International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

Enhancing Threat Detection And Introducing New Vulnerabilities

Rahul Cherekar[1]
[1]Independent Researcher, Boston, USA rahul.cherekar@gmail.com

Abstract
Artificial Intelligence (AI) is increasingly transforming cybersecurity by enhancing threat detection, predictive analytics, and incident response automation. AI-powered systems process vast amounts of data, detect anomalies, and mitigate security risks in real-time, significantly improving defense mechanisms. However, the integration of AI also introduces new vulnerabilities, including adversarial attacks and potential misuse by cybercriminals. This paper examines the dual nature of AI in cybersecurity, highlighting both its advantages and inherent risks. We propose a structured approach to implementing AI-driven security measures while addressing associated vulnerabilities through robust mitigation strategies and ethical considerations.
Keywords: *Artificial Intelligence, cyber security, threat detection, predictive analytics, vulnerabilities, cyber criminals.*

1. INTRODUCTION

Cybersecurity systems heavily rely on advanced methods such as reverse engineering to analyze and identify undocumented software behavior. However, the process—known as the introspection challenge—remains complex due to its resource-intensive nature and the restricted visibility of isolated environments, such as virtual machines. Existing solutions like Virtuoso [1] and VMST [2] primarily focus on kernel-level data but fail to provide comprehensive insights into user-level interactions, such as application logs and program execution traces.

To address these limitations, our research introduces an enhanced monitoring mechanism that utilizes runtime memory access tracking to identify tap points—critical locations within a system that expose security-relevant information, such as accessed URLs or file operations. Unlike prior methodologies that rely on static monitoring, our approach integrates dynamic observation techniques, ensuring adaptability across diverse architectures with minimal overhead [3].

The key challenges associated with this approach include handling vast volumes of real-time data and mitigating the computational overhead associated with continuous security monitoring. To overcome these challenges, we leverage AI based analytics (discussed in Section IV) to efficiently identify and prioritize security-sensitive tap points. Additionally, we implement a record-and-replay framework using QEMU [4] to facilitate low-overhead data collection and offline analysis.

2. Defining tap points

A tap point abstracts memory gets to for investigation. Characterizing it as a triple: (guest, program counter, address space), guarantees intelligent information streams by isolating memory gets to by setting.

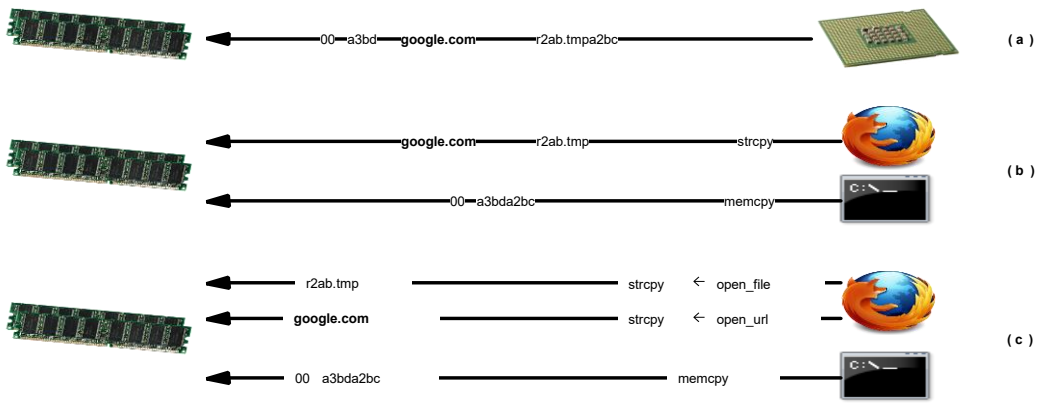

Figure. 1. Tap point definitions: (a) solitary stream of memory gets to; (b) gathered by program and area; (c) refined by calling context.

Figure. 1 shows three degrees of granularity: (a) all central processor to-Smash composes, which incorporate immaterial information; (b) gathered by endlessly program counter, which blends inconsequential information; (c) adding the calling setting, separating the ideal data.

International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

Most cases require one degree of setting, however more profound stacks are upheld depending on the situation. We likewise recognize correlated tap points by identifying nearby memory composes inside a short window, empowering examination of related information structures.
This approach adjusts granularity and soundness, upgrading information thoughtfulness.

3. Scope and assumptions

This study focuses on a runtime-based approach for identifying security-relevant memory accesses within operating system environments. However, certain assumptions and limitations apply:
Tappan Zee Scaffold (TZB) intends to improve on the manual course of distinguishing tap focuses for dynamic observing however doesn't ensure full unwavering quality under all circumstances. Key presumptions:

1. **Streaming data constraints**

Real-time monitoring is optimized for sequential memory access patterns. Spatially unordered or fragmented memory operations are beyond the scope of this framework.

2. **Encoding dependencies**

Identifying security-relevant data assumes a predefined knowledge of encoding schemes (e.g., ASCII, UTF-8). Future extensions could incorporate adaptive encoding detection.

3. **Call stack context**

In most cases, a single-level call stack trace suffices for monitoring. However, edge cases may require deeper context resolution, particularly for applications employing nonstandard memory management techniques. Figure. 2 features TZB's extension. It upholds related memory composes (e.g., memcpy) and transiently requested information (e.g., sequential composes). Limits incorporate opposite arranged duplicates (memmove) and broadly dispersed composes (dmesg). Upgrades like superior setting following could resolve these issues in future work.

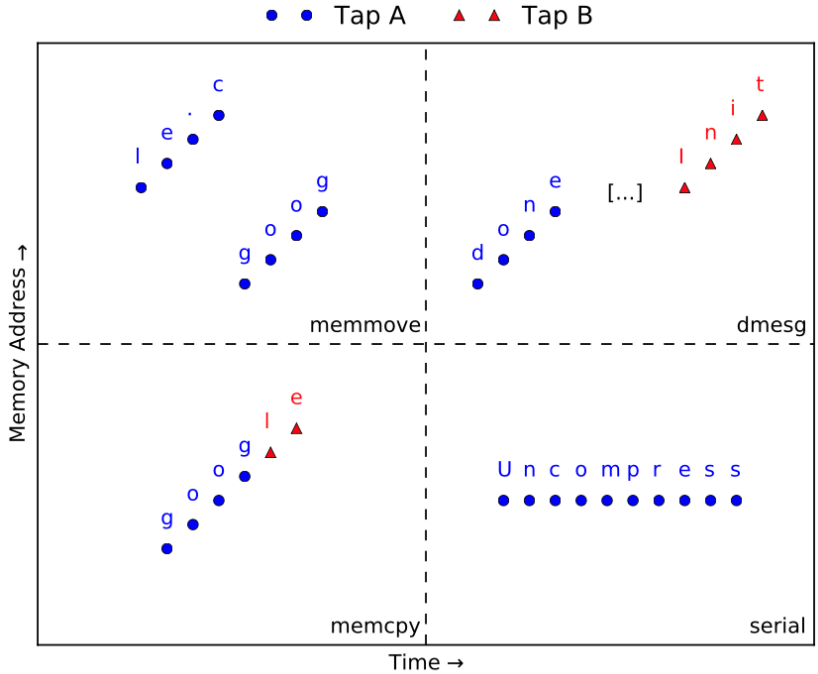

Figure. 2. Patterns of memory access that we might wish to monitor using TZB.

4. Search strategies

Tappan Zee Extension (TZB) distinguishes tap points for information extraction through three systems:
1. **Recording and replay**: Wanted operating system orapplication conduct is kept in PANDA, empowering deterministic replay with low above. Examinations, executed as PANDA modules, process memory gets to during replay to distinguish tap focuses, which are then approved by investigating information or code around the tap point (Figure 3).

2. **Search strategies**:

i **Known knowns**: For unsurprising information (e.g., URLs, filenames), string look recognize tap focuses proficiently.

ii **Known unknowns**: For information of unsure configuration (e.g., bit messages, encryption keys), measurable models or prophets rank tap focuses by importance.

iii **Obscure unknowns**: For exploratory inquiries, bunching bunches comparative tap focuses, empowering productive assessment through models.

These methodologies give a deliberate structure to find and approve helpful thoughtfulness focuses.

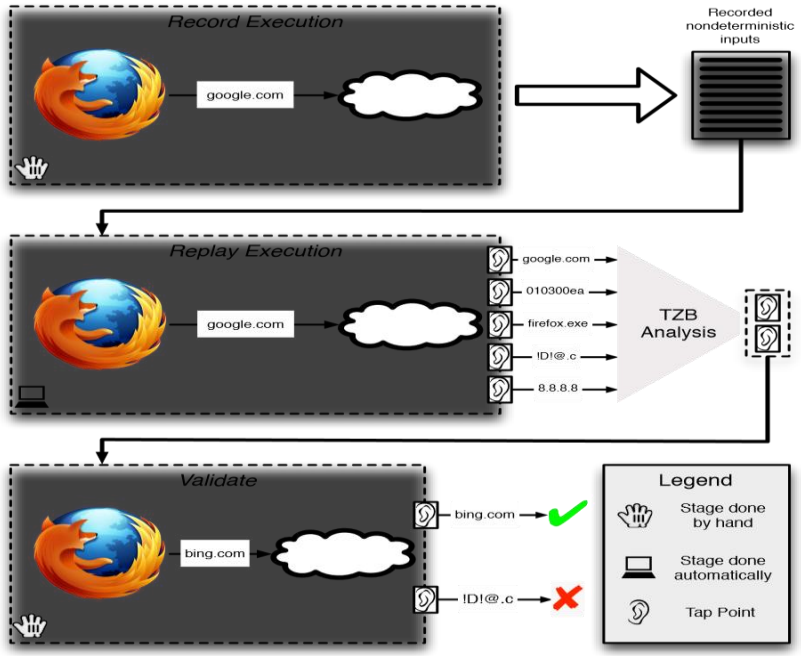

Figure. 3. The workflow for using TZB to locate points at which to interpose for active monitoring.

5. **Implementation**

This segment frames the unique investigation stage used to fabricate TZB and features TZB-explicit algorithmic and information structure arrangements.

1. **Panda**

TZB use the Stage for Design Nonpartisan Unique Investigation (PANDA), based on QEMU 1.0.1 [4]. QEMU's productive copying and Middle of the road Language (IL) empower PANDA to help different models with negligible above (5x log jam).

PANDA's key elements include:

i **Plugin architecture**: Permits dynamic examinations in C/C++ utilizing secluded modules with standard callbacks (e.g., pre/post-fundamental block, memory activities), working with clean detachment and reuse of examinations.

ii **Record and replay (RR)**: Catches computer processor inputs (e.g., ins, interferes) for deterministic replays under weighty instrumentation, basic for TZB's computational requests.

iii **LLVM integration**: Converts QEMU's IL to LLVM code [5], however this component is unused in TZB.

2. **Callstack monitoring**

To follow calling settings, TZB utilizes the callstack module. This module distinguishes engineering explicit call directions (e.g., call on x86, bl on ARM) and keeps a shadow stack for bring addresses back. While powerful for compiler-created code, it might fizzle with non-standard callbring semantics back.

3. **Fixed string searching**

The stringsearch module recognizes fixed strings going through tap focuses. Utilizing a one-byte counter per string/tap point, the module matches strings effectively however restricts lengths to 256 bytes (expandable). This lightweight deterministic limited robot guarantees adaptable execution for TZB's huge information volumes.

4. **Statistical search and clustering**

Bigram measurements empower fluffy hunt and bunching of information at tap focuses. Meager histograms productively store byte-pair frequencies to keep away from inordinate memory use.

For similitude measurements, Jensen-Shannon divergence [6] beats options like Euclidean and cosine distances. This is processed as:

$$JSD(P,Q) = H\left(\frac{P+Q}{2}\right) - \frac{H(P)+H(Q)}{2}$$

where H addresses Shannon entropy. Bunching utilizes kmeans [7] with KMeans++ initialization [8]. Multithreaded executions guarantee down to earth execution.

5. **Finding SSL/TLS keys**

The keyfind module recognizes tap focuses composing SSL/TLS ace mysteries by testing 48-byte successions as unscrambling keys for caught parcels. A checked Message Verification Code (Macintosh) affirms a match. This approach successfully unscrambles SSL/TLS associations for malware examination, as exhibited in Section VI-A4.

6. **Evaluation**

This segment assesses the adequacy of TZB's tap point search methodologies (Section IV) for reflection, zeroing in on genuine applications. Tests length assorted working frameworks, applications, and structures to exhibit TZB's flexibility. Where conceivable, emblematic names are utilized for meaningfulness.

1. **Known knowns**

i **URL access**: Checking URLs can help interruption recognition frameworks (IDS) by confirming human-started demands or boycotting vindictive locales. Since URL warnings are not uncovered through open APIs, we utilized the stringsearch module to recognize tap focuses for URL information in programs across various working frameworks. Table I sums up results, with exact tap focuses found for different setups, including Revelation, Firefox, and Web Voyager. These tap focuses permit exact observing without expecting earlier investigate data.

ii **TLS/SSL expert secrets**: Decoding SSL/TLS traffic for IDS applications generally requires a man-in-the-center assault, presenting dangers to testament confirmation. Utilizing TZB, we found tap focuses to get to dominate privileged insights for associations, empowering secure thoughtfulness. Explores different avenues regarding OpenSSL, Revelation, Firefox, and different clients showed TZB's capacity to recognize dependable tap focuses for ace insider facts (Table II). For SSLv3 executions like Revelation, extra call stack setting was important to detach applicable information because of varieties in pseudo-arbitrary capabilities. TZB works on this cycle contrasted with customary picking apart methods.

iii **File access**: Document access checking is basic for security applications like enemy of infection scanners. Utilizing stringsearch, we recognized tap focuses that caught filenames in ASCII and UTF-16 configurations across numerous working frameworks (Table III). On Windows 7, extra calling setting was expected to avoid irrelevant named objects. By and large, TZB really found tap focuses for exhaustive record access observing.

iv **SSL malware**: Unscrambling SSL-scrambled malware traffic presents remarkable difficulties. In contrast to harmless clients, malware frequently opposes testament based capture. Utilizing the keyfind module, we distinguished tap focuses for encryption keys by means of preliminary decoding. Testing with the Sykipot trojan [9] showed TZB's capacity to find SSL keys, utilizing divided framework components among malware and standard applications like Web Pioneer. The tap point empowered decoding of caught parcels, bypassing conventional difficulties.

v **Finding dmesg**: Framework logs are basic for security and framework organization. A reflection based security framework, for example, requires recognizing tap focuses composing log-like information in view of models. Utilizing the factual pursuit from Section, we prepared on dmesg yields from a Debian sid (amd64) have, figuring bigram probabilities, and applied it to five operating system boot accounts. JensenShannon distances positioned tap focuses, physically checked for accuracy (Table IV).

The right situation log tap focuses were in the best 10 across tried OSes. Outstandingly, Linux-based frameworks coming up short on extraordinary guest due to do_syslog being conjured by numerous capabilities. For Haiku, log fulfillment required distinguishing connected tap focuses shaping a memcpy. On the other hand, no closely resembling Windows 7 tap guide was viewed as due toward its numeric-coded logging framework. Existing apparatuses like Virtuoso [1] might suit such parallel arrangements.

Revelations during inconsequential undertakings, such as investigating QEMU boot issues for Raspberry Pi [10], show extra utility.

vi **Unknown questions**: Clustering: To assess bunching adequacy, we contrasted algorithmic grouping and manual marks on six accounts (FreeBSD 9.0: boot, closure, inactive, applications; Windows 7: inactive, applications). After kimplies bunching (k = 100), tests were chosen from shifting group separates, dazed, and marked physically.

A k-implies grouping (k = 10) accomplished a low Changed Rand Index [11], reflecting frail arrangement with manual marks. Regardless of this, bunching really featured text-like information in FreeBSD boot accounts, gathering dmesg and filename tap focuses in a solitary group. Investigation uncovered piece messages, filenames, shell scripts, process postings, and design information, highlighting grouping's capability to concentrate examination.

vii **Accuracy**: TZB investigations successfully distinguish intervention focuses. For stringsearch, recognizing an adequate tap point required manual assessment of up to 262 sections (e.g., URLs in IE8), frequently requiring under 60 minutes. dmesg recovery accuracy at 10 went from 20% (Minix) to 100% (Haiku), ensuring right outcomes inside the main 10.

In synopsis, factual techniques and bunching are important devices for finding applicable tap focuses, empowering proficient examination and framework bits of knowledge.

2. Comparing TZB with existing solutions

Table VII presents a comparative analysis between our proposed Tappan Zee Bridge (TZB) framework and existing methodologies VMSTin the context of memory introspection.

As observed, TZB introduces an active monitoring paradigm, capable of dynamically identifying securitysensitive memory access points. Unlike static introspection tools, TZB operates across both user and kernel levels, making it more adaptable for real-world cybersecurity applications.

Browser	Caller	PC
Deb Epiphany (arm)	`WebCore::KURL::KURL+0x30`	`WebCore::KURL::init+0x70`
Deb Epiphany (amd64)	`webkit_frame_load_uri+0xc3`	`WebCore::KURL::init+0x368`
Win7 IE8 (x86)	`ieframe!CAddressEditBox::_Execute+0xaa`	`ieframe!StringCchCopyW+0x50`
Win7 Firefox (x86)	`xul!nsAutoString::nsAutoString+0x1a`	`xul!nsAString_internal::Assign+0x1d`
Win7 Chrome (x86)	`msftedit!CTxtEdit::OnTxChar+0x105`	`msftedit!CTxtSelection::PutChar+0xb8`
Win7 Opera (x86)	`Opera.dll+0x2cf6c6`	`Opera.dll+0x142783`
Haiku WebPositive (x86)	`BWebPage::LoadURL+0x3a`	`BMessage::AddString+0x26`

TABLE I
TAP POINTS FOUND THAT WRITE THE URL TYPED INTO THE BROWSER BY THE USER.

Client	Caller	PC	Process
Deb OpenSSL (arm)	`tls1_generate_master_secret+0x9c`	`tls1_PRF+0x90`	openssl
Deb OpenSSL (amd64)	`ssl3_send_client_key_exchange+0x437`	`tls1_generate_master_secret+0x108`	openssl
Deb Epiphany (arm)	`md_write+0x74`	`md5_write+0x68`	epiphany
Deb Epiphany (amd64)	`md_write+0x60`	`md5_write+0x49`	epiphany
Haiku WebPositive (x86)	`tls1_generate_master_secret+0x65`	`tls1_PRF+0x14b`	WebPositive
Win7 Chrome (x86)	`chrome!NSC_DeriveKey+0x1241`	`chrome!TLS_PRF+0xa0`	chrome.exe
Win7 IE8 (x86)	`ncrypt!Tls1ComputeMasterKey@32+0x57`	`ncrypt!PRF@40`	lsass.exe
Win7 Firefox (x86)	`softokn3!NSC_DeriveKey+0xe85`	`freebl3!TLS_PRF+0xbb`	firefox.exe
Win7 Opera (x86)	`Opera.dll+0x2eb06e`	`Opera.dll+0x50251`	opera.exe

TABLE II
TAP POINTS FOUND THAT WRITE THE SSL/TLS MASTER SECRET FOR EACH SSL/TLS CONNECTION.

Target	Caller	PC
Debian (amd64)	`getname+0x13e`	`strncpy_from_user+0x52`
Debian (arm)	`getname+0x88`	`__strncpy_from_user+0x10`
Haiku (x86)	`EntryCache::Lookup+0x27`	`hash_hash_string+0x1b`
FreeBSD (x86)	`namei+0xd1`	`copyinstr+0x38`
Windows 7 (x86)	`ObpCaptureObjectName+0xcb`	`memcpy+0x33`

TABLE III
TAP POINTS FOUND FOR FILE ACCESS ON DIFFERENT OPERATING SYSTEMS.

OS	Caller	PC	Kernel?	Rank
FreeBSD (x86)	`msglogstr+0x28`	`msgbuf_addstr+0x19a`	Yes	1
Haiku (x86)	`ring_buffer_peek+0x59`	`memcpy_generic+0x14`	Yes	1
Debian (arm)	N/A	`do_syslog+0x18c`	Yes	4
Debian (amd64)	N/A	`do_syslog+0x163`	Yes	4
Minix (x86)	`0x190005ee`	`0x190009d4`	No	8
Windows 7 (x86)	Not Found	Not Found	?	?

TABLE IV
TAP POINTS THAT WRITE THE SYSTEM LOG (`DMESG`) ON SEVERAL UNIX-LIKE OPERATING SYSTEMS. ALL TAP POINTS WERE LOCATED IN THE KERNEL, EXCEPT FOR MINIX, WHICH IS A MICROKERNEL. WE WERE UNABLE TO FIND A TAP POINT ANALOGOUS TO `DMESG` IN WINDOWS.

Abbrv.	Description	Count
bp	binary pattern	2318
rd	repeated dword	400
mz	mostly zero	141
rq	repeated quadword	19
fnu	filenames unicode	8
woa	words ascii	8
wou	words unicode	7
inu	integers unicode	6
bu	binary uniform	5
ura	URLs ascii	5
rs	repeated short	4
fna	filenames ascii	2
rb	repeated byte	2
vr	very redundant	1

TABLE V
LABELS GIVEN TO THE SAMPLED TAP POINTS BY HUMAN EVALUATORS, ALONG WITH THE NUMBER OF TIMES EACH OCCURRED.

Recording	ARI
FreeBSD Apps	0.018
FreeBSD Boot	0.048
FreeBSD Idle	0.021
FreeBSD Shutdown	0.074
Win7 Apps	0.029
Win7 Idle	-0.003

TABLE VI
QUALITY OF CLUSTERING AS MEASURED BY THE ADJUSTED RAND INDEX, WHICH RANGES FROM -1 TO 1, WITH 1 BEING A CLUSTERING THAT PERFECTLY MATCHES THE HAND-LABELED EXAMPLES.

Feature	**Virtuoso [?]**	**VMST [?]**	**TZB (Our Work)**
Focus	Kernel-level introspection	VM-based memory access	User and OS-level monitoring
Active Monitoring	Limited	No	Yes
Adaptability	Requires predefined models	Static analysis	Dynamic AI-based detection
Performance Overhead	High	Medium	Low (QEMU-based replay)

TABLE VII
COMPARISON OF TZB WITH EXISTING FRAMEWORKS

7. Limitations and future work

While our proposed framework effectively identifies and monitors security-relevant tap points, several limitations persist:

1. **Lack of post-mortem analysis**

Currently, TZB is designed for live memory monitoring. This restricts its applicability for forensic investigations that require post-execution memory analysis [1].

Proposed solution: Future iterations could integrate forensic memory capture techniques similar to Volatility for offline threat investigation [12], [13].

2. **Performance trade-offs in large-scale deployments**

Continuous monitoring of extensive memory operations may introduce computational overhead, particularly in enterprise-scale environments [14].

Proposed solution: Implement adaptive monitoring mechanisms that dynamically adjust analysis depth based on system load [15].

3. **Dependency on call stack tracing**

TZB relies on call stack information to resolve execution contexts, which may be unreliable for applications utilizing Just-in-Time (JIT) compilation.

Proposed solution: Explore binary instrumentation techniques such as Intel's Pin Tool or DynamoRIO to enhance execution traceability.

By addressing these limitations, TZB can be refined into a more scalable, adaptive, and forensic-ready security framework.

8. Related work

TZB draws from earlier examination in virtual machine reflection, robotized figuring out, and memory access investigation. Connecting the semantic hole problem, apparatuses like Virtuoso [1] progressively separate contemplation instruments yet face difficulties with deficient preparation. Process out grafting [16] and VMST [2] empower direct cycle observing in secure conditions. TZB expands these by distinguishing in application and operating system level tap focuses.

Memory access designs uncover information structures, protocols [17], [18], [19], and record formats [20], [21], with program passages filling in as intermediaries for information types [22], [23], [24]. TZB applies this knowledge to isolate memory gets to into significant streams.

Perception apparatuses have featured the capability of memory access investigation. For example, Burzstein et al. [25] envisioned memory in technique games to distinguish map information, while ICU64 [26] gave full memory perception to Commodore 64 frameworks. However, TZB doesn't utilize representation, it shares the objective of extricating significant bits of knowledge from memory associations.

9. CONCLUSION

In this paper, we presented TZB, an original framework intended to consequently find up-and-comer memory gets to for dynamic checking of the two applications and working frameworks. This capacity tends to an undertaking that has customarily required huge mastery in figuring out, frequently including careful manual exertion by profoundly gifted space specialists. Via mechanizing this interaction, TZB essentially diminishes the intricacy and time related with distinguishing basic memory passageways, democratizing reflection assignments and empowering more extensive availability for investigators and analysts.

We exhibited the flexibility of TZB by effectively distinguishing an extensive variety of tap focuses, including those used to powerfully separate SSL keys, URLs went into internet browsers, and the names of records being gotten to. These models highlight TZB's capacity to uncover high-esteem experiences that can be utilized for security examination, troubleshooting, and measurable examinations. By utilizing these tap focuses, TZB permits clients to acquire profound experiences into framework and application conduct without expecting earlier information on inner execution subtleties, in this manner connecting the semantic hole that frequently entangles such examinations.

A vital strength of TZB is its engineering. Worked on the QEMU-based PANDA stage, TZB works as a bunch of modules, giving a secluded and extensible system for reflection. Significantly, TZB is both working framework and design skeptic, making it comprehensively relevant across different conditions. This adaptability guarantees that TZB can be sent in a large number of situations, from conventional work area frameworks to implanted stages and cloud-based virtual machines. Its broadly useful plan makes it an amazing asset for specialists and professionals across numerous disciplines, including security, frameworks designing, and programming improvement.

TZB addresses a change in how figuring out errands are drawn closer. By reevaluating the generally troublesome undertaking of figuring out as an organized and principled hunt through streaming information delivered by unique investigation, TZB changes the issue space. Rather than spending incalculable hours physically investigating memory follows, investigators can zero in their endeavors on more elevated level errands that require human mastery, like deciphering results and approving discoveries. This increments efficiency as well as decreases the probability of human mistake in the revelation period of figuring out work processes.

Past its useful commitments, TZB gives major areas of strength for a to future innovative work. As powerful investigation procedures proceed to develop and computational stages become more modern, TZB's particular design guarantees that coordinating new abilities and methodologies can be adjusted. Besides, the framework's capacity to break down memory gets to at runtime opens the entryway for investigating progressed methods, for example, AI based bunching and mechanized arrangement of information streams, which could additionally upgrade its utility.

All in all, TZB addresses a huge headway in the field of mechanized figuring out and dynamic checking. By joining strong unique investigation methods with an adaptable and extensible design, TZB has previously started to change how figuring out undertakings are performed. Its capacity to smooth out complex cycles, combined with its expansive materialness, guarantees that TZB will stay a significant

device for specialists and experts the same. As future improvements expand upon this establishment, TZB can possibly additionally change the field, empowering significantly more proficient and powerful reflection in a quickly developing mechanical scene.

10. REFERENCES

[1] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in Proceedings of the IEEE Symposium on Security and Privacy, May 2011.
[2] Y. Fu and Z. Lin, "Space traveling across VM: Automatically bridging the semantic-gap in virtual machine introspection via online kernel data redirection," in Proceedings of the IEEE Symposium on Security and Privacy, May 2012.
[3] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in IEEE Symposium on Security and Privacy, 2008.
[4] F. Bellard, "QEMU, a fast and portable dynamic translator," in USENIX Annual Technical Conference, 2005.
[5] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A platform for invivo multi-path analysis of software systems," ACM SIGARCH Computer Architecture News, vol. 39, no. 1, 2011.
[6] J. Lin, "Divergence measures based on the Shannon entropy," IEEE Trans. Inf. Theor., vol. 37, no. 1, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/18.61115
[7] H. Steinhaus, "Sur la division des corp materiels en parties," Bull. Acad. Polon. Sci, vol. 1, 1956.
[8] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in Proceedings of the ACM-SIAM symposium on Discrete algorithms, 2007. [Online]. Available: http://dl.acm.org/citation.cfm? id=1283383.1283494
[9] K. Gilbert, "Hurricane sandy serves as lure to deliver Sykipot," http://securityblog.verizonbusiness.com/2012/10/31/ hurricane-sandy-serves-as-lure-to-deliver-sykipot/.
[10] "Raspberry Pi: An ARM GNU/Linux box for $25," http://www. raspberrypi.org/.
[11] L. Hubert and P. Arabie, "Comparing partitions," Journal of Classification, vol. 2,no. 1, 1985. [Online]. Available: http: //dx.doi.org/10.1007/BF01908075
[12] Vendicator, "Stack shield: A "stack smashing" technique protection tool for Linux," Available: http://www.angelfire.com/sk/stackshield/, accessed: Nov. 18, 2024.
[13] T. Chiueh and F. Hsu, "RAD: a compile-time solution to buffer overflow attacks," in International Conference on Distributed Computing Systems, 2001.
[14] S. Sinnadurai, Q. Zhao, and W. Wong, "Transparent runtime shadow stack: Protection against malicious return address modifications," http: //citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.5702, 2008.
[15] Z. Yan, "perf, x86: Haswell LBR call stack support," Available: http: //lwn.net/Articles/535152/, accessed: Nov. 18, 2024.
[16] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process outgrafting: an efficient "out-of-vm" approach for fine-grained process execution monitoring," in Proceedings of the ACM conference on Computer and communications security, 2011. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046751
[17] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," in Proceedings of the ACM conference on Computer and communications security, 2007. [Online]. Available: http://doi.acm.org/10.1145/1315245. 1315286
[18] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution," in Network and Distributed Systems Symposium, 2008.
[19] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: automatic protocol reverse engineering from network traces," in Proceedings of the USENIX Security Symposium, 2007. [Online]. Available:
http://dl.acm.org/citation.cfm?id=1362903.1362917
[20] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: automatic reverse engineering of input formats," in Proceedings of the 15th ACM conference on Computer and communications security, 2008. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455820
[21] Z. Lin and X. Zhang, "Deriving input syntactic structure from execution," in Proceedings of the ACM SIGSOFT International Symposium on Foundations of software engineering, 2008. [Online]. Available: http://doi.acm.org/10.1145/1453101.1453114
[22] J. Lee, T. Avgerinos, and D. Brumley, "TIE: Principled reverse engineering of types in binary programs," in Network and Distributed System Security Symposium, 2011.
[23] A. Slowinska, T. Stancescu, and H. Bos, "Howard: A dynamic excavator for reverse engineering data structures," in Network and Distributed Systems Symposium, 2011.
[24] Z. Lin, X. Zhang, and D. Xu, "Automatic reverse engineering of data structures from binary execution," in Network and Distributed System Security Symposium, 2010.
[25] E. Bursztein, M. Hamburg, J. Lagarenne, and D. Boneh, "OpenConflict: Preventing real time map hacks in online games," in Proceedings of the IEEE Symposium on Security and Privacy, 2011. [Online]. Available: http://dx.doi.org/10.1109/SP.2011.28
[26] mathfigure, "ICU64: Real-time hacking of a C64 emulator."
[27]