

pysap

protocols, tools and
offensive capabilities for
SAP security

YOLO edition

Martin Gallo
9/24/2026
Nodo Serrano, Tandil

YOLO edition

Experiment

Can we craft a PoC for a real SAP vulnerability during the talks' time?

SAP What?

SAP

German **enterprise software** company that has spent 50+ years building software for how **businesses run**.

+110k
employees
worldwide

€36.8B total
revenue



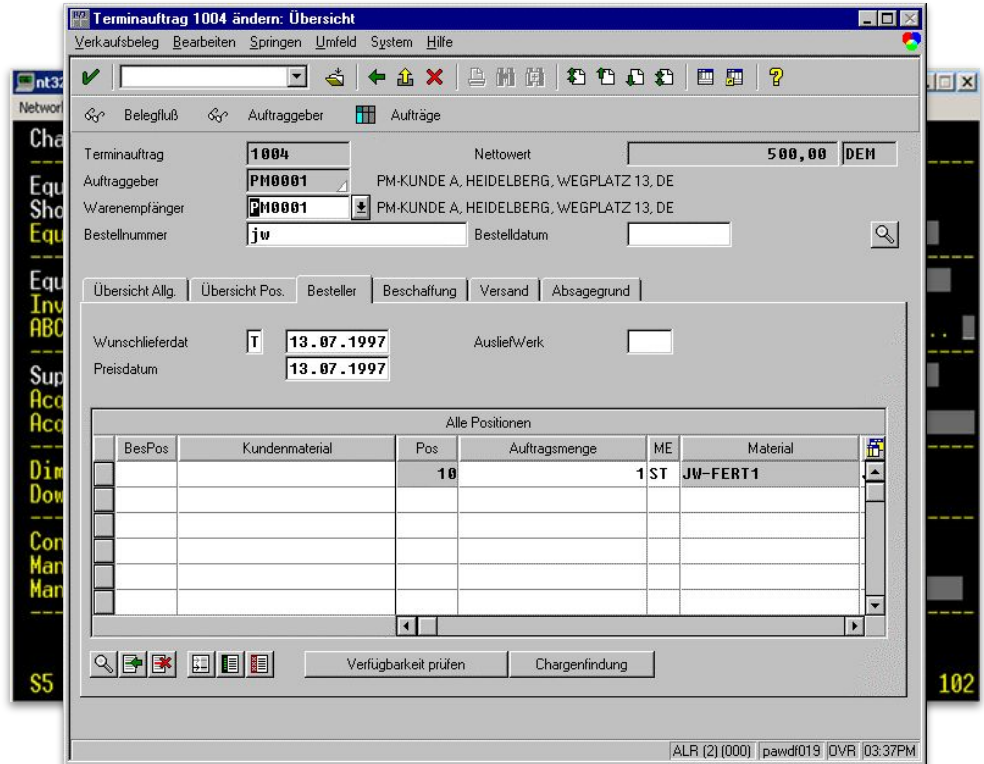
SAP Architecture through the years..

SAP R/1 ~1973

SAP R/2 ~1980

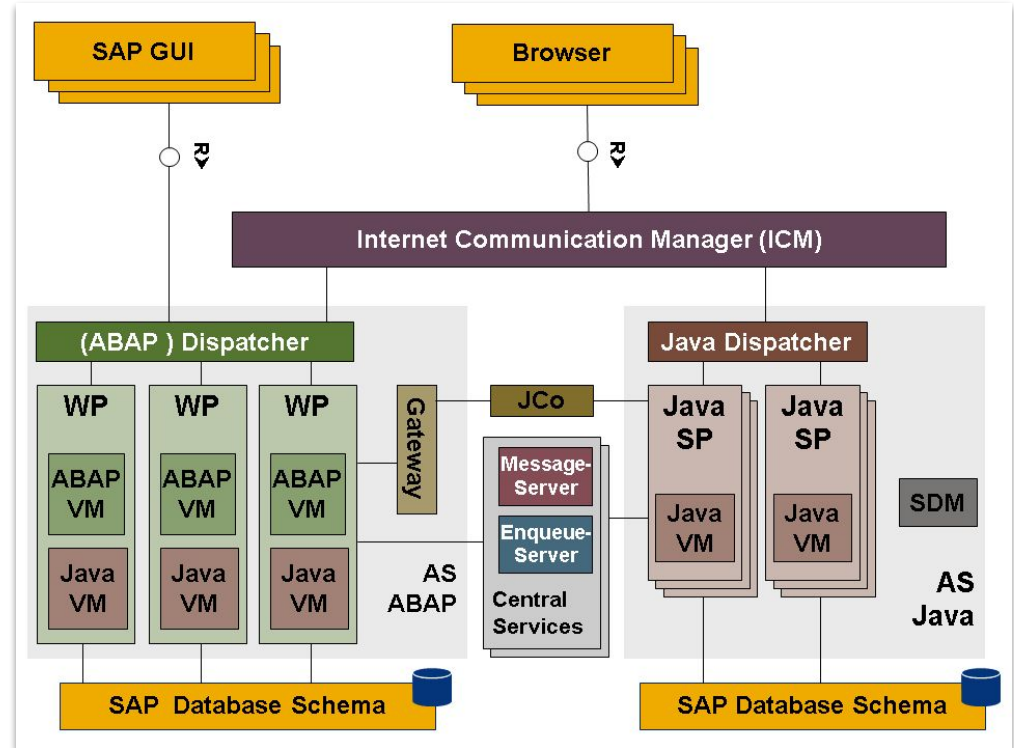
SAP R/3 ~1992

[A Brief History of SAP: An Introduction for the Uninitiated](#)



SAP Architecture through the years..

SAP Netweaver and SAP
ECC
circa ~2004

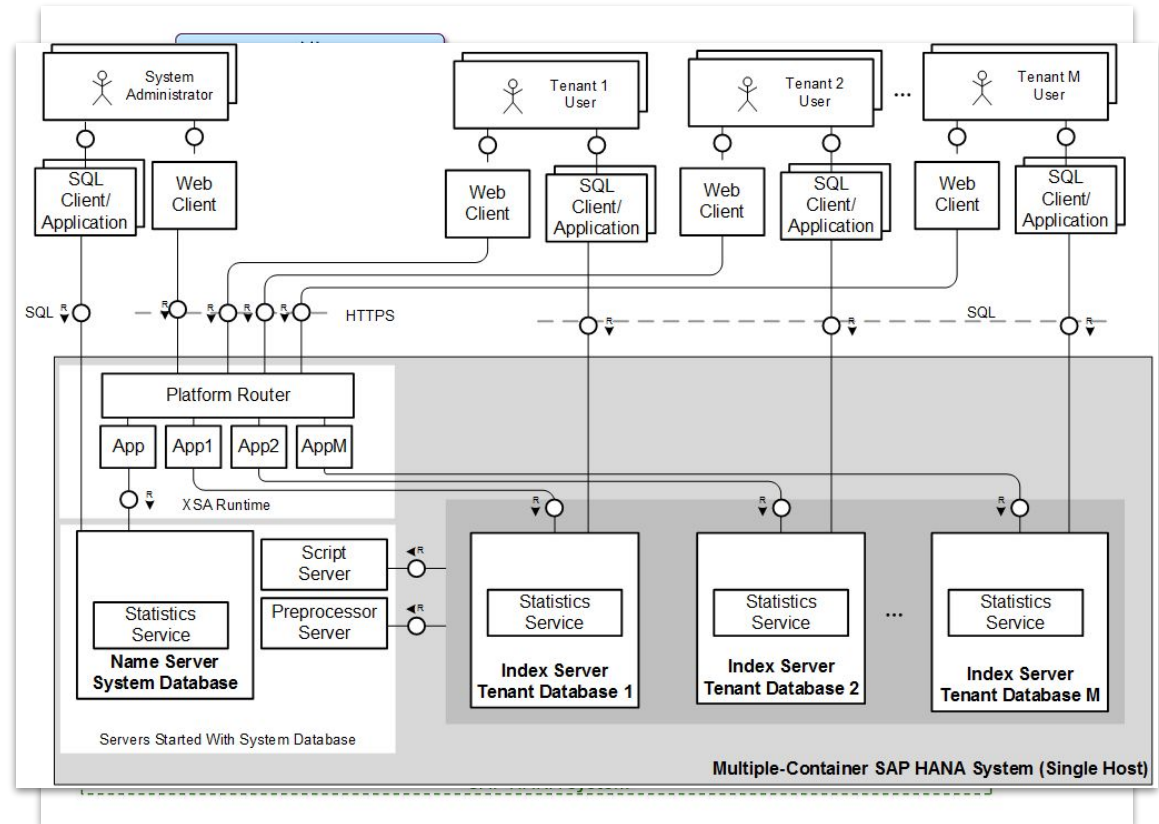


[Architecture of the SAP NetWeaver Application Server](#)

SAP Architecture through the years..

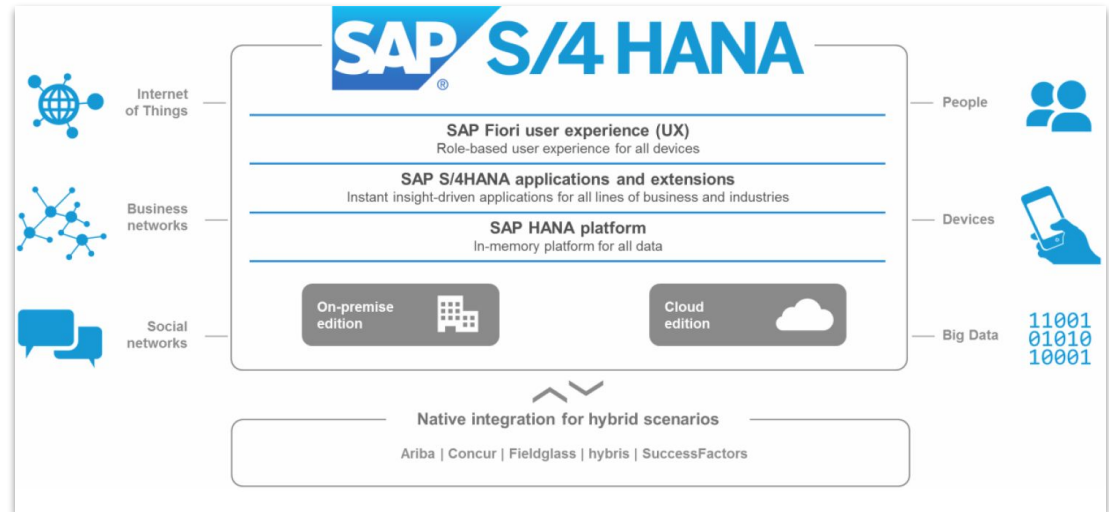
SAP HANA
circa ~2011

[SAP HANA Database Architecture](#)



SAP Architecture through the years..

SAP S4/HANA
circa ~2015

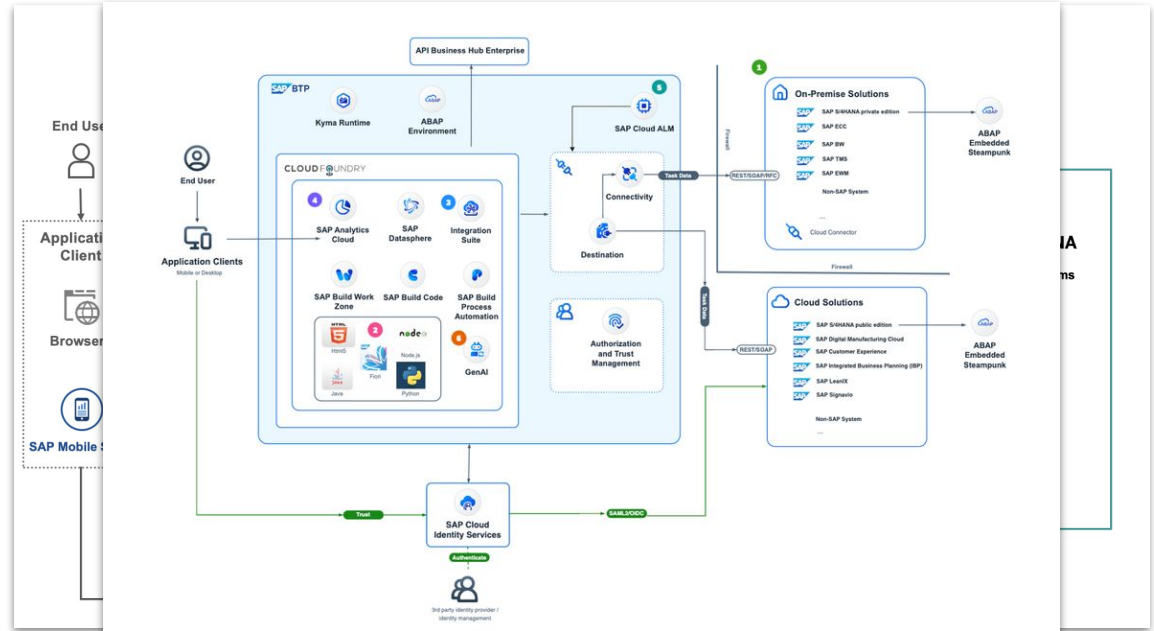


[SAP HANA Database Architecture](#)

SAP Architecture through the years..

SAP BTP (Business
Technology Platform)
circa ~2021

[Beginner Level Understanding on SAP BTP Architecture](#)
[Intermediate Level Understanding of SAP BTP Architecture](#)

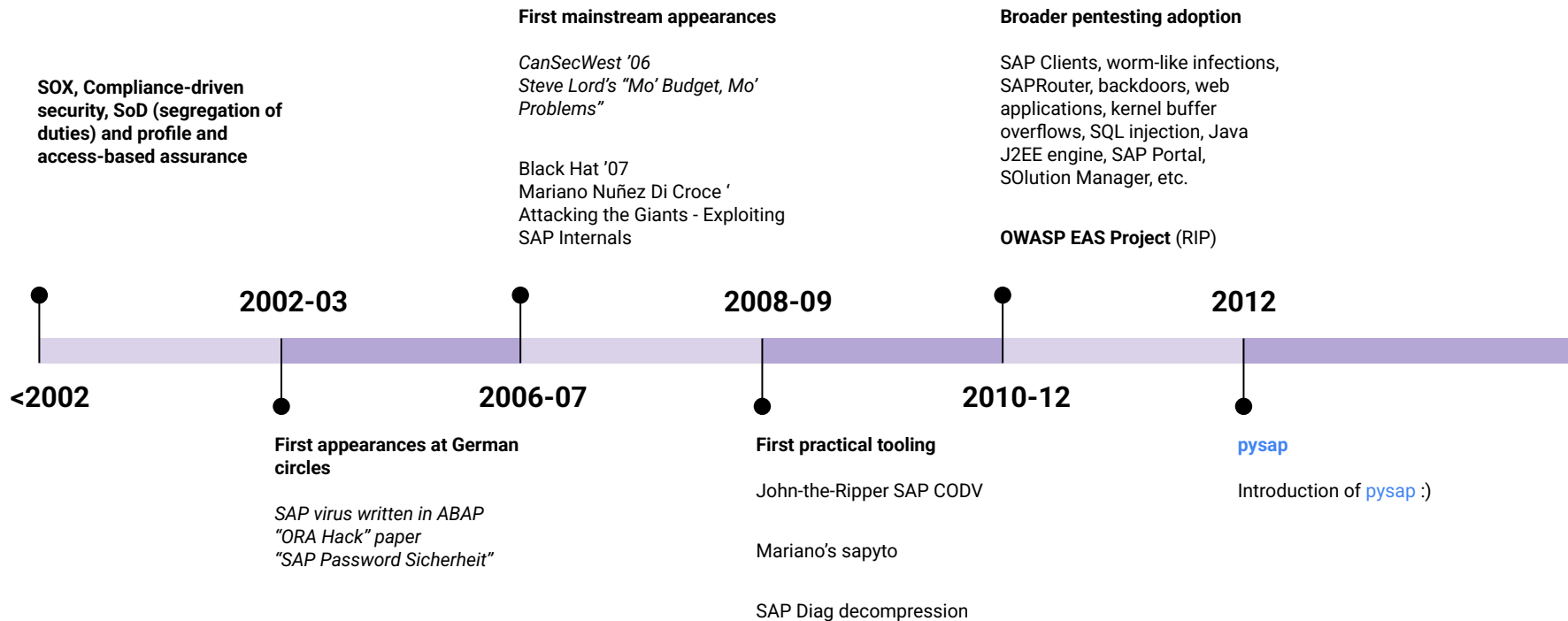


Enter `pysap`

Why **pysap** exists?

- **Access** to SAP packages and systems is challenging
 - \$\$\$, licensing, export restrictions, etc.
- **Setup, maintenance and configuration** ~~is~~ was challenging
 - \$\$\$, scarce knowledge
- SAP security ~~is~~ was very **niche-ish, not** too much **knowledge sharing**
 - closed culture, mostly dominated by SoD/IAM
 - only few vendors still owning the space
- SAP relies heavily on **proprietary protocols** and **custom crypto**
 - licensing, export restrictions, FIPS compliance, etc.
 - custom programming language with VM/compiler
 - roll your own crypto what?
- SAP security research required **first** understanding the protocol
 - understand/reverse the protocol -> implement it enough -> investigating -> cycle
- SAP attack surface is **massive**

SAP early (cyber)security history



pysap at work

Python-based, built on top of [Scapy](#)

Declarative packet definitions for each major protocol and file format

Client and **helper** modules for crafting and sending packets

Scripts for some common “stable” tools



```
# SAP Diag packet
class SAPDiag(PacketNoPadded):
    """SAP Diag packet

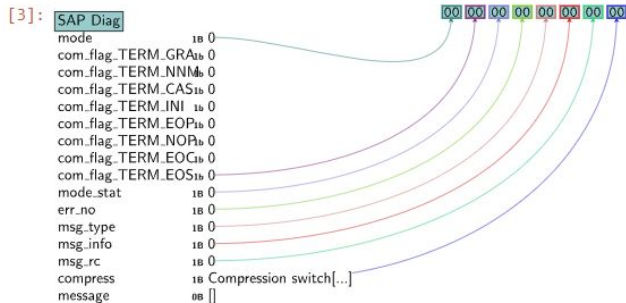
    This packet holds the Diag Header and serve as a container for
    :class:`SAPDiagItem` items. It handles compression/decompression, adding the
    appropriate Compression Header when necessary.
    """
    name = "SAP Diag"
    fields_desc = [
        ByteField("mode", 0),

        # Communication flags
        BitField("com_flag_TERM_GRA", 0, 1),
        BitField("com_flag_TERM_NNM", 0, 1),
```

SAP Diag packet

```
[3]: p = SAPDiag()
     p.canvas_dump()
```

```
[3]: SAP Diag
mode 18 0
com_flag_TERM_GRA 18 0
com_flag_TERM_NNM 18 0
com_flag_TERM_CAS 18 0
com_flag_TERM_INI 18 0
com_flag_TERM_EOP 18 0
com_flag_TERM_NOP 18 0
com_flag_TERM_EOC 18 0
com_flag_TERM_EOS 18 0
mode.stat 18 0
err_no 18 0
msg_type 18 0
msg_info 18 0
msg_rc 18 0
compress 18 Compression switch[...]
message 0b []
```

A diagram showing the mapping of fields from the SAP Diag packet structure to a 16-bit hex dump. The hex dump is represented as a row of 16 colored boxes (00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00). Colored lines connect specific fields to their corresponding positions in the hex dump: mode (green) to the 1st box, com_flag_TERM_GRA (purple) to the 2nd, com_flag_TERM_NNM (blue) to the 3rd, com_flag_TERM_CAS (red) to the 4th, com_flag_TERM_INI (orange) to the 5th, com_flag_TERM_EOP (yellow) to the 6th, com_flag_TERM_NOP (light green) to the 7th, com_flag_TERM_EOC (light blue) to the 8th, com_flag_TERM_EOS (light purple) to the 9th, mode.stat (light green) to the 10th, err_no (light blue) to the 11th, msg_type (light green) to the 12th, msg_info (light blue) to the 13th, msg_rc (light green) to the 14th, and compress (light blue) to the 15th. The message field is shown as an empty list at the end.

pysap at work

Network Protocols

- **NI** - the underlying networking layer
- **Diag** - the dispatcher service - *aka the “GUI” protocol*
- **RFC (Remote Function Call) / NWRFC (Netweaver RFC)** - the gateway for RPC calls
- **Message Server** - the inter service message broker
- **SNC (Secure Network Connections)** - the crypto layer
- **HDB (HANA SQL Command Network)** - the database service layer
- **Router** - the application-level routing/proxying
- **Internet Graphic Server** - the graphic rendering service - *nobody knows why it's there*
- **Enqueue** - the HA enqueue service layer

File Formats

- **SAR** - archive files
- **PSE (Personal Security Environment)** - crypto and certificates storage
- **SSO Credentials** - PIN/cred storage for non-interactive access
- **SSFS (Secure Storage in File System)** - credentials storage

Other capabilities

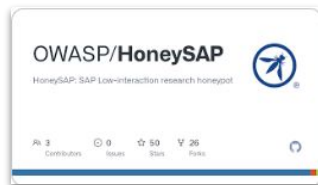
- **LZH / LZC** - compression algorithm in pure Python
- **Client / server classes**, command-line tools, example **scripts**, etc.

pysap at work

What **you** can build?

- **Scripts** and small **tools** to exercise the protocols
- Research known/published **vulnerabilities**
- **Fuzzers**, with mutation that's **protocol-aware**
- **Differential** and **patch analysis** via the exposed protocols
- etc.

Sibling projects

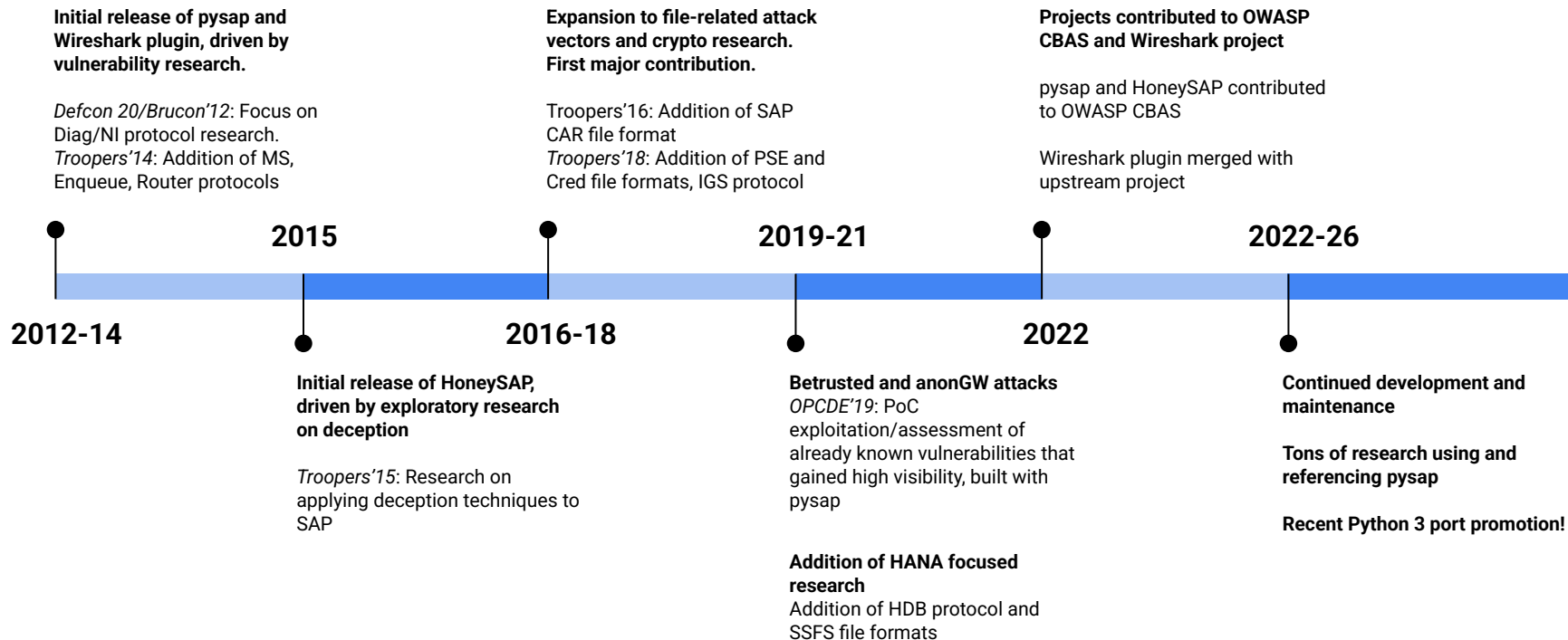


Low-interaction
research honeypot

Protocol dissectors
for network
monitoring tools


WIRESHARK

Little history of pysap



Showtime!

AI + Offensive capabilities

=

?

Thesis: AI + Offensive capabilities = research agility

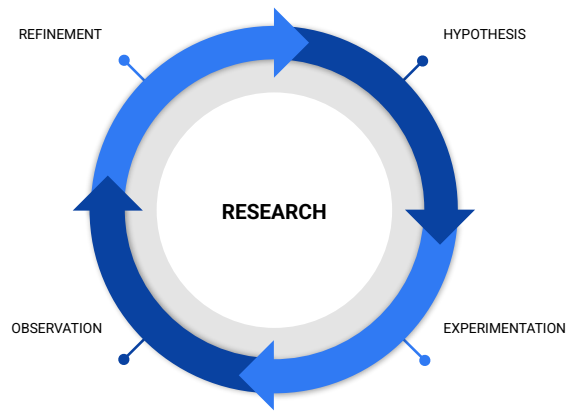
LLMs/Agents can help tremendously

- reverse engineering, generating experiments, analyzing responses, etc.

But LLMs/agents are **not deterministic!**

Harness - protocol primitives = wasted tokens

Harness + protocol abstractions = research agility



YOLO edition

Experiment Results

PoC | GTFO + Offensive capabilities
=
research agility

PoC | | GTFO + Offensive capabilities = **research agility**

Nowadays

- **Generating code** and **tools** is **cheap**
- **Candidate bugs, PoCs** and **one-off tooling** is becoming **cheaper**, but **relevance** is **shorter**

Offensive capabilities become **more valuable** by making **experimentation cheaper**

- Domain knowledge, protocol understanding, reusable primitives, ability to experiment, etc
- Provide agents with *deterministic primitives*

PoC | | GTFO + Offensive
capabilities

>>

CVEs/exploits

Wanna play with **pysap**?

<https://github.com/OWASP/pysap>

<https://github.com/OWASP/HoneySAP>

<https://playbook.securitysilverbacks.com/>

[ABAP Cloud Developer Trial 2025 Available Now - SAP Community](#)
[Trials and Downloads - SAP Developer](#)

Thanks!

@martingalloar

martingallo.ar

