网络空间威胁对抗与防御技术研讨会
暨 第九届安天网络安全冬训营

乱雲飛渡

资源代价与安全算力

量时度力

——威胁检测引擎的优化与算力支撑

安天 | 基础引擎中心

CONTENTS

目 录

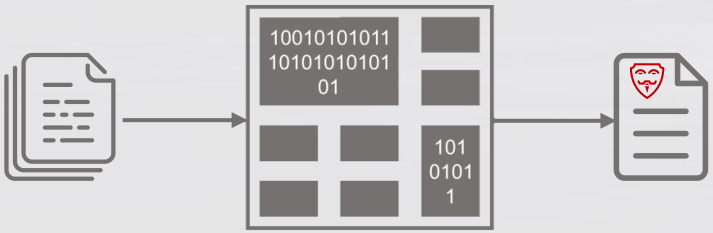
01 威胁检测引擎简介

威胁检测技术与时俱进；引擎支撑高级威胁对抗

什么是威胁检测引擎?

• 威胁检测引擎的定义

一组依靠可维护规则的数据结构 通过接口调用能够对输入对象进行病毒检测处理的程序模块的统称。就像汽车的发动机是汽车的核心动力来源一样，威胁检测引擎为威胁检测产品提供着核心的鉴定能力，只要将待检测对象传入引擎，引擎即能输出对该对象的鉴定结果



• 威胁检测引擎的应用场景

威胁检测引擎不仅可以支持传统主机反病毒产品，还可以应用到各种终端、软件、应用、设备、服务以及新兴场景中。

终端
移动设备、自助终端、POS机、工作站、物联网终端等

设备
交换机、路由器、防火墙、UTM和安全网关等

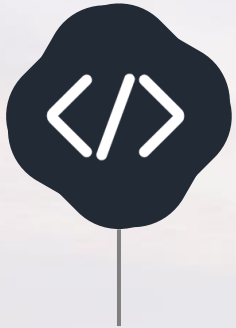
软件
各类主机、服务器和存储系统安全软件等

服务
数据中心服务和运营商级数据处理

应用
邮件服务、Web服务和代理服务等

新兴场景
工业互联网、物联网、大数据、云和5G 等

威胁检测引擎的三高技术门槛



高技术难度

依赖先进科学的体系架构。面对日渐规模化、工程化、产业化的攻击者，只有以**先进的检测技术**与之抗衡，才能为网络安全保驾护航

高开发量

依赖大规模的分析维护团队和足够强大的计算能力。全球每天新产生的恶意代码接近百万，面对如此海量的数据，只有足够**规模的团队**和算力才能够支撑这项工作

高维护量

依赖长时间样本和经验的积累。与恶意代码进行对抗一方面要有对样本的广泛**捕获**和积累，以此作为分析的基础；另一方面要**有经验丰富**的工程师团队针对其进行分析与作业，以此获得检测能力

持续演进的安天威胁检测引擎

下一代威胁检测引擎的定义

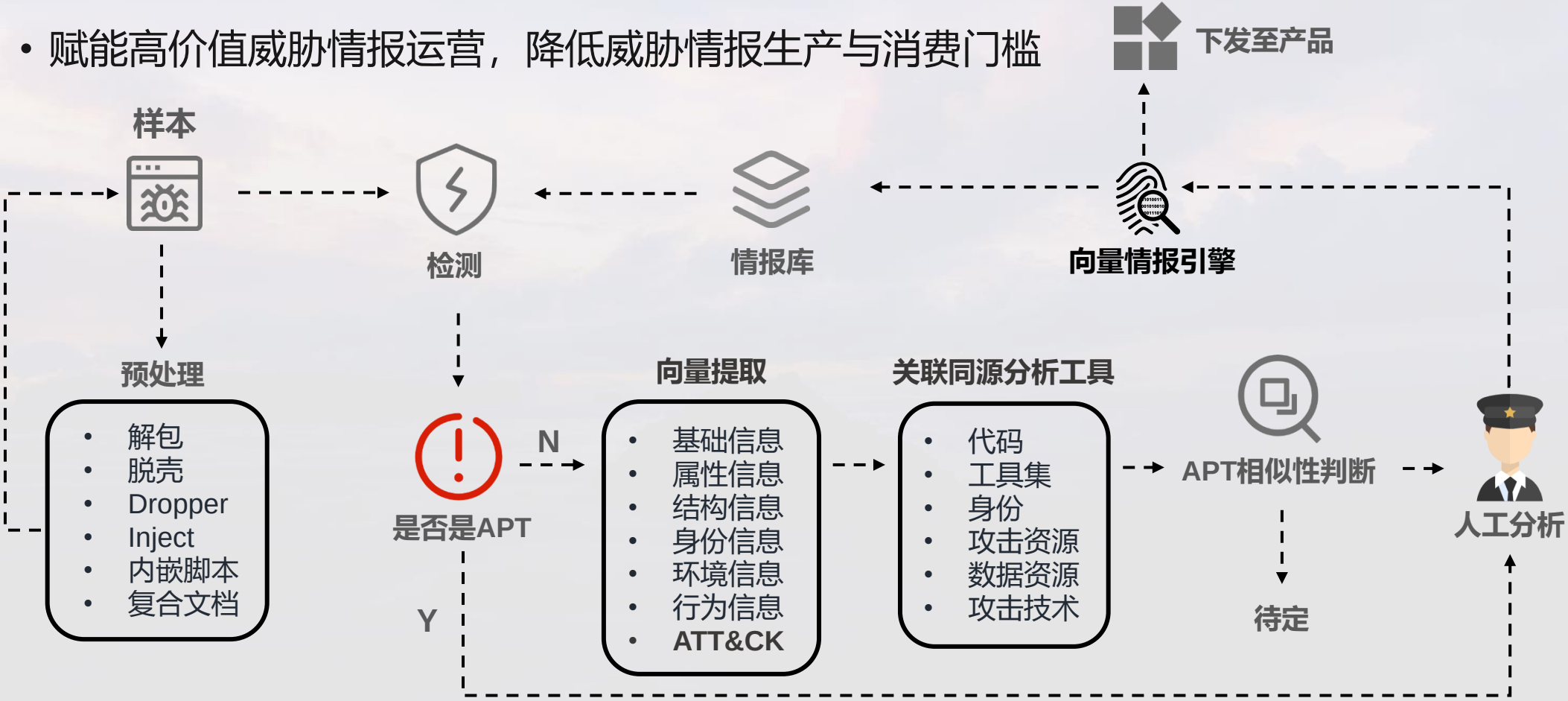
- 下一代威胁检测引擎

下一代威胁检测引擎是一组可通过接口调用，实现对输入的威胁对象进行**识别、鉴定、拆解和分析、输出处理后的信息和向量、供使用者进行威胁情报生产与消费、关联同源等工作**的程序模块的统称。

下一代威胁检测引擎是立足于在攻方可以获得防御方引擎并验证的假想下，立足于引擎本身是可以被绕过的环节来设计的。从输入对象上看，下一代引擎可以把载荷、信标、场景、人机行为作为检测向量，不再是单纯的检测，而是复合关联。把传统的结果鉴定器改为了识别器、鉴定器、拆解器、分析器的复合体，进而支撑用户场景下的**威胁追踪溯源**、技战术揭示、情报信息提取等工作。

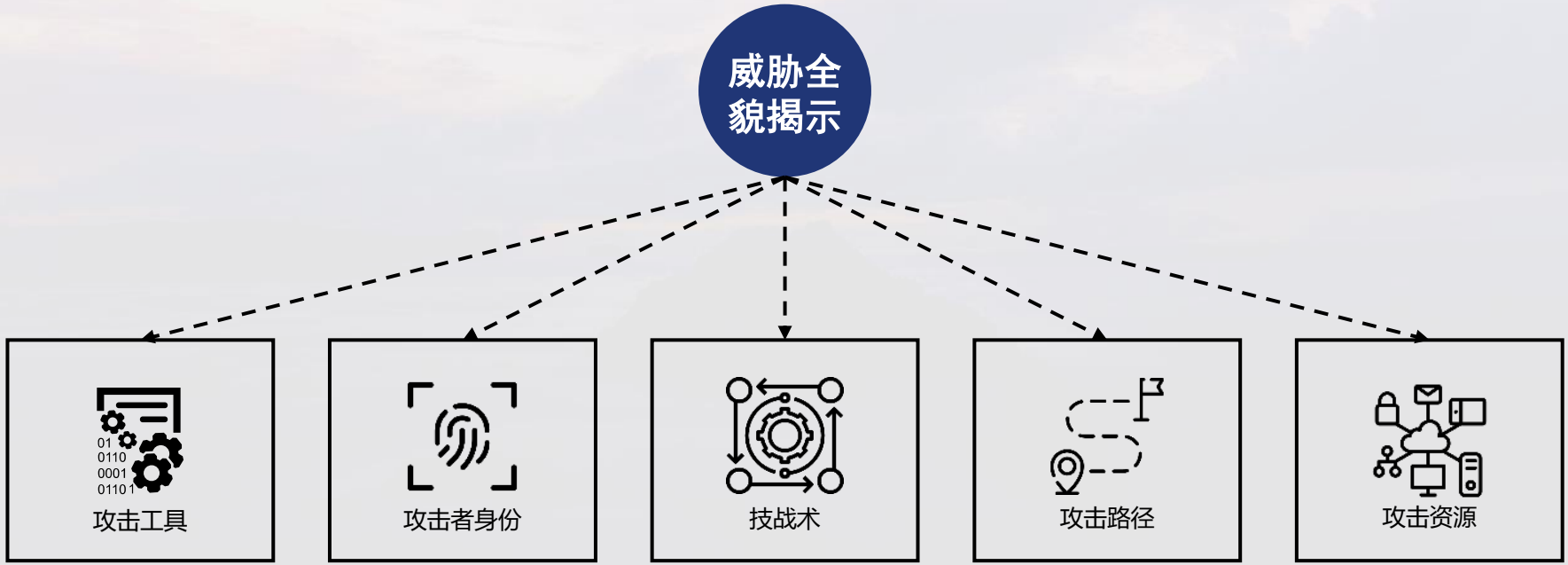
下一代威胁检测引擎的应用—情报的生产与消费

- 赋能高价值威胁情报运营，降低威胁情报生产与消费门槛

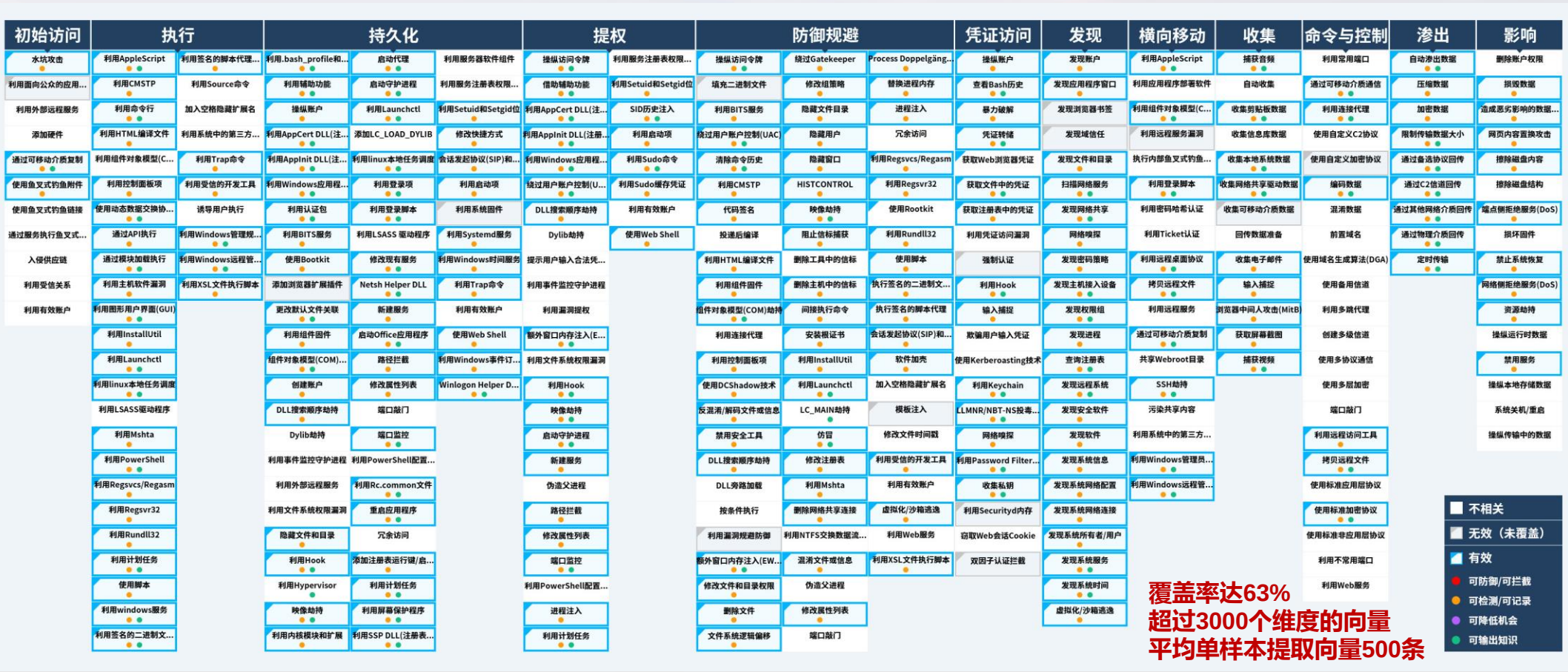


下一代威胁检测引擎的应用—赋能揭示威胁全貌

- 赋能揭示威胁全貌。基于全格式识别、深度预处理和多维向量提取等技术从样本侧揭示攻击身份、技战术、攻击资源和攻击路径等，支撑高级威胁对抗

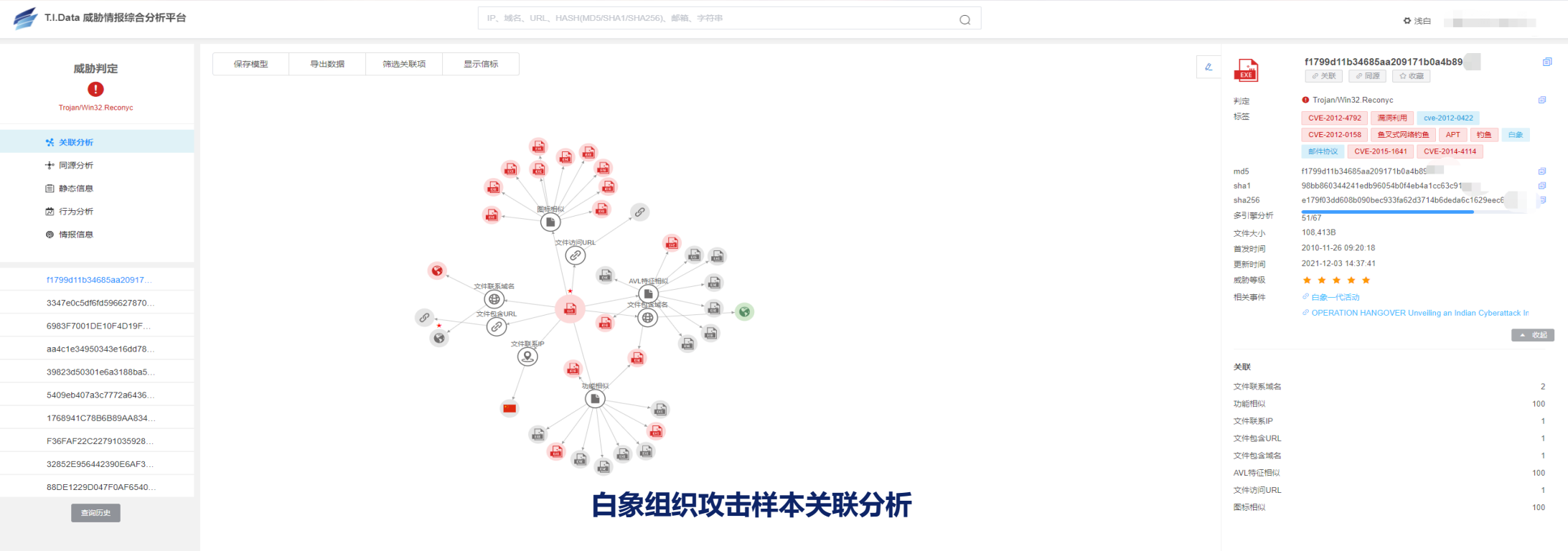


下一代威胁检测引擎的应用—对ATT&CK框架的覆盖



下一代威胁检测引擎的应用—追踪溯源

- 应用于APT关联分析和同源分析，基于提取的多维向量为关联分析与同源分析提供更多维度更细粒度的数据支撑

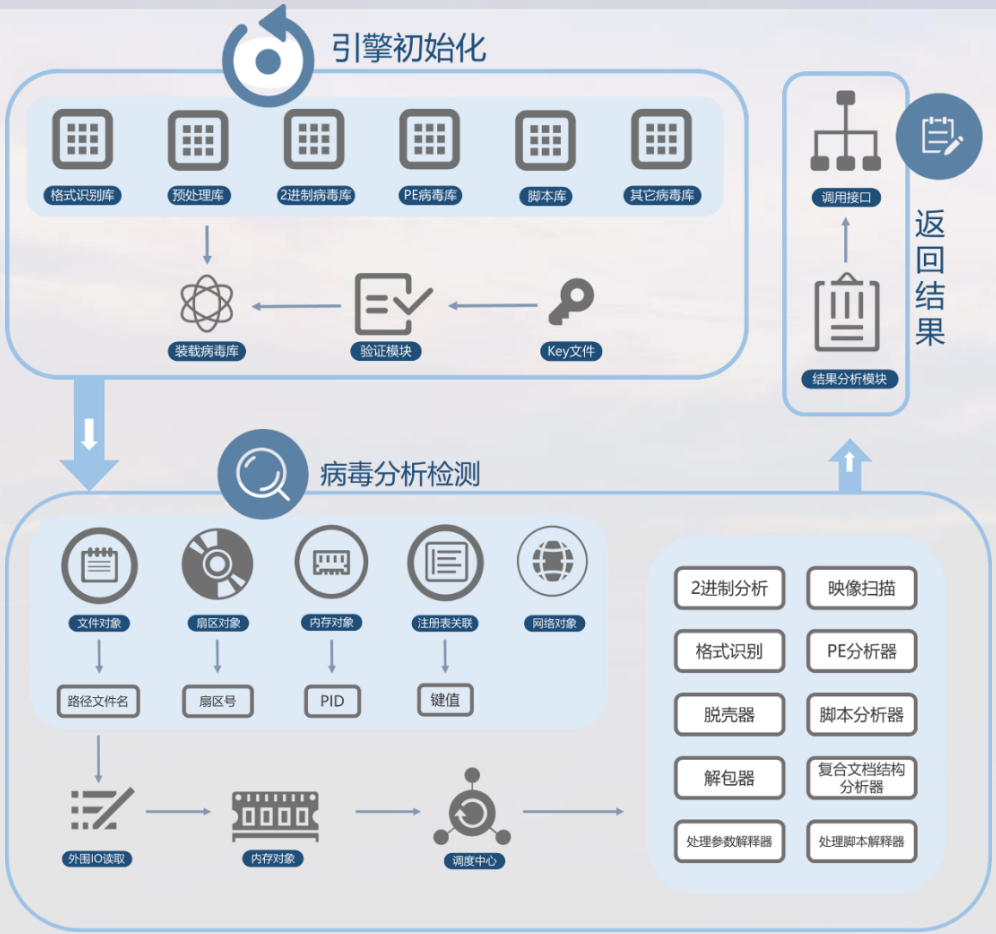


白象组织攻击样本关联分析

02 引擎的算力模型

计算场景对引擎的约束

引擎的工作原理



- 引擎的工作主要分为三个阶段：
  - 一是初始化阶段，引擎会加载格式识别、预处理、二进制病毒、PE病毒、脚本病毒……等特征库到**内存**中
  - 二是分析阶段，引擎会对内存中的待检测对象进行检测分析**计算**，包括格式识别、二进制分析、解包、脱壳、复合文档分析等等动作
  - 三是通过结果分析模块将检测结果输出给用户

引擎的算力模型

- 引擎检测的本质是依托模式识别和特征工程来实现的，这是在恶意代码时代所形成的经验。近些年来，引擎检测的对象从原来的只对部分对象进行识别检测，逐渐扩展到全量的识别与检测，这个过程是需要更大的算力基础设施去支撑的。

IO	CPU	内存
• 库加载 • 临时数据存储 • 日志	• 特征运算 HASH • 特征匹配 多模匹配 决策树 • 数据衍生 解包、元数据拆解 • 数据解密 脱壳 • 数据运算 虚拟执行	• 引擎特征 • 衍生数据 • 待处理对象

预处理对算力的消耗

安天AVLSDK威胁检测引擎 预处理能力参数 （统计时间：2021年12月）	格式识别（300种）	支持识别：可执行文件、包裹、文档、媒体文件、图片文件、软件关联格式、脚本、文本格式、其它格式等九大类格式
	解包（58种）	压缩包（28种）：rar, rar5, zip, tar, 7z, gzip, bzip, arj, cab, chm, iso, udf, lzh, z, zlib, xz, wim, cpio, dmg, hfs, gpt, xar, ace, alz, crx, egg, pyz, rdb
		自解压包（6种）：zlib_sfx, rar_sfx, rar5_sfx, zip_sfx, cab_sfx, 7z_sfx
		安装包（18种）：rpm, deb, nsis, wise, inno, installshield, setupfactory, instyler, ghostinstall, setup2go, smartinstall, vise, installcreator, tarma, autoit, quickbatch, douyou, yinginstall
		其它包裹（6种）：Microsoft Script Encoder, binhex, mso, uue, xxe, yEnc
	脱壳（31种）	upx、pecompact、aspack、upack、mpress、pepatch、nspack、fsg、neolite、petite、mew、packman、kbys、pepack、rlpack、simplepack、npack、expressor、pex、beroexepack、hmimys、exe32pack、xpack、asprotect、asdpack、bambam、 wwpack、depack、mkfpack、dxpack、ahpack
	复合文档拆分能力（12种）	office（7种）：rtf, doc, ppt, xls, docx, pptx, xlsx
		adobe（2种）：pdf, swf
		邮件（3种）：eml, msg, tnef

预处理对算力的消耗

- CPU 单核性能测试 Gzip

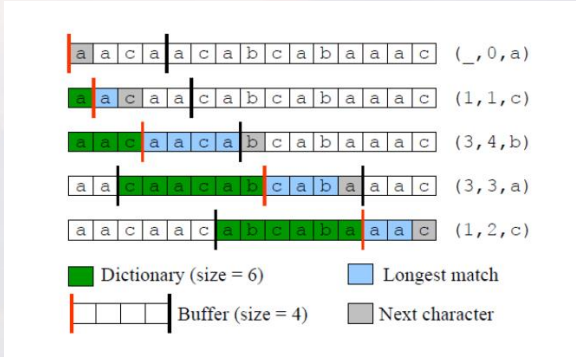
源文件：16Mb

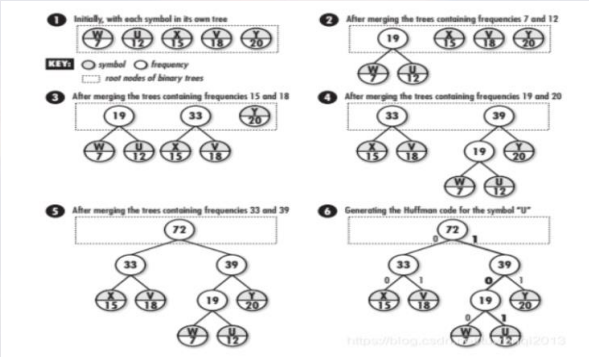
压缩后：1600Kb

每秒解压108次

每秒可解压172.8Mb

Intel(R)Xeon(R)
E5-2620 v4
@ 2.10GHz



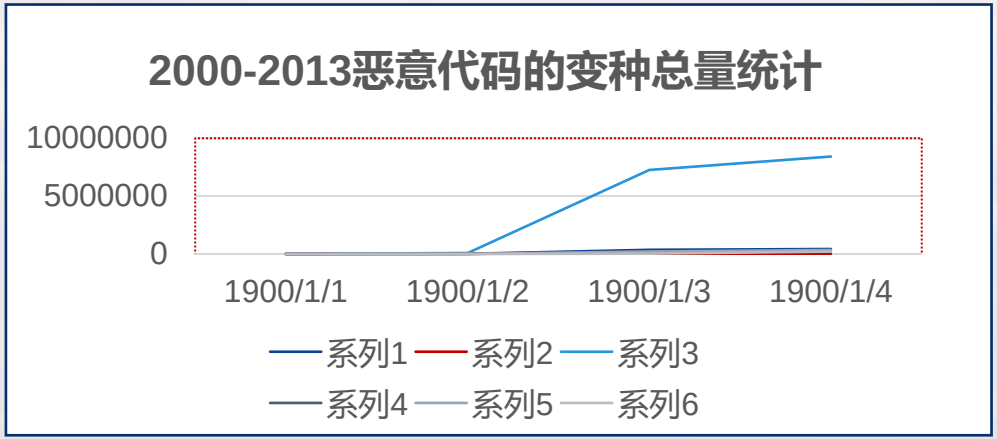


想要达到**千兆级**的性能，大约需要**6**个核

CPU

算力问题的原因—数量更庞大的威胁

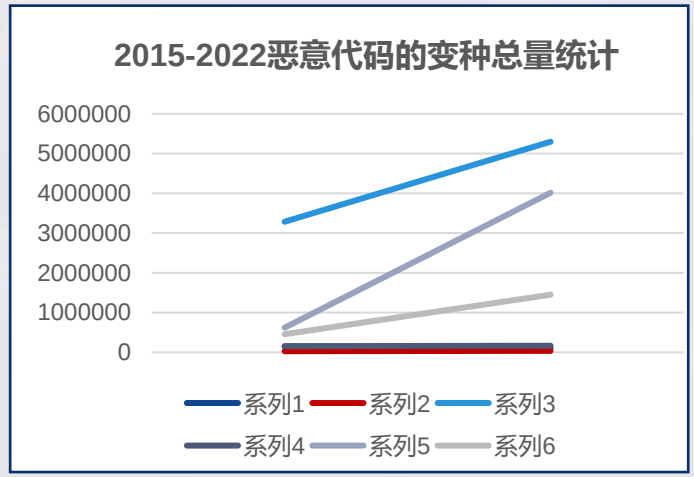
- 数量更加庞大的恶意代码需要更多的检测规则



分类/日期	2000/10/24	2006/11/10	2012/11/27	2013/11/04
Worm	512	8109	354049	435247
Virus	21006	27760	29940	30060
Trojan	3066	84811	7262094	8423751
HackTool	260	4968	217502	301076
Spyware	37	4899	214570	340751
RiskWare	0	88	25800	201401

来源：Kaspersky（卡巴斯基）对应日期病毒名列表

从观测来看，2014年开始，由于卡巴斯基后台分析与同源合并能力的增强，一些病毒家族和变种发生变化，因此无法连续统计，但总体种类膨胀超过**400倍**



2015/06/25	2022/01/11
149137	109359
29397	36821
3283882	5295977
153493	167666
622344	4013398
458035	1451458

算力问题的原因—数量更庞大的威胁

2000年安天病样本恶意代码变种数	
Trojan	3,066
Virus	21,006
Worm	512
HackTool	260
RiskWare	0
GrawWare	37
总计	24,881

2020年安天样本库恶意代码变种数	
Trojan	10,338,715
Virus	49,430
Worm	276,946
HackTool	385,361
RiskWare	1,055,493
GrawWare	2,640,863
总计	14,746,808

2000年安天引擎检测规则数	
总计	约6000条

2020年恶意代码检测规则数	
总计	52,684,446

2020年云检测规则数	
总计	664,870,674

安天引擎的本地检测规则数量，由2000年的不足**1万条**，增加到 2020年超过**5000万条**，20 年间增加了约 **500 倍**

2021安天引擎可识别文件格式规则

文件格式	规则数
PE	41,292,256
安卓	7,918,221
脚本	1,219,258
ELF	189,332
宏病毒	141,812
溢出	6,430
其他文件	191,713

2021安天引擎可解析格式

文件格式	种类数
软件数据	127
包裹	41
可执行格式	41
媒体	35
文档	31
图片	22
文本	18
脚本	9
其他文件	7

核心行为标签			
AdTool	Email	P2P	Spoof
AdWare	Exploit	Packed	Spy
ArcBomb	FakeAV	Porn-Dialer	SuspiciousPacker
AVTool	Flooder	Porn-Downloader	Tool
Backdoor	FraudTool	Porn-Tool	VirTool
BadJoke	GameThief	Proxy	WebToolbar
Banker	Garbage	PSW	Modifier
Clicker	HackTool	PSWTool	AutoRun
Client-IRC	Hoax	Ransom	Injector
Client-P2P	IM	RemoteAdmin	Phishing
Client-SMTP	IRC	RiskTool	Toolbar
Constructor	Joke	Rootkit	KeyLogger
CrackTool	Mailfinder	Server-FTP	Filecoder
DDoS	Monitor	Server-Proxy	AdDisplay
Dialer	MultiPacked	Server-Telnet	LockScreen
DoS	Net	Server-Web	Spammer
Downloader	NetTool	SMS	HLLW
Dropper	Notifier	SpamTool	Sniffer
Bundler	Rogue		

算力问题的原因—手段更复杂的威胁

- 更加复杂的攻击方式使得检测机制也变得更加复杂

1. 原文件—>解包


```
    "description": "fakedir/ba1dfeecb049e654d5888323f0af853b=>quadacrt.dll"
  },
  {
    "md5": "ba1dfeecb049e654d5888323f0af853b",
    "vec": {
      "MB006771.R01": [{ "ArchiveInfo": "zip" }],
      "MB006771.R02": [{ "VecContent": "found_archive_head" }],
      "MB006772.R01": [{ "VecContent": "data_may_be_obfuscated" }],
    },
    "description": "fakedir/aspack"
  }
],
"name": "vector_analyse",
"output_timestamp": "Thu Jan 13 14:47:31 2022"
```


2. 解包后文件—>脱壳


```
  "description": "fakedir/ba1dfeecb049e654d5888323f0af853b=>quadacrt.dll=>aspack"
},
{
  "size": 88612,
  "md5": "e288295f362fe23097d7788e601e8dba",
  "vec": {
    "MB002254.R01": [{ "VecContent": "Encryption_algorithm" }],
    "MB002405.R01": [{ "VecContent": "load_api_by_self" }],
    "MB006759.R12": [{ "certificate": "No digital signature" }],
    "MB002111.R01": [{ "VecContent": "Anti_Kill_process_C" }],
    "MB006770.R01": [{ "PackInfo": "Packer_Compression/StarForce.ASPack" }]
```


3. 脱壳后获取向量


```
{
  "input_timestamp": "Thu Jan 13 14:47:31 2022",
  "version": "1.1.2.1",
  "data": [
    {
      "size": 86016,
      "md5": "e288295f362fe23097d7788e601e8dab",
      "vec": {
        "MB002324.R01": [{ "VecContent": "file_attribute_des" }],
        "MB002410.R01": [{ "VecContent": "found_schtasks" }],
        "MB002259.R01": [{ "VecContent": "ransom_special_file_ops" }],
        "MB001727.R01": [{ "VecContent": "string_cmp_and_copy" }],
        "MB001727.R03": [{ "VecContent": "memory_cmp_and_copy" }],
        "MV000012.R05": [{ "VecContent": "get_clipboard_data" }],
        "MB002392.R01": [{ "VecContent": "cmd_found_in_reg_parameter" }],
        "MV001872.R01": [{ "VecContent": "resume_thread" }],
        "MV001872.R02": [{ "VecContent": "create_thread" }],
        "MB006759.R12": [{ "certificate": "no digital signature" }],
        "MV000029.R01": [{ "VecContent": "enumerate_files_via_apis" }],
        "MB002203.R01": [{ "VecContent": "create_file_to_system_directory" }],
        "MB000916.R01": [{ "VecContent": "delete_shadow_copies" }],
        "MB002429.R01": [{ "VecContent": "Bin_Crypt_RIPEMD160_Constants" }],
```


发现用于设置计划任务的工具schtasks

Scheduled Task/Job

算力问题的原因—手段更复杂的威胁

技术

- 获取访问凭据
- 持久化
- 横向扩散
- 执行
- 环境检测
- 数据回传
- 命令与控制
- 收集信息
- 逃避检测
- 提升权限

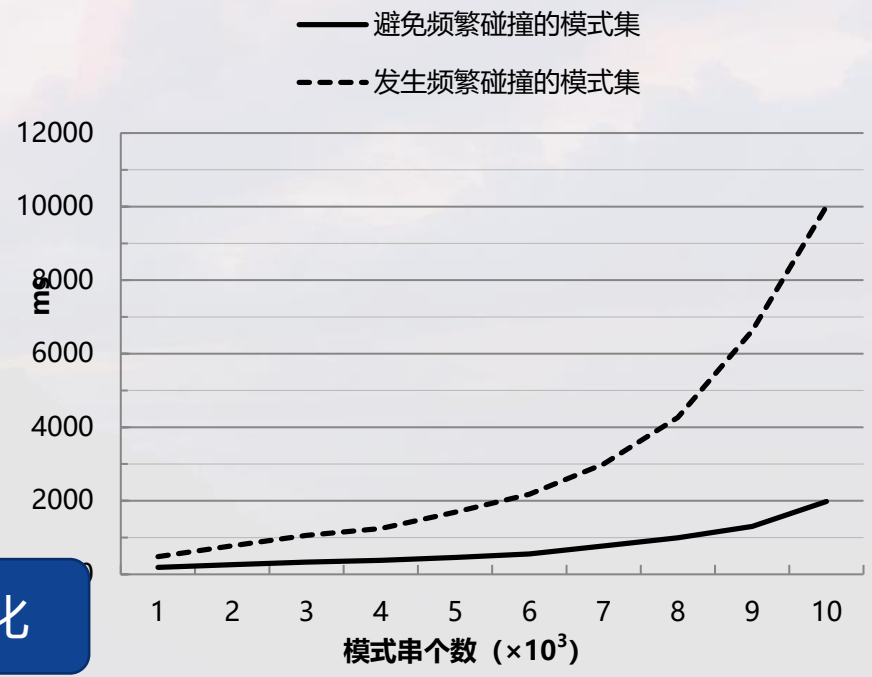
帐户发现	Account Discovery
应用程序窗口发现	Application Window Discovery
文件和目录发现	File and Directory Discovery
网络服务扫描	Network Service Scanning
网络共享发现	Network Share Discovery
外围设备发现	Peripheral Device Discovery
权限组发现	Permission Groups Discovery
过程发现	Process Discovery
查询注册表	Query Registry
远程系统发现	Remote System Discovery
安全软件发现	Security Software Discovery
系统信息发现	System Information Discovery
系统网络配置发现	System Network Configuration Discovery
系统网络连接发现	System Network Connections Discovery
系统所有者/用户发现	System Owner/User Discovery
系统服务发现	System Service Discovery
系统时间发现	System Time Discovery

03 引擎在算力上的优化

引擎的优化；与硬件的结合

引擎的优化—特征维度

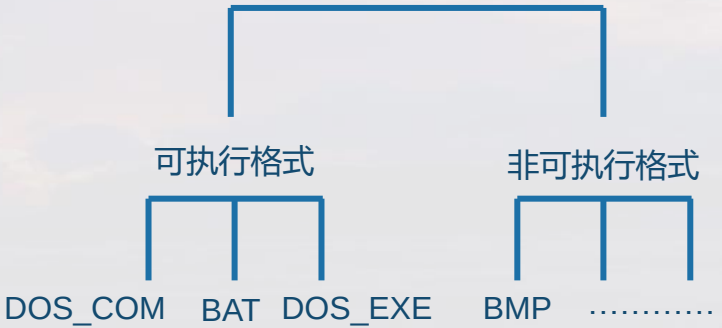
- 在**特征工程**方面，安天提出了网络场景下的特征优化方法，基于特征筛选技术，加速了网络场景下的检测效率，并推出了当时千兆限速检测能力的VDS，这是全球第一个可以形成千兆检测能力的网络安全设备



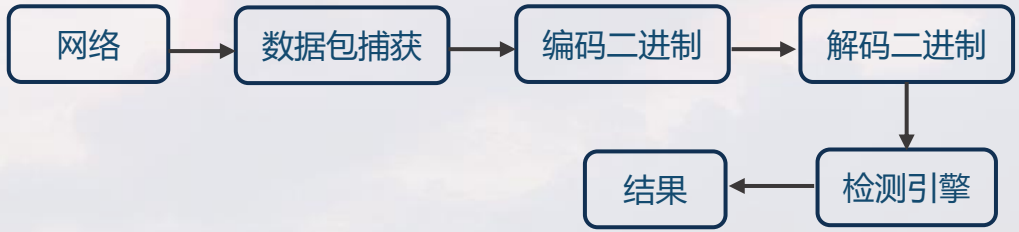
检测范围优化　　算法优化　　特征优化

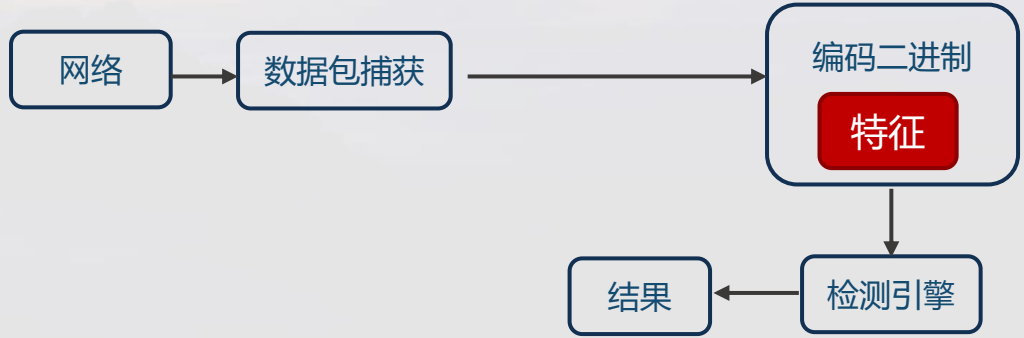
引擎的优化—算法层面

引擎检测的格式分流



- 在**数据包级别**进行动态检测，减轻网络设备还原处理的压力





引擎的优化—指令层面

数据处理优化

包处理基于指令逻辑实现，耗时较高

① 用**跳转表代替指令逻辑实现**，使用缓存机制减少解包时间

时空优化

在数据运算、数据匹配方面，传统的方法性能受限

① 在虚拟运行方面，用**跳转表代替opcode指令解析**
② 用**状态机代替正则匹配**，提升规则匹配速度
③ 用**多模匹配代替单模**匹配，只需一次性匹配，无需一特征一匹配

1. CRC32表

```
local const z_crc_t FAR crc_table[TBLS][256] =
{
  {
    0x00000000UL, 0x77073096UL, 0xee0e612cUL, 0x990951baUL, 0x076dc419UL,
    0x706af48fUL, 0xe963a535UL, 0x9e6495a3UL, 0x0edb8832UL, 0x79dcb8a4UL,
    0xe0d5e91eUL, 0x97d2d988UL, 0x09b64c2bUL, 0x7eb17cbdUL, 0xe7b82d07UL,
    0x90bf1d91UL, 0x1db71064UL, 0x6ab020f2UL, 0xf3b97148UL, 0x84be41deUL,
    0x1adad47dUL, 0x6ddde4ebUL, 0xf4d4b551UL, 0x83d385c7UL, 0x136c9856UL,
    0x646ba8c0UL, 0xfd62f97aUL, 0x8a65c9ecUL, 0x14015c4fUL, 0x63066cd9UL,
    0xfa0f3d63UL, 0x8d080df5UL, 0x3b6e20c8UL, 0x4c69105eUL, 0xd56041e4UL,
    0xa2677172UL, 0x3c03e4d1UL, 0x4b04d447UL, 0xd20d85fdUL, 0xa50ab56bUL,
    0x35b5a8faUL, 0x42b2986cUL, 0xdbbbc9d6UL, 0xacbcf940UL, 0x32d86ce3UL,
    0x45df5c75UL, 0xdcd60dcfUL, 0xabd13d59UL, 0x26d930acUL, 0x51de003aUL,
    0xc8d75180UL, 0xbfd06116UL, 0x21b4f4b5UL, 0x56b3c423UL, 0xcfba9599UL,
    0xb8bda50fUL, 0x2802b89eUL, 0x5f058808UL, 0xc60cd9b2UL, 0xb10be924UL,
    0x2f6f7c87UL, 0x58684c11UL, 0xc1611dabUL, 0xb6662d3dUL, 0x76dc4190UL,
    0x01db7106UL, 0x98d220bcUL, 0xefd5102aUL, 0x71b18589UL, 0x06b6b51fUL,
    0x9fbfe4a5UL, 0xe8b8d433UL, 0x7807c9a2UL, 0x0f00f934UL, 0x9609a88eUL,
    0xe10e9818UL, 0x7f6a0dbbUL, 0x086d3d2dUL, 0x91646c97UL, 0xe6635c01UL,
    0x6b6b51f4UL, 0x1c6c6162UL, 0x856530d8UL, 0xf262004eUL, 0x6c0695edUL,
    0x1b01a57bUL, 0x8208f4c1UL, 0xf50fc457UL, 0x65b0d9c6UL, 0x12b7e950UL,
    0x8bbeb8eaUL, 0xfcb9887cUL, 0x62dd1ddfUL, 0x15da2d49UL, 0x8cd37cf3UL,
    0xfbd44c65UL, 0x4db26158UL, 0x3ab551ceUL, 0xa3bc0074UL, 0xd4bb30e2UL,
    0x4adfa541UL, 0x3dd895d7UL, 0xa4d1c46dUL, 0xd3d6f4fbUL, 0x4369e96aUL,
    0x346ed9fcUL, 0xad678846UL, 0xda60b8d0UL, 0x44042d73UL, 0x33031de5UL,
    0xaa0a4c5fUL, 0xdd0d7cc9UL, 0x5005713cUL, 0x270241aaUL, 0xbe0b1010UL,
    0xc90c2086UL, 0x5768b525UL, 0x206f85b3UL, 0xb966d409UL, 0xce61e49fUL,
```

2. 生成表

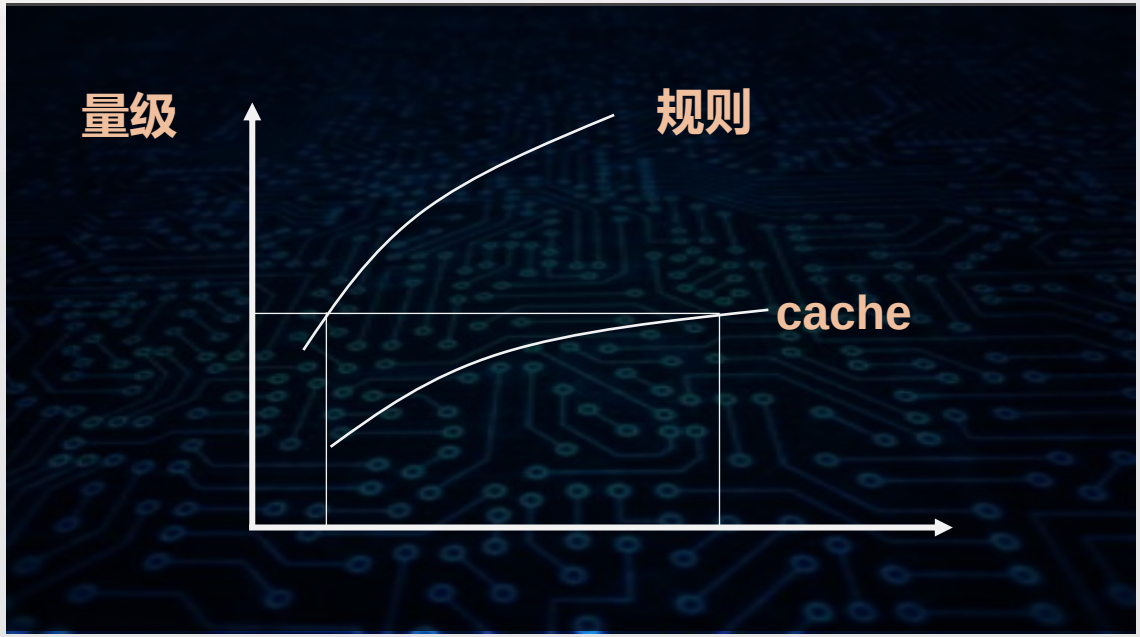
```
    /* Case the advice about DYNAMIC_CRC_TABLE is ignored) */
    if (first) {
        first = 0;

        /* make exclusive-or pattern from polynomial (0xedb88320UL) */
        poly = 0;
        for (n = 0; n < (int)(sizeof(p)/sizeof(unsigned char)); n++)
            poly |= (z_crc_t)1 << (31 - p[n]);

        /* generate a crc for every 8-bit value */
        for (n = 0; n < 256; n++) {
            c = (z_crc_t)n;
            for (k = 0; k < 8; k++)
                c = c & 1 ? poly ^ (c >> 1) : c >> 1;
            crc_table[0][n] = c;
        }
```

探寻算力支撑—通用基础算力的限制 CPU

- 规则数量的膨胀带来存储资源的压力
- 分支众多，预处理深度加深带来计算处理资源的压力
- 热数据集庞大，局部性差异导致cache失效的性能恶化



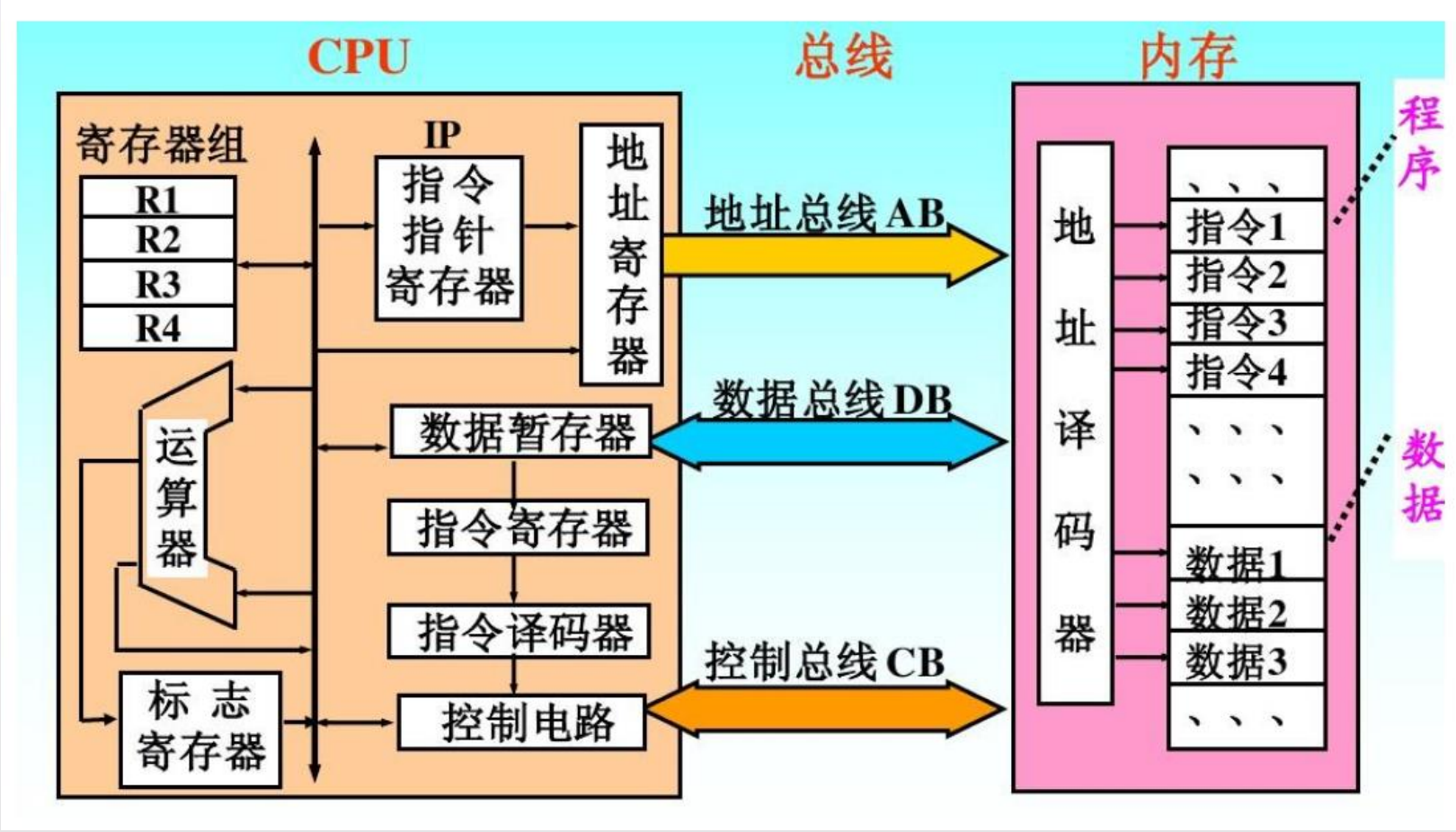
2022年CPU缓存

L1 4MB

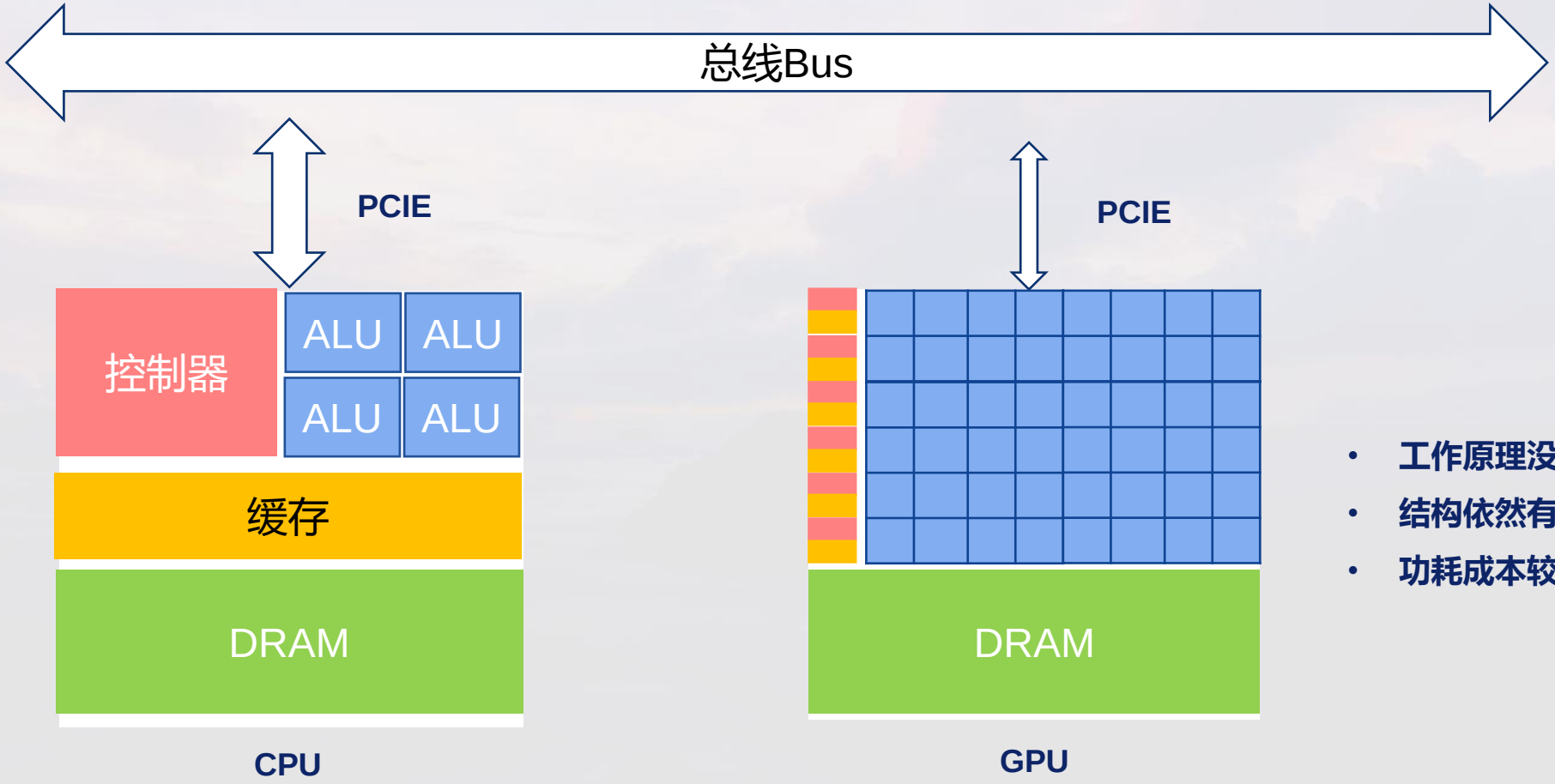
L2 32MB

L3256MB

探寻算力支撑—通用基础算力的限制 CPU



探寻算力支撑—通用基础算力的限制 GPU



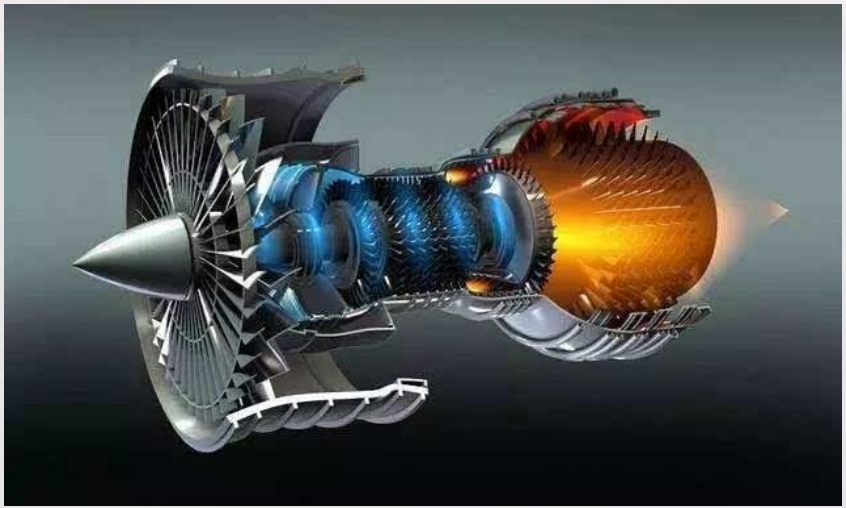
- 工作原理没有改变
- 结构依然有瓶颈
- 功耗成本较高

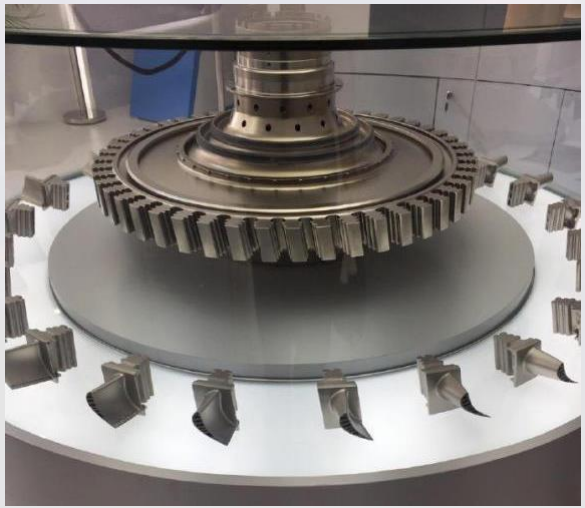
04 引擎结合专用芯片展望

专用芯片应对算力消耗问题

专用芯片上的展望

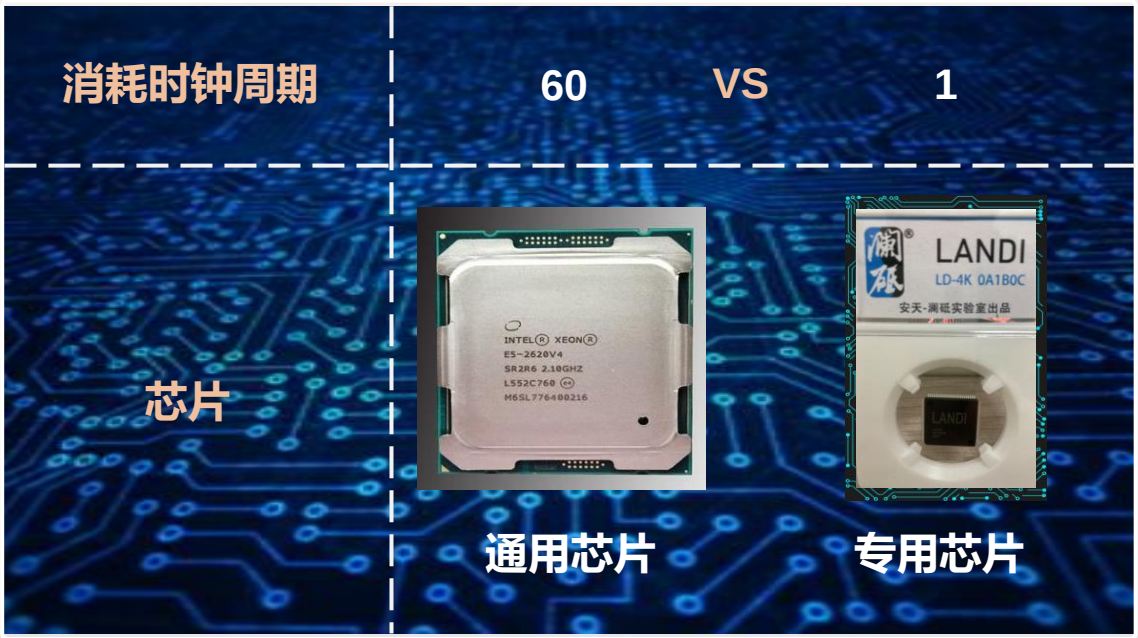
- 专用芯片支撑安天引擎，为检测提供更多安全算力





通用 vs. 专用——性能

- 在memcmp主循环中cmp、sub指令执行时比较消耗时钟周期
- 一次循环消耗60-80个时钟周期



一次循环中的消耗时钟周期对比

```
.text:10189146
.text:10189146 loc_10189146:                           ; CODE XREF: memcmp+24↓j
.text:10189146                 mov     eax, [edx]
.text:10189148                 cmp     eax, [esi]
.text:1018914A                 jnz     short loc_10189158
.text:1018914C                 add     edx, edi
.text:1018914E                 add     esi, edi
.text:10189150                 sub     ecx, edi
.text:10189152
.text:10189152 loc_10189152:                           ; CODE XREF: memcmp+14↑j
.text:10189152                 cmp     ecx, edi
.text:10189154                 jnb     short loc_10189146
.text:10189156                 jmp     short loc_1018915A
.text:10189158 ; ---------------------------------------------------------------------------
.text:10189158
.text:10189158 loc_10189158:                           ; CODE XREF: memcmp+1A↑j
.text:10189158                 mov     ecx, edi
.text:1018915A
.text:1018915A loc_1018915A:                           ; CODE XREF: memcmp+26↑j
.text:1018915A                 test    ecx, ecx
.text:1018915C                 jz      short loc_10189171
.text:1018915E                 sub     esi, edx
.text:10189160
.text:10189160 loc_10189160:                           ; CODE XREF: memcmp+3F↓j
.text:10189160                 mov     al, [edx]
.text:10189162                 mov     bl, [esi+edx]
.text:10189165                 cmp     al, bl
.text:10189167                 jb      short loc_1018917D
.text:10189169                 ja      short loc_10189178
.text:1018916B                 inc     edx
.text:1018916C                 sub     ecx, 1
.text:1018916F                 jnz     short loc_10189160
```

大约20条指令的一个循环

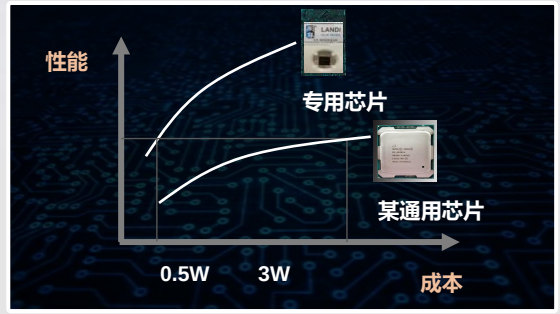
通用 vs. 专用——性能

- 引擎工作时在数据衍生、数据运算、特征运算和特征匹配等环节需要消耗算力
- 以特征运算环节为例，对通用和专用芯片的性能进行对比，专用约为通用的20倍

对平均大小为100k的文件进行特征运算

总结

- 检测能力、效率以及算力成本的完美平衡已经成为一种现实需求，具有广阔的应用前景与市场
- 软件优化虽然在一定程度上能提升检测性能，但还是无法很好应对威胁与用户算力资源的“双重考验”
- 安全算力将会是威胁检测引擎的性能的发力点和主要发展方向之一，专用芯片具备性能和成本双重优势



谢谢大家

安天冬训营 wtc.antiy.cn