observability

Attacker Must Be Able to Observe The Oracle

- Alerts may be encrypted
- Timing works, but is not practical

Observation via Wire or Browser

POODLE Case Study: Attack Requirements



POODLE Case Study: Exploitation Steps

Step 1: Downgrade to SSLv3

- Out of scope for this talk
- Google “TLS Fallback Dance”

POODLE Case Study: Exploitation Steps

Step 2: Generate Request

- JavaScript requests HTTPS from target
- Query has full block of padding

POODLE Case Study: Exploitation Steps

Step 3: Relocate Blocks

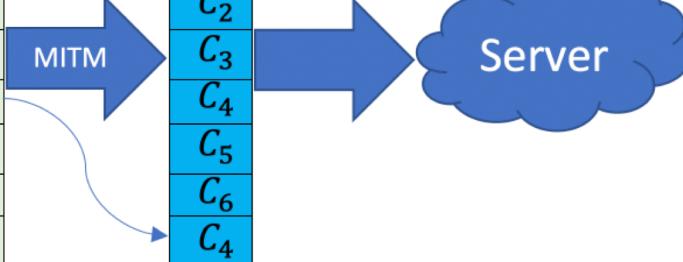
- Padding block replaced by block containing cookie
- Resulting record is sent to server

POODLE Case Study: Block Relocation Visualized

G	E	T		/	f	o	o	b	a	r		H	T	T	P
/	1	.	1	\x0D	\x0A	H	o	s	t	:		E	x	a	m
p	I	e	.	c	o	m	\x0D	\x0A	C	o	n	n	e	c	t
i	o	n	:	k	e	e	p	-	a	l	i	v	e	\x0D	
\x0A	C	o	o	k	i	e	:		S	I	D	=	1	A	3
C	2	0	4	7	F	5	F	\x0D	\x0A	\x0D	\x0A	M	M	M	M
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
\x0F															

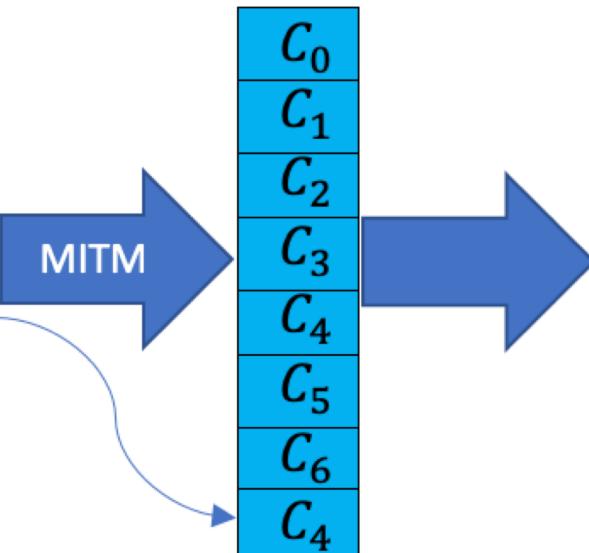
E_{key} : AES Encryption
 $P_{0..7}$: Plaintext Blocks
 $C_{0..7}$: Ciphertext Blocks

$$\begin{aligned}
 C_0 &= E_k(IV \oplus P_0) \\
 C_1 &= E_k(C_0 \oplus P_1) \\
 C_2 &= E_k(C_1 \oplus P_2) \\
 C_3 &= E_k(C_2 \oplus P_3) \\
 C_4 &= E_k(C_3 \oplus P_4) \\
 C_5 &= E_k(C_4 \oplus P_5) \\
 C_6 &= E_k(C_5 \oplus P_6) \\
 C_7 &= E_k(C_6 \oplus P_7)
 \end{aligned}$$



POODLE Case Study: Server Decryption Visualized

$$\begin{aligned}
 C_0 &= E_k(IV \oplus P_0) \\
 C_1 &= E_k(C_0 \oplus P_1) \\
 C_2 &= E_k(C_1 \oplus P_2) \\
 C_3 &= E_k(C_2 \oplus P_3) \\
 C_4 &= E_k(C_3 \oplus P_4) \\
 C_5 &= E_k(C_4 \oplus P_5) \\
 C_6 &= E_k(C_5 \oplus P_6) \\
 C_7 &= E_k(C_6 \oplus P_7)
 \end{aligned}$$



G	E	T	/	f	o	o	b	a	r	H	T	T	P	
/	1	.	1	\r\n	H	o	s	t	:	E	x	a	m	
p	l	e	.	c	o	m	\r\n	C	o	n	n	e	c	
i	o	n	:	k	e	e	p	-	a	l	i	v	e	
\r\n	C	o	o	k	i	e	:	S	I	D	=	1	A	3
C	2	0	4	7	F	5	F	\r\n	\r\n	\r\n	[MAC BYTES]			
M	M	M	M	M	M	M	M	M	M	M	M	M	M	
Pseudorandom Bytes: $D_k(C_4) \oplus C_6$														

POODLE Case Study: Exploitation Steps

Step 4: Observe Oracle

- TLS Alert?
 - Back to Step 2
- No TLS Alert?
 - Onto Step 5

POODLE Case Study: Example Decryption Error

G	E	T		/	f	o	o	b	a	r		H	T	T	P
/	1	.	1	\x0D	\x0A	H	o	s	t	:		E	x	a	m
p	l	e	.	c	o	m	\x0D	\x0A	C	o	n	n	e	c	t
i	o	n	:	k	e	e	p	-	a	l	i	v	e	\x0D	
\x0A	C	o	o	k	i	e	:		S	I	D	=	1	A	3
C	2	0	4	7	F	5	F	\x0D	\x0A	\x0D	\x0A	M	M	M	M
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
38	72	4A	66	2B	FE	39	B4	85	27	83	A4	90	15	17	14

Diagram illustrating the POODLE case study example decryption error. The table shows the structure of the plaintext and MAC. The last byte of the MAC (14) is highlighted in orange, indicating an incorrect MAC. The padding length is also indicated at the bottom right.

Plaintext

Incorrect MAC

Padding Bytes

Padding Length

POODLE Case Study: Decryption Error Frequency

Most records trigger TLS alert

- About 1/256 decrypts will not error
- Think of a 256-side die landing on 15

POODLE Case Study: Successful Decryption Example

G	E	T		/	f	o	o	b	a	r		H	T	T	P	
/	1	.	1	\x0D	\x0A	H	o	s	t	:		E	x	a	m	
p	l	e	.	c	o	m	\x0D	\x0A	C	o	n	n	e	c	t	
i	o	n	:	k	e	e	p	-	a	l	i	v	e	\x0D		
\x0A	C	o	o	k	i	e	:		S	I	D	=	1	A	3	
C	2	0	4	7	F	5	F	\x0D	\x0A	\x0D	\x0A	M	M	M	M	
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	
38	72	4A	66	2B	FE	39	B4	85	27	83	A4	90	15	17	\x0F	

Plaintext

Correct MAC

Padding Bytes

Padding Length

POODLE Case Study: Exploitation Steps

Step 5: Byte Decryption

- $D_k(C_4)[15] = 15 \oplus C_6[15]$
- $P_4[15] = (15 \oplus C_6[15]) \oplus C_3[15]$

POODLE Case Study: Exploitation Steps

Step 6: Adjust Request

- Put next byte of cookie as $P_n[15]$
- Return to step 2 until done

POODLE Case Study: Root Cause Analysis

Mac-Then-Pad-Then-Encrypt

- MAC does not include padding

SSLv3 Padding is Underspecified

- No way to recognize tampering

The POODLE Attack and TLS

TLS specifies padding bytes

No more POODLE, right?

POODLE Scanning

Researchers Made Tools To Scan
For Similar Padding Oracles

POODLE TLS

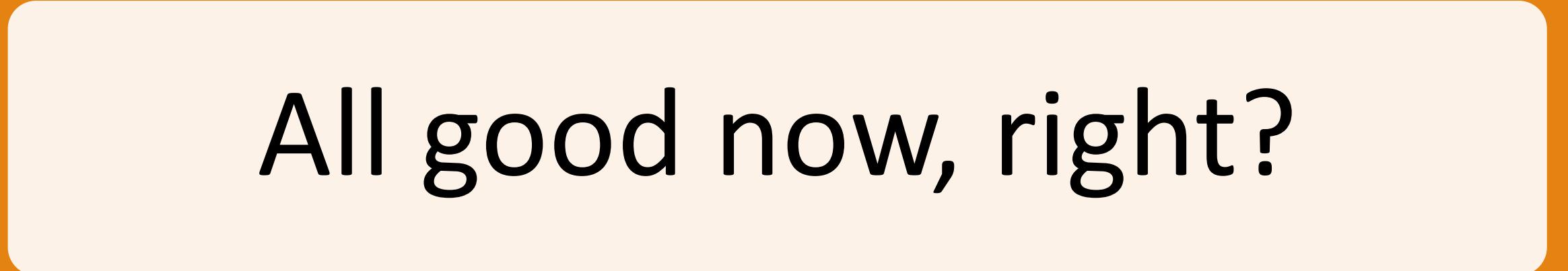
Multiple vendors were using
SSLv3 unpadding in TLS

POODLE TLS

POODLE Was Exploitable Again...



Patches To The Rescue!



All good now, right?



Maybe?

POODLE TLS Scanning

POODLE TLS is SSL Unpad Used in TLS

Test by Connecting With Invalid Client

- Scanner encrypts a badly padded Finished
- Vulnerable systems allow this connection

POODLE TLS Scanning Doesn't Match Exploit

POODLE Doesn't 'Bite' Finished

- Message is Forwarded Untouched

Research Questions

RQ1: Do stacks behave differently post-handshake?

RQ2: What other remote side-channels exist?

RQ3: How common are CBC oracles on the web?

Research Methodology

Build New Tool



Devise New Testcases



Scan Top Ranked Sites

Building a Tool

Based on Adam Langley's `scanpad.go` Example

- Uses patched Golang `crypto/tls` to break padding

Hacked it to only do bad padding for app data

Identifying New Signals

What Else Might Distinguish Error States?

- Received data quantity?
- TCP headers?

Distinction Must Be Observable

- Attacker must learn oracle response
- May observe via MitM or via JavaScript

Testcase Behavior

Complete Handshake

Send HTTP Request with Padding via Testcase

Observe/Record Response

- How many bytes?
- Socket aborted?

Scan Methodology

Responses from each test are compared

Differences considered possible vulns

Scan Reliability

Inconsistent Responses May Not Be Exploitable

Vulnerable Systems Should Get Triple Tested

- Any variation will frustrate attacks

Initial Testcases (August 2018)

Each line represents the “padding block” of a malformed TLS record:

Test #1 - Valid Pad/Invalid MAC

\x00	\x01	\x02	\x03	\x04	\x05	\x06	\x07	\x08	\x09	\x0A	\x0B	\x0C	\x0D	\x0E	\x00
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #2 - Incomplete Padding

\xFF															
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #3 - Correct Length w/ Bad Values

\x10	\x0F	\x0E	\x0D	\x0C	\x0B	\x0A	\x09	\x08	\x07	\x06	\x05	\x04	\x03	\x02	\x0F
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #4 - Pad Len is Plaintext Len

\x5F															
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

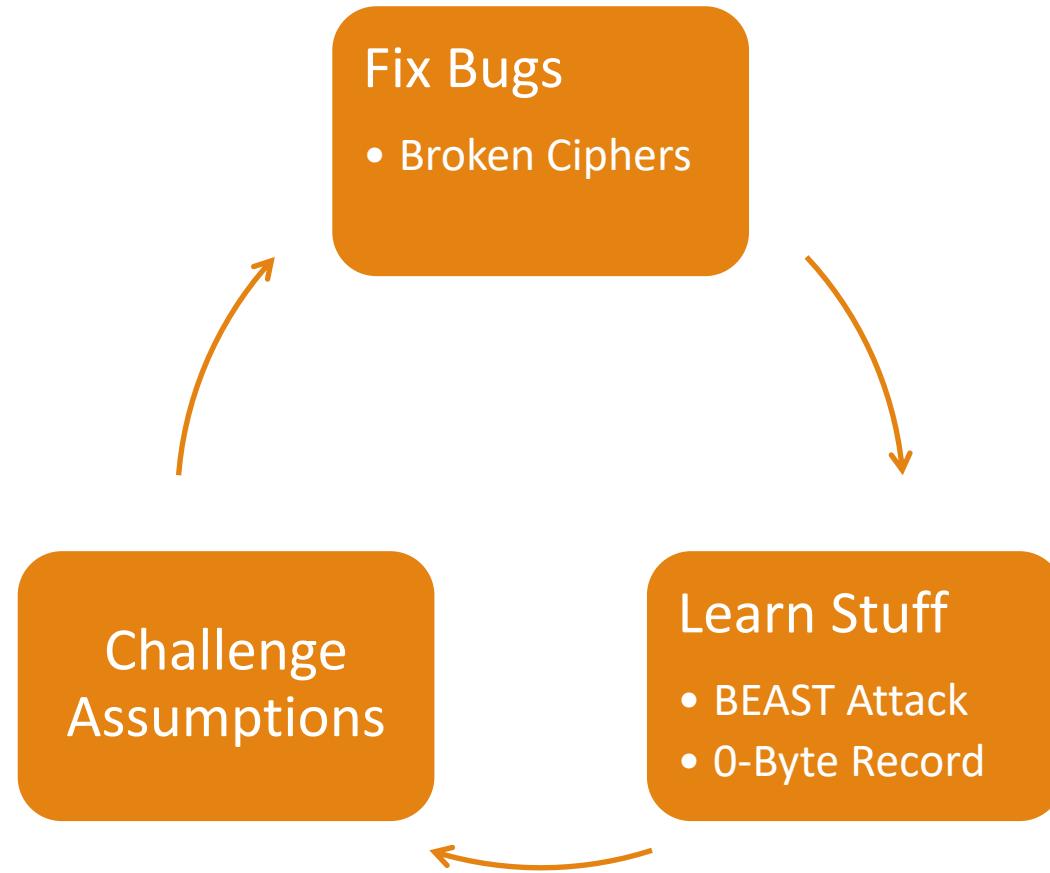
Valid Pad

Invalid Pad

MAC Bytes

Pad Length

Development Process



Current Testcases (March 11, 2019)

Each line represents the “padding block” of a malformed TLS record:

Test #1 - Valid Pad/Invalid MAC

\x00	\x01	\x02	\x03	\x04	\x05	\x06	\x07	\x08	\x09	\x0A	\x0B	\x0C	\x0D	\x0E	\x00
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #2 - Incomplete Pad/No MAC

\xFF																
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #3 - Invalid Pad/Valid MAC

\x10	\x0F	\x0E	\x0D	\x0C	\x0B	\x0A	\x09	\x08	\x07	\x06	\x05	\x04	\x03	\x02	\x0F
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #4 - Valid Pad/No MAC

\x5F																
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Test #5 - 0-Length Record*

\x0F																
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

*CVE-2019-1559 found by Somorovsky, Merget, and Aviram

Valid Pad

Invalid Pad

MAC Bytes

Pad Length

Testcase 1: Valid Padding, Invalid MAC

```
\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x00
```

Zero-length padding is represented by a single null byte.

Testcase 2: Invalid Padding, Not Enough Record

\xFF																	
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Padding length 255 exceeds record length.

NOTE: As of March, plaintext is also \xFF bytes for valid (incomplete) padding.

Testcase 3: Invalid Padding, Valid MAC (POODLE)

\x10	\x0F	\x0E	\x0D	\x0C	\x0B	\x0A	\x09	\x08	\x07	\x06	\x05	\x04	\x03	\x02	\x0F
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Padding bytes are non-deterministic.

Testcase 4: Invalid Padding, Missing MAC



Padding length is record length

NOTE: As of March, this is 6 blocks of bytes with value (6*blockSize-1).

Testcase 5: 0-Length Record (Added in March)

\x0F																
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

Padding length is record length minus MAC length.
(This record would also have 2 blocks for a 32 byte SHA256 MAC.)

This can trigger CVE-2019-1559 as found by Juraj Somorovsky, Robert Merget, and Nimrod Aviram
More info @ <https://github.com/RUB-NDS/TLS-Padding-Oracles>

Scanning Tripwire's Lab For Calibration

Detected expected POODLE TLS targets

- Compared against IP360 detection results

Detected a non-POODLE target too

- Cisco ASA with CVE-2015-4458

Cisco ASA CVE-2015-4458

MAC Validation Failure Due to Cavium Bug

- Yngve Pettersen found this with TLS Prober

“MAC Error” (MACE) Vulnerability

- Detailed on, “The POODLE has friends” blog post:
<https://yngve.vivaldi.net/2015/07/14/the-poodle-has-friends/>



Cisco ASA CVE-2015-4458

Bug is actually a classic padding oracle

CVE-2015-4458: MITRE's Description

The TLS implementation in the Cavium cryptographic-module firmware, as distributed with Cisco Adaptive Security Appliance (ASA) Software 9.1(5.21) and other products, does not verify the MAC field, which allows man-in-the-middle attackers to **spoof TLS content** by modifying packets, aka Bug ID CSCuu52976.

Cisco Advisory: <https://tools.cisco.com/security/center/viewAlert.x?alertId=39919>

A successful exploit of this vulnerability
does not allow an attacker to decrypt
the packets in transit or obtain
information about the session keys being
used for the TLS connection.

Can't Decrypt Packets?



**CHALLENGE
ACCEPTED**

Meet POODLE's Friend GOLDENDOODLE

Same attack scenario as POODLE

- MitM + CBC Ciphers

Same impact as POODLE

- Decryption of authentication headers/cookies

Much faster than POODLE

- Decryption is deterministic

Example GOLDENDOODLE Transform

$$\begin{aligned}
 C_0 &= E_k(IV \oplus P_0) \\
 C_1 &= E_k(C_0 \oplus P_1) \\
 C_2 &= E_k(C_1 \oplus P_2) \\
 C_3 &= E_k(C_2 \oplus P_3) \\
 C_4 &= E_k(C_3 \oplus P_4) \\
 C_5 &= E_k(C_4 \oplus P_5) \\
 C_6 &= E_k(C_5 \oplus P_6) \\
 C_7 &= E_k(C_6 \oplus P_7)
 \end{aligned}$$

MITM

$$\begin{array}{c}
 C_0 \\
 C_1 \\
 C_2 \\
 C_3 \\
 C_4 \\
 C_5 \\
 C_6' \\
 C_4
 \end{array}$$

G	E	T	/	f	o	o	b	a	r	H	T	T	P
/	1	.	1	\r\nH	o	s	t	:	E	x	a	m	
p	l	e	.	c	o	m	\r\nC	o	n	n	e	c	
i	o	n	:	k	e	e	p	-	a	l	i	v	
\nC	o	o	k	i	e	:	S	I	D	=	1	A	3
C	2	0	4	7	F	5	F	\r\n\r\n\r\n	[MAC	BYTES			
M	M	M	M	M	M	M	M	M	M	M	M	M	M
Pseudorandom Bytes: $D_k(C_4) \oplus C_6$													G

$$(C_6')_i = C_6[0..14] \mid (G_i \oplus C_3[15])$$

Where G_i is a guess for $P_4[15]$

Example GOLDENDOODLE Transform

If $G_i = P_4[15]$ then $C_7[15] = 0$

G	E	T		/	f	o	o	b	a	r		H	T	T	P
/	1	.	1	\x0D	\x0A	H	o	s	t	:		E	x	a	m
p	l	e	.	c	o	m	\x0D	\x0A	C	o	n	n	e	c	t
i	o	n	:		k	e	e	p	-	a	I	i	v	e	\x0D
\x0A	C	o	o	k	i	e	:		S	I	D	=	1	A	3
C	2	0	4	7	F	5	F	\x0D	\x0A	\x0D	\x0A	M	M	M	M
M	M	M	M	M	M	M	M	M	M	M	M	M	M	G	
38	72	4A	66	2B	FE	39	B4	85	27	83	A4	90	15	17	\x00

Plaintext

Invalid MAC

No Padding Bytes

Padding Length

GOLDENDOODLE Performance

Attack can guess 1 byte with each intercepted request

Any byte can be decrypted with at most 255 requests

- POODLE requires an average of 256 requests/byte

HTTP headers use a limited ASCII character set

- Max 94 requests for any printable character
- Max 15 requests for any fixed case hexadecimal

GOLDENDOODLE Proof-of-Concept

Quick PoC Completed in a Day

- Used Python + iptables
- cURL in a loop as “victim”
- Tested with SSL-VPN on ASA 9.1(6)

Next Finding: “Zombie POODLE”

Not POODLE TLS -- But Similar

Mishandling Application Data Records with SSLv3 Style Pad

- Most commonly an extra TLS alert only on testcase #3

Exploited with POODLE algorithm almost verbatim

- Oracle is basically just inverted from POODLE
- TLS alert means good padding length in Zombie POODLE

Scanning the Internet

Alexa Top 1M

Includes Most
Popular Stacks

~85% Supports
SSL/TLS

Used for
ROBOT scans

Findings (August-December 2018 Combined Results)

At least 100 discernible behaviors

- Most are observed on only 1 or 2 hosts
- 46 were observed on at least 10 domains each

Over 5800 domains are readily exploitable

- Only counting hosts with consistent and observable oracles
- Under 1000 with POODLE TLS behavior
- About 1000 can be exploited via GOLDENOODLE
- The rest can be exploited via “Zombie POODLE”

March 2019 Findings (After 3 Public Advisories)

129 discernible behaviors

- Used second Read() and Close() error returns for more signals
- Fixed broken ciphersuites
- Testing with multiple blocks

7,947 domains with oracles (March 11-12, 2019)

- 3,689 marked as 'Observable'
- 129

Impact

Around 1.6% of the TLS enabled Alexa top 100K

- Nearly 1% out of all Alexa ranked sites with TLS
- Scans conducted March 11-12, 2018

Many high-profile sites

- Financial
- Government
- Commerce

Caveats/Limitations

Far more hosts are vulnerable

- Some oracles are cipher/protocol specific
- Padcheck tests only the preferred CBC cipher/protocol
- More test cases are possible

Sleeping POODLE?

Some Hosts Only Look Like POODLE TLS

- Servers reject malformed Finished
- Tested tools did not report a padding oracle

App Records Handled Differently

- Padding check on Finished, but not on Application Record

Disclosures

Citrix (CVE-2019-6485)

- Mostly Zombie POODLE and Some GOLDENDOODLE

F5 (CVE-2019-6593)

- Mostly GOLDENDOODLE but also Zombie POODLE

Three More Vendor Disclosures In Progress

- Load Balancer / Firewall / VPN

Missed Opportunities

Possibly Important Things That Aren't Top Ranked Sites

- Browser Based VPN
- Devices Found Only on LANs

More Subtle Padding Oracles

- No Individual MAC Byte Checks



TLS 1.3 To Save Us All?

No More CBC in TLS 1.3!



Moving Forward

STOP using TLS CBC ciphers

START using TLS 1.3

Acknowledgements

Hanno Böck

- Inspired me to investigate CBC padding oracles

Robert Merget, Juraj Somorovsky, and Nimrod Aviram

- Developed a far more extensive padding oracle test tool
- Provided guidance on testing and exploitation methodology

Thanks!

- Padcheck scanner available on GitHub
 - Link TBD: Check @CraigTweets (or on-screen during talk)
- More information on modern TLS padding oracles:
<https://github.com/RUB-NDS/TLS-Padding-Oracles>