R210 Essay: Capability Systems

Chongyang Shi (*cs940*)

February 23, 2018

1 Summaries of research

Wagner and Tribble [1] reported findings from a limited-time security analysis on an early capability-based web browser *DarpaBrowser*, designed to protect the browser and the underlying operating system from malicious renderers, as well as from the insecure web renderers are exposed to. Starting from the user perspective, they analysed the design and implementation of each layer of the capability system. Most severe vulnerabilities discovered in the system's design resulted from insufficient "taming" of the interoperable Java libraries, highlighting the potential security impacts of supporting legacy codebases. Significant vulnerabilities in the implementation were also revealed. While a vast majority of discovered vulnerabilities were relatively simple to fix, the authors highlighted their reservations about DarpaBrowser as a capability system due to limited analysis coverage and DarpaBrowser's flawed design. The authors' exemption of a focus-related vulnerability due to DarpaBrowser using a prior non-vulnerable version of Java SDK would not be seen as a reliable argument today.

Watson et al. [2] designed and implemented a lightweight capability framework *Capsicum*, which was shown to be easily integrable into a wide range of applications. As an extension to UNIX APIs, Capsicum could be integrated into existing applications with the choice of entering a capability mode or integrating with the userspace *libcapsicum*. While some manual checking of delegated rights at runtime are still required to ensure security, Capsicum was shown in evaluations to require relatively minor code changes and produce low overheads, especially for applications already partially implementing sandboxing or privilege compartmentalisation, such as Chromium. In some cases, Capsicum outperformed existing sandbox APIs through efficient design, such as against *setuid* in *chroot*. A possible criticism regarding Capsicum's on-demand initialisation support is the lack of discussions on Capsicum's workarounds for compatibility issues other than DNS queries [2, 4.1].

A few years after the publication of Capsicum, Watson et al. [3] reported the design and implementation of a compartmentalisation-based capability system CHERI, which spanned across the entire system stack, including a userspace library for application integration, a specialised LLVM compiler and kernel, and a FPGA coprocessor with CHERI-specific ISA. As a hybrid capability system, CHERI is scalable, and provides both excellent programmability and compartmentalisation guarantees, without significantly compromising performance. This was demonstrated with micro and macro-benchmarks against both non-compartmentalised systems and Capsicum, including on system applications adapted by both studies. While designed for unified protection, the authors acknowledged that future work was still needed to increase the CHERI stack's flexibility. The lack of Chromium as evaluated in Capsicum may have been a result of flexibility limitations.

2 Key themes of research

2.1 Maintaining the principle of least privilege

One of the key underlying goals of capability systems is to maintain the principle of least privilege, which grants benefits by limiting damages done both by malicious applications and by capabilities that can be insecurely delegated by vulnerable applications. As a web browser can be broadly delegated with the current user's capabilities while remain exposed to the insecure web, Wagner and Tribble [1, 4.1] recognised the need to minimise the browser's privileges through compartmentalisation. This was also the ultimate goal of Capsicum [2, Sec. 1], which implemented easy-to-adapt compartmentalisation. Capsicum's mitigation was further improved through more fine-grained compartmentalisation in CHERI [3, Sec. II].

2.2 Backward compatibilities in capability systems are tricky

One of the primary obstacles and sources of vulnerability in capability systems is the need to support legacy and existing applications through backward compatibility. The ability of renderer developers to call existing Java libraries from within DarpaBrowser [1, 5.2] introduced many breakout vulnerabilities into the capability system, due to insufficient suppression and filtering on Java SDK's vast interfaces. Despite requiring application modifications, Capsicum [2, Sec. 4] sought to minimise adaptation efforts by offering two modes of integration with varying capability implications. As a hardware-software hybrid capability system, CHERI [3, Sec. IV] provided specialised compiler and kernel to allow convenient mapping of C-language constructs to CHERI-compatible representations.

2.3 Choices and impacts in layer of capability system implementation

Vertically, capability systems presented in the three papers represented radically different approaches in choosing the insertion layer of capability implementation, with varied positive and negative impacts. DarpaBrowser [1] is language-based and operates on the application-level only, failing to anticipate vulnerable system calls available from the connected Java SDK. Capsicum [2] is primarily OS-based, offering low adaptation complexity but cannot support un-sandboxed applications requiring on-demand initialisation very well. CHERI [3] however offers a hybrid language and OS-based full stack capability solution, addressing a lot of drawbacks encountered by DarpaBrowser and Capsicum.

3 Ideas in current context

Wagner and Tribble [1, Sec. 3] speculated the dangers of browsers leaking authorities to a renderer exposed to the insecure web. This has since been validated in practice, as demonstrated in iOS 4.3's FreeType parser vulnerability, enabling a PDF downloaded by the web browser to gain arbitrary code execution privileges on the system and enable an iOS jailbreak through a kernel vulnerability [4]. Concurrency-related vulnerabilities in capability systems as discussed in the papers have also enabled a more recent jailbreak [5].

In order to reduce adaptation complexity, Capsicum [2] avoided adopting a full message-passing model in compartmentalisation. Watson et al. however noted that mask rights can be further simplified in a pure message-passing capability system. This was realised in the design of Unikernels [6], which enforced type-safety through fully compartmentalising OCaml applications on the Xen hypervisor, with message-passing as the only form of communication. This approach of Unikernels however severely increases adaptation complexity.

Capability systems are vulnerable to be the single point of failure as part of a complex system, preventing regular applications from functioning should they become faulty. This distinct risk was exhibited but not directly addressed in any of the papers [1, Sec. 4.2] [2, 5.5] [3, III. B.]. However, it may be possible to address this problem through replicating capability systems and operating in conjunction with a consensus system such as ZooKeeper [7], especially in multi-system environments.

4 Literature review

In analysing DarpaBrowser, Wagner and Tribble [1, 2.1] referred to the capability-based browser's protection against malicious renderers as a confinement mechanism, which was achieved through the elimination of reachability [1, 4.1]. This was further shown to be effectively enforceable by Miller et al. [8]. Miller and Shapiro [9] also extended DarpaBrowser's capability model into an abstraction for access control systems. DarpaBrowser's language-based capability control on Java was also an inspiration for a recent compartmentalisation reasoning tool by Gudka et al. [10].

In addition to providing ideas and performance references for the later CHERI [11, 3], Capsicum also inspired a direction of future work in enabling permission dropping in an Android-based capability system by Backes et al. [12]. Capsicum's language-dependent capability abstraction was described as lacking in portability for Ansel and Marchenko's [13] language-independent software sandboxing design. It was also suggested as a possible OS protection module for the two-way sandboxing system by Li et al. [14].

In developing a control-hijacking attack against control flow integrity-checking systems, Evans et al. [12] noted CHERI's capability-based tagging as a possible mitigation that could achieve complete memory safety. This was also suggested in a related research by Evans et al. [15] in their attack against code pointer integrity-checking systems. CHERI's capability-compatible C implementation was also used as a testing platform for the formal model of C by Memarian et al. [16], identifying several possible design and implementation drawbacks in CHERI. A light-weight context-based OS abstraction system for privilege separation was developed by Litton et al. [17] as an alternative to CHERI.

(1174 words according to texcount.)

References

[1] D. Wagner and D. Tribble, "A security analysis of the combex darpabrowser architecture," 2002.

[2] R. N. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical capabilities for unix." in *USENIX Security Symposium*, vol. 46, 2010, p. 2.

[3] R. N. Watson, J. Woodruff, P. G. Neumann, S. W. Moore, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie *et al.*, "CHERI: A hybrid capability-system architecture for scalable software compartmentalization," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 20–37.

[4] NIST, "Cve-2011-0226," October 2011. [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0226

[5] I. Beer, "XNU kernel UaF due to lack of locking in set_dp_control_port," October 2016. [Online]. Available: https://bugs.chromium.org/p/project-zero/issues/detail?id=965

[6] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *Acm Sigplan Notices*, vol. 48, no. 4. ACM, 2013, pp. 461–472.

[7] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX annual technical conference*, vol. 8, no. 9. Boston, MA, USA, 2010.

[8] M. S. Miller, K.-P. Yee, J. Shapiro *et al.*, "Capability myths demolished," Technical Report SRL2003-02, Johns Hopkins University Systems Research Laboratory, 2003. http://www. erights. org/elib/capability/duals, Tech. Rep., 2003.

[9] M. S. Miller and J. S. Shapiro, "Paradigm regained: Abstraction mechanisms for access control," in *Annual Asian Computing Science Conference*. Springer, 2003, pp. 224–242.

[10] K. Gudka, R. N. Watson, J. Anderson, D. Chisnall, B. Davis, B. Laurie, I. Marinos, P. G. Neumann, and A. Richardson, "Clean application compartmentalization with soaap," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1016–1031.

[11] J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe, "The cheri capability model: Revisiting risc in an age of risk," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3. IEEE Press, 2014, pp. 457–468.

[12] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. v. Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock android," 2015.

[13] J. Ansel, P. Marchenko, Ú. Erlingsson, E. Taylor, B. Chen, D. L. Schuff, D. Sehr, C. L. Biffle, and B. Yee, "Language-independent sandboxing of just-in-time compilation and self-modifying code," in *ACM SIGPLAN Notices*, vol. 46, no. 6. ACM, 2011, pp. 355–366.

[14] Y. Li, J. M. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry, "Minibox: A two-way sandbox for x86 native code." in *USENIX Annual Technical Conference*, 2014, pp. 409–420.

[15] I. Evans, S. Fingeret, J. Gonzalez, U. Otgonbaatar, T. Tang, H. Shrobe, S. Sidiroglou-Douskos, M. Rinard, and H. Okhravi, "Missing the point (er): On the effectiveness of code pointer integrity," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 781–796.

[16] K. Memarian, J. Matthiesen, J. Lingard, K. Nienhuis, D. Chisnall, R. N. Watson, and P. Sewell, "Into the depths of c: elaborating the de facto standards," in *ACM SIGPLAN Notices*, vol. 51, no. 6. ACM, 2016, pp. 1–15.

[17] J. Litton, A. Vahldiek-Oberwagner, E. Elnikety, D. Garg, B. Bhattacharjee, and P. Druschel, "Light-weight contexts: An os abstraction for safety and performance." in *OSDI*, 2016, pp. 49–64.