Electronic Communications of the EASST
Volume 080 (2021)

Conference on Networked Systems 2021 (NetSys 2021)

Polymorphic Protocols for Fighting Bots

August See

5 pages

Guest Editors: Andreas Blenk, Mathias Fischer, Stefan Fischer, Horst Hellbrueck, Oliver Hohlfeld, Andreas Kassler, Koojana Kuladinithi, Winfried Lamersdorf, Olaf Landsiedel, Andreas Timm-Giel, Alexey Vinel
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Polymorphic Protocols for Fighting Bots

August See

richard.august.see@uni-hamburg.de
Universität Hamburg, Germany

Abstract: Web Robots (bots) that automate communication with a service on the Internet via their API are efficient and easy to scale. A large number of bots leads to significant losses for providers and can frustrate users of social media, games or online stores. Existing solutions such as CAPTCHAs or complex registrations either frustrate users or are easy to circumvent. Current solutions that make it difficult to create bots are only effective for the first bot. Once the first bot is created, it can be easily duplicated to build an army of bots. This paper presents an approach inspired by polymorphic malware and censorship resistance to change this. Each client that communicates with a service does so by using its own application protocol that is syntactically different but not semantically. Thus, a bot creator is forced to either find a way to automatically extract the whole application protocol from a client or to reverse engineer a new protocol for each bot that is created.

Keywords: bots, protocol obfuscation, reverse engineering

1 Introduction

Services on the Internet are more and more interconnected. They communicate and exchange data. For example, price comparison portals automatically retrieve data from a number of different third party services. For other services, like social media, or games, an automated usage is often not wanted, since this might affect the satisfaction of human users. In online dating for example, probably no one wants to chat and meet with an automated program.

There are several ways to automate [ASLN20]. One approach is to automate the use of the user interface where a bot presses buttons or writes in text fields just like the user. This approach is inefficient because each new bot needs its own user interface. This is especially expensive for applications that run natively (desktop or mobile) and do not have a web application. When the applications can not be run in parallel even a new instance of the operating system needs to be launched. A much more efficient way is to communicate directly with the service via its API, sticking to the application protocol. Here a simple script is enough. Neither a user interface is needed nor a specific operating system or a browser instance. The script can simply be executed multiple times to spawn a multitude of bots e.g., for nation wide spamming and influencing voting opinions in social media [BF16]. This is why this paper focuses on automation via the API of a service.

The main problem of internet services is that automation cannot be prevented, it can only be made more difficult. If a human user can use a service, it can also be automated. Currently mainly CAPTCHAs coupled with risk assessment approaches are used to combat bots. Depending on the calculated risk that a user is a bot, a more difficult CAPTCHA to no CAPTCHA at all is

presented [Liu18]. However, the effectiveness of CAPTCHAs in general decreases with the advancement in machine learning [SPK16, AA20]. Another method to combat bots is to make the creation as difficult as possible. This can be done by using obfuscation or anti-reverse engineering techniques. These techniques increase the cost required to create one initial bot. However they do not affect the cost required to simply duplicate the first initial bot. Simultaneously, this is what makes bots so dangerous: the ability to build many bots quickly and cost-effectively.

The main contribution of this paper is a novel approach that increases the labor and computational cost of duplicating a bot. This is done by using so-called polymorphic protocols which are inspired by polymorphic malware and censorship resistance. The idea is that each client that communicates with a service does so with its own application protocol. The protocol can be considered as an identifier to distinguish between clients. This effectively eliminates the convenient way to simply duplicate a bot. The duplicated bots would all share the same protocol and can therefore all be detected together and excluded from further communication with the service. This increases the cost of bot creators to create multiple different bots. Bot creators must now either:

1. Manually extract the respective protocol from clients again and again. Ideally the effort required for this is close to the effort to build the first bot.

2. Find a way to automatically extract the respective protocol from clients.

It is important to note that the approach is not about making it harder to create a bot or even to make an application or service bot secure. It is a proactive obfuscation approach that restricts the cost-effective duplication of bots. Interoperability between different services or legitimate bots can still be ensured e.g., by giving the creators an API key after thorough verification.

The rest of the paper is structured as follows. Section 2 discusses other approaches that make it harder to create bots. Section 3 explains how polymorphic protocols can be created and applied, and proposes possible research questions. Finally, Section 4 concludes the paper.

2 Related Work

The word polymorphism comes from biology and refers to different forms (morphs) within the population of one species [HGE76]. In the field of computer science this term is often used in the area of computer malware that evades signature detection due to that each instance of the same malware is different in its form [YY10]. Polymorphic protocols are mostly mentioned in literature in connection with censorship resistance and privacy [WDA+15]. A similar idea to polymorphic protocols is used by the authors of [RS10] which aim to achieve less shared vulnerabilities in server replicas by transforming the code while keeping the semantic.

Depending on the application there are different ways to make the creation of a bot more difficult. A common way is that some sort of identification is required for interacting with the service e.g., by registering. Depending on the information required for registering, this process is more or less hindering the creation of bots. A registration where only some username and password is required is less secure than a form where in addition a valid Email is required, which is less secure than when a valid phone number is required or even a valid document e.g., a passport. However, a registration needs time and may disturb user experience [GN16]. Furthermore, not

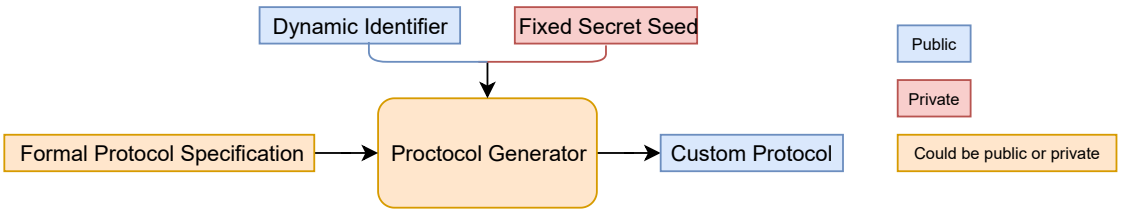

Figure 1: Polymorphic Protocol Generation Overview

every user might be willing to share a phone number or even documents with some service. Another way is to make the protocol itself more resilient to analysis. A common way is to encrypt and authenticate the protocol in addition to TLS[1]. When reconstructing the protocol one does not only need to know the specification of the protocol, but also the application key(s) to encrypt and authenticate messages. Those application keys are then protected, using anti-reverse engineering techniques, like obfuscation [GTG07]. However, the protection is not ideal. On a controlled device those techniques only increase the cost of the bot creator in obtaining the key but they do not prevent it. Even if the key would be different and hidden in different parts for each application, at some point the key has to be used. One of these points is when a message needs to be encrypted or authenticated. There the key can be extracted automatically e.g., by using hooks. Than a bot can be duplicated again without much cost.

3 Polymorphic Protocols

The objective is to limit the cost-effective duplication of bots. This is achieved by using a so-called polymorphic protocol. The creation of one bot is as difficult as without a polymorphic protocol. However, the creation of further bots should then be similarly difficult. The main assumption is that compared to automatically extracting application keys, the extraction of a whole communication protocol is a much greater effort. This assumption is supported by literature [NSC15, NBFS06, LJLW13] but not confirmed. A polymorphic protocol could also encrypt messages. As a result the automatic extraction of such a protocol can become at least as difficult as the extraction of an application key. The challenge is to estimate how much more demanding it is.

3.1 Functionality and Approach

The basic functionality is shown in Figure 1. The *Protocol Generator* takes a *Formal Protocol Specification* as well as a *Dynamic Identifier* and *Fixed Secret Seed* as input, and outputs a *Custom Protocol*, i.e., bound to the *Dynamic Identifier*. The *Dynamic Identifier* can be a device id, a username or some nonce, depending on the application. The *Fixed Secret Seed* is there so that if all components are public, it is not possible to create a valid custom protocol without knowing this Secret Seed. The *Formal Protocol Specification* is needed for the *Protocol Generator* to randomize the protocol, while making sure that any mandatory semantic dependencies of messages

[1] https://github.com/see-aestas/SINoALICE-API, https://github.com/see-aestas/JodelApi

are preserved. The *Protocol Generator* can randomize the order, transmission method, encoding and other features of the protocol or even inject extra dummy bytes, creating a custom protocol. The *Custom Protocol* is then merged with a client application e.g., website, smartphone app, or desktop app. This means that each client has a different protocol for communicating with the same service. A challenge for this usage is to ensure that a custom protocol can not easily be extracted from a binary, in contrast to the usage of polymorphic protocols in censorship resistance. A combination with additional encryption on the custom protocol and the use of application packers like UPX[2] can amplify the effort of extracting a protocol from an application.

Depending on the application, a custom protocol can be assigned once to each client, or the same protocol to a group, or a new protocol every other day. However, this requires significant effort in terms of computing time and transmission.

3.2 Arising Research Questions

Possible research questions are:

1. How much additional cost does the use of polymorphic protocols cause for an attacker who wants to set up multiple bots for a service?

2. How much additional cost does the use of polymorphic protocols cause for a service provider and how does the approach scale?

The evaluation should include a comparison with methods where only an application key is used to secure the protocol against bots. However, evaluating the first research question is challenging in some aspects. It is hard to quantify the cost of extracting a protocol (automatically) for a generic attacker who could use any method. One possibility is to examine this question argumentatively on the basis of empirical research in this area. Another possibility would be to exemplarily apply different automatic protocol extraction techniques on a client with polymorphic protocol.
The second question is more straightforward to evaluate. Aspects such as application size, data consumption, time and resources to create a custom protocol in comparison to non-polymorphic protocol can be quantified. It is also possible to investigate which common technologies can only be used to a limited extent due to the use of polymorphic protocols e.g., CDNs.

4 Conclusion

This paper presents an approach to make it harder to duplicate bots. It adopts the technique of syntactically modification while maintaining the semantics of polymorphic malware and censorship resistance. The basic idea of the approach is to equip each application with a syntactic different protocol. This obfuscation makes it very expensive for a bot creator to assemble an army of bots because it is either necessary to find a way which extracts the protocol automatically or to extract each protocol manually.
In future work such a polymorphic protocol will be implemented and evaluated. In addition, it will be tested whether the findings can be applied to censorship resistance.

[2] https://upx.github.io/

Bibliography

[AA20] F. H. Alqahtani, F. A. Alsulaiman. Is image-based CAPTCHA secure against attacks based on machine learning? An experimental study. *Computers & Security* 88:101635, 2020.

[ASLN20] B. Amin Azad, O. Starov, P. Laperdrix, N. Nikiforakis. Web runner 2049: Evaluating third-party anti-bot services. In *Proceedings of the 17th DIMVA*. 2020.

[BF16] A. Bessi, E. Ferrara. Social bots distort the 2016 US Presidential election online discussion. *First monday* 21(11-7), 2016.

[GN16] R. Gafni, I. Nagar. CAPTCHA–Security affecting user experience. *IISIT* 13:063–077, 2016.

[GTG07] M. N. Gagnon, S. Taylor, A. K. Ghosh. Software protection through anti-debugging. *IEEE Security & Privacy* 5(3):82–84, 2007.

[HGE76] P. W. Hedrick, M. E. Ginevan, E. P. Ewing. Genetic polymorphism in heterogeneous environments. *Annual review of Ecology and Systematics* 7(1):1–32, 1976.

[Liu18] W. Liu. Introducing reCAPTCHA v3: the new way to stop bots. https://developers.google.com/search/blog/2018/10/introducing-recaptcha-v3-new-way-to, 2018. Accessed: 2021-05-20.

[LJLW13] M. Liu, C. Jia, L. Liu, Z. Wang. Extracting sent message formats from executables using backward slicing. In *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*. Pp. 377–384. 2013.

[NBFS06] J. Newsome, D. Brumley, J. Franklin, D. Song. Replayer: Automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and communications security*. Pp. 311–321. 2006.

[NSC15] J. Narayan, S. K. Shukla, T. C. Clancy. A survey of automatic protocol reverse engineering tools. *CSUR* 48(3):1–26, 2015.

[RS10] T. Roeder, F. B. Schneider. Proactive obfuscation. *TOCS* 28(2):1–54, 2010.

[SPK16] S. Sivakorn, I. Polakis, A. D. Keromytis. I am robot:(deep) learning to break semantic image captchas. In *2016 IEEE EuroS&P*. Pp. 388–403. 2016.

[WDA+15] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, T. Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Pp. 57–69. 2015.

[YY10] I. You, K. Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*. Pp. 297–300. 2010.