SoK: Toward Protecting Internet-Accessible Legacy Systems

William Yurcik[†‡]
Centers for Medicare & Medicaid Services
Baltimore, MD USA
Email: william.yurcik@cms.hhs.gov

Gregory Koenig
Independent Researcher
Atlanta, GA USA
Email: koenig@acm.org

Gregory Pluta
University of Illinois at Urbana-Champaign
Champaign, IL USA
Email: gpluta@illinois.edu

Gianni Pezzarossi
Stuart Turner
University of Illinois at Urbana-Champaign
Champaign, IL USA
Email: {gpezza2, slturne1}@illinois.edu

Fabio Roberto de Miranda
Luciano Pereira Soares
Insper
São Paulo, SP Brazil
Email: {fabiomiranda, lucianops}@insper.edu.br

***Abstract*— The security problem with legacy systems is large, persistent, and structurally embedded in critical sectors. It is not just a patching problem but rather a systemic issue driven by economics, operational dependency, and architectural constraints. In this Systemization-of-Knowledge paper, we survey and summarize literature on legacy systems and identify several potential solutions we intend to pursue.**

Keywords - cross domain solutions; technical debt; technical obsolescence; network pump; data diode; end-of-life.

I. INTRODUCTION

Legacy systems are outdated computer systems, software, or other technologies that are still in active use despite being superseded by newer solutions [1]-[13]. This includes hardware, software, file formats, and programming languages. Organizations retain legacy systems in order to serve business needs even though they may be inefficient, have high maintenance costs, high cybersecurity risk, and incompatibility with current technologies. The real problem is not age, but lack of adaptability since most legacy systems were never intended for their current uses. Legacy systems typically struggle with modern requirements for real-time data, streaming workloads, and evolving platforms. Table I has the major distinguishing characteristics of legacy systems.

TABLE I. CHARACTERISTICS OF LEGACY SYSTEMS

Beyond End-of-Life Support	*can no longer receive updates, support, or maintenance from the original manufacturer or third-party vendor*
No Longer Available for Purchase	*no longer available for purchase and/or rely on now-obsolete technology for maintenance*
Shrinking Workforce	*requires developers with expertise in outdated programming languages to maintain.*
High Maintenance	*requires frequent and/or time-intensive repairs often relying on third-party or salvaged spare parts (eBay)*
Vulnerability Exploitation	*Internet-exposed cybersecurity vulnerabilities not being remediated with software patches*

[†] Corresponding Author

[‡] Official Organizational Disclaimer: "The views presented herein do not represent the views of the Federal Government."

The term "technical debt", or what we prefer to call "technical obsolescence", refers to the accumulated cost of keeping old systems running in ways they were never designed to support [18]-[21]. This technical obsolescence occurs because of the widening gap between the functions a system was originally designed to perform versus the functions an organization currently has the system performing.

Viewing the situation in which many organizations find themselves through the lens of technical obsolescence, the challenge becomes apparent: organizations purchase hardware or software systems, many of which have a substantial cost, and are forced to operate these systems when they transition into being legacy systems due to the cost-prohibitive nature of replacing them before the organization's budget might allow. Thus, technical obsolescence accumulates over years (or decades) through:

- quick fixes layered on top of old architectures
- workarounds instead of redesigns
- outdated languages, frameworks, or platforms
- deferred upgrades because "replacement is coming"
- patches added to remediate software vulnerabilities

Thus, many systems have become legacy systems because they do not have the timely resources to upgrade and support new requirements. Each decision not to re-architect and/or upgrade makes sense at the time—but together the end result is an increasingly inefficient, fragile, and risky system.

Maintaining legacy systems is likewise costly. Among other factors, legacy system degradation occurs with the use of outdated computer languages, and poor documentation, which inevitably increases maintenance costs [14]. This largely explains the high proportion of total IT expenditure organizations commit to system maintenance. By some estimates, as much as 75% of the IT budgets of banks and insurance companies have been consumed by legacy systems maintenance costs [15] [16]. In 2019, the U.S. government spent over $90B on Information Technology (IT), of which about 80% was used to operate/maintain legacy systems [17].

Given myriads of challenges, why do organizations continue to rely on legacy systems? Despite their limitations, these legacy systems still work and are relatively stable, predictable and well-understood by the organization. If a system has been running payroll, benefits, billing, or claims processing for many years without major outages, leadership is often reluctant to touch it. From a risk perspective, *"if it's not broken then don't fix it"* is powerful.

We are surrounded by legacy technologies that are still in active use: cellphones, televisions, radios, keyboards, data storage devices, cars, medical devices, microwave ovens, refrigerators, traffic lights, credit card readers, and computers. These technologies survive because they are universally compatible, cheap and reliable, "good enough" for the job, and supported by massive infrastructure. Newer technology has to be dramatically better to displace something that already works everywhere.

The decision to replace a legacy system is both expensive and inherently risky. Modernization is not just buying new software. It often means rewriting or migrating decades of data, reengineering business processes, training staff, and handling downtime and transition risk [1]-[13]. Replacement of large legacy systems typically goes over budget, takes years, and/or fails outright. Compared to the expense and risk of replacement, keeping a legacy system limping along is often the safest short-term bet.

Legacy systems are rarely standalone, instead they are typically deeply embedded in the organization integrated with dozens of other systems and hard-coded to specific workflows which have been developed over years [1]-[13]. Institutional knowledge is locked into legacy systems such that organizations adapt to the limitations of the legacy system and workarounds become "how things are done". Thus, changing out a legacy system can trigger a domino effect across the entire enterprise. This all-encompassing impact alone can stall legacy system modernization indefinitely.

In summary, organizations do not keep legacy systems because they love them—they keep them because the perceived risk of change tradeoffs is greater than the pain of staying the same. The perceived risk of change tradeoffs being greater including actual cost (Capex), plus time cost, migration cost, and training costs.

However, the viability of continuing to use a legacy system indefinitely without end is problematic. There is a significant time period between when a legacy system is unsupported but still highly-functional, to the time when a legacy system is no longer operational. This is the precise period of time the technical challenges we describe in this paper occur.

The remainder of this paper is organized as follows: Section II provides a literature summary of legacy systems issues. Section III describes a specific context for legacy systems for illustrative purposes. Section IV presents several potential solutions that have been used, and we plan to explore in the future. We end with a summary and conclusions in Section V.

II. THE CYBERSECURITY RISKS OF LEGACY SYSTEMS

The technical obsolescence of legacy systems is not just messy code—it's accumulated cybersecurity risk [18]-[21]. Legacy systems create outsized cybersecurity risk not because they were poorly designed, but because they were designed in the past for a different threat environment. Modern attackers actively exploit the assumptions those systems were built upon in the past.

A. Unsupported Unpatched Legacy Systems

Unsupported and unpatched software is the single biggest risk to legacy systems. Security patches protect against cybersecurity incidents by adding new security features and fixing software bugs (i.e., coding errors or remediating known vulnerabilities). Since legacy systems do not receive patches, they are more vulnerable to cyberattacks.

Cybersecurity support is "a reasonable expectation of a predictable effective response to a new cybersecurity risk" [22]. When vendors discontinue cybersecurity support for legacy systems, known vulnerabilities persist and remain susceptible to exploitation. As a result, legacy systems become easy targets for inexpensive, automated, and widely available public exploits.

No system can be supported forever, but this fact has not been integrated into organizational processes. In particular the cybersecurity risk of outdated software is not being communicated very well if at all [22]. These risks of outdated software are typically included in the unread small print of disclosure agreements. Table II describes the different terms being used to describe the status of legacy systems in regard to support for cybersecurity vulnerability patches.

TABLE II. LEGACY SYSTEM STATUS RELEVANT TO CYBERSECURITY

End-of-Support (EOS): *Hardware devices, firmware, and software versions that no longer receive timely, supported updates from the original equipment manufacturer, including security patches, security updates, software fixes (hotfixes), and defects. Vendor stops taking responsibility for system/software operation.*
End-of-Life (EOL): *Point at which a technology is no longer supported by its vendor. Importantly: EOL does not mean the system stops working — it means the vendor stops taking responsibility for it.*
End-of-Service: *Point at which a system is no longer actively serviced or maintained although it may still technically be supported in limited ways for critical fixes. It sits between "fully supported" and "end-of-life" on the vendor lifecycle.*
End-of-Maintenance (EOM): *Point at which a vendor stops providing routine maintenance fixes for a product, even though the product may still be usable and may still receive very limited support.*
End-of-Renewal (EOR): *Point at which a vendor no longer allows customers to renew licenses, subscriptions, or support contracts for a product. The system may still be operational but contractual support obligations are soon expiring.*
End-of-Sale: *Point at which you can no longer buy the product, but it may still be supported*
End-of-Expansion (EOX): *Point at which a vendor no longer allows the system to be expanded in size, capacity, or scope even though the system may still be operational and supported*

End-of-Life is not necessarily a cybersecurity vulnerability by itself [22]. There are many legacy systems still in active use today that are no longer supported by their original vendor that are instead being protected by compensating cybersecurity controls provided by the host organization. In fact, this situation is common in industries where systems typically outlive the companies, products, or strategies that created them. Well-known examples include: (1) COBOL-based systems (typically the original vendor no longer exists); (2) VAX systems, DEC defunct 1990s, still used in: transportation, manufacturing, government, utilities; and (3) Windows XP Embedded, EOL 2014-2019 (depending on variant), still used in: ATMs, medical devices, kiosks, industrial systems.

End-of-life is usually final, although exceptions have happened in situations where an otherwise End-of-Life system was still widely deployed. If system failure would be catastrophic then revival/re-support is possible. For example, legacy systems that have reached End-of-Life and have been brought back to life include: (1) Windows XP, EOL 4/2014, revived 2017 due to WannaCry computer virus; (2) IBM mainframe OS/390 variants, revived as mainframes "too big to fail"; (3) Oracle 9i/10g databases EOL 2000s-2010s (depending on variant), now "actively supported"; (4) Siemens industrial control systems EOL revived due to national infrastructure risks; (5) VMS EOL multiple times under different owners (DEC/HP/Compaq → VMS Software) currently under active support.

Is it possible to determine whether a legacy software system is supported without access to formal contractual documentation? The answer is both yes and no [22]. In a development environment, there are signals for "Signs-of-Life" support including number of maintainers, stars (bookmarks, endorsements) and forking (forking allows someone to modify the code independently), recent activity (opening issues), and the existence of dedicated security communications channels for the software in question. There is also the possibility that small portions of code may be stable without needing support.

The imminent threat of exploitation for unpatched unsupported legacy systems is substantial and constant, resulting in a significant threat to an entire organization. Advanced threat actors search to identify and target unsupported legacy systems as a means to pivot into other systems and enterprise networks. Legacy systems are attractive targets due to their extensive reach into an organization's network and integrations with identity management systems and are especially vulnerable to cyber exploits targeting newly discovered, unpatched vulnerabilities. Additionally, since they no longer receive supported updates from the original equipment manufacturer, they are exposed to disproportionate and unacceptable risks.

While we have focused on lack of vendor support for patching legacy systems, there are also non-patchable vulnerabilities.

B. Non-Patchable Legacy Systems

A non-patchable vulnerability for a legacy system is one where there is no available patch leaving the only true fix as system replacement—everything else is damage control. There are many examples of vulnerabilities in legacy systems that are effectively non-patchable, meaning there is no fix available from the manufacturer and remediation would require architectural modification or system replacement. Table III shows a comparison between patchable/non-patchable vulnerabilities. Examples of non-patchable vulnerabilities in legacy systems include:

- Hard-coded cryptographic algorithms
- Systems that only support TLS 1.0 or SSLv3
- Firmware with hard-coded credentials
- Applications dependent on deprecated authentication protocols
- Embedded system firmware that cannot be updated

TABLE III. COMPARISON OF PATCHABLE VS NON-PATCHABLE VULNERABILITIES IN LEGACY SYSTEMS

DIMENSION	*PATCHABLE VULNERABILITIES*	*NON-PATCHABLE VULNERABILITIES*
Definition	*cybersecurity weaknesses correctable by applying a vendor-provided patch*	*cybersecurity weaknesses not patchable due to architecture, end-of-life, and/or hard-coded design*
Nature of Flaw	*implementation bugs (buffer overflows, logic errors, missing input validation)*	*design or architectural flaws (insecure protocols, weak cryptography, trust assumptions)*
Availability of Fixes	*patch exists, available. & documented*	*no fix exists; no patch will ever be released*
Remediation Time	*days to weeks (depending on testing/change control)*	*indefinite — permanent risk for entire life of system*
Operational Impact of Remediation	*episodic with patch sandbox testing & application off-hours*	*continuous (monitoring, controls, documentation)*
Required Actions	*apply patch, test, verify, close vulnerability*	*formally accept risk & apply compensating controls*
Compensating Controls	*optional & temporary*	*mandatory & permanent*
Example Controls	*patch management, change management*	*isolation, access restrictions, protocol blocking, monitoring, gateways*
Cost Profile	*predictable, ongoing maintenance cost*	*technical obsolescence in form of hidden & growing costs (controls, audits, incidents)*
Change Risk	*pre-testing and verification results in low/moderate & predictable change risks*	*high – system may break with any changes*
Risk Trend Over Time	*low if patched quickly & significant reduction after patch applied*	*high due to public, well-known exploits & increasing overtime as threats evolve*
Audit Status	*closed finding after patch installed*	*finding remains open with compensating controls and risk acceptance*
Long-Term Resolution	*patch management as part of normal operations*	*fixing requires replacement, major redesign & reimplementation, and/or current system shutdown*

Cybersecurity teams are not ignoring non-patchable vulnerabilities, they are managing permanent risk. In non-patchable cases, the vulnerability is structural in architectural design - frozen in time.

Patching can also be impossible to perform — or patching itself may create greater operational risk than the vulnerability itself. Even when patches exist:

- Applying a patch may break the legacy system
- Sandbox patch testing environments may not exist
- Knowledge of safely applying patches may not exist

C. Summarizing Legacy System Vulnerabilities

Legacy systems pose additional significant cybersecurity risks because they are often incompatible with modern cybersecurity monitoring tools, hindering detection and automated responses. This results in attacks going undetected for longer periods before being identified and addressed. Given this longer time to detection, attacks on legacy systems typically cause more damage.

In summary of cybersecurity risk, legacy systems increase the likelihood of compromise from both non-patchable vulnerabilities and patchable vulnerabilities that are no longer provided under vendor support. When a compromise occurs for a legacy system, the impact is large and existential. Legacy systems turn manageable threats into existential risks because they have not evolved to be resilient to modern cybersecurity attacks.

III. A SPECIFIC LEGACY SYSTEM CONTEXT: SCIENTIFIC INSTRUMENTS

Legacy systems exist in an almost infinite number of contexts. We now describe one specific context for illustrative purposes: a legacy system involving high value and sensitive scientific instruments shared globally with scientists and students.

Advanced cyber-infrastructure is increasingly needed to support data-driven and interdisciplinary research. Many universities have scientific instruments in their laboratories that still use outdated operating systems (OSs) like Windows XP, NT, and 2000, which pose security risks [23]-[25]. These instruments include Scanning Electronic Microscopes (SEMs), Transmission Electronic Microscopes (TEMs), and Atomic Force Microscopes (AFMs) [23]-[25].

The challenges of remote experiments on high-value scientific instruments are: (1) Scientific instruments, often multi-million-dollar investments, age slower than computing and networking technologies. Instruments last 20-30 years, while their software typically lasts 6-8 years, and cyber-components need upgrades every 6 months to 2 years for security and performance. (2) Scientific software is embedded in an instrument's OS for performance, meaning OS upgrades require simultaneous software upgrades, causing mismatches between them. (3) Frequent OS patches are essential to keep an instrument securely connected to a university network, preventing cybersecurity risks and ensuring access to cloud services. Without these patches, instruments become vulnerable, leading to research slowdowns due to limited local storage and reduced productivity as researchers spend more time managing data manually.

This situation persists because instrument companies may not update their software as frequently as OS providers due to software development lifecycles, support costs, and business life cycle (businesses leave the market). Even newer OSs like Windows 10 will soon be outdated. Consequently, the current networked solution for scientific instruments cannot evolve, hindering the advancement of discovery and cyberinfrastructure deployment.

IV. POTENTIAL SOLUTIONS

Legacy systems are not insecure because of neglect or poor management. They are insecure because they cannot be fundamentally upgraded with patches against the latest known cybersecurity threats without unacceptable operational risk. Most legacy systems:

- Are mission-critical
- Are architecturally frozen
- Are no longer fully supported by vendors
- Contain non-patchable/removable vulnerabilities

For these systems, eliminating risk is not technically or operationally feasible.

A foundational concept for legacy systems is that they need to focus on containment, isolation, and survivability, not elimination of risk. For legacy systems, risk cannot be eliminated—because the system itself cannot be fundamentally changed. The objective is not to make it secure but rather to limit damage when something goes wrong.

Containment focuses on limiting how far an attack can spread, turning potential catastrophic attacks into manageable incidents. Even if a system is compromised, exfiltration can be blocked and the attacker cannot pivot easily to attack peer systems such that peer systems remain protected and damage stays localized.

Most exploits require network access, unrestricted communication, and discovery/scanning ability. **Network isolation** removes those conditions—even if the vulnerability still exists. By restricting the flow of network traffic into and out of the system, limiting the points from where the system may be accessed. Network isolation can be accomplished using air-gaps (where feasible), firewalled/segmented network zones, jump hosts, and protocol restrictions.

Survivability includes strong monitoring, logging, and alerting. The assumption is that compromise is possible, and that any system compromise should be detected addressed using a variety of recovery techniques resulting in minimal disruption to mission delivery. This ensures the organization can withstand and recover from incidents, rather than being paralyzed by them.

A. General Techniques

The goal is to reduce the likelihood and impact of compromise on legacy systems while still meeting a wide variation of different needs. Leveraging domain-specific contexts can help but there is no silver bullet solution [26] [27].

The first consensus technique is to keep legacy systems isolated from the Internet using network isolation and segmentation. If attackers cannot reach the legacy system, they cannot exploit it. External compensating security controls that can block attack patterns and detect suspicious activity include Web Application Firewalls (WAFs).

Application and protocol hardening involves disabling unused ports and removing unnecessary services (every disabled port and removed service eliminates an attack vector), enforcing least-privilege service accounts, restricting allowed protocols, enforcing the use of secure tunnels or gateways for required legacy protocols.

Survivability includes strong monitoring, logging, and alerting -- assume compromise is possible but detect it quickly and respond using:

- Network intrusion detection/prevention (IDS/IPS)
- Host-based intrusion detection (where supported)
- File integrity monitoring
- Offline, immutable backups
- Tested restoration procedures
- Manual fallback processes

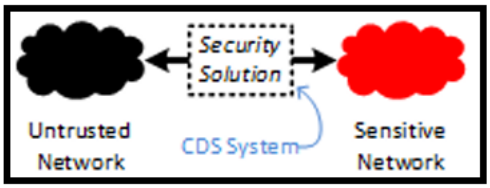

Figure 1. Cross-Domain Solutions Between Disparate Domains

Cross Domain Solutions (CDS) are another potential solution that may enable organizations to share information across physically, logically, and administratively separated networks (known as security domains) in a reliable, secure and interoperable manner as shown in Figure 1 [28]-[30]. A Cross-Domain Solution is typically a combination of hardware/software mechanisms that enable data transfer between two or more information systems operating at different security domains (e.g., different classification levels, networks, or trust zones) [28]-[30]. Thus, CDS allows controlled information sharing between networks that are intentionally separated for security reasons.

In Figure 2 the taxonomy of these solutions can be broadly classified into broad categories of unidirectional transfer and bidirectional transfer. Unidirectional transfer solutions, often built around hardware like data diodes, enforce a strict one-way flow of information, which is ideal for sending data from a less secure network to a more secure one for monitoring or analysis without creating a return path for threats. Bidirectional transfer solutions allow for a controlled, two-way exchange of information. These systems, often called "guards," use a "protocol break" to terminate and deeply inspect all network traffic. They employ sophisticated filtering mechanisms—including content filtering, data sanitization to remove hidden metadata, and malware scanning—to ensure that only authorized data that complies with security policies can pass between the domains.

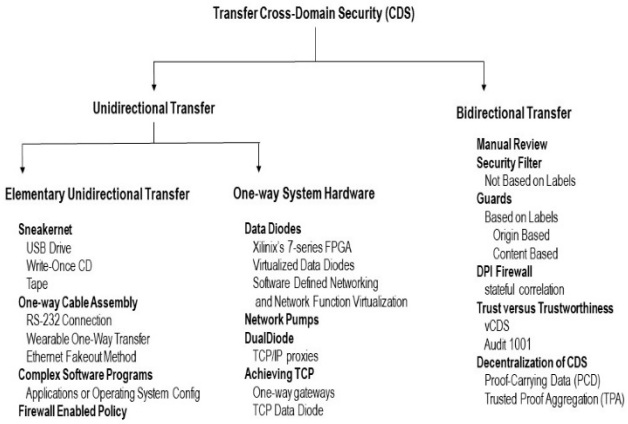

Figure 2. Classification Taxonomy of Cross-Domain Solutions (CDS)

B. The NRL Network Pump

The NRL Network Pump (developed by the U.S. Naval Research Laboratory) is a high-assurance, one-way network transfer device designed exactly for CDS -- to securely move data from a lower-classification or lower-trust network to a higher-classification or higher-trust network. It is often described as a trusted guard or controlled data transfer mechanism for cross-domain environments.

As shown in Figure 3 the key idea is the NRL Network Pump (aka Pump) allows information to move in one direction only, with no return path [31]-[33]. This could potentially protect legacy systems from compromise originating in less trusted environments.

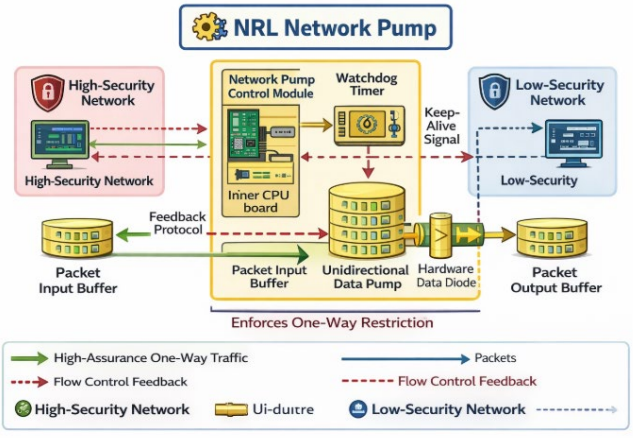

Figure 3. Data Flow within the NRL Network Pump

The Pump allows one-way information transfer only from source → destination. However, the Pump must have middleware designed for specific supported protocols such that it parses and reconstructs messages. In this way the Pump is not the same as a firewall since it does not forward packets but rather recreates packets. The original Pump was designed for the reconstruction of SMTP protocol email messages, enforcing strict rules blocking anything that does not

conform. With no bidirectional responses back from the destination back to the source (e.g., acknowledgments, flow control), the Pump breaks end-to-end TCP connections while preventing covert channels, interactive exploits, and protocol tunneling. The Pump protects legacy systems using:

Isolation from Direct Network Exposure: the legacy system never communicates directly with untrusted systems, the legacy system is shielded from malformed traffic, the legacy system is protected from remote interactive exploitation.

Protocol Normalization: The pump terminates incoming connections and reconstructs well-formed protocol-compliant limited structured data as input to the legacy system. This protects legacy software that may crash or be exploited by malformed packets, buffer overflow attempts, executable malware, and/or protocol fuzzing.

Elimination of Back Channels: Legacy systems often cannot defend against covert exfiltration channels, reverse shells, and interactive command-and-control sessions.

In summary the Pump turns fragile and vulnerable legacy systems into isolated data consumers rather than exposed network participants. The Pump protects legacy systems by enforcing one-way communication, breaking direct network sessions, normalizing protocol data traffic, eliminating back channels, and eliminating exposure to malicious traffic.

C. The Data Diode

As shown in Figure 4, a data diode is a hardware-enforced unidirectional network device that allows data to flow in only one direction between two networks [34]-[38]. Unlike firewalls, which rely on software rules, a data diode enforces one-way communication at the physical layer — typically by removing or disabling the receive path in one direction, using transmit-only (Tx-only) and receive-only (Rx-only) network interfaces, using optical fiber with a physically absent return channel (air gap) and implementing hardware logic that makes reverse signaling impossible. Because the restriction is physical rather than logical, it provides high-assurance isolation between networks.

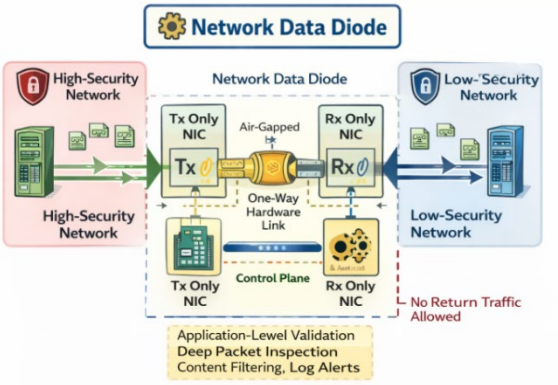

Figure 4. Data Flow within the Network Data Diode

A data diode can protect legacy systems fundamentally by eliminating the inbound attack surface. Even if a legacy system is compromised internally, it cannot establish outbound connections, it cannot receive instructions, it cannot participate in interactive sessions -- this dramatically reduces post-compromise impact.

In summary, a data diode can serve as a compensating control for legacy systems by physically isolating, restricting communication direction, and dramatically reducing inbound network exposure. A data diode turns fragile legacy systems from exposed network participants into isolated data emitters — dramatically reducing their cybersecurity risk.

D. Edge Computing via Virtual Machines & Cloudlets

Another technique is the use of edge computing to protect legacy systems by using virtual machine (VM) encapsulation to relocate execution closer to users or data—without modifying the legacy software itself [39]. The key mechanism introduced is called Edge-Based Virtual Desktop Infrastructure (EdgeVDI) which works as shown in Figure 5. While traditional VDI only works well on LANs because RDP is latency-sensitive, edge computing solves this by moving the VM from the central cloud (Tier-1) to a nearby cloudlet (Tier-2), restoring LAN-quality latency.

This edge computing solution protects legacy applications by removing direct exposure of legacy systems. Instead of running legacy applications on user laptops/exposed desktops/unmanaged distributed systems, they can instead run inside controlled VM instances at cloudlets.

This technique has the security benefit of reducing the attack surface and end system risk. Legacy apps are isolated from host environments and configuration drift. Encapsulation ensures a fixed OS version, controlled library versions, controlled configuration state, and a snapshot/rollback capability. Instead of moving large datasets across WAN links, EdgeVDI moves the VM to the data location.

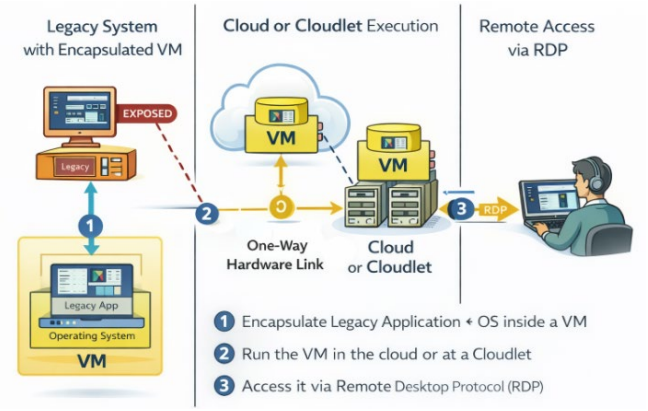

Figure 5. Data Flow within the Cloudlet Edge Computing Approach

Edge computing protects legacy applications by encapsulating them in VMs and dynamically relocating execution to nearby cloudlets, reducing latency, minimizing WAN data transfer, isolating execution environments, and eliminating the need to rewrite fragile legacy software. Rewriting legacy software for cloud-native operation is only economically viable for a tiny fraction of applications. Thus, edge computing offers a protection and modernization strategy that requires zero changes to legacy code, improves performance, enhances mobility, and reduces network exposure for protection. Edge computing transforms legacy applications from vulnerable, device-bound software into secure, centrally managed, migratable execution environments.

The challenge with EdgeVDI is if the legacy software happens to be hosted on a machine that attaches to a device that it is controlling, such as via a serial connection and/or USB/PCI connection, then virtualization cannot be used to protect the software because it needs to be able to communicate directly with the underlying hardware.

V. CONCLUSION AND FUTURE WORK

The size of the security problem with legacy systems is widespread, economically entrenched, and concentrated in the most operationally critical environments. It is one of the dominant structural cybersecurity challenges across government, healthcare, industrial control systems, and industries of all types in all countries around the world.

Protecting legacy systems requires layered, architectural controls because many legacy platforms cannot be patched, upgraded, or instrumented with modern security agents. No single solution has yet proven to be sufficient but potential solutions include the following which we presented:

- System hardening
- Controlled mediation (NRL Network Pump)
- One-way transfer controls (Data Diodes)
- Architectural relocation (EdgeVDI computing with VM encapsulation)

Common aspects of these potential solutions include:

- Eliminating direct Internet exposure
- Expose APIs instead of raw interfaces
- Middleware in front of the legacy systems
- Enforcing strict one-way communications
- Validating input at a one-way boundary
- Eliminating reverse channels

For future work, we intend to experiment, test, and share empirical research results for the potential solutions we presented in this paper. It is our current intuition that the best solution to protect a particular legacy system may depend on the specific context of the legacy system in question so there may not be one general solution but multiple solutions given different contexts.

ACKNOWLEDGMENT

We would first like to acknowledge and sincerely thank the anonymous CROSS-SEC peer reviewers whose constructive feedback led to significant improvements to this paper. Authors Miranda and Soares were supported by a joint funding support agreement between the Insper Institute of Education & Research and the Computer Science Department at the University of Illinois at Urbana-Champaign.

REFERENCES

[1] PacketLabs,"What is the Definition of Legacy Systems?" Sep 2024.

[2] J. Crotty and I. Horrocks, "Managing Legacy System Costs: A Case Study of a Meta-Assessment Model to Identify Solutions in a Large Financial Services Company," Appl Computing and Information 13(2) 2017.

[3] H. K. A. Bakar and R. Razali, "A Preliminary Review of Legacy Information Systems Evaluation Models," IEEE Intl Conf on Research and Innovation in Info Systems (ICRIIS), 2013.

[4] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage, "How Do Professionals Perceive Legacy Systems and Software Modernization," 36th ACM Intl Conf on Software Engineering, 2014.

[5] C. Brooke and M. Ramage, "Organizational Scenarios and Legacy Systems, Int. J. f Inform. Mgmt J. Info. Professionals 21(5) 2001.

[6] I. Sommerville, Software Engineering, 10th ed., Pearson Education, Harlow, 2015.

[7] W. K. Assunção, L. Marchezan, A. Egyed, and R. Ramler, "Contemporary Software Modernization: Perspectives and Challenges to Deal with Legacy Systems," 2024.

[8] K. G. Wesley, L. M. Assunção, L. Arkoh, A. Egyed, and R. Ramler, "Contemporary Software Modernization: Strategies, Driving Forces, and Research Opportunities," ACM Trans. Softw. Eng. Methodol. 34(5) Article 142 June 2025.

[9] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, "Legacy Information Systems: Issues and Directions," IEEE Software 16(5) Sept 1999. <https://doi.org/10.1109/52.795108>

[10] K. Bennett, "Legacy Systems: Coping with Success," IEEE Software, 12(1) Jan. 1995. <doi:10.1109/52.363157>

[11] L. Ellot, "Legacy Systems, Legacy Options," Computer World, pp. 86, 88 Jul 11 1994.

[12] M. Ali, S. Hussain, M. Ashraf, and M. K. Paracha, "Addressing Software Related Issues On Legacy Systems – A Review," Intl J of Scientific & Technology Research, 9(03) Mar 2020.

[13] European Union Agency for Cybersecurity (ENISA), "Risks of Using Discontinued Software," Flash Note 01, Jan 29 2014.

[14] A. De Lucia, A.R. Fasolino, and E. Pompelle, "A Decisional Framework for Legacy System Management," IEEE Intl Conf on Software Maintenance, 2001.

[15] M. Arnold and T. Braithwaite, "Banks' Ageing IT Systems Buckle Under Strain', Financial Times, Jun 18 2015.

[16] G. R. Gangadharan, E. J. Kuiper, M. Janssen, and P. O. Luttighuis, "IT Innovation Squeeze: Propositions and a Methodology for Deciding to Continue or Decommission Legacy Systems, Grand Successes and Failures in IT," Intl Federation of Information Processing, Springer Link, 2013.

[17] Government Accounting Office (GAO), "Information Technology: Agencies Need to Develop Modernization Plans for Critical Legacy Systems," Report 19-471. 2019.

[18] C. Mims, "The Invisible $1.52 Trillion Problem: Clunky Old Software," Wall Street Journal, Mar 1 2024.

[19] H. Krasner, "The Cost of Poor Software Quality in the U.S.: A 2022 Report - From Problem to Solutions," Consortium for Information & Software Quality (CISQ), Dec 15 2022.

[20] H. Krasner, "The Cost of Poor Software Quality in the U.S.: A 2020 Report," Consort for Info & SW Quality (CISQ) 2021.

[21] H. Krasner, “The Cost of Poor Software Quality in the U.S.: A 2018 Report - From Problem to Solutions,” Consortium for Information & Software Quality (CISQ), Sep 26 2018.

[22] A. Friedman, “All Good Things: End-of-Life and End-of-Support in Policy and Practice,” RSA Conference, 2024.

[23] P. Nguyen, T. Elgamal, S. Konstanty, T. Nicholson, S. Turner, P. Su, K. Nahrstedt, T. Spila, R. H. Campbell, J. Dallesasse, M. Chan, and K. McHenry “BRACELET: Edge-Cloud Microservice Infrastructure for Aging Scientific Instruments,” Intl Conf. on Computing, Networking and Com (ICNC), 2019.

[24] P. Nguyen, S. Konstanty, T. Elgamal, T. Nicholson, S. Turner, P. Su, K. Nahrstedt, T. Spila, R. H. Campbell, J. Dallesasse, M. Chan, and K. McHenry, “BRACELET: Hierarchical Edge-Cloud Microservice Infrastructure for Scientific Instruments’ Lifetime Connectivity,” UIUC Technical Report, 2018.

[25] P. Nguyen and K. Nahrstedt, “MONAD: Self-adaptive Micro-service Infrastructure for Heterogeneous Scientific Workflows,” IEEE Intl. Conf. on Autonomic Computing (ICAC), 2017. <doi: 10.1109/ICAC.2017.38>

[26] Australian Signals Direct, “Managing the Risks of Legacy IT: Practitioner Guidance,” Austral Cyber Sec Centr, Apr 2024.

[27] Australian Signals Directorate, “Managing the Risks of Legacy IT: Executive Guidance,” Austral Cyber Sec Centre, Apr 2024.

[28] V. Sundaravarathan et al. “Cross-Domain Solutions (CDS): A Comprehensive Survey,” IEEE Access, Vol. 12, pp. 163551-163620, 2024. <doi:10.1109/ACCESS.2024.3483659>

[29] Australian Signals Directorate, “Fundamentals of Cross Domain Solutions,” Austral Cyber Security Centre, Oct 2021.

[30] B. Adam, “Cross Domain Solutions Overview,” Cross Domain Solutions Overview Def Acquistn Univ, Feb 2023.

[31] M.H. Kang, I. S. Moskowitz, and D. C. Lee, ‘‘A Network Pump,’’ IEEE Trans. Softw. Eng., 22(5) May 1996.

[32] M. H. Kang, I. S. Moskowitz, and S. Chincheck, ‘‘The Pump: A Decade of Covert Fun,’’ 21st Ann. Comput. Sec. Appl. Conf. (ACSAC), 2005.

[33] S. K. Gorantla, S. Kadloor, N. Kiyavash, T.P. Coleman, I.S. Moskowitz, and M.H. Kang, ‘‘Characterizing the Efficacy of the NRL Network Pump in Mitigating Covert Timing Channels,’’ IEEE Trans. Inf. Forensics Secur., 7(1) Feb. 2012.

[34] A. Almaazmi, M. S. Al Shehhi, O.A. Alkhoori, S.J. Al Shehhi and Y. Hamid, "Data Diode for Cybersecurity: A Review," Intl Conf on Artificial Intel of Things (ICAIoT) 2022.

[35] B.-S. Jeon and J.-C. Na, "A Study of Cyber Security Policy in Industrial Control System using Data Diodes," 18th Intl Conf on Advanced Comm Techn (ICACT) 2016.

[36] J. H. Yun, Y. Chang, K. H. Kim, and W. Kim, “Security Validation for Data Diode with Reverse Channel,” In: G. Havarneanu, et al., (editors) Critical Information Infrastructures Security. Springer LNCS Vol 10242. 2017.

[37] D. W. Jones and T. C. Bowersox, “Secure Data Export and Auditing Using Data Diodes,” USENIX/ACCURATE Electronic Voting Technology Workshop, 2006,

[38] E. D. Knapp, “Chapter 11: Implementing Security and Access Controls,” within book: Industrial Network Security 3rd edition, Elsevier 2024.

[39] M. Satyanarayanan et. al., “Edge Computing for Legacy Applications,” IEEE Pervasive Computing 19(4) 2020.