Contents

1 Executive Summary **1**

2 Innovative Claims and Deliverables **1**

3 Running Example: Countering Cyberadversary Systems **2**

4 Related Work **2**

5 Technical Plan: System Overview **2**
5.1 R1. Lifting to Retargetable IR (RIR) 2
5.2 R2. Semantic-Preserving Transformation 3
5.3 R3. Refining Decompilation Results for μpatch Placement 4

6 Technical Plan: TA1/TA2 Integration **4**

7 Evaluation Plan **4**
7.1 Evaluation Metrics 4
7.2 Success Criteria 5

8 Management Plan **5**
8.1 Team Organization 5
8.2 Risk-Management Plan 6

9 Personnel, Qualifications, and Commitments **7**

10 Capabilities **9**

11 Statement of Work (SOW) **10**
11.1 Phase 1: Design (16 months) 10
11.2 Phase 2: Development (16 months) 13
11.3 Phase 3: Integration (16 months) 16

12 Schedule and Milestones **18**

Executive Summary

PRESERVE aims to provide a foundational ecosystem that is suitable for μpatching. μpatching is when a security patch is formulated in a different context from its original source code or binary, but is still applicable with sufficient theoretical or empirical confidence. Since the current practice demands intensive human involvement—manually retrofitting a μpatch from one system to a different, yet similar, environment, PRESERVE strives to automate the underpinnings of μpatching practices, thereby reducing the time required for common practices from months to a few hours. In particular, PRESERVE focuses on developing three foundational techniques: 1) *retargetable lifting*, which can guarantee the lifted low-level IR always has an isomorphic transformation to the original binary; 2) *semantic-preserving transformation*, which provides a contextual representation of low-level IR in a resilient form for searching patching targets; and 3) *refinement techniques*, which iteratively refine the decompilation output to be closer to the original output based on known hints, such as code snippets or build environments. The proposed technique is different from traditional lifting (e.g., McSema, LibVEX), which generates a radically different binary to support a variety of code-level changes, whereas ours enables *retargetable* transformation for a security patch; and it is different from existing decompilation (e.g., Hex-Rays, Ghidra), which aims to produce readable output to human analysts whereas ours aims to produce *formal, semantic-preserving* output for automation. The foundation PRESERVE builds can not only situate the μpatch in a precise manner (TA1) but also assist TA2 in collecting enough assurance to formulate the μpatch and applying it to a different binary in a seamless manner.

The proposed team has an extensive background in the underlying technologies of RENAISSANCE, namely, vulnerabilities analysis, exploit generation, botnet detection and response, malware analysis, intrusion detection, and formal methods. The PIs have a long history of close collaboration with joint projects, co-authored papers, and co-advised Ph.D. students. All of the PIs are supported by academic institutions and will release all software developed for RENAISSANCE under an academic, open-source license.

The proposed project will cost $7,470,661 over the three phases in a total of 48 months.

Innovative Claims and Deliverables

- **TA2.** A new approach to repeatedly enhance, develop, and validate N-day exploits for open-source and closed-source userspace applications by using public information from multiple online resources and using the environment replication framework (**??**).
- **TA2.** Automatic techniques for generating robust (i.e., resilient) and safe (i.e., no side-effect) exploits against open-source and binary-only OSes using security patches (i.e., code commits or binary patches) and crash PoCs (**??**). The generated exploits by design are formulated to bypass the state-of-the-art defenses like KASLR, SMEP and SMAP (**??**, **??**).
- **TA3.** A novel approach to synthesize a botnet agent based on the extracted, cloned malware sample and its techniques to analyze and neutralize the malicious behaviors (**??**, **??**).
- **TA3.** New formal techniques for verifying operational characteristics of the synthesized agent (i.e., cloned malware) (**??**).

Deliverables. PRESERVE will be developed in an open manner; the entire project will be open sourced (i.e., technical documents and evaluation) and maintained in a public repository. We will deliver monthly financial status reports and a yearly status summary according to the BAA.

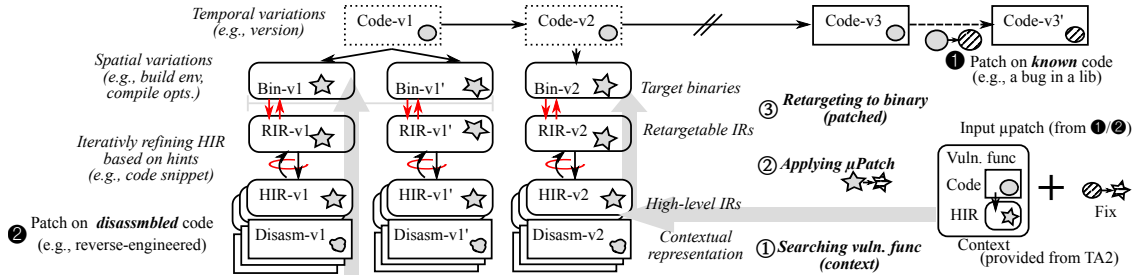

Figure 1: Design and workflow. Given ❶/❷, PRESERVE situates the proper patch candidates by ① lifting the target binary to RIR (§5.1), ② transforming RIR to HIR (§5.2), and ③ iteratively refining its representation (§5.3).

Running Example: Countering Cyberadversary Systems

In this section, we study how severe the problems of modern cyberadversary systems (e.g., botnets) are in terms of their size, purpose, and evasion techniques (**??**). As an in-depth case study, we explain how much effort was needed to take down a recent large-scale botnet, Mirai, and the limitations of such take-down approaches (**??**). Last, we define the major challenges of developing a systematic approach to disarm cyberadversary systems (**??**).

Related Work

XXX: hong: 1pg

Technical Plan: System Overview

Overview. Given a μpatch—including its surrounding context and fix, we would like to search for appropriate candidates (i.e., vulnerable code) to be repaired. We assume the μpatch is formulated in a certain context (e.g., against known source code or against a decompilation result during reverse-engineering of a similar binary), which is similar but not identical to the targeted binaries to be patched. The targeted binaries can be similar in two senses: 1) the original source code is the same but compiled by different compilers with different optimizations enabled, 2) each source code has evolved over time but contains the same bug, or perhaps mixes of 1) and 2). It is worth emphasizing that the targeted binaries and source code may not be completely one-to-one; source may be just a small part of the binary (e.g., statically linked yet common libraries).

Given a μpatch and a target binary, PRESERVE searches the appropriate patching candidates in three steps: 1) lifting the binary to the retargetable IR (RIR) (§5.1 R1), 2) transforming the RIR to a high-level IR (HIR) while respecting its original semantics in the binary (§5.2 R2), and 3) incrementally refining the HIR to be closer to the original form based on known hints such as a tuple of matched code snippet and binary output or the faithful build environment (§5.3 R3). PRESERVE extracts such hints in either automatic ways (e.g,. inferring applied optimization passes), or manually from reverse-engineers or TA2 (e.g., linking a part of a binary to the known source code). Once PRESERVE locates μpatch's surrounding context in the decompilation output, the fix finally applies to the HIR while performing the formal reasoning on its changes (by TA2). PRESERVE enabled retargeting back to RIR and the original binary.

R1. Lifting to Retargetable IR (RIR)

TODO: lead: insu/kyhong, 4pg

Traditional IRs that are widely used for decompilation are not suitable for μpatching as they relax the semantics of the original binary—otherwise, the decompiled output is not really helpful for analysts. Such lossy transformation is not ideal for PRESERVE as it has to preserve existing binary as much as possible to maintain its original functionality. For example, both `[inc eax]` and `[add eax, 1]` instructions in x86 would be lifted to a single IR, but its semantics are profoundly different—`inc` preserves the `CF` flag while `add` updates the flag, allowing `inc` to be used as a loop counter. Another good example is control-flow diverging. To produce readable output, a decompiler often splits or removes basic blocks to reverse popular compiler optimization during IR lifting. Moreover, CISC architectures include instructions that have additional control flow (e.g., cmp-and-swap or str cmp). These operations make control-flow diverged, complicating the IR lifting.

To minimize modification during μpatching, we define *Retargetable IRs (RIRs)* that extend LLVM IRs with additional metadata for lowering. RIRs contain machine-dependent instructions and locations of the original binary in the metadata. Lowering unmodified RIRs is reasonably straightforward; PRESERVE can emit the original instructions to the original locations. This design can guarantee isomorphism between RIRs and the original binary, i.e., lowering RIRs produces precisely the same binary. Moreover, such additional information would as well be leveraged during μpatching. For instance, if a patch requires moving an instruction, PRESERVE can preserve the original structure from RIRs and only adjust locations, resulting in small offset changes. When retargeting the code, RIRs can also bound its impact by analyzing the corresponding location of the binary. For example, if the modified code is 1-byte smaller than the original, PRESERVE can prefer to add 1-byte `nop` instead of re-locating every instruction based on the location of the next instruction. To improve the quality of RIRs, PRESERVE employs compiler optimizations, but only those that maintain the isomorphism such as in-block dead code elimination. PRESERVE also uses the metadata to bridge semantic gaps between the original and the patched code; enforcing the use of specific machine registers for correct data-flow after patching.

R2. Semantic-Preserving Transformation

XXX: lead: hong/insu, 4pg

The next step is to transform RIR to our high-level IR, called HIR, while *preserving the semantics* of the binary. The design goal of HIR is to match the structure of the available source code, which could be fully available or partially available (*e.g.*, only the vulnerable function), so that we can situate a security patch. HIR does not keep the low-level information of the binary and represents the program with a more neat format and high-level structural information. However, it is functionally equivalent to the corresponding RIR and the binary.

We can apply any traditional transformations that are widely used by current decompilers to simplify the RIR, as long as the transformed IR preserves the original semantics. For example, dead-code elimination will remove completely unnecessary instructions based on the data-flow analysis to reduce the size of the IR. This is essential to the decompilation as the RIR is designed to contains verbose, but likely unnecessary instructions. For example, all side effects of arithmetic operations (*e.g.*, `add`) on the flag register (*i.e.*, `eflags` or `rflags`) are conservatively made explicit and kept in the RIR, but most of them are not used in any following operations. Removing them makes the code more readable and easier to analyze. These transformations also construct the control-flow and data-flow of the IR to facilitate other analyses, like SSA form generation.

Different from existing decompilers, any destructive transformations are prohibited, as they may

lead to divergent functionality and violate the goal of AMP. For example, existing decompilers infer the number of function arguments with principle-based calling conventions and best-effort heuristics, which may find fewer arguments than the ground truth if some arguments are not used in the current function. When we compile the IR back into the binary, the incorrect number of arguments will affect the register allocation algorithm and may overwrite registers that hold the value of missed arguments. In any case where we cannot accurately determine the information, we should label the result as best-effort and not use it for other analysis.

R3. Refining Decompilation Results for μpatch Placement

XXX: lead:daehee/hong, 0.75pg

The last step is to refine the transformed HIR in order to match it with the source code to which the μpatch applies. The challenge is that the binary, which the HIR originates from, has been highly optimized and reconstructed from its source code during the compilation. Moreover, the source code could have been updated and differ from the source used to generate the binary. Directly comparing the structures (*e.g.*, control-flow) of two parties will likely fail to match. For instance, inlining functions preserves the original semantics but changes the control-flow significantly.

To this end, we *iteratively* refine the HIR towards structures similar to the source code, with the help of any available supplementary information. First, we try to match the current version of HIR with the source code with our best-effort attempt. From the imperfect matching results, we identify the most similar components between two parties, like functions or basic blocks. Then, we adjust the structural information of these components and try to make them be as close to the source code as possible. We can use both semantics-preserving and heuristic-based transformations in order to achieve the match quickly. When applying heuristic-based transformations, we define the confidence of functionality equivalence to measure the divergence between the refined HIR and the original version. Meanwhile, we measure the distance between the refined IR and the source code to identify the benefits of each transformation on converging two parties. Such quantitative measurements will help us find the optimal method to refine the HIR so that we can match it with the source code with high confidence. After each refinement, we go back to the first step to do another matching attempt until either we cannot find ways to improve the matching results or the confidence of functionality equivalence is too low. The matching process will benefit by utilizing supplementary information. For example, if the build script defines the threshold of inline function size, we can identify common, small-enough code snippets from the HIR and create new functions, effectively re-inlining the HIR. Once we match the HIR level functions with the source code, we will focus on these functions for applying the μpatch.

Technical Plan: TA1/TA2 Integration

TODO: lead: kennon, 1pg

Evaluation Plan

TODO: lead: kevin, port r4 here.

Evaluation Metrics

Specifically, we plan to evaluate RENAISSANCE following the below metrics:

TA2. Coverage of N-day exploits. To show that PRESERVE can support a wide range of applications, services, and operating systems (including open source and commercial software), we will first survey target software-deployed emerging systems, including IoT devices and cloud environments that were previously targeted by recent botnets (e.g., Mirai, Petya, etc), and demonstrate that PRESERVE can automatically formulate exploits or any form of penetration against them.

TA2. Robustness, safety and quality of N-day exploits. To show that PRESERVE's automatic exploit generation can scale without compromising the robustness, safety and quality of exploits, we will evaluate the generated exploits on the systems with various modern mitigation in applications and kernel, including DEP, ASLR, SMA/EP, Sandboxing, and Web Application Firewall, etc.

TA2. Performance of testing, validation framework. To be agile in reacting to malware, PRESERVE provides a fast mechanism to replicate environments for testing and validation. We will measure its performance and characteristics with micro-benchmark and end-to-end testing cases using recent botnet samples.

TA3. Effective malware analysis. To synthesize agents from malware, XANALYST will attempt to discover control paths that can be executed by the original malware. We will evaluate the ability of XANALYST to efficiently discover a set of instructions that can be executed by the original malware that is sufficiently large to support the implementation of the synthesized agent.

TA3. Effective sandboxing mechanism. XAGENT will attempt to mitigate the side-effects of a given malware by synthesizing agents that each maintain a copy-on-write sandbox. We will evaluate whether agents synthesized by XAGENT can maintain such sandboxes efficiently, and whether such sandboxes preserve sufficient functionality of the original malware for the agent to be effective.

TA3. Verification efforts and techniques. XAGENT will synthesize agents whose design enables verification as a combination of manually-verified, correct-by-construction libraries and automatically-verified monitoring properties. We will evaluate the ability of the design to integrate functions that can be manually verified with a reasonable amount of effort, and we will evaluate the ability of XAGENT's analysis to verify monitoring properties automatically and efficiently.

Success Criteria

We will consider our effort with RENAISSANCE to be successful if

- The answers to all questions under the evaluation (see §7.1) are "yes";
- **TA2.** PRESERVE automatically generates robust, safe, and high-quality exploits against N-day vulnerabilities of emerging systems, including Cloud and IoT, that botnets are mainly targeting;
- **TA3.** XAGENT correctly clones and safely neutralizes the modern malware binaries and synthesized each of them as an autonomous agent with required operational features but without disturbing the behavior of the cyberadversary system.

Management Plan

Team Organization

PI. Kim at Georgia Tech will act as the PI for the entire project and as the point-of-contact. Table 1 lists all the team members and their expertise relevant to the DARPA HACCS program. The PI and co-PIs from Georgia Tech are multidisciplinary across various areas in computer science, namely, operating systems, system/network security, and programming analysis with a solid focus

Institution	Team member	Expertise
Georgia Tech	Taesoo Kim (PI)	Systems security, Operating systems
	William R. Harris (co-PI)	Program verification, Program language security
	Brendan Saltaformaggio (co-PI)	Cyber forensics, Malware analysis
	Wenke Lee (co-PI)	Malware detection/analysis, Network security
Columbia University	Salvatore Stolfo (PI)	Data-driven botnet identification, IDS systems
Oregon State University	Yeongjin Jang (PI)	Offensive security, Penetration testing

Table 1: Team members and their expertise areas relevant to the HACCS program.

on security. Both PIs from Oregon State University and Columbia University are leading the development of the testing and validation framework (TA2) and bringing expertise in real malware binaries and analysis experiences of leading the IDS industry (TA3), respectively.

Coordination plan. We will coordinate among the various team members using three mechanisms: joint meetings, jointly supervised students and postdocs, and shared code repositories.

- **Joint meetings.** All team members will attend the HACCS PI meetings and contribute to the monthly reports. We plan to have biweekly project meetings with the PIs at Georgia Tech and monthly project meetings with all universities together.
- **Joint supervision.** Drs. Kim, Harris, Lee, and Jang continue to supervise graduate students and postdocs jointly and run a weekly group meeting together. The project team has already started collaborating on some of the research problems related to this proposal; in particular, PI. Kim and PI. Jang designed and developed an automatic exploit generation engine to participate in DARPA Cyber Grand Challenge (CGC) and DEFCON CTF last year.
- **Shared, central code management.** We have a central GIT repository at IISP to host resources, including reference papers, design documents, source code, and test cases.
- **Engineering task force.** A full-time programmer (50%) and a postdoc will contribute to this project to produce a robust, stable version of the prototype.

Risk-Management Plan

TODO: Add to this and write up. Some may move to technaicl sections instead of risk

- !!! Destructive transforms may become necessary (called out in abstract feedback)
- Inability to converge on a matching decompilation
- Ground truth issues (disassembly, control flow recovery, points-to analysis) - for disassembly, we can cite the probabilistic disassembly paper
- Little/no source of comparison (i.e. original source code)
- Multi-threaded data flow analysis
- Distinguishing memory mapped device IO and analyzing it properly (more generally, challenges of lifting an embedded image vs. a userspace program)

This section discusses the potential risks that we foresee and their impact on the proposed work.

Risk-1: Insufficient or inaccurate vulnerability data for exploit generation (Task-TA2)

We estimate the impact of this risk to be *low*. As PRESERVE relies on publicly available vulnerability information, a potential risk might be that PRESERVE requires a targeted exploit for which no or only inaccurate vulnerability information exists. Our preliminary investigation (e.g., **??**) shows that systems often match hundreds or thousands of distinct CVEs. We expect that when an exploit cannot be generated for any single vulnerability, PRESERVE can find an alternate or even combine several exploits to accomplish the same goal.

Risk-2: Scalability of binary analysis techniques (Task-TA2/Task-TA3)

We estimate the risk of this to be *moderate*. PRESERVE performs offline binary analysis both to generate and verify exploits. However, the symbolic techniques we leverage may suffer from state explosion, and dynamic or fuzzing-based techniques may require traversing thousands of unimportant control paths. We have designed PRESERVE to leverage the strengths of one analysis to avoid the weaknesses of another. For example, our hybrid fuzzer complements fuzzing with symbolic analysis. If scalability becomes a serious issue, we may adopt techniques to reduce our analysis via models of select system components via manual analysis.

When applying binary analysis to synthesize an agent, even if many control paths are only discovered after deploying an agent, these discoveries do not cause our approach to fail; they only cause XANALYST to spend a proportionate amount of time to synthesize a new variant of a given malware.

Risk 3: Failure to mitigate malware side effects and ensure effectiveness (Task-TA3)

We believe that the degree of this risk is *low*. It is feasible that the copy-on-write sandbox implemented by agents will break critical functionality of the malware that must be reused by the agent to be effective. However, our choice to design such a sandbox is informed by our extensive previous work on collecting and analyzing malware, which indicates that such a sandbox will preserve critical functionality. We can mitigate such a risk by allowing a domain expert to implement manual sandbox *validators*, which will determine if particular writes to a filesystem may escape a sandbox without causing harm to a target system. XAGENT will automatically synthesize agents to consult given validators when implementing a sandbox.

Risk 4: Failure to verify synthesized agents (Task-TA3)

We believe that the degree of this risk is *moderate*. While verifying properties similar to control interposition has been the subject of previous work, our work will be the first to result in a verified library used in the design of a malware agent. Verifying that programs satisfy such a novel class of properties could lead to challenges that are difficult to anticipate. To mitigate such a risk, we will develop the TAB and libraries so that they satisfy *partial* security properties, even in cases in which particular libraries do not satisfy their specification. We will also complement verification efforts with rigorous, semi-automated testing techniques.

Personnel, Qualifications, and Commitments

Dr. Taesoo Kim

Qualifications. Dr. Taesoo Kim is an Assistant Professor of Computer Science at Georgia Tech and a Director of the Systems Software and Security Center (GTS3). He received his Ph.D. in Computer Science from MIT in 2014. His main focus of research is to design systems that have a

Key Individual	Project	Status	Hours on Project				
			2018	2019	2020	2021	2022
Taesoo Kim	**DARPA HACCS**	Proposed	173	347	347	347	173
	DARPA TC	Current	174	260	0	0	0
	LM Malware	Current	64	0	0	0	0
	ONR Debloat	Current	87	173	347	347	87
	ONR RHIMES	Current	130	260	260	0	0
	NSF SaTC	Current	43	43	43	43	0
	NSF TWC	Current	22	43	22	0	0
	NSF SFS	Current	22	43	43	43	0
	NSF CSR	Pending	43	87	87	87	43
	NSF CAREER	Pending	65	130	130	87	0
Wenke Lee	**DARPA HACCS**	Proposed	87	173	173	173	87
	ARO Multi	Current	87	87	0	0	0
	DARPA TC	Current	260	390	0	0	0
	LM Malware	Current	64	0	0	0	0
	ONR Adapt	Current	43	87	87	43	0
	ONR Debloat	Current	87	173	121	52	0
	NSF SaTC	Current	43	43	43	43	0
	NSF SFS	Current	26	52	52	52	0
William Harris	**DARPA HACCS**	Proposed	87	173	173	173	87
	ONR Debloat	Current	260	520	520	520	260
	SRC STARSS	Current	61	0	0	0	0
Brendan Saltaformaggio	**DARPA HACCS**	Proposed	87	173	173	173	87
	DARPA Eupalinos	Current	347	347	347	347	0
Yeongjin Jang	**DARPA HACCS**	Proposed	173	347	347	347	173
	ADD RTOS	Current	174	174	87	0	0
	Intel CRC	Current	0	0	87	0	0
Salvatore Stolfo	**DARPA HACCS**	Proposed	65	130	130	130	65
	ONR EMBED	Current	275	629	629	275	0
	ONR TCB	Current	275	629	629	275	0

Table 2: Time commitments of key individuals, the status of proposed, pending, and current projects, and the hours per year for 2018–2022, assuming a 40-hour work week.

secure yet comprehensive foundation with strong security guarantees.

Previous accomplishments. Dr. Kim's thesis work focused on building the foundation of "undo computing," which provides a new, retroactive way to respond to intrusions. He demonstrated the idea by building such systems in various settings, including operating systems [12], web applications [3], distributed web services [4] and OSes [13], and web framework [5]. His thesis became a foundation technology for a startup company, Nerati. His recent work on software hardening [15–19] and bug-finding tools [20, 26] found over 200 bugs in GNU Libc, Android, Firefox, and Linux kernel, which earned the 2015 Internet Defense Prize ($100K Prize). He also participated in the DARPA CGC as a team, Disekt.

Dr. Brendan Saltaformaggio

Qualifications. Dr. Brendan Saltaformaggio is an Assistant Professor in the School of Electrical and Computer Engineering at Georgia Tech, with a courtesy appointment to the School of Computer Science. His research focuses on memory image forensics, binary analysis and instrumentation, and the vetting of untrusted software. He received his Ph.D. in Computer Science in 2016 from Purdue University, where he was honored with two fellowships: the 2016 Symantec Research Labs Graduate Fellowship and the inaugural Emil Stefanov Memorial Fellowship in Computer Science.

Previous accomplishments. Dr. Saltaformaggio's research has introduced new paradigms in the investigation of advanced cyber crime [21–24] and the analysis and prevention of next-generation malware attacks [6, 14], particularly in mobile and IoT environments. His work in mobile app vetting [6] has led to the identification and removal of hundreds of privacy-violating iOS apps from the Apple App Store. He received a Best Paper Award from ACM CCS 2015 [23] and a Best Student Paper Award from USENIX Security 2014 [22]. His research has garnered acclaim from highly regarded media outlets such as the Stanford Cyber Initiative, The Register, NSF News, ACM TechNews, IEEE Electronics360, and Homeland Preparedness News.

Dr. Wenke Lee

Qualifications. Dr. Wenke Lee is a Professor of Computer Science at Georgia Tech and a Co-Director of the Institute for Information Security & Privacy Security (IISP). He received his Ph.D. in Computer Science from Columbia University in 1999. His research interests include systems and network security, applied cryptography, and data mining. In 2006, Dr. Lee co-founded Damballa, Inc., a company that focuses on botnet detection and mitigation, which originated from his lab.

Previous accomplishments. Dr. Lee has worked on malware analysis and botnet detection for more than 10 years, with oft-cited papers, e.g., [1, 2, 9, 10, 25], and widely used open-source tools, e.g., [7, 8]. Since 2016, Dr. Lee and his students have been developing a collaborative, cloud-based and virtualized malware analysis framework with the goal of enabling analysts to produce sharable, reusable and repeatable malware analysis results. The framework will serve as the foundation of XANALYST described in this proposal. Dr. Lee and his students have also started exploring techniques to create "safe copy" of a malware once its behaviors have been analyzed, and will use the lessons learned for the XAGENT work described in this proposal. Dr. Lee has been the PI of several large projects funded by DARPA, NSF, and ONR MURI. He has published over 100 articles in highly-refereed conference and prominent journals. He has won several best-paper awards as well as the NSF CAREER award in 2002.

Kennon Bittick

Qualifications.

Previous accomplishments.

Dr. Sukarno Mertoguno

Qualifications.

Previous accomplishments.

Dr. Michael Brown

Qualifications.

Previous accomplishments.

Capabilities

Georgia Tech will lead the project to provide the state-of-the-art general-purpose computing, storage, and networking facilities needed for this project and furnish sufficient office space for supporting students and engineers participating in the project. The Institute for Information Security & Privacy (IISP) has significant competency in administrating IT infrastructure. In particular, IISP administers tens of server racks and a dedicated router with a gigabit Internet connection. Combined with its

experience in collecting, storing, and disseminating large-scale threat intelligence, it can meet any need of computation, storage, and sharing required by this project.

Statement of Work (SOW)

We describe our proposed work in the following three phases: design (Phase 1), development (Phase 2), and integration (Phase 3). We use **U** to refer userspace exploit generation; **K** for kernel exploit generation; **E** for exploit environment building; **A** for analysis environment building; **S** for synthesizing agent; **V** for verifying agent operation; and **I** for TA2-TA3 integration.

Phase 1: Design (16 months)

In Phase 1, for the first year, we will focus on developing the core components of RENAISSANCE ready to demonstrate the idea of robust exploit generation schemes and efficient agent behavior analysis.

Task-U1: Collecting a toolkit of vulnerabilities and software (8 months)

Team: Brendan Saltaformaggio (coordinator) and Taesoo Kim

General description: We will build a framework for collecting publicly available N-day vulnerability data for a variety of OSS libraries and common closed-source binaries.

Detailed description: We will automatically build a database of vulnerabilities and software from the sources cited in **??**. We will identify and develop 10 or more N-day exploit instances from our collection that can be leveraged to demonstrate autonomous agent lateral movement.

Measurable evaluation: We will evaluate the performance and scalability of the framework to (1) support vulnerability collection for hundreds of thousands of OSS repos or binaries and (2) provide various hooks for TA3 agents to request targeted collections matching newly encountered binaries.

Deliverables: (1) PRESERVE collection framework. (2) Applicable exploit instances identified.

Task-U2: Detecting the vulnerabilities in the indexed software (8 months)

Team: Brendan Saltaformaggio (coordinator) and Taesoo Kim

General description: Develop a schema and analysis infrastructure for detecting the presence of the collected vulnerabilities in the indexed software/binaries.

Detailed description: In order to support scalable and accurate detection of exploit targets, we will build an analysis infrastructure that is able to (1) detect the inclusion of vulnerable OSS within a binary program and (2) accurately compare and tag vulnerable versions of CSS software.

Measurable evaluation: We will evaluate the accuracy of our analysis infrastructure on detecting OSS versions, partial-inclusions, and tagging CSS vulnerabilities. The accuracy should be above 80% (ideally 90%), with the only false negatives coming from incomplete or inaccurate information.

Deliverables: (1) Core analysis infrastructure. (2) Experiment results and evaluation dataset.

Task-K1: Collecting existing vulnerabilities and generating kernel PoCs (8 months)

Team: Taesoo Kim (coordinator) and Brendan Saltaformaggio

General description: Collect information for kernel vulnerabilities from public sources and build a hybrid fuzzer to generate kernel PoCs (a cornerstone for constructing exploits for a vulnerability).

Detailed description: We will collect information for kernel vulnerabilities from public sources. Based on existing kernel fuzzers and symbolic executors, we will build a kernel hybrid fuzzer that takes the collected information for vulnerability to guide the execution to reach vulnerable points.

Measurable evaluation: To evaluate the effectiveness of this system, we will measure (1) the number of vulnerabilities and their classes that we can cover, (2) the number of generated PoCs and code coverage of the fuzzer, and (3) elapsed time and required resource to generate PoCs.

Deliverables: (1) A vulnerability information collector. (2) A hybrid fuzzer framework. (3) Generated kernel PoCs as database. (4) Evaluation results of the combined system.

Task-K2: Developing full-fledged exploits for local privilege escalation (8 months)

Team: Taesoo Kim (coordinator) and Brendan Saltaformaggio

General description: Calibrate PoCs from Task-K1 by identifying vulnerability types and program constraints for exploitation to develop a corresponding exploit that guarantees privilege escalation.

Detailed description: We will analyze kernel PoCs to understand how the input affects the execution at the crashing point. We will identify the vulnerability types and program constraints on the PoC. We will build exploits through heuristic-based strategies, *e.g.*, code injection and ROP.

Measurable evaluation: From given PoCs of kernel vulnerabilities, we will show that PRESERVE can (1) calibrate the PoCs and control the context values at the crashing point and (2) develop exploit code to achieve essential exploit primitives (*e.g.*, arbitrary kernel memory access).

Deliverables: (1) A kernel PoC analyzer. (2) A kernel exploit generator.

Task-E1: Replicating userspace environment (5 months)

Team: Yeongjin Jang (coordinator) and Taesoo Kim

General description: Develop a mechanism that replicates the userspace environment of a target host based on the software stack information provided by TA1.

Detailed description: We will extend XSHOP to build a userspace replica environment by matching dependencies accurately. We will extend XSHOP to support a more general architecture (e.g., X86, ARM and PowerPC) in order to support a variety of infected devices.

Measurable evaluation: We will measure software compatibility (a list of userspace elements and systems that we can replicate), construction efficiency (the time and storage for replica building), and the exploit compatibility (the success rate of the generated exploits on the real host).

Deliverables: (1) A binary/library dependency resolver. (2) A software configuration dependency resolver. (3) A system that generates replica containers for various architectures.

Task-E2: Replicating kernel environment (5 months)

Team: Yeongjin Jang (coordinator) and Taesoo Kim

General description: Develop a mechanism that replicates the kernel space environment of a target host based on the software stack information provided by TA1.

Detailed description: We will (1) tailor and extend KUP to support UNIX-like operating systems on multiple architectures and (2) develop a kernel image replacement mechanism based on virtual machine snapshots to support proprietary operating systems, such as Windows and MacOS.

Measurable evaluation: We will evaluate software compatibility (a list of OS kernels that we can

reliably update), construction efficiency (the time and storage to construct a replica), and exploit compatibility (the success rate of the generated exploits on the real host).

Deliverables: (1) A kernel update mechanism based on KUP. (2) A virtual machine based kernel update mechanism for supporting proprietary OSs.

Task-E3: Building a replica cloud system by integrating Task-E1 and Task-E2 (6 months)

Team: Yeongjin Jang (coordinator) and Taesoo Kim

General description: Integrate both kernel and userspace replica environment generation system to support full-chain exploit testing.

Detailed description: We will integrate mechanisms for building both userspace and the kernel space replica, and put the system on the cloud infrastructure to extend its scalability. The integrated system will be used for generating full-chain exploits and testing/enhancing exploits' reliability.

Measurable evaluation: We will measure device compatibility (a list of host devices that we can replicate), construction efficiency and scalability (the time, the storage to construct a replica in the cloud), and exploit compatibility (the success rate of full-chain exploits on the real host).

Deliverables: (1) XGRIND, a replica environment generation system (both for kernel and userspace). (2) An orchestration mechanism for replica environment generation in the cloud infrastructure.

Task-A1: Building an environment for static malware analysis (16 months)

Team: Wenke Lee (coordinator), Salvatore Stolfo, and Yeongjin Jang

General description: Build an environment for analyzing the static behavior of malware.

Detailed description: We will create a tool to transform a binary into the LLVM IR representation. Then we will develop several static analyses on LLVM IR to understand malware's behavior, such as the CFG, communication with the C2 server, and commands handling.

Measurable evaluation: To measure the lifting correctness, we compile the lifted malware into binary and then compare its execution with the original one. We also measure the logic detection accuracy regarding the precision and recall for both known and unknown malware samples.

Deliverables: (1) A binary lifting platform that can lift malware binary into the LLVM IR. (2) A prototype environment to perform static malware analysis.

Task-S1: Synthesizing the Trusted Agent Base (16 months)

Team: William R. Harris (coordinator) and Wenke Lee

General description: Build the trusted component base of the XAGENT.

Detailed description: We create (1) the standard utilities for network scanning, malware identification, and communication with the PRESERVE and (2) a shim component that will sandbox the safe copy of the malware. In this stage, we will build a general shim that enforces a strong sandbox policy.

Measurable evaluation: We run it in an artificial botnet to test its functions on network scanning and malware identification; we run an artificial program in the shim to perform as many privileged functionalities as possible to test the sandbox strength.

Deliverables: The standard utilities and the general shim code.

Task-V1: Building a partial program model from software testing **(16 months)**

Team: William R. Harris (coordinator) and Wenke Lee

General description: Design a tool that, given a malware binary, generates a partial CFG of the binary to include control paths where the binary interacts critically with the target botnet.

Detailed description: Design the tool to discover malware control paths using a combination of random and directed testing, and construct a partial CFG from explored paths.

Measurable evaluation: Evaluate the ability of the model constructor to generate a control-flow graph that models relevant control paths of known malware.

Deliverables: A command-line utility that generates a partial CFG for a given malware binary.

Phase 2: Development (16 months)

In Phase 2, we will further develop matured components that can generate reliable exploits and safe-copying agents in automated manner. RENAISSANCE will extend to be able to attack on various types of architectures as well as real-world botnet cases.

Task-U3: Generating reliable exploits for targeting software **(12 months)**

Team: Brendan Saltaformaggio (coordinator) and Taesoo Kim

General description: We will generate exploits based on the extraction and combination of semantic information from the collected vulnerability data.

Detailed description: We will build a symbolic execution framework to generate robust exploits by selectively incorporating dynamic analysis. It will extract restrictions on vulnerabilities, induce crashes from CSS CVEs, and package exploit execution paths into building blocks for TA3.

Measurable evaluation: We will evaluate the robustness of user-level exploit generation by building 100 exploits against selected OSS and CSS targets. We will measure its efficiency regarding analysis time, complexity of symbolic constraints, and percentage of exploitable constraints.

Deliverables: (1) PRESERVE symbolic execution framework. (2) Evaluation results.

Task-U4: Extracting pre- and post-conditions of vulnerabilities **(4 months)**

Team: Brendan Saltaformaggio (coordinator) and Taesoo Kim

General description: We will extract the preconditions and postconditions of exploits to ensure the stability of the exploits generated in Task-U3.

Detailed description: We will use precondition and postcondition information to efficiently prevent failures on exploit generation. (1) We improve the stability via precondition checks and (2) provide side-effect mitigation via postcondition checks.

Measurable evaluation: We will first evaluate the stability provided by the addition of precondition and postcondition validation. We will then evaluate what conditions most benefit the successful deployment of exploits and reveal limitations of our exploit generation techniques.

Deliverables: (1) Target precondition generator. (2) Target postcondition generator. (3) Experimental results of precondition and postcondition validation.

Task-K3: Hardening the exploit by avoiding kernel-level crash **(8 months)**

Team: Taesoo Kim (coordinator) and Brendan Saltaformaggio

General description: Enhance working exploits to reliable exploits that are crash-resistant, self-adapted, traceless, and able to bypass modern kernel-level defenses.

Detailed description: We will embed various exploit techniques into PRESERVE to bypass kernel-level defenses (e.g., SMEP and KASLR). PRESERVE will generate configuration-agnostic exploits and will clean up the kernel, fix potential threats and achieve persistent control via hot patching.

Measurable evaluation: We will show that the exploits generated by PRESERVE can still work on target machines equipped with modern kernel-level defenses. We will also evaluate the success rate of the kernel exploits generated by PRESERVE.

Deliverables: Flexible exploit modules for (1) bypassing kernel-level defenses, (2) de-randomizing kernel space and building ROP chain, and (3) cleaning up and patching kernel.

Task-K4: Handling binary-only COTS OS kernels (8 months)

Team: Taesoo Kim (coordinator) and Brendan Saltaformaggio

General description: Extend PRESERVE to binary-only COTS OS kernels (i.e., Windows) to cover a wider range of target hosts and vulnerabilities.

Detailed description: We will extract patches by contrasting unhatched/patched binaries to enrich the vulnerability database. We will use Intel PT to speed up fuzzing for binary-only COTS kernels. We will apply the boundary-aware and context-sensitive mechanism [11] to scale PRESERVE.

Measurable evaluation: To evaluate how effectively PRESERVE can find crashes in binary-only COTS OS kernels such as Windows, we will measure (1) the overall system performance as in Task-K1 and (2) the effectiveness of each technique.

Deliverables: (1) The extended vulnerability information collector. (2) The Intel PT module for speeding up hybrid fuzzing. (3) Generated PoCs as database. (4) Evaluation results.

Task-A2: Building an interactive environment for dynamic malware analysis (8 months)

Team: Wenke Lee (coordinator), Salvatore Stolfo, and Yeongjin Jang

General description: Create an interactive virtual environment to execute the malware and reveal its hidden behavior dynamically.

Detailed description: We first create a virtual runtime to contain and detect malicious behaviors. We then symbolically execute malware's conditional branch to figure out the condition to unfold the malicious behavior and then trigger malware's hidden behavior interactively.

Measurable evaluation: We will measure (1) correctness by feeding inputs satisfying the conditions to the malware to see whether or not the malware reveals new behavior and (2) performance as the speed and scalability of our analysis framework.

Deliverables: A malware analysis environment that can reveal malware's hidden functionalities.

Task-A3: Building an automatic malware analysis framework (8 months)

Team: Wenke Lee (coordinator), Salvatore Stolfo, and Yeongjin Jang

General description: Automatically analyze the malware behaviors and classify the malware.

Detailed description: We tailor the interactive environment to analyze the malware and collect its behavior as much as possible, like the communication mechanisms with the C2 server, the valid commands. We then provide a tool to classify malware based on the collected information.

Measurable evaluation: We use the known malware to measure the effectiveness of the behavior collection and the classification.

Deliverables: A framework for automatic malware analysis and classification.

Task-S2: Safe-copying malware from execution trace (8 months)

Team: William R. Harris (coordinator) and Wenke Lee

General description: We will execute the malware in the analysis environment and create a malware copy that retains only the functionality to communication with the C2 server.

Detailed description: We run the malware in the analysis environment to collect the execution trace for communicating with the C2 server. We use the instructions in the trace to build a program and fill the missed instructions with illegal instructions.

Measurable evaluation: (1) We initialize the communication from the C2 server to the safe copy, and watch the response. The received messages should be the same as those from the original malware. (2) We feed arbitrary input to the safe-copy to detect unhandled exceptions.

Deliverables: A platform that efficiently synthesizes the safe-copy of the given malware.

Task-S3: Circumventing side-effects of synthesized agents (8 months)

Team: William R. Harris (coordinator) and Wenke Lee

General description: Instrument the synthesized agent to avoid potentially harmful side-effects on the file system of the infected device.

Detailed description: We instrument the agent to create a copy of the accessed file once the file is opened for writing. The shadow file system is accessible only by the agent and is deleted when the agent has neutralized the infected device.

Measurable evaluation: (1) We evaluate the correctness of the communication by comparing the packets received by the C2 server with and without the shadow file system. (2) We evaluate the effectiveness by comparing the file system status before and after the execution of the agent.

Deliverables: A platform to inspect and instrument the agent for minimal file-system side-effects.

Task-V2: Developing a synthesizer of verified agents (16 months)

Team: William R. Harris (coordinator), Wenke Lee, and Salvatore Stolfo

General description: Extend XAGENT to synthesize agents that are provably safe and effective.

Detailed description: Extend XAGENT to generate a verifiable shim that it completely monitors the control flow of an original malware, completely intermediates system side-effects, and makes progress toward disabling a botnet. Verify implementations of all manually-developed routines.

Measurable evaluation: Evaluate the ability of XAGENT to generate proofs of control monitoring efficiently. Evaluate if the architecture of instrumented malware enables us to develop a manually verified library of agent utilities.

Deliverables: An extension of XAGENT as a command-line utility that synthesizes an agent accompanied by an independently-certifiable proof that the agent is safe and effective.

Phase 3: Integration (16 months)

In Phase 3, we will further integrate RENAISSANCE components into an end-to-end system and prove RENAISSANCE's capability by taking down real-world botnets.

Task-UK1: Integrating userspace and kernel exploits **(6 months)**

Team: Taesoo Kim (coordinator), Brendan Saltaformaggio and Yeongjin Jang

General description: Combine userspace exploit and kernel exploit into a full exploit chain.

Detailed description: After gaining access to a host via userspace exploits, PRESERVE will collect various kernel information, like installed kernel modules. PRESERVE will accordingly develop a targeted kernel exploit based on this information and launch it on the target machine.

Measurable evaluation: We will show that PRESERVE can smoothly combine userspace and kernel exploits together for any target botnet machines. We will also evaluate the overall success rate of the full exploit chain developed by PRESERVE.

Deliverables: (1) A user level probe that helps to develop targeted kernel exploits. (2) Evaluation results on the success rate of the full exploit chains with different target machine settings.

Task-UK2: Infiltrating hosts infected by botnet malware **(6 months)**

Team: Taesoo Kim (coordinator), Brendan Saltaformaggio, and Yeongjin Jang

General description: Apply PRESERVE to victim machines infected by real world botnets.

Detailed description: We will collect specifications of real-world botnet samples, emulate botnet hosts according to the specifications, and apply PRESERVE to infiltrate into botnet hosts and test PRESERVE's capability for deploying TA3 agents on the host.

Measurable evaluation: We will evaluate PRESERVE's ability to infiltrate real-world botnet hosts efficiently and silently by measuring (1) the success rate of launching exploits generate by PRESERVE and (2) time and resource taken to create and launch exploits to emulated hosts.

Deliverables: (1) Evaluation testbed. (2) Evaluation results with emulated hosts.

Task-ASV1: Developing a functional synthesizer of agents **(6 months)**

Team: William R. Harris (coordinator), Wenke Lee, and Salvatore Stolfo

General description: Develop an agent synthesizer that, given a body of malware on an infected system and library of exploits, synthesizes an agent that uses the exploits to disable the botnet.

Detailed description: Develop an iterative synthesizer, XAGENT, to synthesize an agent as a malware instrumented with a shim that monitors malware's control flow using a combination of code fragments in the original malware and a manually-designed library of utilities.

Measurable evaluation: Evaluate how effectively XAGENT can discover malware control paths that implement desired functionality and mitigate harmful side-effects of modified malware. Evaluate whether a small, fixed set of routines can implement basic agent tasks in proof-of-concept scenarios.

Deliverables: An agent synthesizer that takes a malware collected from a target botnet and synthesizes an agent that disables the target botnet.

Task-ASV2: Deploying agents to disable a real-world botnet **(6 months)**

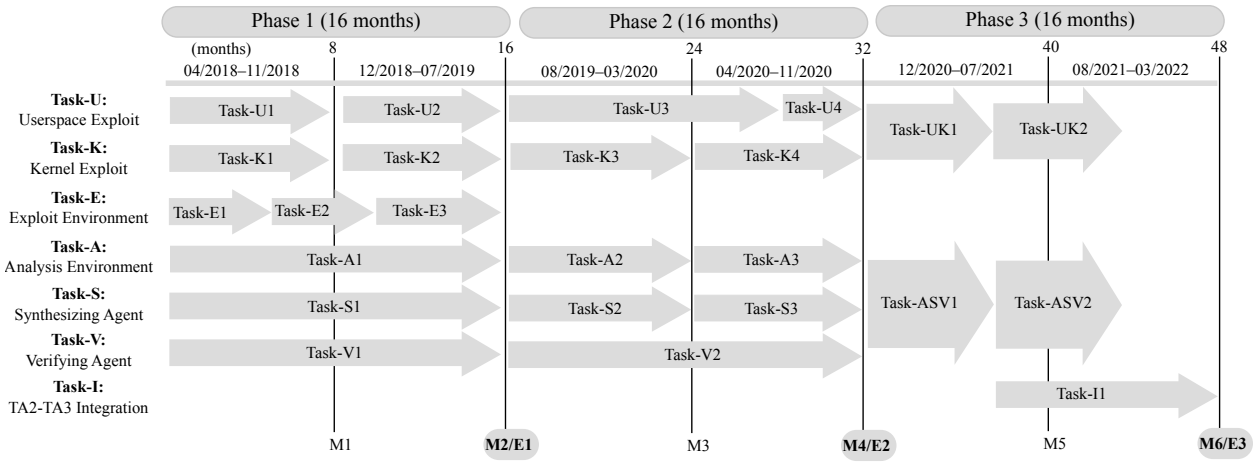

Figure 2: Schedule and milestones for the proposed work.

Team: William R. Harris (coordinator), Wenke Lee, and Salvatore Stolfo

General description: Apply agents synthesized by XAGENT to real-world botnets and evaluate their ability to disable real-world botnets.

Detailed description: We will run the following operations for the agents synthesized by XAGENT: (1) build a secure analysis environment, (2) analyze bot malware to identify their malicious actions and build the safe-copy, and (3) apply XAGENT to neutralize the botnet.

Measurable evaluation: We will evaluate how XAGENT neutralizes real-world botnets safely and autonomously by measuring (1) decrease of the malicious actions when applying XAGENT and (2) the time and resources used for neutralization regarding the botnet size.

Deliverables: (1) Evaluation testbed. (2) A report of case studies with the real-world botnets.

Task-I1: Integrating the entire operation of TA2 and TA3 (10 months)

Team: Yeongjin Jang (coordinator) and Taesoo Kim

General description: Integrate the components of both TA2 and TA3 to make them interactively work on taking down real-world botnets.

Detailed description: We integrate TA2 and TA3 by providing interfaces to make them interactively work on taking down real-world botnets. On the TA2 system, we will create an interface that XAGENT can send fingerprinted information for the hosts in private networks and retrieve generated exploits for them. On the TA3 system, we will create an interface that generates an intermediate TA3 agent that safely launches the exploit-and-gather step of the TA2 system upon request. Optionally, our TA3 system will provide a proxy interface to TA1 systems that enables reusing their technology for fingerprinting hosts behind private networks.

Measurable evaluation: We will measure the enhanced take-down operation by comparing the number of bots taken down by RENAISSANCE before and after the integration, and lateral movement by testing integrated systems against botnets constructed over a private network to demonstrate its capability for taking down botnets by lateral movement.

Deliverables: (1) An integrated system of TA2 and TA3. (2) A TA3 interface to TA1 systems.

Schedule and Milestones

Figure 2 shows the schedule of the proposed tasks (see §11) and corresponding milestones with success criteria. In Phase 1, we will build the foundation of RENAISSANCE's key components:

M1: Task-U1, Task-K1, Task-E1/2 **(End of 8 months)**

(1) Building a toolkit for crawling user-level vulnerabilities, OSS libraries, kernel vulnerabilities, and kernel crashes; and (2) environment for testing user-level exploits.

Success criteria. (1) A ready-to-use database for user- and kernel-level vulnerabilities, patch history, and crashes. (2) The userspace environment replication framework.

M2: Task-U2, Task-K2, Task-E3, Task-A1, Task-S1, Task-V1 **(End of 16 months)**

(1) Detecting the vulnerabilities in the indexed user-level software; (2) developing full-fledged exploits for local privilege escalation; (3) building an environment for kernel exploit testing; (4) implementing a prototype environment for malware behavior analysis; (5) safe copying malware from execution trace; and (6) applying approximate model of a program from software testing.

Success criteria. (1) Algorithm for automatic kernel exploit generation from a PoC and its crash dump. (2) A rapid kernel replication framework. (3) Algorithm for malware behavior analysis. (4) A prototype to demonstrate safe-copying malware.

E1: End-to-end integration and testing **(End of 16 months)**

Our team will integrate the project and perform field exercises using a new or unknown botnet to see if our design can efficiently infiltrate target systems and take down well-known botnet agents.

Success criteria. (1) Robust exploit on the specified systems that are running the indexed software. (2) Successful safe-copying malware deployment and neutralization of malicious agent.

In Phase 2, the main goal is to develop matured components that can generate reliable exploits and safe-copying agents in an automated manner, as described as follows:

M3: Task-K3, Task-A2, Task-S2 **(End of 24 months)**

(1) Hardening the exploit by avoiding kernel-level crash; (2) building secure and repeatable experiment environment; and (3) recovering missing code by monitoring malware behavior.

Success criteria. (1) Develop a module for bypassing cutting-edge kernel-level defense and generating robust kernel exploit. (2) Build a safe botnet testing environment. (3) Automatically synthesize executable safe-copying malware.

M4: Task-U3/4, Task-K4, Task-A3, Task-S3, Task-V2 **(End of 32 months)**

(1) Generating reliable exploits for targeting software; (2) extracting pre- and post-conditions of vulnerabilities; (3) handling binary-only COTS OS kernels; (4) fine-grained monitoring behavior of bot malware; and (5) performing verified instrumentation to control behavior of malware.

Success criteria. (1) Build PRESERVE's own symbolic execution environment. (2) Evaluate correctness of exploit on OSS/CSS software. (3) Evaluate effectiveness of pre- and post-condition checker. (4) Provide PT module for hybrid kernel fuzzing with experiment result.

E2: End-to-end integration and testing **(End of 32 months)**

Our team will integrate the project and perform field testing to see if how RENAISSANCE can efficiently infiltrate a real-world botnet, promptly analyze preconditions and postconditions, adaptively generate exploits, and take down running bot agents by safely replacing them with cloned malware.

Success criteria. (1) TA2 achieves successful user-level exploit and privilege escalation through kernel exploit for N-day vulnerabilities used in known malware samples. (2) TA3 achieves a safe-copying of botnet malware samples using the developed malware analysis.

In Phase 3, the main goals are to integrate RENAISSANCE's components into an end-to-end system and demonstrate RENAISSANCE's capability by taking down real-world botnets under various operating requirements and conditions:

M5: Task-UK1, Task-ASV1 **(End of 40 months)**

(1) Integrating userspace and kernel exploits for a specified botnet machine and (2) rapidly generating verified agents from neutralized malware using side-effect mitigation and operation verification schemes.

Success criteria. (1) Generate a user-level probe that contributes to developing a targeted kernel exploit. (2) Correctly synthesize a full-chain exploit with user- and kernel-level exploits. (3) Build an end-to-end system for synthesizing verified agent.

M6: Task-UK2, Task-ASV2, Task-I1 **(End of 48 months)**

(1) Evaluating the neutralization techniques by measuring decrease of the malicious actions, time taken to fully neutralize depending on the size of the botnets, resource usage of victim machines in the procedure and (2) initiating the TA2 and TA3 integration.

Success criteria. (1) Demonstrate an end-to-end system with TA2 and TA3 capabilities. (2) An entire process should be demonstrated without causing any crash or side-effects.

E3: End-to-end integration and testing **(End of 48 months)**

Our team will complete the implementation of RENAISSANCE and perform an end-to-end integration and testing to show that it can efficiently infiltrate real-world botnets and can deploy clone-based botnet agents under realistic environment constraints and operational requirements. In the final exercise, we will show that RENAISSANCE can analyze an unknown botnet and counter the botnet via N-day exploits by synthesizing and deploying a verified agent.

Success criteria. (1) Completing an end-to-end system. (2) Showing that automatically generated N-day exploits are effective. (3) Showing that the neutralization and verification techniques are effective in synthesizing an agent based on a malware sample.

References

[1] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *Proceedings of 19th USENIX Security Symposium (USENIX Security '10)*, 2010.

[2] M. Antonakakis, R. Perdisci, W. Lee, D. Dagon, and N. Vasiloglou. Detecting Malware Domains at the Upper DNS Hierarchy. In *Proceedings of 20th USENIX Security Symposium (USENIX Security '11)*, 2011.

[3] R. Chandra, T. Kim, M. Shah, N. Narula, and N. Zeldovich. Intrusion recovery for database-backed web applications. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, Cascais, Portugal, Oct. 2011.

[4] R. Chandra, T. Kim, and N. Zeldovich. Asynchronous intrusion recovery for interconnected web services. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)*, Farmington, PA, Nov. 2013.

[5] H. Chen, T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek. Identifying information disclosure in web applications with retroactive auditing. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Broomfield, Colorado, Oct. 2014.

[6] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu. iRiS: Vetting private api abuse in ios applications. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, Colorado, Oct. 2015.

[7] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, 2008.

[8] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security '07)*, 2007.

[9] G. Gu, , R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure- independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security '08)*, 2008.

[10] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proc. Network and Distributed System Security (NDSS)*, 2008.

[11] S. Y. Kim, S. Lee, I. Yun, W. Xu, B. Lee, Y. Yun, and T. Kim. CAB-Fuzz: Practical Concolic Testing Techniques for COTS Operating Systems. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC)*, Santa Clara, CA, July 2017.

[12] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek. Intrusion recovery using selective re-execution. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, Oct. 2010.

[13] T. Kim, R. Chandra, and N. Zeldovich. Recovering from intrusions in distributed systems with Dare. In *Proceedings of the 3rd Asia-Pacific Workshop on Systems (APSys)*, Seoul, South Korea, July 2012.

[14] Y. Kwon, B. Saltaformaggio, I. L. Kim, K. H. Lee, X. Zhang, and D. Xu. A2c: Self destructing exploit executions via input perturbation. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb.–Mar. 2017.

[15] B. Lee, L. Lu, T. Wang, T. Kim, and W. Lee. From Zygote to Morula: Fortifying weakened ASLR on Android. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2014.

[16] B. Lee, C. Song, Y. Jang, T. Wang, T. Kim, L. Lu, and W. Lee. Preventing use-after-free with dangling pointers nullification. In *Proceedings of the 2015 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2015.

[17] B. Lee, C. Song, T. Kim, and W. Lee. Type Casting Verification: Stopping an emerging attack vector. In *Proceedings of the 24th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2015.

[18] K. Lu, C. Song, B. Lee, S. P. Chung, T. Kim, and W. Lee. ASLR-Guard: Stopping Address Space Leakage for Code Reuse Attacks. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, Colorado, Oct. 2015.

[19] K. Lu, C. Song, T. Kim, and W. Lee. UniSan: Proactive Kernel Memory Initialization to Eliminate Data Leakages. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*

(CCS), Vienna, Austria, Oct. 2016.

[20] C. Min, S. Kashyap, B. Lee, C. Song, and T. Kim. Cross-checking Semantic Correctness: The Case of Finding File System Bugs. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP)*, Monterey, CA, Oct. 2015.

[21] B. Saltaformaggio, R. Bhatia, X. Zhang, D. Xu, and G. G. Richard III. Screen after Previous Screens: Spatial-Temporal Recreation of Android App Displays from Memory Images. In *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014.

[22] B. Saltaformaggio, Z. Gu, X. Zhang, and D. Xu. DSCRETE: Automatic Rendering of Forensic Information from Memory Images via Application Logic Reuse. In *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014.

[23] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. GUITAR: Piecing Together Android App GUIs from Memory Images. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, Colorado, Oct. 2015.

[24] B. Saltaformaggio, R. Bhatia, Z. Gu, X. Zhang, and D. Xu. VCR: App-Agnostic Recovery of Photographic Evidence from Android Device Memory Images. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, Colorado, Oct. 2015.

[25] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee. Eureka: A framework for enabling static malware analysis. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS '08)*, 2008.

[26] I. Yun, C. Min, X. Si, Y. Jang, T. Kim, and M. Naik. APISan: Sanitizing API Usages through Semantic Cross-checking. In *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016.