

# CSE 207B: Applied Cryptography

**Nadia Heninger**

UCSD

Fall 2025 Lecture 7

# Announcements

1. HW 3 is due next lecture.
2. HW 4 online soon, due before class in 1.5 weeks.

**Last time:** Hash functions

**This time:** MD5 cryptanalysis, hash-based MACs

# 2009: MD5 considered harmful today

Sotirov, Stevens, Appelbaum, Lenstra, Molnar, Osvik, de Weger

	real certificate		rogue CA certificate
issuer	signature algorithm "MD5 with RSA" (15)	block 1	signature algorithm "MD5 with RSA" (15)
	country "US" (15)		country "US" (15)
issuer	organization "Equifax Secure Inc." (15)	block 2	organization "Equifax Secure Inc." (15)
	common name "Equifax Secure Global eBusiness CA-1" (15)		common name "Equifax Secure Global eBusiness CA-1" (15)
validity	validity "from 3 Nov. 2008 7:52:02 to 4 Nov. 2008 7:52:02" (15)	block 3	validity "from 31 Jul. 2004 0:00:00 to 2 Sep. 2004 0:00:00" (15)
	country "US" (15)		common name "MD5 Collisions Inc. (http://www.phredom.org/md5)" (15)
subject	organization "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phredom.org" (15)	block 4	public key algorithm "RSA" (15)
	organization unit "See www.rapidial.com/resources/cpe (c)08" (15)		modulus (1024 bits) (15)
subject	organization unit "Domain Control Validated - Rapid(R)" (15)	block 5	2BA6F8C3C8262A 8F9D8D9F6A6A437 88218F8D8C8263 8F818C8A8E878E 1280888F8444F8 A38E1AF18D83818 2D8C8A837841F7 8E8D8A8F78E8F8 2EAC8D848E8288 8E8C8D8A8E878C 7E1E8A8F87888A 838E878A89F7818 7118484E8A8A8F 2E8C8D8E878887 8E1AC8F78D8488 7E88C1AF2AF8181
	common name "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phredom.org" (15)		Key Usage "..." (15)
subject	public key algorithm "RSA" (15)	block 6	basic constraints "CA = TRUE" (15)
	modulus (2048 bits) (15)		subject key identifier "..." (15)
public key	modulus (2048 bits) (15)	block 7	header "tumor (Netscape comment)" (15)
	3A810C0F28478A 8F04810E888C87 1780888 8F8D8E8C788C8A8		header "..." (15)
public key	31C4888888818 83AC888888888F8 88F7848188888 8F88888888888A 81C888848C4888 84C88888888A8 2AC8888A817C88 87888888888C8	block 8 block 9 1 <sup>st</sup> near collision block	31C4888888818 83AC888888888F8 88F7848188888 8F88888888888A 81C888848C4888 84C88888888A8 2AC8888A817C88 87888888888C8
	2F888882288888 8FAC178A288F82 82811F788F88F8 878F7818F818A 84818888888888 8C8A888888888A 818C818C8188 888888 85788A88888F8		block 10 2F888882288888 8FAC178A288F82 82811F788F88F8 878F7818F818A 84818888888888 8C8A888888888A 818C818C8188 888888 85788A88888F8
public key	62888888888888 88888888888888 8A878881881878 81178A88888C8 8E88888F888888 1881888A888888 88818888888888 82888888888888	block 11 block 12 2 <sup>nd</sup> near collision block	62888888888888 88888888888888 8A878881881878 81178A88888C8 8E88888F888888 1881888A888888 88818888888888 82888888888888
	8E88888F888888 1881888A888888 88818888888888 82888888888888 8E88888F888888 1881888A888888 88818888888888 82888888888888		block 13 (identical)
extensions	key usage "..." (15)	block 14 (identical)	key usage "..." (15)
	subject key identifier "..." (15)		subject key identifier "..." (15)
extensions	or distribution points "..." (15)	block 15 (identical)	or distribution points "..." (15)
	authority key identifier "..." (15)		authority key identifier "..." (15)
extensions	extended key usage "..." (15)	block 15 (identical)	extended key usage "..." (15)
	basic constraints "CA = FALSE" (15)		basic constraints "CA = FALSE" (15)
	signature algorithm "MD5 with RSA" (15)		signature algorithm "MD5 with RSA" (15)
	signature "..." (15)		signature "..." (15)

# RapidSSL collision generation

- About 2 days on a cluster of 200 PS3s
- Equivalent to about \$20k on Amazon EC2



# Colliding SSL certificates

serial number	<b>chosen prefix (difference)</b>	serial number
validity period		validity period
real cert domain name		<b>rogue cert domain name</b>
real cert RSA key	<b>collision bits (computed)</b>	real cert RSA key
X.509 extensions		X.509 extensions
signature	<b>identical bytes (copied from real cert)</b>	signature



*Defense Guided by Experience*

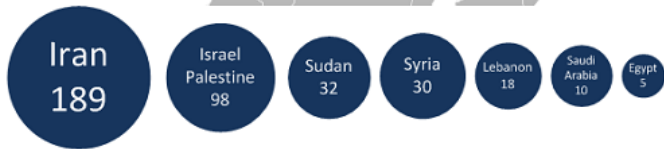
# Analyzing the MD5 collision in Flame

Alex Sotirov

Co-Founder and Chief Scientist

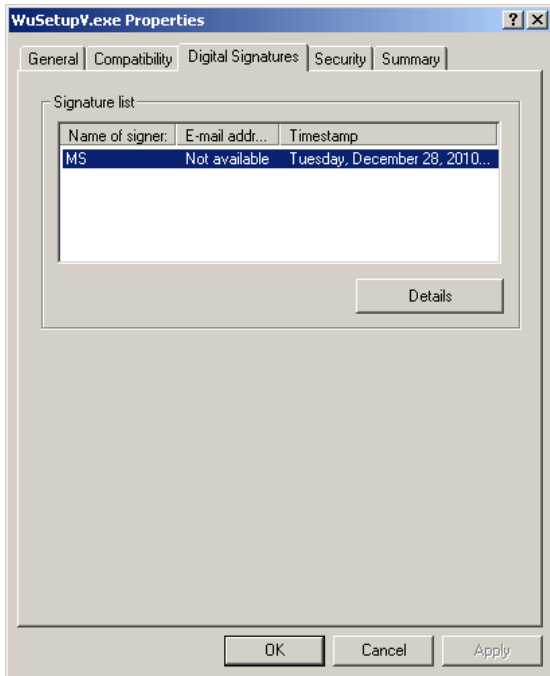
Trail of Bits, Inc

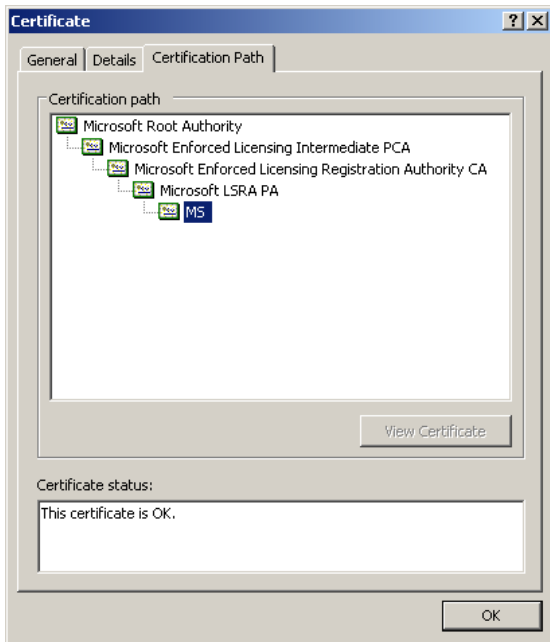
- Discovered sometime in 2012
- Active since at least 2010
- Complex malware
  - almost 20MB in size
  - multiple components
- Very limited targeted attacks



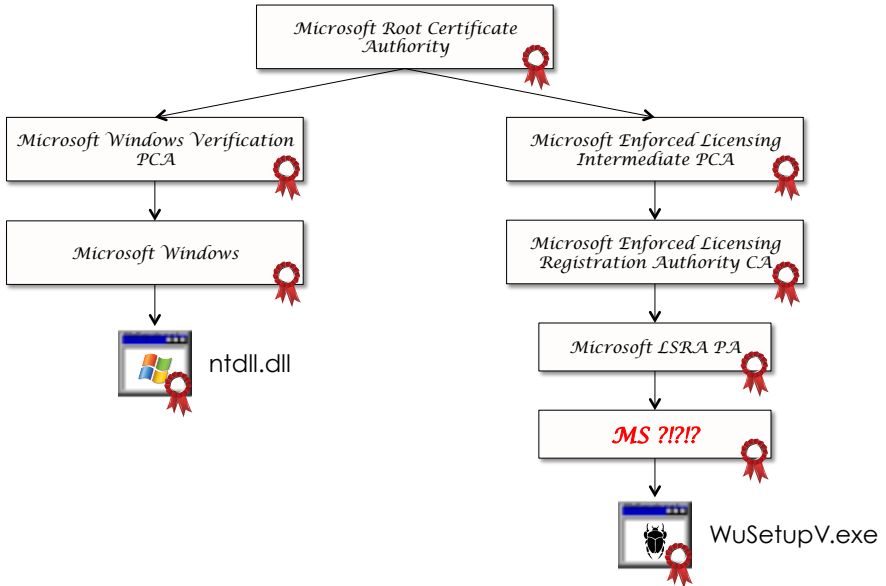
Source: Kaspersky Lab

- Flame registers itself as a proxy server for `update.microsoft.com` and other domains
  - WPAD (Web Proxy Auto-Discovery Protocol)
  - local network only
- Man-in-the-middle on Windows Update
  - SSL spoofing is not needed, Windows Update falls back to plaintext HTTP
  - serves a fake update signed with a Microsoft code-signing certificate





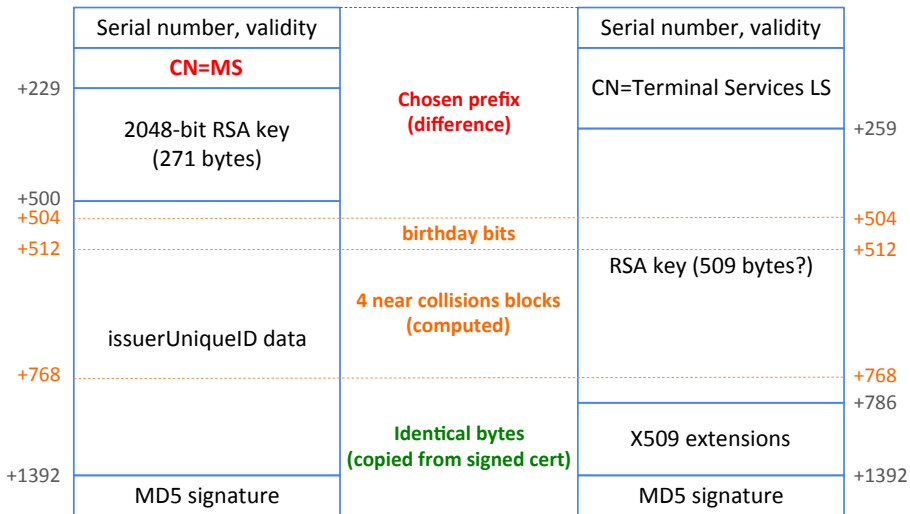
# Certificate hierarchy



# Colliding certificates

Flame certificate

Certificate signed by Microsoft



The bit differences in the near collision blocks can be used to determine what technique produced them:

Using our forensic tool, we have indeed verified that a chosen-prefix collision attack against MD5 has been used for Flame. More interestingly, the results have shown that not our published chosen-prefix collision attack was used, but an entirely new and unknown variant. This has led to our conclusion that the design of Flame is partly based on world-class cryptanalysis.

Marc Stevens, CWI.nl

# Constructing a MAC from a hash function

## Recall:

- Collision-resistant hash function: Unkeyed function  $H : \{0, 1\}^* \rightarrow \{0, 1\}^n$  hard to find inputs mapping to same output.
- MAC: Keyed function  $\text{Mac}_k(m) = t$ , hard for adversary to construct valid  $(m, t)$  pair.

## Candidate MAC constructions

- $\text{Mac}(k, m) = H(k||m)$
- $\text{Mac}(k, m) = H(m||k)$
- $\text{Mac}(k, m) = H(k||m||k)$
- $\text{Mac}(k_1, k_2, m) = H(k_2||H(k_1||m))$

## Candidate MAC constructions

- $\text{Mac}(k, m) = H(k||m)$

**Insecure** for Merkle-Damgård functions: Vulnerable to length extension attacks.

**Secure** for SHA3 sponge.

- $\text{Mac}(k, m) = H(m||k)$

- $\text{Mac}(k, m) = H(k||m||k)$

- $\text{Mac}(k_1, k_2, m) = H(k_2||H(k_1||m))$

## Candidate MAC constructions

- $\text{Mac}(k, m) = H(k||m)$

**Insecure** for Merkle-Damgård functions: Vulnerable to length extension attacks.

**Secure** for SHA3 sponge.

- $\text{Mac}(k, m) = H(m||k)$

Ok, but vulnerable to offline collision-finding attacks against  $H$  for Merkle-Damgård functions. Your next assignment!

- $\text{Mac}(k, m) = H(k||m||k)$

- $\text{Mac}(k_1, k_2, m) = H(k_2||H(k_1||m))$

## Candidate MAC constructions

- $\text{Mac}(k, m) = H(k||m)$

**Insecure** for Merkle-Damgård functions: Vulnerable to length extension attacks.

**Secure** for SHA3 sponge.

- $\text{Mac}(k, m) = H(m||k)$

Ok, but vulnerable to offline collision-finding attacks against  $H$  for Merkle-Damgård functions. Your next assignment!

- $\text{Mac}(k, m) = H(k||m||k)$

Ok, but nobody uses.

- $\text{Mac}(k_1, k_2, m) = H(k_2||H(k_1||m))$

## Candidate MAC constructions

- $\text{Mac}(k, m) = H(k||m)$

**Insecure** for Merkle-Damgård functions: Vulnerable to length extension attacks.

**Secure** for SHA3 sponge.

- $\text{Mac}(k, m) = H(m||k)$

Ok, but vulnerable to offline collision-finding attacks against  $H$  for Merkle-Damgård functions. Your next assignment!

- $\text{Mac}(k, m) = H(k||m||k)$

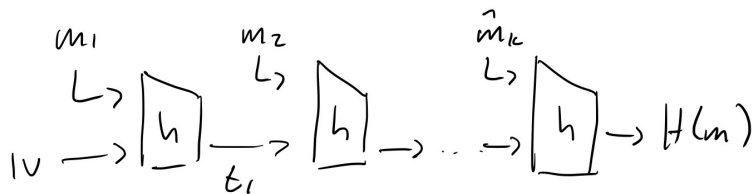
Ok, but nobody uses.

- $\text{Mac}(k_1, k_2, m) = H(k_2||H(k_1||m))$

Secure for Merkle-Damgård functions, similar to HMAC.

# Length extension attacks

Recall the Merkle-Damgård construction:



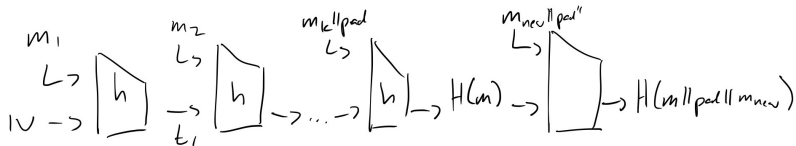
$$\hat{m}_k = m_k || \text{pad} || \text{len}(m)$$

The final output is equivalent to an intermediate state for  $H(m || \text{pad} || \dots)$ .

# Length extension attacks

**Input:** Bad MAC:  $(m, H(k||m))$

**Attack:** Forge valid bad MAC:  $(m||pad||m', H(k||m||pad||m'))$



In general, we can construct the hash  $H(m||pad||m_{new})$  for any  $m_{new}$  from only  $H(m)$  even if we don't know  $m$ .

Just need to know (or guess)  $\text{len}(m)$  to compute padding.

# The Blast-RADIUS attack: Inspiration for HW 4

## **RADIUS/UDP Considered Harmful**

Sharon Goldberg, Miro Haller, Nadia Heninger, Mike Milano, Dan Shumow, Marc Stevens, and Adam Suhl.  
USENIX Security 2024.

`https://www.blastradius.fail`

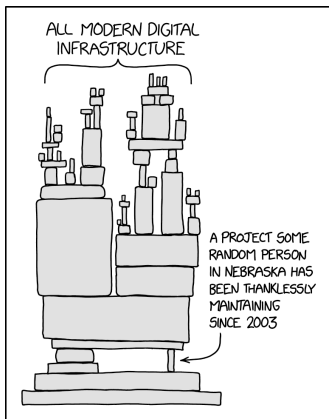
# Attack Summary

A man-in-the-middle network attacker can forge arbitrary RADIUS responses for non-EAP authentication modes:

- Turning an Access-Reject into an Access-Accept
- Adding arbitrary network access attributes to Access-Accept.

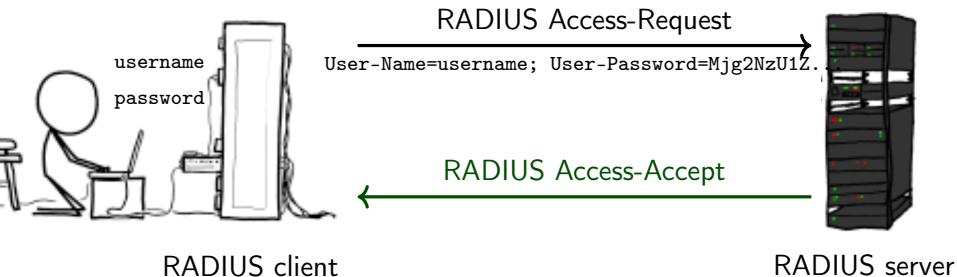
This is a **protocol vulnerability**: RADIUS hard codes weak authentication based on broken MD5 hash function.

# Obligatory XKCD



- RADIUS is the de facto standard lightweight protocol for authentication, authorization, and accounting (AAA) for networked devices.
- RADIUS is *everywhere*: ISPs (DSL/FTTH), 802.1X, WiFi, mobile roaming, IoT, router admin access...

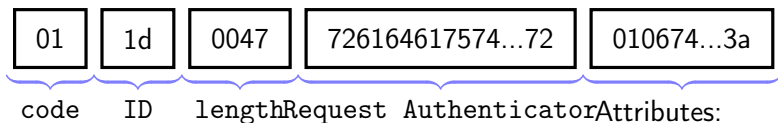
# RADIUS Protocol Reminder



- RADIUS requests and responses are often sent over UDP.
- RFC 6614 TLS Encryption for RADIUS (2012) never left experimental status.
- (D)TLS Encryption for RADIUS:  
draft-ietf-radext-radiusdtls-bis

# RADIUS Packet Formats

## Access-Request



User-Name test

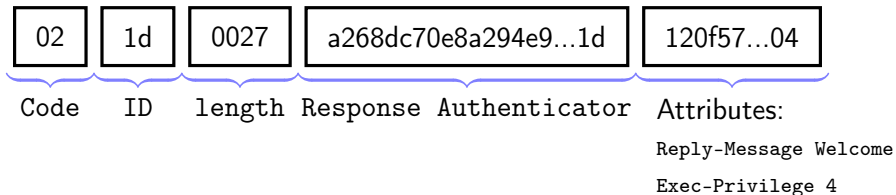
Password Mjg2NzU1z

NAS-Identifier = localhost

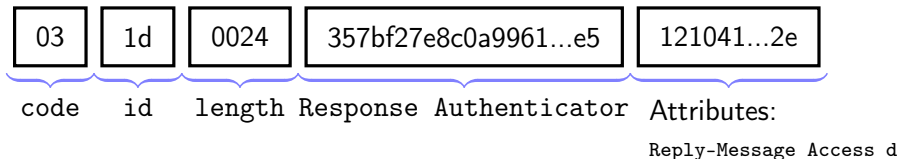
- Access-Request packet contents unauthenticated.
- UDP source IP address is used to identify/validate client.
- Client and server share fixed shared secret.
- Passwords obfuscated using MD5+shared secret.

# RADIUS Packet Formats

## Access-Accept



## Access-Reject

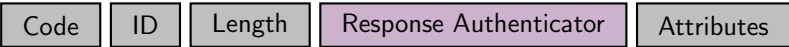


- Response Authenticator: Ad hoc authentication with MD5 hash.

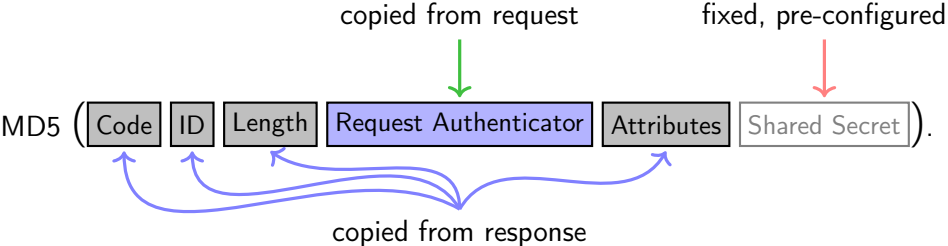
# Response Authenticator

**Goal:** Prevent forgery of packets, e.g., by man-in-the-middle attacker.

The Response Authenticator from packet



is computed as



## Blast-RADIUS: Turning Access-Reject Into Access-Accept

- Our attack allows a MITM attacker to produce a valid Response Authenticator *without* knowing the Shared Secret.
- It creates an MD5 collision such that Access-Accept and Access-Reject produce the same Response Authenticator (very simplified):

$$\text{MD5}(\text{Access-Accept}) = \text{MD5}(\text{Access-Reject}).$$

# MD5 Collision Attack History

1993 Known weaknesses in MD5.

2004 First full MD5 collision. Produced unstructured strings  $G_1$ ,  $G_2$  with

$$\text{MD5}(G_1) = \text{MD5}(G_2).$$

⇒ Unstructured  $G_1$ ,  $G_2$  hard to exploit in realistic contexts.

2004 Identical-prefix collision. Given prefix  $P$ , produce  $G_1$ ,  $G_2$  such that

$$\text{MD5}(P||G_1) = \text{MD5}(P||G_2).$$

⇒ Takes seconds on a laptop. Used to create colliding PDFs etc.

# MD5 Collision Attack History

2007 Chosen-prefix collision. Given prefixes  $P_1$  and  $P_2$ , produce  $G_1$  and  $G_2$  such that

$$\text{MD5}(P_1||G_1) = \text{MD5}(P_2||G_2).$$

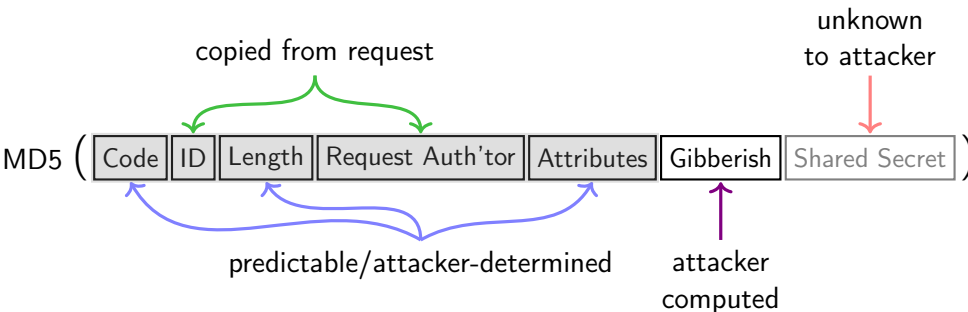
Because of MD5 structure, can append any fixed suffix  $S$  and still collide:

$$\text{MD5}(P_1||G_1||S) = \text{MD5}(P_2||G_2||S)$$

⇒ This is what we want to exploit in a real protocol.

Took days to compute when we started this project.

# A RADIUS Response Authenticator MD5 Collision



1. Given a RADIUS request and two possible responses, an attacker can add an attribute that causes them to have colliding Response Authenticators.
2. E.g., a forged Access-Accept and expected observed Access-Reject.
3. Attacker can copy valid Response Authenticator from observed response to desired forged response.

## A RADIUS Response Authenticator MD5 Collision

Hence, the adversary's goal is to compute the following chosen prefix collision:

$$\text{MD5}\left( \begin{array}{|c|c|c|} \hline \text{AcceptPrefix} & \text{AcceptGibberish} & \text{Shared Secret} \\ \hline \end{array} \right) \\ = \\ \text{MD5}\left( \begin{array}{|c|c|c|} \hline \text{RejectPrefix} & \text{RejectGibberish} & \text{Shared Secret} \\ \hline \end{array} \right)$$

**Problem 1:** Collision prefixes `AcceptPrefix` and `RejectPrefix` depend on Request Authenticator and ID from Access Request.

**Problem 2:** Attacker needs to hide collision gibberish `AcceptGibberish` in an attribute that is echoed back from the server.

# Resolving Problem 1: Online Collision Computation

**Problem 1:** Collision prefixes `AcceptPrefix` and `RejectPrefix` depend on Request Authenticator and ID from Access Request.

**Solution:** Attacker

1. intercepts request from victim client,
2. learns Request Authenticator,
3. predicts prefixes,
4. and computes MD5 collision before client timeout.

We optimized collision to get time from multiple hours to under 5 minutes for proof-of-concept attack.

We could have decreased it further by implementing on GPU/FPGA but did not think this was a good use of grad student time.

## Resolving Problem 2: Reflection Via Proxy-State Attribute

**Problem 2:** Attacker needs to hide collision gibberish in an attribute that is echoed back from the server.

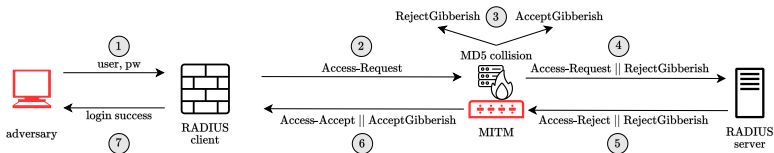
**Solution:** The Proxy-State attribute.

*This Attribute is available to be sent by a proxy server to another server when forwarding an Access-Request and MUST be returned unmodified in the Access-Accept, Access-Reject or Access-Challenge.*

*(RFC 2058)*

⇒ Attacker intercepts Access-Request and injects malicious Proxy-State attribute to force collision.

# Blast-RADIUS Attack Overview



1. Adversary logs into victim NAS with invalid password.
2. Victim NAS makes RADIUS Access-Request to RADIUS server.
3. MITM intercepts request and computes MD5 collision.
4. MITM injects collision gibberish into Proxy-State attribute in request.
5. Victim RADIUS server rejects request.
6. MITM copies Response Authenticator from reject into forged Access-Accept.
7. Victim NAS RADIUS client receives forged accept and permits login.

# Impact and Mitigation

## Impact:

- PAP, CHAP, MS-CHAP are vulnerable
- Modes *requiring* a Message-Authenticator attribute are not vulnerable.
  - E.g., EAP.
  - HMAC-MD5 is not vulnerable to MD5 collision attack.

## Threat model: Requires MITM network access

- RADIUS/UDP traffic over open internet is exposed/vulnerable.
- RADIUS/UDP traffic over VLAN or IPSEC requires attacker to have network access to exploit; useful for lateral movement within org.

**Short-term mitigation:** All requests and responses should include and verify Message-Authenticator attribute:

draft-ietf-radext-deprecating-radius-02.

**Long-term mitigation:** All RADIUS traffic should be encapsulated in (D)TLS tunnel: draft-ietf-radext-radiusdtls-bis-02.

## HMAC: A PRF for Merkle-Damgård functions

$$F_k(m) = H(k \oplus \text{opad} || H(k \oplus \text{ipad} || m))$$

$$\text{ipad} = 0x36 \quad \text{opad} = 0x5C$$

Under the heuristic assumption that  $k \oplus \text{opad}$  and  $k \oplus \text{ipad}$  are “independent” keys, this is a secure PRF.

HMAC is standardized and HMAC-SHA256 is a good choice. Historically HMAC-SHA1 was also common.

$H(k || m)$  is a secure MAC for SHA3.

1. HW 3 due next lecture!
2. HW 4 will be online soon!