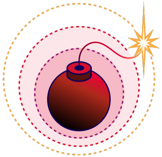
RADIUS/UDP Considered Harmful

The Blast-RADIUS Attack

Sharon Goldberg[1], Nadia Heninger[2], **Miro Haller**[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

August 16, 2024



Attack Summary

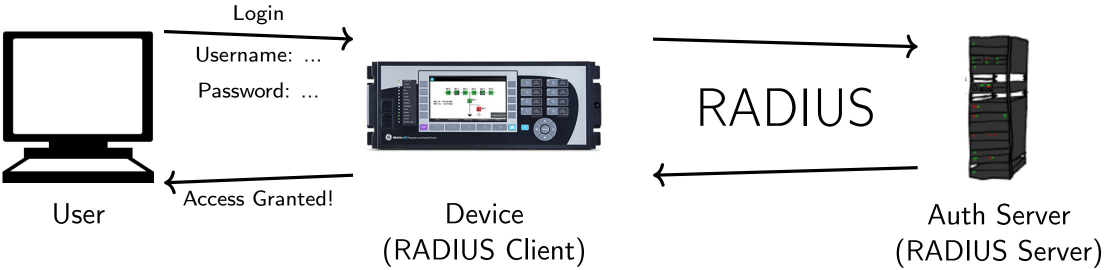
MitM network attacker can forge arbitrary RADIUS responses (for non-EAP authentication modes)

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.

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 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, lead developer of FreeRADIUS, [DeK24])*

- Backbone routers
- VPNs
- ISP infrastructure (DSL/FTTH)
- IoT devices
- Identity Providers and MFA (Okta, Duo)
- Power grid equipment
- Not vulnerable to this attack: 802.1X, enterprise WiFi, eduroam

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)*

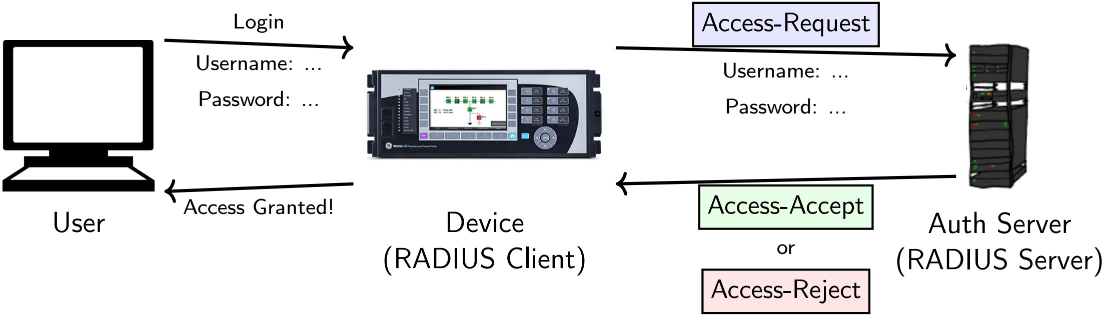
RADIUS still uses 90s-era cryptography

- 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

$$\text{Access-Request} = \underbrace{\boxed{\text{Request Header}}}_{\texttt{4 bytes}}\ \underbrace{\boxed{\text{Request Nonce}}}_{\texttt{16 random bytes}}\ \underbrace{\boxed{\text{Attributes}}}_{\begin{array}{l}\texttt{User-Name test}\\ \texttt{Password Mjg2NzU1z}\end{array}}$$

$$\text{Access-Accept} = \underbrace{\boxed{\text{Accept Header}}}_{\texttt{4 bytes}}\ \underbrace{\boxed{\text{Response Authenticator}}}_{\texttt{16 byte ``MAC''}}\ \underbrace{\boxed{\text{Attributes}}}_{\begin{array}{l}\texttt{Reply-Message Welcome test!}\\ \texttt{Exec-Privilege 4}\end{array}}$$

$$\text{Access-Reject} = \underbrace{\boxed{\text{Reject Header}}}_{\texttt{4 bytes}}\ \underbrace{\boxed{\text{Response Authenticator}}}_{\texttt{16 byte ``MAC''}}\ \underbrace{\boxed{\text{Attributes}}}_{\texttt{Reply-Message Access denied}}$$

Response Authenticator

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

The Response Authenticator from packet

Response Header	Response Authenticator	Attributes

is computed as

copied from request   fixed, pre-configured

MD5 (| Response Header | Request Nonce | Attributes | Shared Secret |).

copied from response

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

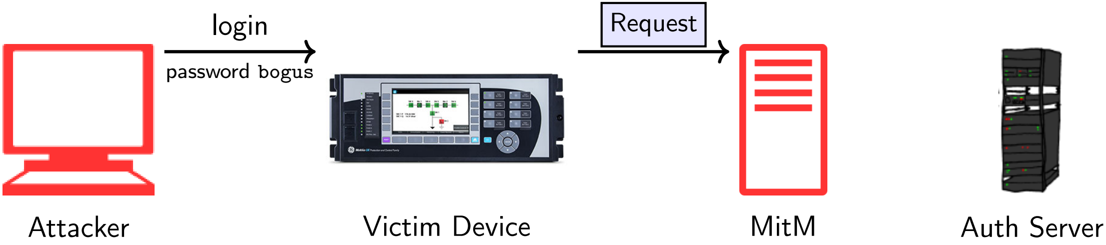
- MitM attacker wants to forge an Access-Accept
  - Don't know shared secret, so can't compute Response Authenticator
- Attack: create an MD5 collision such that Access-Accept and Access-Reject will produce the same Response Authenticator (very simplifed):

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

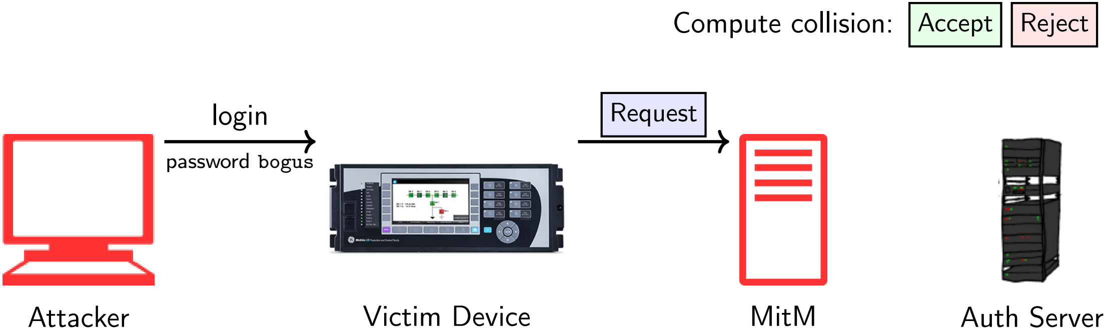
- Trick server into sending the Access-Reject

Blast-RADIUS Attack Overview

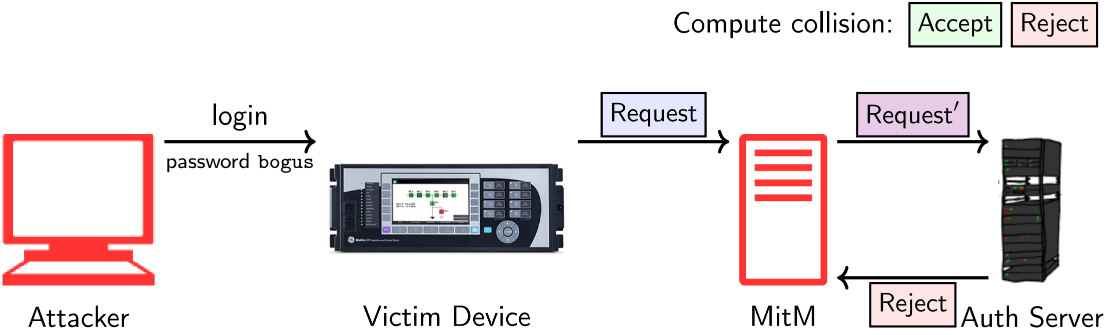
Blast-RADIUS Attack Overview



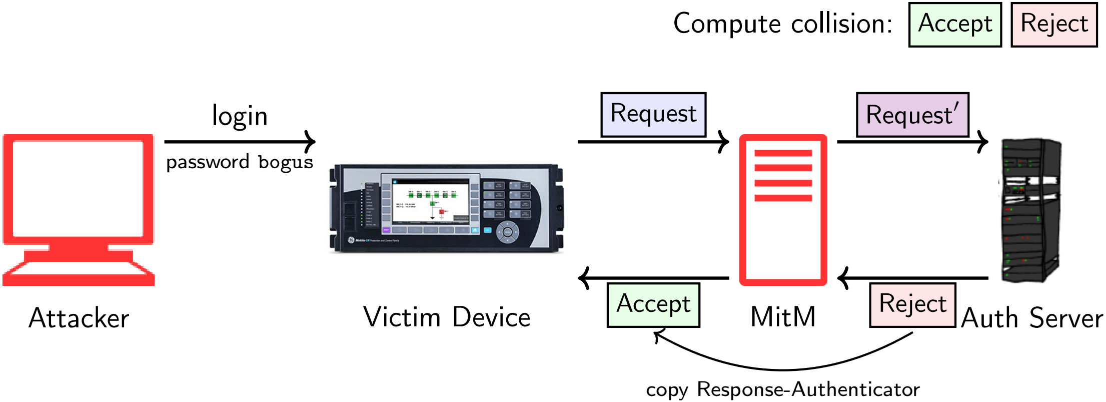
Blast-RADIUS Attack Overview



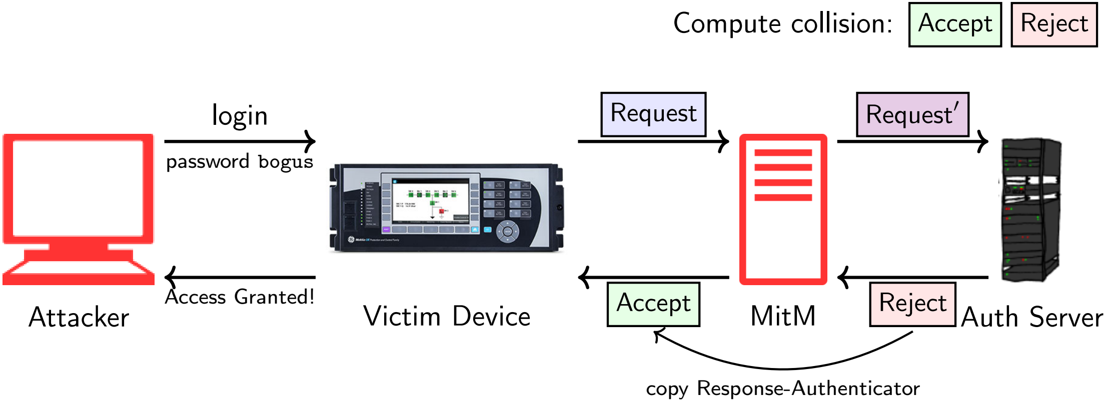
Blast-RADIUS Attack Overview



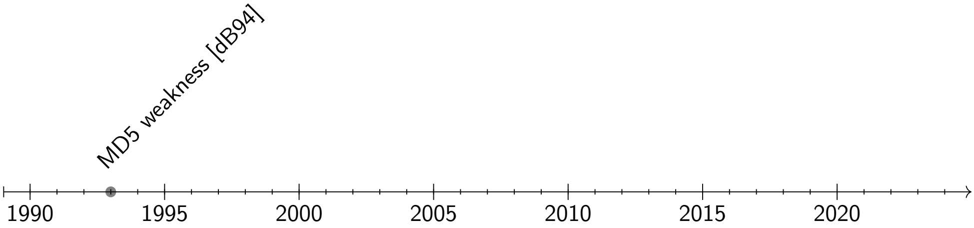
Blast-RADIUS Attack Overview



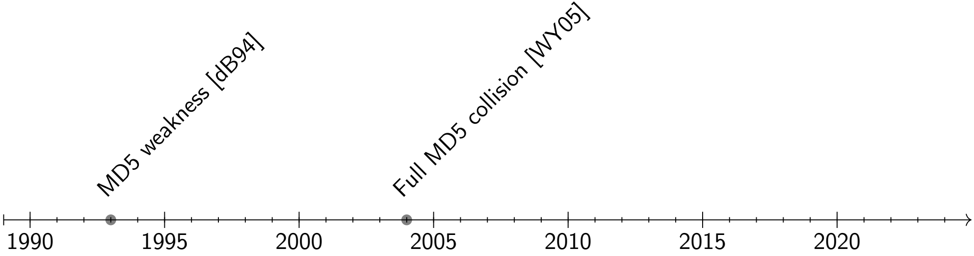
Blast-RADIUS Attack Overview



MD5 Collision Attack History

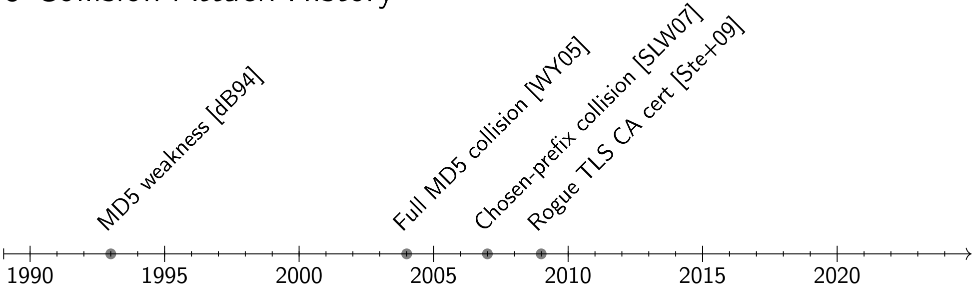


MD5 Collision Attack History



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

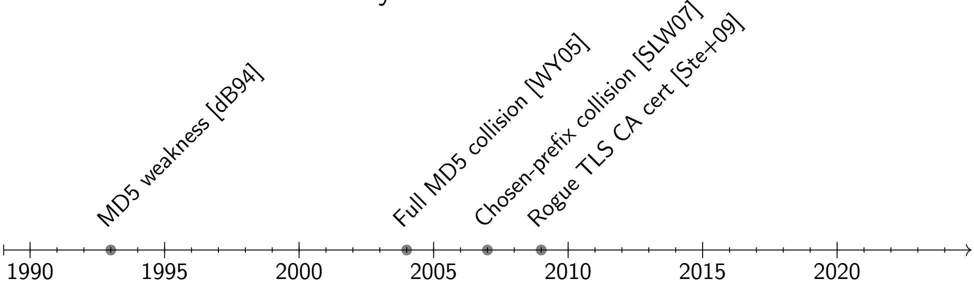
MD5 Collision Attack History



- 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



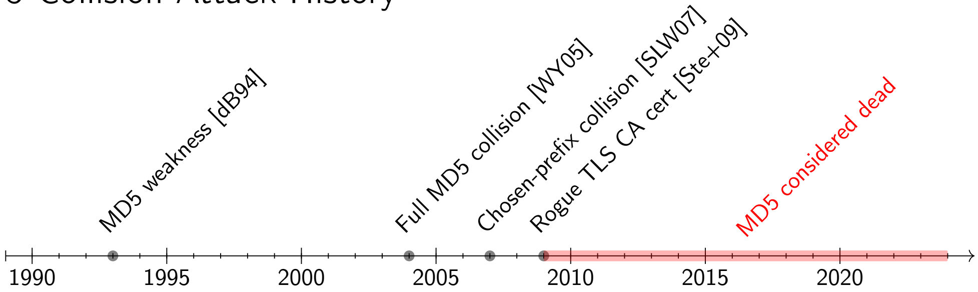
- MD5 collision: unstructured strings G_1, G_2 with $\text{MD5}(G_1) = \text{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

- Appending any common suffix S still collides:

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

MD5 Collision Attack History



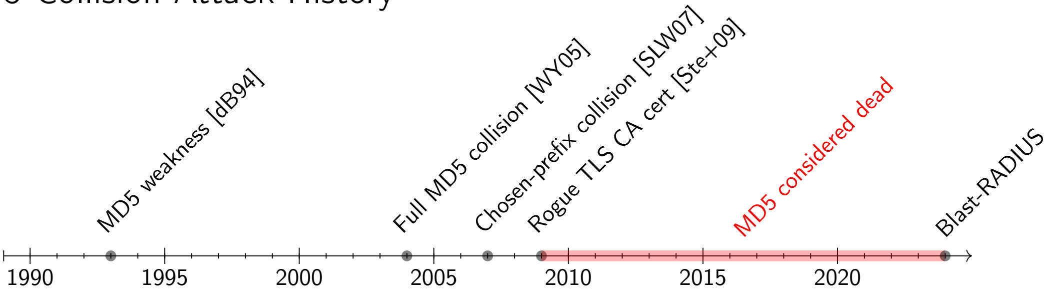
- MD5 collision: unstructured strings G_1, G_2 with $\text{MD5}(G_1) = \text{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

- Appending any common suffix S still collides:

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

MD5 Collision Attack History



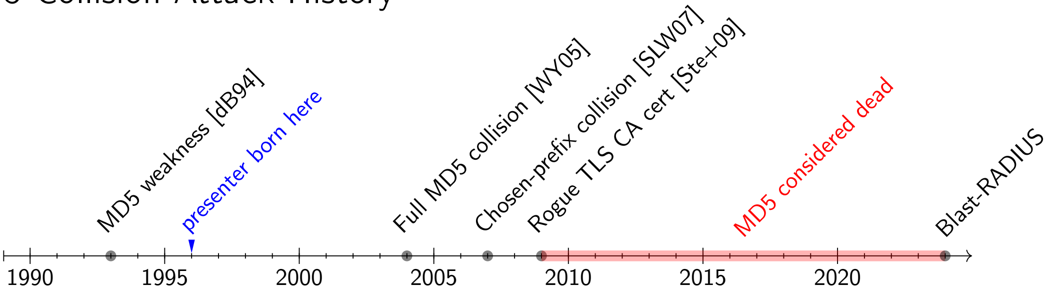
- MD5 collision: unstructured strings G_1, G_2 with $\mathrm{MD5}(G_1) = \mathrm{MD5}(G_2)$.
- Chosen-prefix collision: given prefixes P_1, P_2, produces G_1, G_2 such that:

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

- Appending any common suffix S still collides:

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

MD5 Collision Attack History



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

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

- Appending any common suffix S still collides:

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

MD5 Collision for RADIUS Response Authenticator

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

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

Applied to RADIUS:

$$\boxed{\text{Response Authenticator}}$$

$$= \mathrm{MD5}\left(\boxed{\text{Accept Header}}\boxed{\text{Request Nonce}}\boxed{\text{Accept Gibberish}}\boxed{\text{Shared Secret}} \right)$$

$$= \mathrm{MD5}\left(\underbrace{\boxed{\text{Reject Header}}\boxed{\text{Request Nonce}}}_{\text{predicted prefixes } P_1,\ P_2}\underbrace{\boxed{\text{Reject Gibberish}}}_{\text{gibberish } G_1,\ G_2}\underbrace{\boxed{\text{Shared Secret}}}_{\text{suffix } S \text{ (unknown)}} \right)$$

Challenge 1: RejectGibberish Injection

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

$$\mathrm{MD5}\left(\ \boxed{\text{Reject Header}}\boxed{\text{Request Nonce}}\boxed{\text{Reject Gibberish}}\boxed{\text{Shared Secret}}\ \right)$$

Challenge 1: RejectGibberish Injection

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

  MD5(| Reject Header | Request Nonce | Reject Gibberish | Shared Secret |)

- 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:

$$\mathrm{MD5}\Big(\boxed{\text{Reject Header}}\boxed{\text{Request Nonce}}\boxed{\text{Reject Gibberish}}\boxed{\text{Shared Secret}}\Big)$$

- 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)

$$\boxed{\text{Reject Gibberish}} = \overbrace{\boxed{\text{Header}}\boxed{\text{Gibberish}}}^{\text{Proxy State}}\overbrace{\boxed{\text{Header}}\boxed{\text{Gibberish}}}^{\text{Proxy State}}$$

$$\text{Access-Request} = \boxed{\text{Request Header}}\boxed{\text{Request Nonce}}\boxed{\text{Attributes}}\boxed{\text{Reject Gibberish}}$$

Challenge 2: Online Collision Computation

Access-Request = $\boxed{\text{Request Header}}\boxed{\text{Request Nonce}}\boxed{\text{Attributes}}$

Reject Prefix = $\boxed{\text{Reject Header}}\boxed{\text{Request Nonce}}$

- Prefixes require knowing the `Request Nonce`.

Challenge 2: Online Collision Computation

Access-Request =	Request Header	Request Nonce	Attributes
Reject Prefix =	Reject Header	Request Nonce	

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

Challenge 2: Online Collision Computation

Access-Request = | Request Header | Request Nonce | Attributes |

Reject Prefix = | Reject Header | Request Nonce |

- 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.

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:

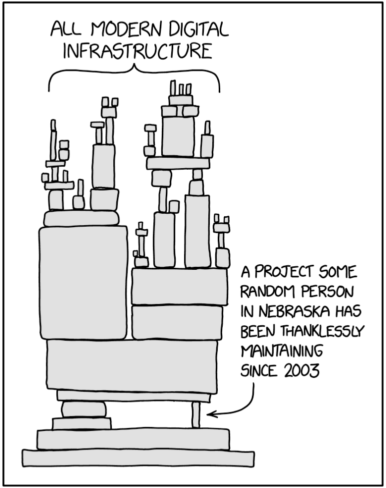
- Encapsulate all RADIUS traffic in (D)TLS tunnel.
- Current IETF draft is being standardized [RW24].

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.

[DeK23] Alan DeKok. *Deprecating Insecure Practices in RADIUS*. Internet-Draft draft-ietf-radext-deprecating-radius-00. Work in Progress. Internet Engineering Task Force, Nov. 2023. 34 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-radext-deprecating-radius/00/.

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

References II

[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/`.

[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.

References III

[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.

[Wie+12] Klaas Wierenga et al. *Transport Layer Security (TLS) Encryption for RADIUS*. RFC 6614. May 2012. DOI: 10.17487/RFC6614. URL: https://www.rfc-editor.org/info/rfc6614.

[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.

Backup Slides

Blast-RADIUS Attack Example (1/3)

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

$$\boxed{01}\ \boxed{1d}\ \boxed{0047}\ \underbrace{\boxed{726164617574...72}}_{\texttt{Request Authenticator}}\ \boxed{010674...3a}$$

3. MITM attacker predicts the following prefixes

$$\text{AcceptPrefix} = \boxed{02}\ \boxed{1d}\ \boxed{01c0}\ \boxed{726164617574...72}$$

$$\text{RejectPrefix} = \boxed{03}\ \boxed{1d}\ \boxed{01c0}\ \boxed{726164617574...72}$$

to compute the MD5 chosen-prefix collision gibberish.

$$\text{AcceptGibberish} = \boxed{21}\ \boxed{ec}\ \boxed{3d...86}\ \boxed{21}\ \boxed{c0}\ \boxed{f5...9e}\ \text{(428 bytes)}$$

$$\text{RejectGibberish} = \underbrace{\boxed{21}\ \boxed{ec}\ \boxed{96...86}}_{\text{Proxy State}}\ \underbrace{\boxed{21}\ \boxed{c0}\ \boxed{f5...9e}}_{\text{Proxy State}}\ \text{(428 bytes)}$$

PoC example packets

`blastradius.fail/example.py`

Blast-RADIUS Attack Example (2/3)

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

01	1d	0047	726164617574...72	010674...3a	21	ec	96...86	21	c0	f5...9e

`RejectGibberish`

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

03	1d	01c0	6034d0ff16e4...30	21	ec	96...86	21	c0	f5...9e

`Response Authenticator`

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

02	1d	01c0	6034d0ff16e4...30	21	ec	3d...86	21	c0	f5...9e

`AcceptGibberish`

Blast-RADIUS Attack Example (3/3)

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

$$\overbrace{\boxed{\text{6034d0ff16e4...30}}}^{\texttt{Response Authenticator}}$$

$$= \text{MD5}\Big(\underbrace{\boxed{\text{02}}\boxed{\text{1d}}\boxed{\text{01c0}}\boxed{\text{726164617574...72}}}_{\texttt{AcceptPrefix}}\underbrace{\boxed{\text{21}}\boxed{\text{ec}}\boxed{\text{3d...86}}\boxed{\text{21}}\boxed{\text{c0}}\boxed{\text{f5...9e}}}_{\texttt{AcceptGibberish}}\boxed{\text{Shared Secret}}\Big)$$

$$= \text{MD5}\Big(\underbrace{\boxed{\text{03}}\boxed{\text{1d}}\boxed{\text{01c0}}\boxed{\text{726164617574...72}}}_{\texttt{RejectPrefix}}\underbrace{\boxed{\text{21}}\boxed{\text{ec}}\boxed{\text{96...86}}\boxed{\text{21}}\boxed{\text{c0}}\boxed{\text{f5...9e}}}_{\texttt{RejectGibberish}}\boxed{\text{Shared Secret}}\Big)$$

Attack Extensions

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

$$\text{AcceptPrefix} = \boxed{02}\,\boxed{1d}\,\boxed{01c0}\,\boxed{726164617574...72}\,\underbrace{\boxed{1a0b000007db1d04}}_{\substack{\text{Attribute:} \\ \texttt{Exec-Privilege 04}}}$$

- 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.