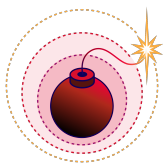
RADIUS/UDP Considered Harmful

The Blast-RADIUS Attack

Sharon Goldberg[1], **Miro Haller**[2], Nadia Heninger[2], Mike Milano[3], Dan Shumow[4], Marc Stevens[5], Adam Suhl[2]

[1]Cloudflare, [2]UC San Diego, [3]BastionZero, [4]Microsoft Research, [5]Centrum Wiskunde & Informatica

October 7, 2024



Attack Summary

MitM network attacker can forge arbitrary RADIUS responses (for non-EAP authentication modes)

e.g., can log into victim device with bogus credentials

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

Attack Summary

MitM network attacker can forge arbitrary RADIUS responses (for non-EAP authentication modes)

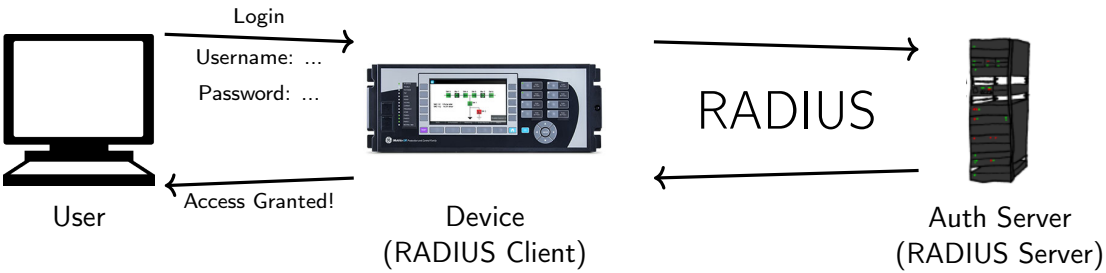
e.g., can log into victim device with bogus credentials

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

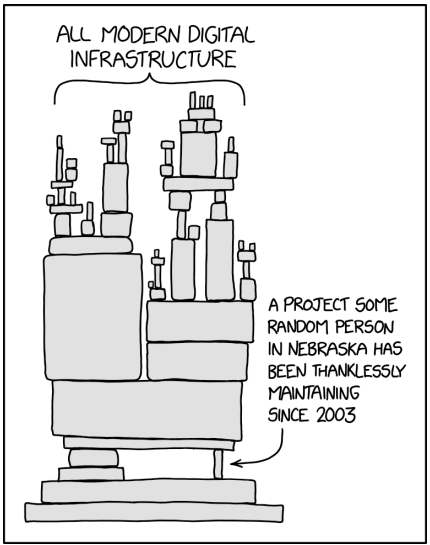
This is a USENIX Security 2024 paper, further information on `https://www.blastradius.fail`.

What is RADIUS?

- RADIUS is the de facto standard lightweight protocol for authentication, authorization, and accounting (AAA) for networked devices.
- Log into X but handle auth on server Y



What uses RADIUS?



- RADIUS is *everywhere*: backbone routers, VPNs, ISP infrastructure (DSL/FTTH), IoT devices, identity providers and MFA (Okta, Duo), power grid equipment, router admin access
- Not vulnerable uses: 802.1X, enterprise WiFi, eduroam

RADIUS is in wide-spread use, and is supported by essentially every switch, router, access point, and VPN concentrator product sold in the past twenty-five years.

(Alan DeKok, FreeRADIUS [DeK24])

RADIUS still uses 90s-era cryptography

- MD5 was broken 20 years ago
- Perceived lack of urgency to deprecate

> *As of the writing of this specification, RADIUS/UDP is still widely used, even though it depends on MD5 and "ad hoc" constructions for security. While MD5 has been broken, it is a testament to the design of RADIUS that there have been (as yet) no attacks on RADIUS Authenticator signatures which are stronger than brute-force.*
>
> *("Deprecating Insecure Practices in RADIUS" IETF draft, 2023)*

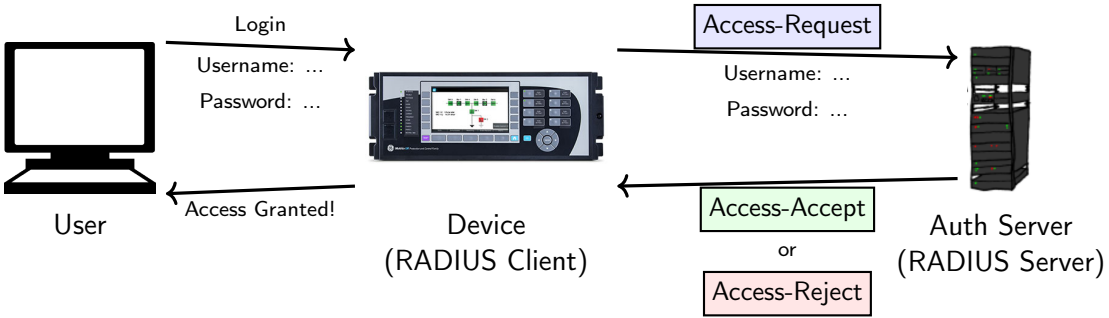
RADIUS still uses 90s-era cryptography

- MD5 was broken 20 years ago
- Perceived lack of urgency to deprecate

> *As of the writing of this specification, RADIUS/UDP is still widely used, even though it depends on MD5 and "ad hoc" constructions for security. While MD5 has been broken, it is a testament to the design of RADIUS that there have been (as yet) no attacks on RADIUS Authenticator signatures which are stronger than brute-force.*
>
> *("Deprecating Insecure Practices in RADIUS" IETF draft, 2023)*

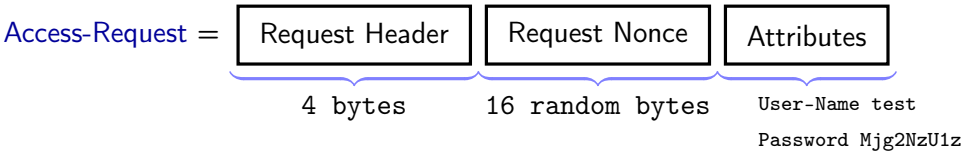
..until now!

How does RADIUS work?

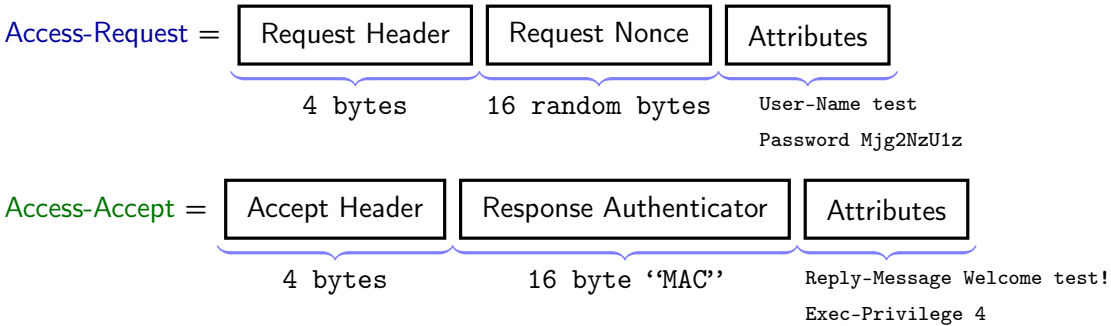


- RADIUS requests and responses are often sent over UDP.
- Client and server share fixed shared secret for authenticating responses and obfuscating passwords.

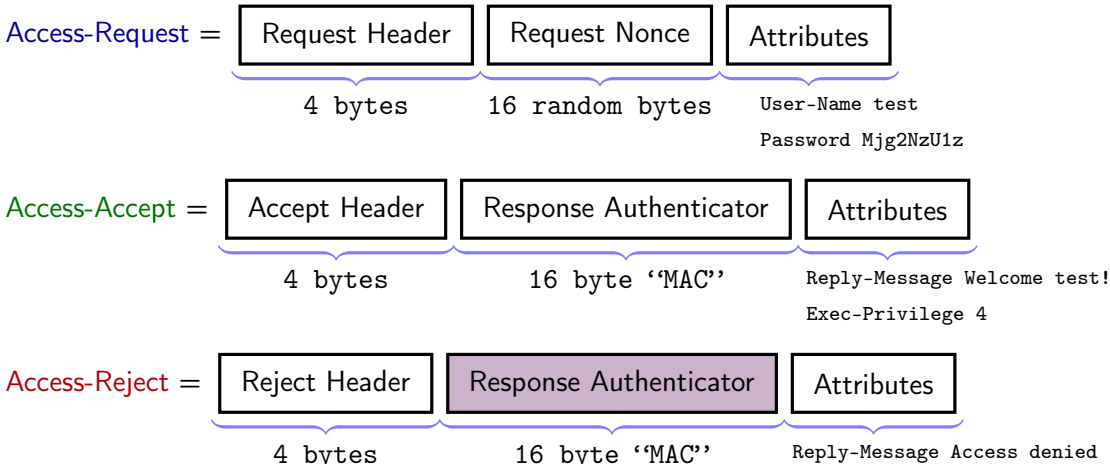
Packet Formats



Packet Formats



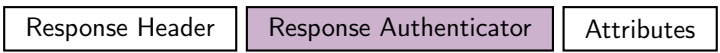
Packet Formats



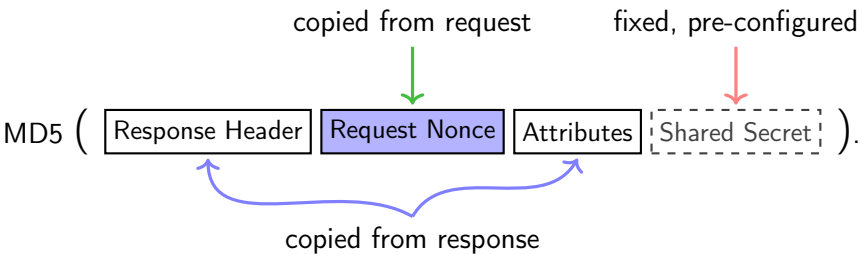
Response Authenticator

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

The Response Authenticator from packet



is computed as



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

- MitM attacker wants to forge an Access-Accept
  - Don't know shared secret, so can't compute Response Authenticator
- Attack: create an MD5 collision such that Access-Accept and Access-Reject will produce the same Response Authenticator (very simplifed):

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

- Trick server into sending the Access-Reject

Blast-RADIUS Attack Overview

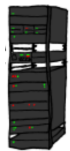
Attacker

login

password bogus

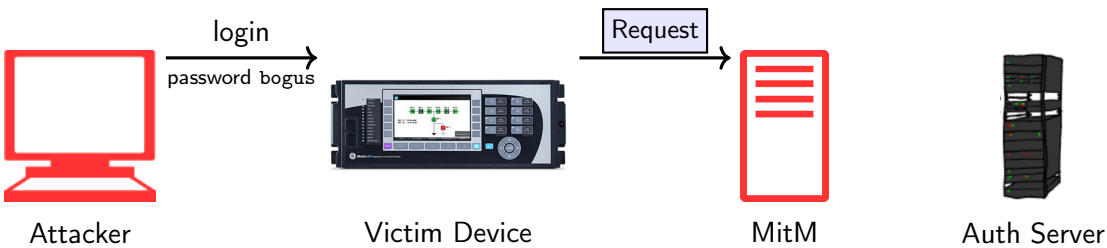
Victim Device

MitM

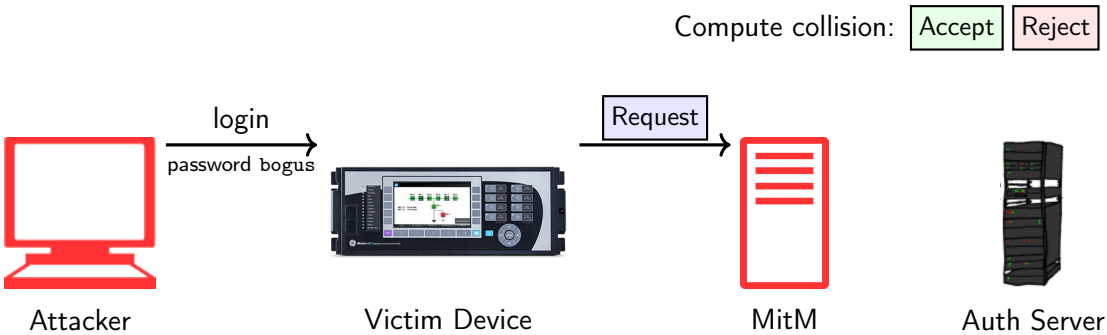


Auth Server

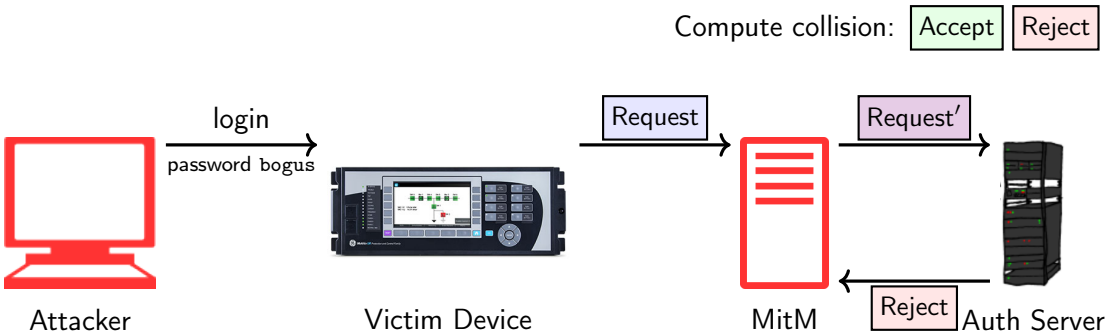
Blast-RADIUS Attack Overview



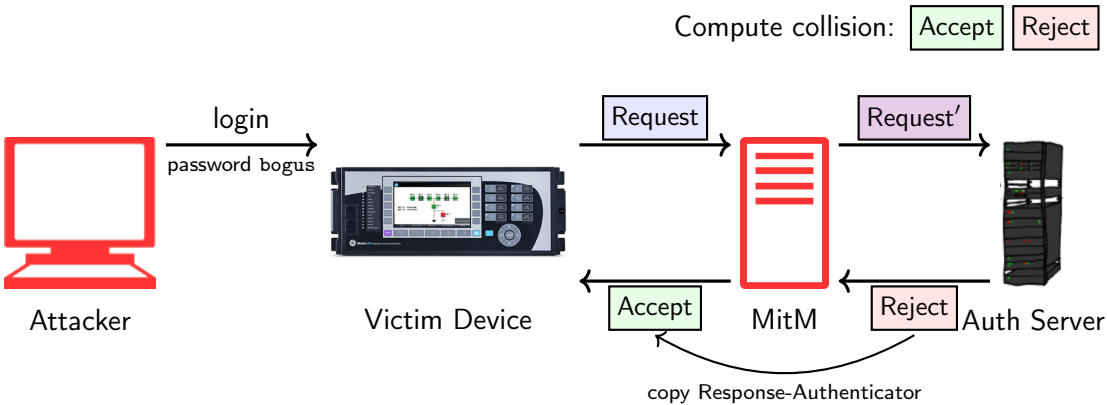
Blast-RADIUS Attack Overview



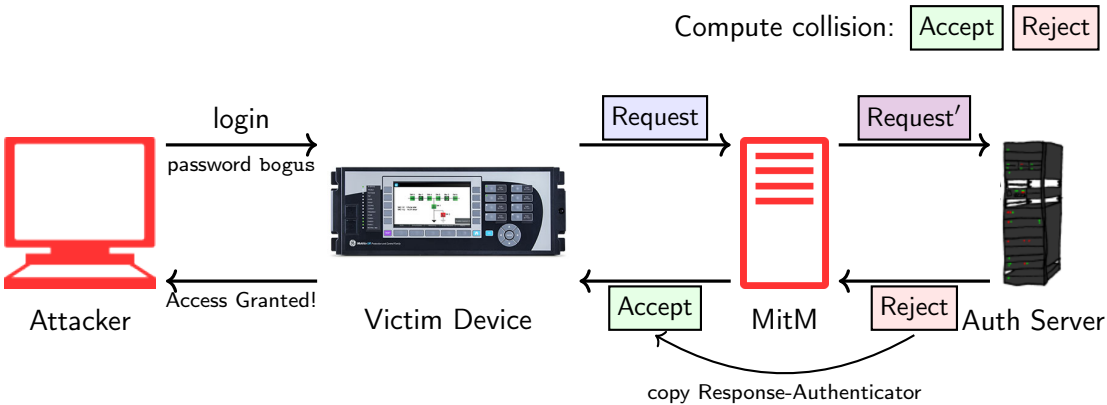
Blast-RADIUS Attack Overview



Blast-RADIUS Attack Overview



Blast-RADIUS Attack Overview



Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

A MAC is a keyed checksum of the message that is sent along with the message. It takes in a fixed-length secret key and an arbitrary-length message, and outputs a fixed-length checksum. A secure MAC has the property that any change to the message will render the checksum invalid.

(Computer Security textbook [Wag+])

Is this a secure MAC?

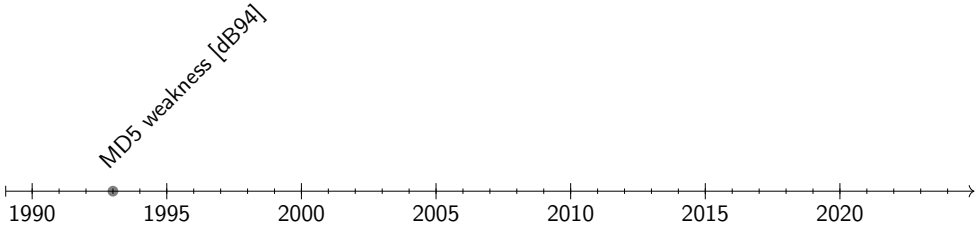
$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

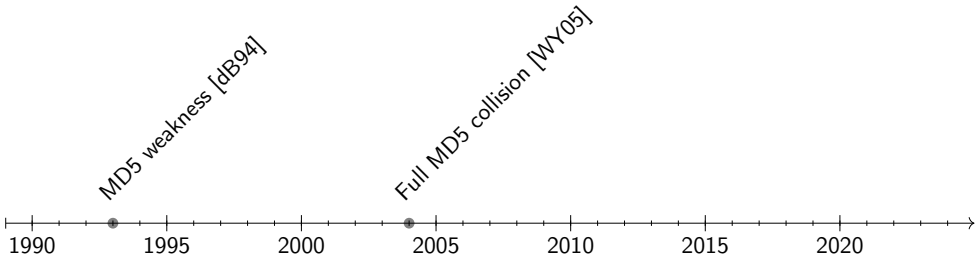
A MAC is a keyed checksum of the message that is sent along with the message. It takes in a fixed-length secret key and an arbitrary-length message, and outputs a fixed-length checksum. A secure MAC has the property that any change to the message will render the checksum invalid.

(Computer Security textbook [Wag+])

MD5 Collision Attack History

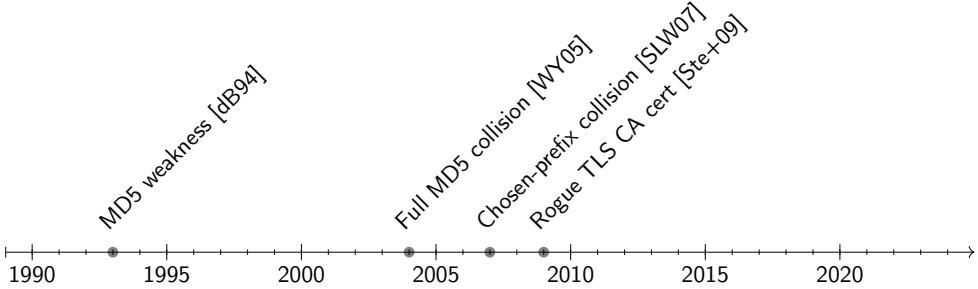


MD5 Collision Attack History



- MD5 collision: unstructured strings G_1, G_2 with $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$.

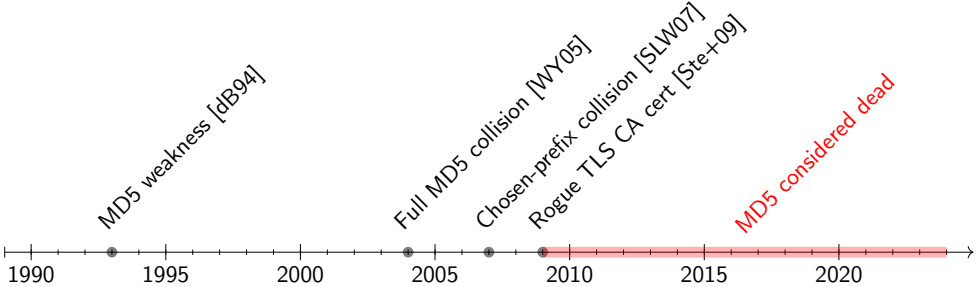
MD5 Collision Attack History



- MD5 collision: unstructured strings G_1, G_2 with $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

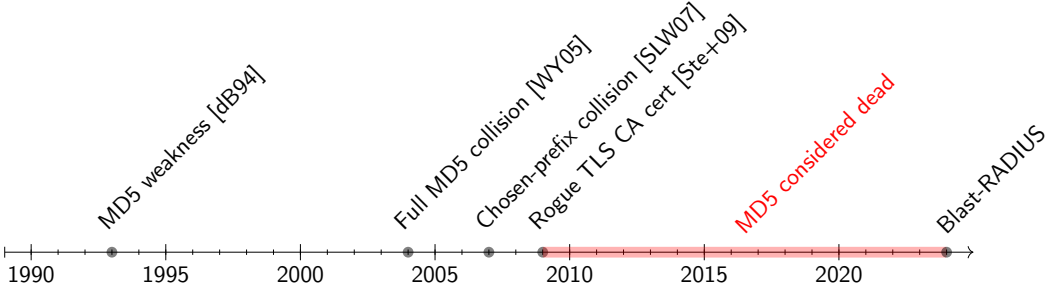
MD5 Collision Attack History



- MD5 collision: unstructured strings G_1, G_2 with $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

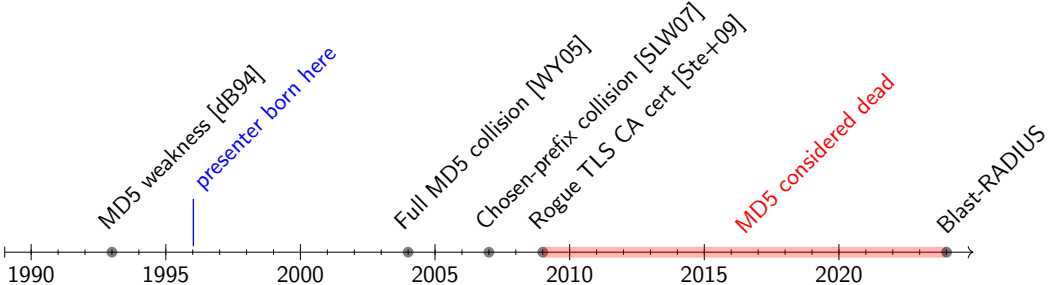
MD5 Collision Attack History



- MD5 collision: unstructured strings G_1, G_2 with $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

MD5 Collision Attack History



- MD5 collision: unstructured strings G_1, G_2 with $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

- Find collision $\texttt{MD5}(M_1) = \texttt{MD5}(M_2)$, then

$$\texttt{MD5}(M_1 \| S) = \texttt{MD5}(M_2 \| S).$$

Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

- Find collision $\texttt{MD5}(M_1) = \texttt{MD5}(M_2)$, then
$$\texttt{MD5}(M_1 \| S) = \texttt{MD5}(M_2 \| S).$$
- In particular, with chosen-prefix collision ($\mathrm{MD5}(P_1 || G_1) = \mathrm{MD5}(P_2 || G_2)$), appending any common suffix S still collides:
$$\mathrm{MD5}(P_1 || G_1 || S) = \mathrm{MD5}(P_2 || G_2 || S)$$

Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

- Find collision $\texttt{MD5}(M_1) = \texttt{MD5}(M_2)$, then
$$\texttt{MD5}(M_1 \| S) = \texttt{MD5}(M_2 \| S).$$
- In particular, with chosen-prefix collision $(\mathrm{MD5}(P_1 || G_1) = \mathrm{MD5}(P_2 || G_2))$, appending any common suffix S still collides:
$$\mathrm{MD5}(P_1 || G_1 || S) = \mathrm{MD5}(P_2 || G_2 || S)$$

Side note: what about

- $\mathrm{MAC}_S(M) = \texttt{MD5}(S \| M)$?

Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

- Find collision $\texttt{MD5}(M_1) = \texttt{MD5}(M_2)$, then
$$\texttt{MD5}(M_1 \| S) = \texttt{MD5}(M_2 \| S).$$
- In particular, with chosen-prefix collision $(\mathrm{MD5}(P_1 || G_1) = \mathrm{MD5}(P_2 || G_2))$, appending any common suffix S still collides:
$$\mathrm{MD5}(P_1 || G_1 || S) = \mathrm{MD5}(P_2 || G_2 || S)$$

Side note: what about

- $\mathrm{MAC}_S(M) = \texttt{MD5}(S \| M)$? No (length extension)

Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

- Find collision $\texttt{MD5}(M_1) = \texttt{MD5}(M_2)$, then
$$\texttt{MD5}(M_1 \| S) = \texttt{MD5}(M_2 \| S).$$
- In particular, with chosen-prefix collision $(\mathrm{MD5}(P_1 || G_1) = \mathrm{MD5}(P_2 || G_2))$, appending any common suffix S still collides:
$$\mathrm{MD5}(P_1 || G_1 || S) = \mathrm{MD5}(P_2 || G_2 || S)$$

Side note: what about

- $\mathrm{MAC}_S(M) = \texttt{MD5}(S \| M)$? No (length extension)
- $\mathrm{MAC}_S(M) = \texttt{MD5}(S \| M \| S)$?

Is this a secure MAC?

$$\mathrm{MAC}_S(M) = \texttt{MD5}(M \| S)$$

No!

- Find collision $\texttt{MD5}(M_1) = \texttt{MD5}(M_2)$, then
$$\texttt{MD5}(M_1 \| S) = \texttt{MD5}(M_2 \| S).$$
- In particular, with chosen-prefix collision $(\mathrm{MD5}(P_1 || G_1) = \mathrm{MD5}(P_2 || G_2))$, appending any common suffix S still collides:
$$\mathrm{MD5}(P_1 || G_1 || S) = \mathrm{MD5}(P_2 || G_2 || S)$$

Side note: what about

- $\mathrm{MAC}_S(M) = \texttt{MD5}(S \| M)$? No (length extension)
- $\mathrm{MAC}_S(M) = \texttt{MD5}(S \| M \| S)$? Yes?* (sandwich/envelope MAC)

*assuming proper padding

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

- MitM attacker wants to forge an Access-Accept
  - Don't know shared secret, so can't compute Response Authenticator

- Attack: create an MD5 collision such that Access-Accept and Access-Reject will produce the same Response Authenticator (simplified):

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

$$\text{implies}$$

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

- Trick server into sending the Access-Reject

MD5 Collision for RADIUS Response Authenticator

Given prefixes P_1, P_2, generated collision gibberish G_1, G_2, and suffix S:

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

Applied to RADIUS:

Response Authenticator

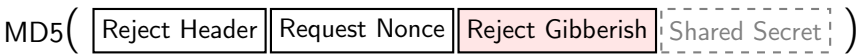
$= \text{MD5}($ Accept Header | Request Nonce | Accept Attributes | Accept Gibberish | Secret $)$

$= \text{MD5}($ Reject Header | Request Nonce | Reject Attributes | Reject Gibberish | Secret $)$

predicted prefixes P_1, P_2 — gibberish G_1, G_2 — suffix S (unknown)

Challenge 1: RejectGibberish Injection

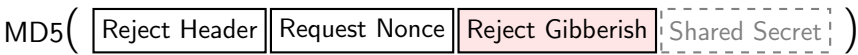
- Server needs to include `Reject Gibberish` in Response Authenticator:



How do we get it to include Reject Gibberish in its Access-Reject?

Challenge 1: RejectGibberish Injection

- Server needs to include `Reject Gibberish` in Response Authenticator:



How do we get it to include Reject Gibberish in its Access-Reject?

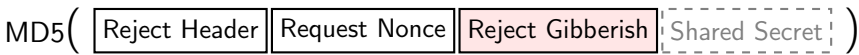
- The Proxy-State attribute:

  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, emphasis added)

Challenge 1: RejectGibberish Injection

- Server needs to include `Reject Gibberish` in Response Authenticator:

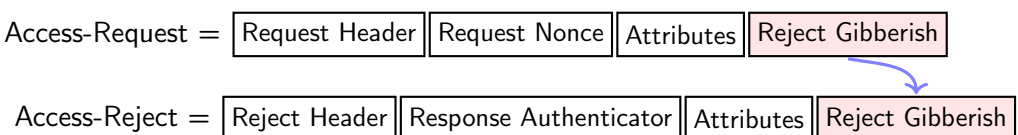


How do we get it to include Reject Gibberish in its Access-Reject?

- 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, emphasis added)



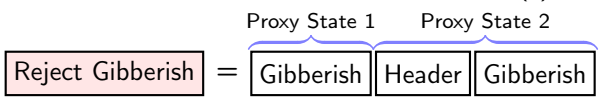
Challenge 2: Gibberish Length

Maximum length of Proxy-State is 253 bytes.
Gibberish that short would take too long to compute (we want $\approx$ 400 bytes)

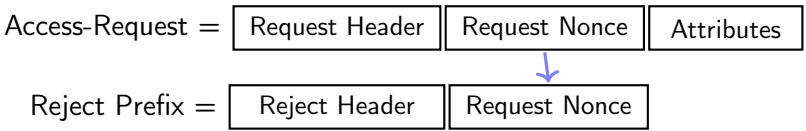
Challenge 2: Gibberish Length

Maximum length of Proxy-State is 253 bytes.
Gibberish that short would take too long to compute (we want ≈ 400 bytes)

Solution: Embed extra Proxy-State header(s) inside gibberish

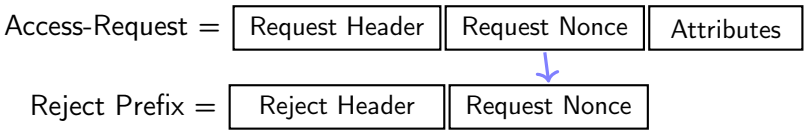


Challenge 3: Online Collision Computation



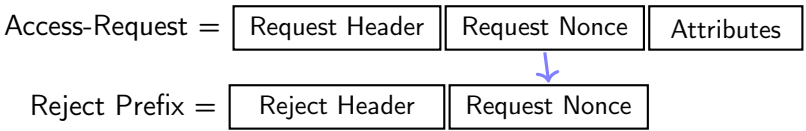
- Prefixes require knowing the `Request Nonce`.

Challenge 3: Online Collision Computation



- Prefixes require knowing the `Request Nonce`.
- Collision must be computed before RADIUS client times out.

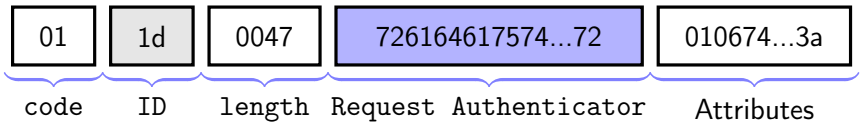
Challenge 3: Online Collision Computation



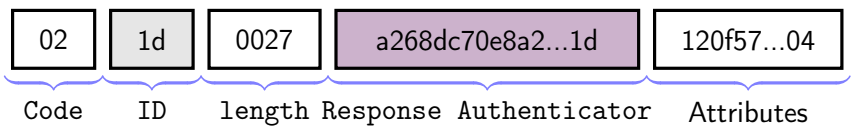
- Prefixes require knowing the `Request Nonce`.
- Collision must be computed before RADIUS client times out.
- Collision time depends on collision length and type:
  - $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$ and $\mathrm{MD5}(P||G_1) = \mathrm{MD5}(P||G_2)$ takes seconds.
  - Chosen-prefix collision of [Ste+09]: 204-byte G_1 and G_2 in 28h on 215 PS3.
  - We optimized our 428-byte collision from days to $\leq$ 5m on 47 servers.

The Juicy Details: End-to-End Example Attack (1/4)

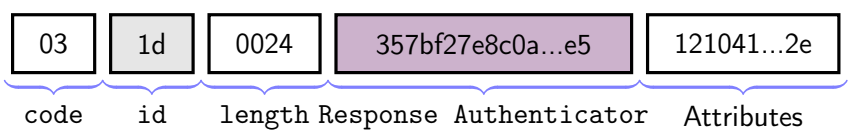
Access-Request



Access-Accept

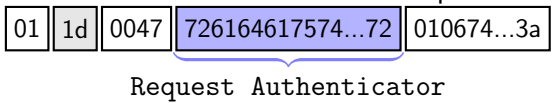


Access-Reject



The Juicy Details: End-to-End Example Attack (2/4)

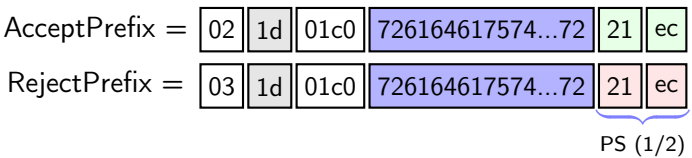
1. Attacker triggers Access-Request.
2. MITM attacker observes Access-Request.



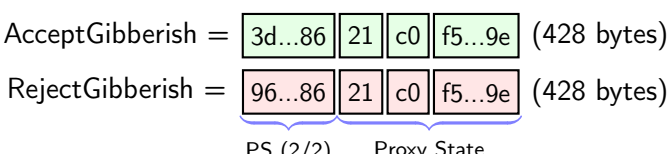
PoC example packets

`blastradius.fail/example.py`

3. MITM attacker predicts the following prefixes

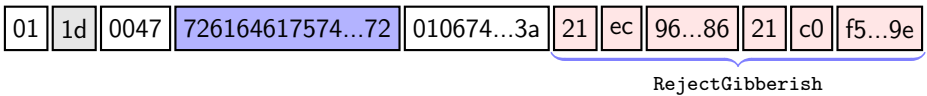


to compute the MD5 chosen-prefix collision gibberish.

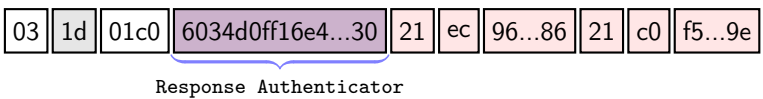


The Juicy Details: End-to-End Example Attack (3/4)

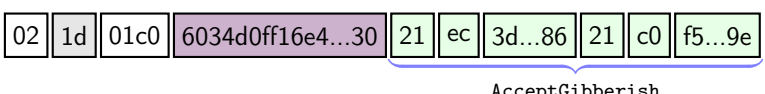
4. MITM sends Access-Request with appended `RejectGibberish` to server.



5. MITM intercepts Access-Reject, learning the Response Authenticator.

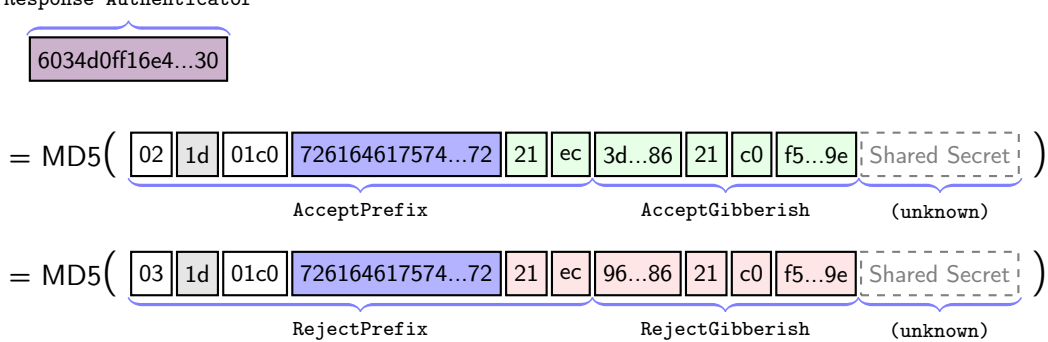


6. MITM puts Response Authenticator in Access-Accept packet with appended `AcceptGibberish`.



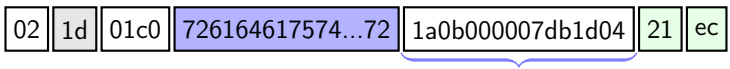
The Juicy Details: End-to-End Example Attack (4/4)

7. Access-Accept and Access-Reject produce the same Response Authenticator, and, hence, pass the RADIUS client authentication check.



Attack Extensions

- Adversary can add arbitrary attributes in prefix for Access-Accept.

AcceptPrefix = 

- Proxy-State attributes are *not* the only way to inject the `RejectGibberish`.
  - Any reflected user input could work, e.g. the User-Name or Vendor-Specific attributes.
    - In Access-Request:
      `User-Name: 0PZjN-_ayr83S-nc6q...Mt85`
    - In Access-Reject:
      `Reply-Message: Login for 0PZjN-_ayr83S-nc6q...Mt85 failed!`
  - The client does not need to support or parse these attributes.

Impact

Affected modes:

- PAP, CHAP, MS-CHAP are vulnerable
- EAP modes likely not vulnerable (require Message-Authenticator)

Impact

Affected modes:

- PAP, CHAP, MS-CHAP are vulnerable
- EAP modes likely not vulnerable (require Message-Authenticator)

Affected deployments: Requires MITM network access

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

Impact

Affected modes:

- PAP, CHAP, MS-CHAP are vulnerable
- EAP modes likely not vulnerable (require Message-Authenticator)

Affected deployments: Requires MITM network access

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

Timing:

- RADIUS client timeouts $\leq$ 1m, our PoCs take $\approx$ 5m.
- Optimizations feasible: parallelizes well, hardware implementation.

Mitigations

- Massive disclosure with 90+ vendors.
- Challenges: widespread, backwards compatibility.

Some power plants use RADIUS [TKSA14].

Mitigations

- Massive disclosure with 90+ vendors.
- Challenges: widespread, backwards compatibility.

Short-term:

- Message-Authenticator attribute uses HMAC-MD5 not vulnerable to MD5 collisions.
- All requests and responses should include and verify Message-Authenticator.

Some power plants use RADIUS [TKSA14].

Mitigations

- Massive disclosure with 90+ vendors.
- Challenges: widespread, backwards compatibility.

Short-term:

- Message-Authenticator attribute uses HMAC-MD5 not vulnerable to MD5 collisions.
- All requests and responses should include and verify Message-Authenticator.

Long-term:

- Encapsulate all RADIUS traffic in (D)TLS tunnel.
- Current IETF draft is being standardized [RW24].

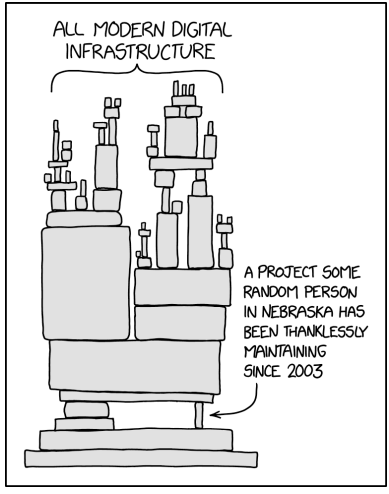
Some power plants use RADIUS [TKSA14].

Blast-RADIUS attack

Attack summary: MD5 collision attack on RADIUS authentication by MitM adversary.

`https://blastradius.fail`

RADIUS/UDP Considered Harmful
Sharon Goldberg, Miro Haller, Nadia Heninger, Mike Milano, Dan Shumow, Marc Stevens, and Adam Suhl.
USENIX Security, August 2024.



References

References I

[dB94] Bert den Boer and Antoon Bosselaers. "Collisions for the Compression Function of MD5". In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, Germany, May 1994, pp. 293–304. DOI: 10.1007/3-540-48285-7_26.

[DeK24] Alan DeKok. *RADIUS and MD5 Collision Attacks*. https://networkradius.com/assets/pdf/radius_and_md5_collisions.pdf. 2024.

[RW24] Jan-Frederik Rieckers and Stefan Winter. *(Datagram) Transport Layer Security ((D)TLS Encryption for RADIUS*. Internet-Draft draft-ietf-radext-radiusdtls-bis-02. Work in Progress. Internet Engineering Task Force, July 2024. 38 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-radext-radiusdtls-bis/02/.

References II

[SLW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. "Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities". In: *EUROCRYPT*. Vol. 4515. Lecture Notes in Computer Science. Springer, 2007, pp. 1–22.

[Ste+09] Marc Stevens et al. "Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate". In: *CRYPTO*. Vol. 5677. Lecture Notes in Computer Science. Springer, 2009, pp. 55–69.

[TKSA14] Henrik Thejl, Nagaraja K S, and Karl-Georg Aspacher. "A method for user management and a power plant control system thereof for a power plant system". Pat. 2765466. Siemens Gamesa Renewable Energy A/S. Jan. 24, 2014. URL: `https://data.epo.org/publication-server/rest/v1.0/publication-dates/20190904/patents/EP2765466NWB1/document.pdf`.

References III

[Wag+] David Wagner et al. *Computer Security*. https://textbook.cs161.org/. accessed on Oct 6, 2024.

[WY05] Xiaoyun Wang and Hongbo Yu. "How to Break MD5 and Other Hash Functions". In: *EUROCRYPT*. Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 19–35.